

B E T S Y

Deliverable D2a

Report on temporal impact of identified entities on end-to-end-timing

Public

Project Number : IST-004042

Project Title : BETSY
Deliverable Type : Report

Deliverable Number : D2a
Title of Deliverable : Report on temporal impact of identified entities on end-to-end-timing
Nature of Deliverable : Report, Confidential
Internal Document Number : BETSY D2a-temporalimpact.doc
Contractual Delivery Date : 31 August 2005
Actual Delivery Date : 14 September 2005
Contributing WPs : WP2
Author(s) : D. Iovic, G. Fohler (MDH) A. Prayati, C. Koulamas, G. Papadopoulos (ISI) J-D. Decotignie, M. Sénéclauze (CSEM) M. Joosten, R. Bernhardi (C-LAB) M. Geilen, B. Theelen, T. Basten (TU/e) C.Otero Perez, L. Steffens, P. van der Stok (Philips) C. Blanch, K. Denolf (IMEC)

Abstract

Processing of high-quality streams is quite computationally expensive at the same time as video processing applications are more and more deployed in small embedded systems that traditionally exhibit limited processing and network resources, e.g. set top boxes, mobile phones or PDAs. Hence, for some combinations of streams and devices, the timing constraints on stream processing within a functional component will not automatically be met.

This task of WP2 analyses the identified functional components of a breeze processing and transmission with respect to their impact on the timing of end-to-end delivery chain processing. In particular it identifies parameters at a high level of abstraction, i.e., hiding the full internal details (which are addressed in WP3). The analysis is based on the list of devices selected in WP1 and the resources identified in close collaboration with WP3.

WP2-3 results can be used to make trade-offs between timeliness, resource usage and energy consumption at system design time.

Keyword list

timing constraints, end-to-end delivery chain, high-level parameters, temporal aspects of functional components

Table of Contents

Table of Contents	2
List of figures	5
1 Introduction	6
1.1 Position of the deliverable in project	6
1.2 Document organization and approach	7
2 Definitions and Approach	8
2.1 Resources.....	8
2.2 Temporal aspects	8
2.2.1 Timing constraints.....	8
2.2.2 Decomposition of constraints.....	9
2.3 Approach	9
3 Capturing	10
3.1 Function description	10
3.2 Resources consumed.....	11
3.2.1 Memory	11
3.2.2 Energy	11
3.2.3 CPU	11
3.3 Temporal aspects	11
3.3.1 Latency	12
3.3.2 Buffer overflow and underflow	12
3.3.3 Timing constraints.....	12
3.4 Example devices	13
4 Encoding	14
4.1 Function description	14
4.2 Resources consumed.....	14
4.2.1 CPU	14
4.2.2 Memory	14
4.2.3 Energy	15
4.3 Temporal aspects	15
4.3.1 Latency	15
4.3.2 Buffer underflow and overflow	16
4.3.3 Encoder rate constraints: Rate control mechanism	16
4.3.4 Timing constraints for Encoding	19
4.4 Example devices	20

5	Multiplexing	21
5.1	Function description	21
5.1.1	FlexMux	22
5.2	Resources consumed	22
5.3	Temporal aspects	23
5.3.1	Timing constraints for multiplexing	23
5.4	Example devices	24
6	Transporting	25
6.1	Function description	25
6.2	Resources consumed	25
6.2.1	CPU	26
6.2.2	Memory	26
6.2.3	Bandwidth	26
6.2.4	Energy	26
6.3	Temporal aspects	26
6.3.1	Error management	27
6.3.2	Timing constraints for transporting	28
6.4	Example devices	28
7	Delay buffering	29
7.1	Function description	29
7.2	Resources consumed	29
7.3	Temporal aspects	29
8	Demultiplexing	31
8.1	Function description	31
8.2	Resources consumed	32
8.3	Temporal aspects	32
8.3.1	Timing constraints for demultiplexing	33
8.4	Example devices	33
9	Decoding	35
9.1	Function description	35
9.2	Resources consumed	35
9.2.1	CPU	35
9.2.2	Memory	36
9.2.3	Energy	36
9.3	Temporal aspects	36
9.3.1	Latency	36
9.3.2	Buffer underflow and overflow	37

9.3.3	Timing constraints for decoding.....	38
9.4	Example devices	39
10	Rendering.....	41
10.1	Function description	41
10.2	Resources consumed.....	41
10.2.1	Processor.....	41
10.2.2	Memory.....	41
10.2.3	Communication bandwidth	41
10.2.4	Energy	42
10.3	Temporal aspects	42
10.4	Example devices	42
11	Temporal aspects table.....	43
12	Conclusion	48
	References	49

List of figures

Figure 2-1: Decomposition of end-to-end constraints	9
Figure 3-1: Digital sensor image interface	10
Figure 4-1: Encoding Model.....	14
Figure 4-2: Delay components of a communication system	15
Figure 4-3: Delay constraints in live video case.....	17
Figure 4-4: Bit rate variation in Foreman	18
Figure 4-5 : Smoothing of variable bit rate.....	18
Figure 5-1: Multiplexing function usage models.....	21
Figure 6-1: Transporting function usage models	25
Figure 6-2: Initial end-to-end latency of Transporting function.....	26
Figure 7-1: Delay buffering creation	29
Figure 8-1. Demultiplexing function usage models	31
Figure 9-1: MPEG processing model.....	35
Figure 9-2: End-to-end latency for MPEG playout.....	37
Figure 9-3: Input buffer overflow and underflow	38
Figure 10-1 Rendering chain	41

1 Introduction

In the BETSY project, the transport of a video stream from a producer to a number of consumers is decomposed into a number of functional components, such as encoding, transmitting, decoding etc. Each functional component can be parameterized and each parameter will have an impact on the end-to-end quality of the service of stream transport. Depending on these parameters, the entities will use more or less resources. For example, software decoding of high-quality video streams requires extensive processing power due to complex coding techniques, more memory for buffering the input stream and decoded frames, and more energy consumption.

At the same time, video and audio, as well as stream processing in general, have throughput requirements and real-time deadlines. For example, decoding a 25 fps (deinterlaced) video stream requires periodically a newly decoded frame every 40 ms. Delays in this process may result in severe video quality degradation of the played stream.

Hence, for some combinations of streams and devices, the timing constraints on stream processing within a functional component will not automatically be met. Furthermore, in order to fulfill end-to-end timing constraints on a global basis, devices and resources have to meet specific constraints locally.

In this task, we analyze the identified functional components for their impact on end-to-end timing and look into trade-offs between resources imposed by the function of the component and how those trade-offs influence the timing of performed operations.

For each component we identify parameters and constraints that influence the timing of the functions performed by the component, such as display times of frames, latencies, jitter and reactions to changes imposed by the system status, e.g., resource availability, user input or quality change. Proposed timing constraints are then analyzed for their influence on end-to-end timing of the delivery chain of a breeze.

1.1 Position of the deliverable in project

The research described in this deliverable has been performed on the basis of input from WP1 and WP3. WP1 provided a set of different functional components operating on a breeze (D1b), while WP3 identifies the concrete resources involved in the production, transport and display of multimedia streams in an extended home context (D3a). We use the results and findings from both work packages to analyze functional components for their temporal behavior and their impact on end-to end timing of the breeze delivery chain.

WP2 identifies high-level parameters that have an impact on the end-to-end timing, such as CPU load, available network bandwidth. It studies "resource knobs" to turn on a system wide level, while WP3 goes into the device internal, detailing the implementation of these "knobs" for the devices. Close collaboration with WP3 ensured that the results of D3a - i.e. the functional component internals and more specifically their resource models – match/interface well with the high-level parameters identified in this task.

In particular, the work presented here is closely related to deliverable D3a [7] of WP3. In collaboration with WP3, we have analyzed the functional components of D1b [6] and proposed a methodology to decouple resources into abstract resources and concrete resources, and provided a set of parameters for both of them. D3a analyses concrete resources for their aspects of the usage demand, the settings of the resource and their intrinsic attributes, while D2a analyzes the trade-offs of the abstract, high-level resources for their impact on end-to-end timing. We refer to D3a [7] for full details on the classification of the resources as well as their parameters and settings.

Together with D1b and D3a, D2a completes the initial “inventory work” for BETSY, i.e., defining the scope of devices and resources, providing a common understanding, which is essential for the further project work across WPs. WP2-3 results will be used in WP4 to make trade-offs between timeliness, resource usage and energy consumption at system design time. This way the models of WP2 and WP3 are not only validated, but also put into very practical use, making industrial exploitation of the research results much easier.

1.2 Document organization and approach

The document is organized as follows: First we define the framework for representing resource demands and timing constraints of functional component. In following chapters, we give details about temporal behavior of each functional component and describe how individual timing constraints of the components influence end-to-end delivery chain. For each functional component identified in WP1, we analyze all high-level resources that are consumed by the component for their temporal behavior, identifying the timing constraints imposed by the component on the end-to-end timing of breeze processing and transmission. This is followed by a table overview of temporal aspects for each component, with parameters that influence timing of the functions performed by components.

2 Definitions and Approach

Here we define the framework for representing resource demands and describe the approach for obtaining timing constraints from identified temporal aspects of functional components – as defined in D1b and D3a.

2.1 Primary resources

To be able to realize the functions, the components employ their resources. Different devices have different kinds of resources, and resources can also be shared between active components. Spending different amounts of resources on functions may result in the function having different qualities.

To be able to make trade-offs at the level of the entire Breeze, these device dependent resources have to be represented as generic types of resources. They need to have high level of abstraction, hiding the full internal details of the devices. This way we can describe a system in functional entities and their needed resources, temporal and power consumption characteristics without actual hardware characteristics, which will be addressed in WP3.

The following high-level resources (identified in exchange with D3a [7]) will be analyzed for their impact on timing:

- *CPU budget* – for a particular processing unit, possibly expressed as a fraction of the total processing capabilities. Through DVS or similar techniques, the available processing resources may be traded for reduced power consumption.
- *Network bandwidth* – expressed in bits per second or possibly with an intrinsic trade-off of BER vs. bandwidth vs. latency
- *Memory / storage* – For buffering and processing, memory may be needed to store or hold intermediate results.
- *Energy* – Various components spend energy on performing their functions in a streaming application. The energy spent in a device determines its battery life, although the relation between the profile of energy usage along the time axis and the actual battery life may be subtle.

Note that the availability of resources themselves may be subject to trade-offs. It may be possible to reduce the available resources in favor of reduced power consumption, but to the expense of quality.

2.2 Temporal aspects

One objective of this task is being able to express end-to-end timing constraints and decompose it to requirements on individual functional components.

2.2.1 Timing constraints

We can identify the following types of timing constraints:

- *End-to-end timing constraints* – these constraints describe the temporal demands between user action and user perception. They include *end-to-end deadlines* for breeze transport and processing, such as display times of frames, latencies, jitter and reactions to

changes imposed by the system status, e.g., resource availability, or user input, e.g., quality change

- *Device and resource timing constraints* – In order to fulfill end-to-end timing constraints on a global basis, devices and resources have to meet specific constraints locally. We will look into the timing constraints of functional components that operate locally on devices, and then identify their impact on end-to-end timing.

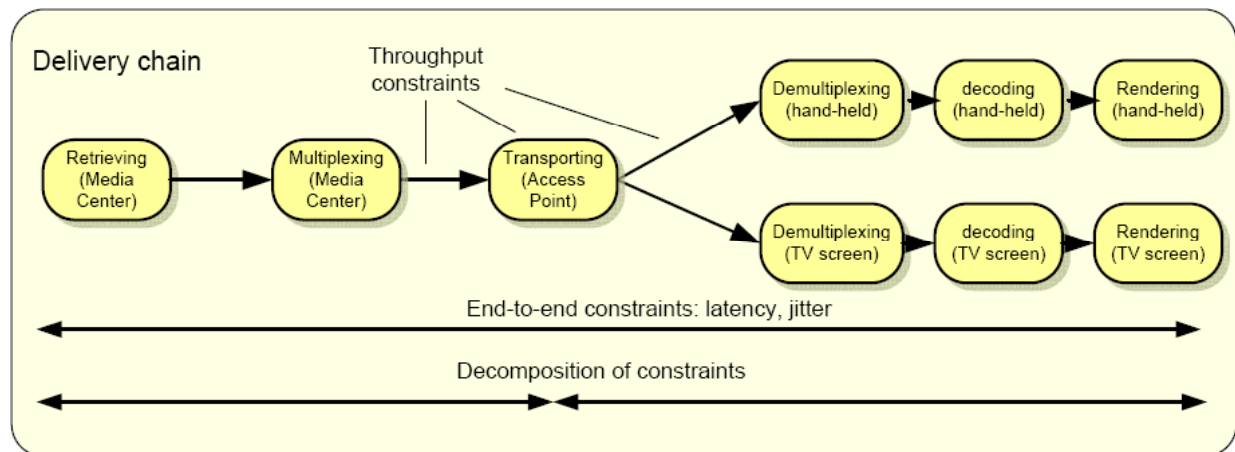


Figure 2-1: Decomposition of end-to-end constraints

2.2.2 Decomposition of constraints

Decomposition of constraints maps end-to-end requirements into device and resource constraints such that when all local constraints are met, the end-to-end timing of breezes is kept. It should be noted that such decomposition trades optimality of the solution with complexity of the process, which we believe has to go in favor of efficiency of the process.

Decomposition can be carried out offline or online: offline is less flexible with respect to changes in system state, but provides significantly better efficiency of the online process. It avoids determining the solution online as well as does not need to explicitly negotiate on a per device basis with the rest of the system.

2.3 Approach

For each component, we identify consumed resources and look into the trade-offs between resource usage. These trade-offs are then analyzed for their impact on temporal behavior of the functional components, in order to propose a set of timing constraints imposed by performing the functions of the components.

For example, if we take the decoding functional component, input buffer underflow will decrease decoding latency, while buffer overflow will increase it, directly influencing decoding start times and finishing times. We analyse how variations in available resource memory lead to latency variations and hence have impacts on the timing constraints for video frame decoding.

3 Capturing

3.1 Function description

Capturing means the acquisition of video data by means of an image sensor and its provision in memory buffers in a suitable format for the encoding. This function usually comprises two hardware components, the actual sensor module with a 4-8 bit wide data interface and a control interface, and a DMA controller or a suitable software task to receive the data and deposit them into memory.

The image sensor module contains the necessary hardware to control the resolution (number of rows and columns), the data format (RGB, YUV/YCbCr) with the pixel depth (RGB: 5:5:5, YUV: 4:2:2 or 4:2:0), and the frame rate (16, 25 or 50 fps). For still images and direct transfer to displays the RGB pixel format is preferred, whereas for video streaming and compression the YUV format is preferred which already provides a lossy reduction of the video data by favoring the brightness data (luminance, Y) over the color information (U and V, or Cb and Cr normalized) as does the human eye.

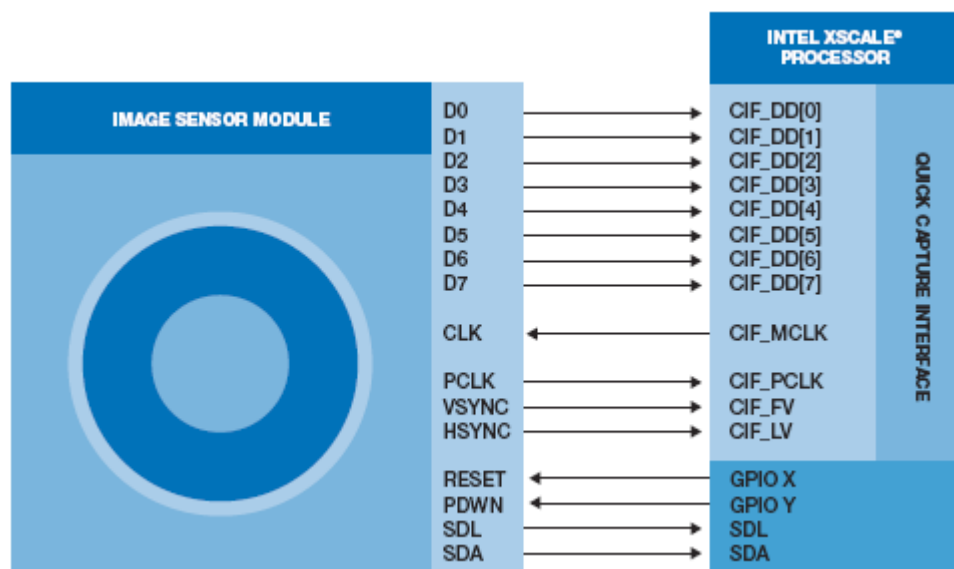


Figure 3-1: Digital sensor image interface

Once the transfer of one frame has started, the sensor transmits synchronously (usually with a separate clock signal) the whole frame in one swoop. This means, that depending on the selected resolution (640x480 or 320x240 for small 'movie' sensors) and the pixel format, $640 * 480 * 8-16\text{bits} = 300-600\text{Kbyte}$ must be processed for each frame, and there is no hand-shaking to slow down the transfer.

In fact, even more data is transferred, because the data format provides self-synchronization by means of blanking data after each line and after a frame, about 40 pixels per line and 5 lines per frame.

Consequently, this data stream is usually processed by a dedicated interface controller with FIFOs and DMA, integrated with the actual CPU in a SoC (System on Chip). In many cases, this controller (QuickCapture in Intel's XScale SoCs) can store the pixel data in either packed or planar format. Packed means that each pixel's color component values (RGB or YUV) are stored linearly in the memory, whilst with a planar format, the YUV data are stored in separate

blocks, one each for Y, U and V. Which one is more preferable depends on the implementation of the MPEG encoder.

3.2 Resources consumed

Capturing is mainly a bandwidth-intensive process, so few CPU resources are needed. But because of the memory-intensive nature of the capturing of the video stream, use of main memory, i.e. buffers, bus bandwidth and energy due to the switching activity of bus drivers are consumed.

3.2.1 Memory

As already stated in the functional description, capturing provides the video data in input frame buffers. Depending on the encoding process and the complexity of the video scene, the processing (encoding, multiplexing and transmission) of the video data might take longer than a frame duration for a short time, such that the frame rate is only met on average. Consequently, more than two input frame buffers (one to be written by the image sensor, the other to be read by encoding) are required to provide a FIFO to overcome short encoding bottlenecks. Again, this represents the typical tradeoff of memory buffers versus time or processing power.

In addition to the amount of memory consumed by input frame buffers, the high volume of data traffic ($600\text{Kbytes} * 25 \text{ fps} = 14.6 \text{ Mbytes/s}$) is a considerable share of the memory data rate. Assuming a 100MHz system/memory clock and 32 (4 bytes) wide data bus, the maximum (burst) memory data rate is about 400Mbytes/s, though in practice slower memory speeds are probably used to reduce the SDRAM's power consumption.

3.2.2 Energy

The capturing process consumes an almost constant amount of power, because of the constant data rate. A typical CMOS image sensor requires about 20-40mA at 2.5V (50-100mW) during capture, depending on the pixel clock. In active mode, the power consumption of the SDRAM is also almost constant, due to the usual data traffic and the refresh cycles, meaning that the image data traffic does not increase memory energy consumption in a significant way. The dynamic part of the SDRAM's I/O drivers is rather small (<10%) compared with the static part.

3.2.3 CPU

Depending on the actual hardware configuration, the CPU load is either very small if an integrated DMA controller is used, or quite considerable, if a polling scheme is implemented. It is not so much for the amount of processing as rather for the strict timing deadlines of the image sensor data stream which does not provide (asynchronous) handshaking. This requires a small periodic real-time task with high priority that reads the image data from a GPIO port or, better, some internal controller like QuickCapture providing FIFOs, and stores them into input frame buffers.

3.3 Temporal aspects

Capturing, as the source of media streams, provides data in a strictly periodic way. For video data with a frame rate of 25fps, this means that every 40ms about 300-600KB video data must be processed by Encoding. Buffering may be used to overcome short bottlenecks in encoding, if the compression of a few video scenes takes longer than their actual time, but this cannot take longer than few 100ms, because of the large buffer size.

3.3.1 Latency

The internal latency, that is the time between the start-of-capture command and the last pixel value sent, is fixed, depending only on setup parameters of the image sensor, like master clock, pixel format and frame resolution. These values determine the amount of time a whole frame needs to be transferred and, thus, also determine the maximal frame rate. This means that a higher master clock or lower frame rate leaves more time for subsequent processing before the next frame capturing starts. The time for the entire transfer can reach up to but not exceed the frame period, while the encoding and transmitting takes place in the next frame periods. Consequently, there is at least a latency of one frame period.

3.3.2 Buffer overflow and underflow

In order to decouple the capturing and the encoding process, at least two input frame buffers are required: One to be written by the capturing task or the DMA controller, the other to be read independently by MPEG-4 encoder. After the frame is processed, the buffers are exchanged.

In case the encoding takes longer because of the complexity of the current scene, additional input buffers can be used. But because of the large size of such buffers, this is a rather expensive option in terms of memory consumption, and only useful for very short processing bottlenecks, which, of course, cannot be anticipated.

In case of a buffer overflow, the current frame must be skipped, which might lead to noticeable jerkiness on the display at the end of the transmission chain.

A buffer underflow, i.e. the processing is finished before the next frame is captured, is the optimal situation, allowing in turn to reduce the CPU or coprocessor speed and conserve some energy.

Using buffers and dynamic frequency and voltage scaling (DVFS), the necessary average speed can be adjusted by comparing the required processing time with the frame period. If the processing time increases because of different video frame complexity, the input frame buffers buy some time to readjust the processing speed.

3.3.3 Timing constraints

Capturing, as the source of multimedia data, only imposes constraints on the subsequent processing components. Depending on the selected configuration of frame resolution and frame rate, the rate and amount of data is fixed and requires a strict deadline.

Start time constraints

Capturing is started by sending the image sensor the configuration (setup), setting up the capture and DMA controller or the capturing task, providing the necessary input frame buffers and sending the sensor a start command (usually by switching from IDLE into RUN mode). Line and Frame Valid signals indicate a complete read of a scan line or entire frame.

Finishing time constraints

The time of the last pixel value depends on the master clock and the selected frame resolution, which in turn determines the frame rate.

3.4 Example devices

Capturing requires dedicated hardware, namely the image sensor to convert light into a digital data stream, and usually a dedicated controller to accommodate the high data rate. Video capturing devices with a network connection are usually surveillance cameras. Such cameras usually provided only still images (JPEG) in short intervals, then MJPEG (Motion JPEG) and recently even MPEG-4 video streams via network streaming became possible, due to the increased multimedia processing performance of microcontrollers and digital signal processors (DSPs). Since these devices are designed for video streaming, the necessary hardware resources (memory, CPU/coprocessor, image sensor with interface) are generally provided, but usually only for a limited choice of configurations, in particular either a smaller resolution (QVGA, 320x240) with higher frame rate (25-30fps) or a large resolution (VGA, 640x480) with lower frame rate (10-15 fps). For surveillance purposes, reducing the frame rate in case of resource bottlenecks is more acceptable than reducing the resolution.

4 Encoding

4.1 Function description

To encode an MPEG video stream an input function is needed to provide the raw data to be encoded. This is usually done by a camera producing raw video data at a constant rate. Encoding means transforming raw digital information in an MPEG-4 Elementary Stream. During this process the data is compressed in order to fit in the available network bandwidth. The encoded data is placed in the Output encoder buffer at a variable rate (see Figure 4-1).

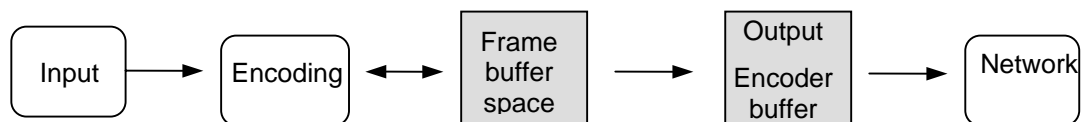


Figure 4-1: Encoding Model

4.2 Resources consumed

Factors such as the available processing power, the size and number of available buffers, the available energy (also dependent on the network conditions) and the available network bandwidth have an impact on the performance of the encoding process.

4.2.1 CPU

Encoding of high-quality video streams is a computationally expensive process. In general the processing power required during encoding can be up to one order of magnitude higher than the processing involved in the decoding [9].

The processing power required for video encoding is not constant but varies from frame to frame. This is partly due to the different types of frames: I, P and B with varied processing demands. Moreover the amount of motion and detail in the scene has a big effect on the associated coding complexity. The accuracy with which the texture information is encoded, given by the quantization parameter (QP), impacts as well the complexity of the encoding process of a particular frame as a higher amount of accesses to external memories is required increasing both encoding delay and energy consumption.

If the processor cannot be fast enough to meet the time constraints, the encoder can speed up by degrading the video quality. This can be done by quantizing more coarsely (by means of a higher QP), or by reducing the spatial or temporal resolution (frame rate). Clearly, frame skipping at the encoder results in a smaller quality loss than frame skipping at the decoder. Especially when motion compensated video coding is used, skipping frames at the decoder cannot be done without a heavy quality penalty.

4.2.2 Memory

Encoding requires an output buffer and frame buffers. The input buffer from which the encoder takes the input data is filled by the row data source at a regular rate.

Frame buffers are required to store the current frame to be coded and the reference frame (used for prediction of the current frame). When Bi-directional frames (B frames) are used,

these frames involve both backward and forward prediction so the memory requirements are increased by one extra frame memory.

The output bit rate of the encoder is variable as every frame takes a different amount of bits to be encoded based on the coding mode, quantisation parameter applied and motion and detail of the content. The output buffer of the encoder can serve to smooth the rate variations before the data is transmitted through the channel.

At the encoder the Video Buffer Verifying (VBV) model is used to model a hypothetical decoder buffer that will not overflow or underflow when fed with a conforming MPEG bit stream. Thus, part of the definition of a compliant stream is that it does not cause underflow or overflow of this model buffer. Parameters such as buffer size and fullness are transmitted to specify decoder buffer action according to this model. However, decoders do not have to use these quantities, but instead can rely on redundant information provided by time stamps.

4.2.3 Energy

The energy consumption for encoding a video frame will vary between different frames, as the complexity of the frame encoding varies. The energy consumption for frame encoding can be reduced by degrading the compression efficiency or the video quality. For example, the use of P frames instead of B frames will reduce the encoding energy but also the compression efficiency will decrease as P frames are predicted from only one reference frame. Degrading the video quality by means of increasing the quantization parameter or reducing the spatial resolution will also help reduce the energy consumption. Additionally increasing the encoding latency by reducing the processing speed will also decrease the energy but in order to meet the end timing constraints this will pose stronger requirements on both network and decoder latency.

4.3 Temporal aspects

Encoding of video is subject to throughput and real-time constraints. Furthermore the encoding latency has an impact on the end-to-end latency that needs to be respected to avoid severe quality degradation.

4.3.1 Latency

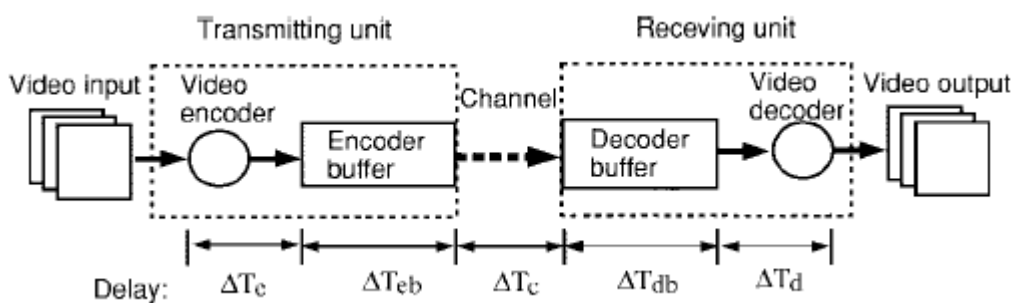


Figure 4-2: Delay components of a communication system

In typical video communications systems, the end-to-end delay each frame experiences (from the time it is obtained from the input video buffer to the time it is placed in the display video buffer) consists of several delay components. The total delay experienced by each frame (see Figure 4-2) can be broken up into:

$$\begin{aligned} \Delta T &= \Delta T_e \quad (\text{Encoder delay}) \\ &+ \Delta T_{eb} \quad (\text{Encoder buffer delay}) \\ &+ \Delta T_c \quad (\text{Channel delay}) \\ &+ \Delta T_{db} \quad (\text{Decoder buffer delay}) \\ &+ \Delta T_d \quad (\text{Decoder delay}). \end{aligned}$$

Different video applications have different requirements in terms of delay. In interactive video communications (e.g., video conferencing) low delay is required, while in one-way video transmission, (e.g., broadcast or video on demand) the end-to-end delay is only noticeable to the user as an initial latency, i.e., the time interval between the start of the video transmission session and the time the first video frame is displayed.

4.3.2 Buffer underflow and overflow

Buffer overflows or underflows of the decoder input buffer would highly degrade the video quality. The encoder uses the Video Buffer Verifier mechanism (VBV) to produce a bitstream, which does not overflow or underflow the decoder input buffer.

Figure 4-3 provides an illustration of the relevant rate constraints, [10] A represents the total accumulated bits in the transmitter buffer (each step represents the bits corresponding to one frame), and B represents the bits corresponding to frames that have been removed from the decoder buffer (each step corresponds to removing one frame so that it can be decoded). Note that A and B are simply shifted versions of each other, since the frames that are put in the transmitter buffer are eventually removed from the decoder buffer to be decoded (this assumes that the transmission delay is constant). Also note that in this example the delay, the difference in the time axis between the A and B, is made up of two components, a delay in starting the transmission and a delay in starting the decoding. C represents the rate at which bits are transported from encoder to decoder. In this case it represents a Constant Bit Rate (CBR) transmission mode, since C is shown as a straight line. Thus A – C corresponds to the bits remaining in the encoder buffer and C-B to those present in the decoder buffer.

In the coding process we have to take into account overflow and underflow conditions at encoder and decoder buffers. Encoder buffer overflow occurs when the buffer memory at the encoder is exceeded, that is, if the difference A-C becomes greater than the maximum memory. Conversely, if the difference A-C becomes zero (it cannot become negative), what we observe is encoder buffer underflow, and in this situation no transmission takes place, or “filler bits” are sent. At the decoder, if C-B is too large, the decoder buffer may overflow. On the other hand, if C-B becomes negative (as in the example of Figure 4-3), that means that the decoder demand exceeds the bits supplied by the channel and therefore decoder buffer underflow occurs (frames arriving too late at the receiver).

4.3.3 Encoder rate constraints: Rate control mechanism

Typical compressed video bitstreams are variable bit rate (VBR), i.e., each frame is compressed using a different number of bits. When transmitting over a constant bit rate (CBR) channel, buffers at the encoder and decoder are required to smooth out the variations in the

encoding rates. Buffering data requires extra memory at both encoder and decoder and introduces additional delay to data transmission. As the encoder is allowed to produce a more variable rate (more bits for difficult frames, fewer bits for easy frames, for example) the overall quality will be better. Thus, larger buffers, or equivalently, increased end-to-end delay, will tend to result in higher video quality.

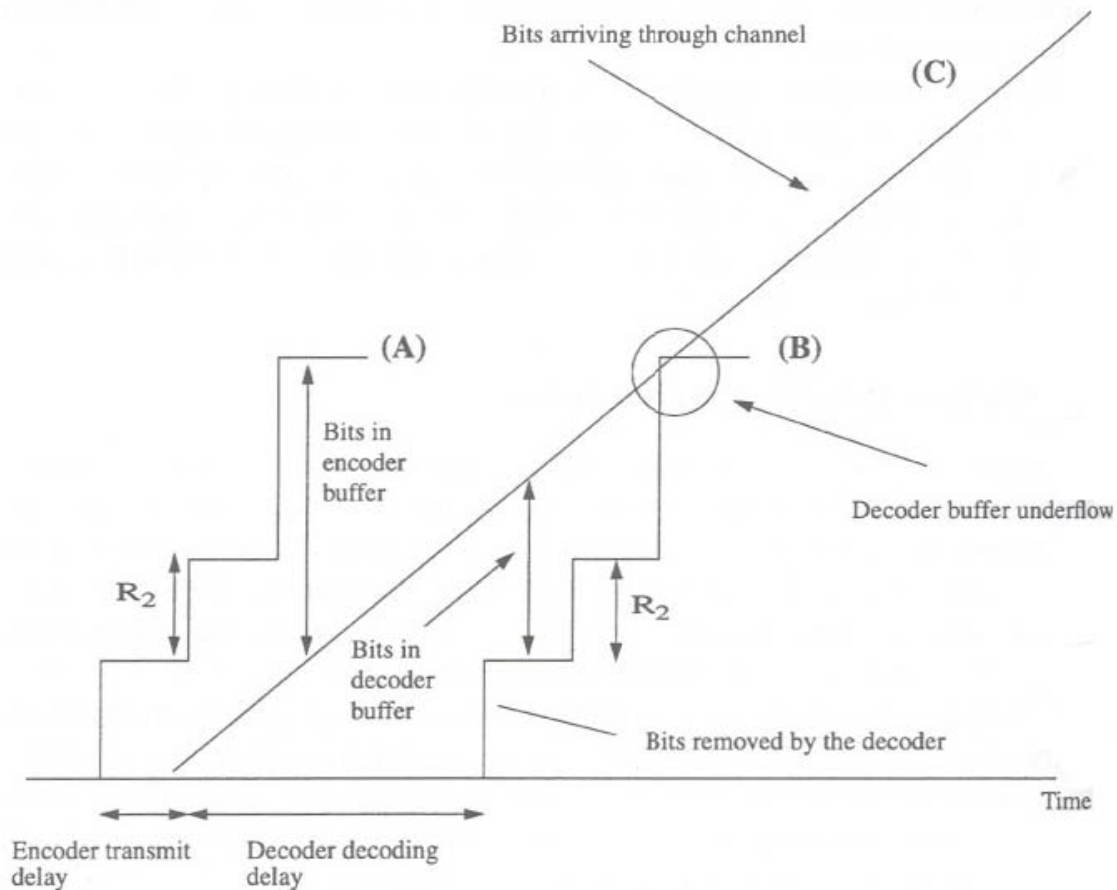


Figure 4-3: Delay constraints in live video case

The MPEG-4 Visual standard requires each video frame or object to be processed in units of a macroblock. If the control parameters of a video encoder are kept constant (e.g. motion estimation search area, quantisation step size, etc.), then the number of coded bits produced for each macroblock will change depending on the content of the video frame, causing the bit rate of the encoder output (measured in bits per coded frame or bits per second of video) to vary. Typically, an encoder with constant parameters will produce more bits when there is high motion and/or detail in the input sequence and fewer bits where there is low motion and/or detail.

Figure 4-4 shows an example of the variation in output bit rate produced by coding the Foreman sequence (25 frames per second) using an MPEG-4 Simple Profile encoder, with a fixed quantiser step size of 3.

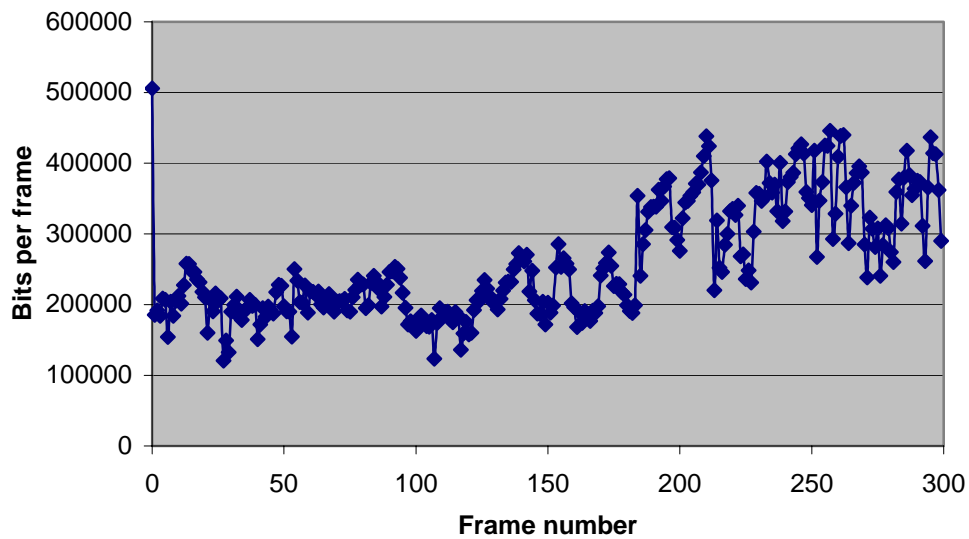


Figure 4-4: Bit rate variation in Foreman

The first frame is coded as an I-VOP (and produces a large number of bits because there is no temporal prediction) and successive frames are coded as P-VOPs. The number of bits per coded P-VOP varies between 120000 and 440000 bits (equivalent to bit rate range of 4.6Mbps - 13.2 Mbits per second).

This variation in bit rate can be a problem for many practical delivery and storage mechanisms. For example, a constant bit rate channel cannot transport a variable-bit rate data stream. A packet-switched network can support varying throughput rate but the mean throughput at any point in time is limited by factors such as link rates and congestion. In these cases it is necessary to adapt or control the bit rate produced by a video encoder to match the available bit rate of the transmission mechanisms.

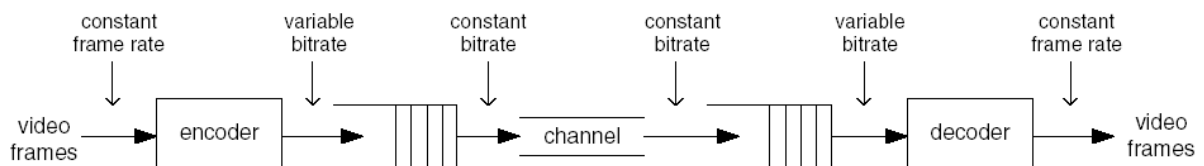


Figure 4-5 : Smoothing of variable bit rate

The variable data rate produced by an encoder can be 'smoothed' by buffering the encoded data prior to transmission. Figure 4-5 shows a typical arrangement, in which the variable bitrate output of the encoder is passed to a 'First In/First Out' (FIFO) buffer. This buffer is emptied at a constant bitrate that is matched to the channel capacity. Another FIFO is placed at the input to the decoder and is filled at the channel bitrate and emptied by the decoder at a variable bitrate (since the decoder extracts P bits to decode each frame and P varies).

These examples show that a variable coded bitrate can be adapted to a constant bitrate delivery medium using encoder and decoder buffers. However, this adaptation comes at a cost of buffer storage space and delay and (as the examples demonstrate) the wider the bitrate variation, the larger the buffer size and decoding delay. Furthermore, it is not possible to cope with an arbitrary variation in bitrate using this method, unless the buffer sizes and decoding delay are set at impractically high levels. It is usually necessary to implement a feedback mechanism to control the encoder output bitrate in order to prevent the buffers from overflowing or underflowing. *Rate control* involves modifying the encoding parameters in order to maintain a target output bitrate. The most obvious parameter to vary is the quantiser parameter or step size (QP) since increasing QP reduces coded bitrate (at the expense of lower decoded quality) and viceversa. A common approach to rate control is to modify QP during encoding in order to (a) maintain a target bitrate (or mean bitrate) and (b) minimise distortion in the decoded sequence. Optimising the tradeoff between bitrate and quality is a challenging task and many different approaches and algorithms have been proposed and implemented.

4.3.4 Timing constraints for Encoding

The factors that determine the timing constraints for an MPEG video encoder are the following:

- The video sequence characteristics (degree of motion and detail): this will have an impact on the complexity of the encoding process and therefore on the capacity to meet the timing constraints.
- Frame ordering and dependencies: there are dependencies between P and B frames. In a frame sequence such as I – P1 – B – P2, frames P1 and P2 need to be encoded prior to frame B as they are both used in the prediction of the B frame. Nevertheless, frame B needs to be displayed before P2.
- Camera rate constraints: the camera produces input data for the encoding process at a certain rate. If the camera rate is too high for the encoder, this can skip some frames and encode only a subset of them (for example halving the frame rate).
- Timing constraints from other components in the communication system: as seen in Figure 4-2 the total delay experienced by each frame is formed of several delay components, one of them being the encoding delay. To meet the total delay constraint, an excess delay of one component (such as the network for example) could motivate the need of a decrease in the delay of another component. This way, a network with low capacity (high transmission delay) or a slow decoder (high decoding delay) can pose harder constraints on the encoding delay that will need to be reduced in order to meet the overall delay.

Start time constraints

The starting time for encoding a video frame depends on the following conditions:

- Raw data available at the input buffer coming from the capturing process
- For B frames, the encoding finishing time of the forward/backward reference frames
- Free frame buffer available. Predicted frames such as P frames require at least two buffers, one for the current frame and one for the previous frame that is used for prediction

Finishing time constraints

The latest time at which encoding a frame has to be completed

- Encoder output buffer overflow: if the network bandwidth is too low and the data is not removed fast enough from the encoder output buffer, the finishing time could be extended to avoid buffer overflow or frame discard when buffer overflow is already occurring
- The allowed Encoding latency determines the finishing time for encoding

Encoding deadlines

The encoding deadline is determined by the end-to-end delay in the system, which consists of the sum of encoder output buffer delay, transmission delay, decoder input buffer delay and decoding delay as shown in Figure 4-2.

This way, higher transmission delays or decoding delays will force to speed up the encoding process in order to meet the overall timing constraints.

4.4 Example devices

The encoding of high-quality streams is computationally very expensive, and of an order of magnitude higher than the processing involved in decoding the stream. For devices with limited processing power such as PDAs or mobile phones meeting the encoding timing constraints can be a hard task.

5 Multiplexing

5.1 Function description

Multiplexing is responsible for combining different elementary streams produced by Encoding and retrieving function to a transport stream with the required format for transportation, as shown in Figure 5-1. At this layer, multiplexing may be used, for example, to group elementary streams with similar QoS requirements, reduce the number of network connections or the end-to-end delay.

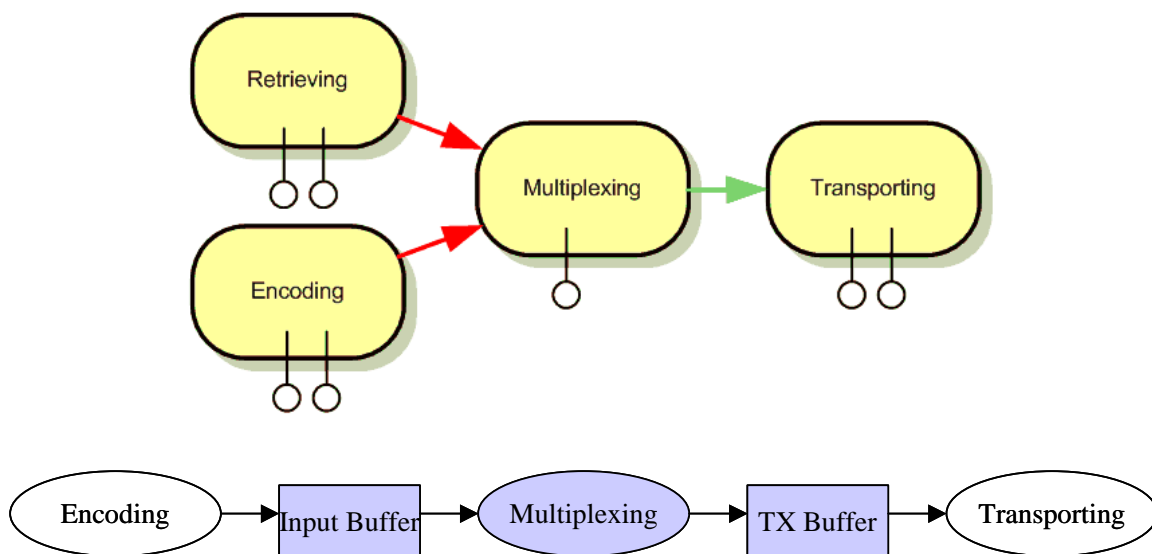


Figure 5-1: Multiplexing function usage models

Multiplexing must guarantee that the elementary streams after synchronization (SL, synchronization layer, according to MPEG4) are transformed to a packet format appropriate for transporting, designated by the communication protocol stack. The Delivery Multimedia Integration Framework (DMIF) application Interface (DAI) is defined by the MPEG4 standard to cope with information exchanged between the synchronization layer and the transporting mechanism. Part of the DAI is multiplexing of elementary streams for transporting through the underlying delivery mechanism. Most of the currently available transport layer systems provide native means for multiplexing information with a few exceptions, such as RTP Internet connections. However, the existing multiplexing mechanisms do not always fit MPEG-4 needs in terms of low delay, or they may incur prohibitive overhead in handling the expected large number of streams associated with an MPEG-4 session. As a result, MPEG-4 has defined a multiplexing tool that can optionally be used on top of the existing transport delivery layer.

Low complexity multiplexing tools like FlexMux [IEC-14496-2002] or DSMux [Lee2003] should take care of interleaving of data with low overhead and low delay. The FlexMux is a multiplexer with simple packet syntax, defined by IEC-614496 for low bitrate, low delay streams, as for example object descriptor, scene description, animation or speech streams. It is a tool intended for use if the cost, in terms of management load, overhead or delay to set up and use transport channels for each individual elementary stream, would be too high. This may be the case if a presentation consists of dozens of audio-visual objects with a similar amount of corresponding elementary streams.

5.1.1 FlexMux

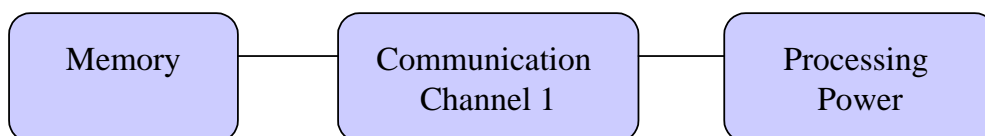
The FlexMux tool provides identification of synchronization packets originating from different elementary streams by means of FlexMux channels. FlexMux packets with data from different SL-packetized streams are then arbitrarily interleaved, forming a FlexMux stream, which is then passed to the transporting mechanism. However, FlexMux packets require framing by the underlying layer for random access or error recovery. The FlexMux packet structure has two different modes, providing alternative features and complexity, namely Simple mode and MuxCode mode. In Simple mode, the header consists of an index that corresponds to the FlexMux channel (FMC), or stream number, and the packet length in bytes, both 8 bit, limiting the number of streams and payload size to 256 bytes each. The MuxCode mode is active for index values greater than or equal to 240. These values reference a payload template instead of a FMC. Payload templates, called MuxCode table entries, further reduce the multiplex overhead by describing how the payload of a single FlexMux packet is shared between multiple streams. This mode has an initial cost: each of the 16 possible templates needs to be conveyed before it can be used. MuxCode table entries should be conveyed by the same protocol that sets up this transport channel and can be changed dynamically. In order to maintain correct state and to discover transmission errors, a version number is used that must match between the current packet and the MuxCode table entry.

5.2 Resources consumed

Based on the afore-mentioned functionality, concrete resources for supporting multiplexing are identified to be:

- CPU to handle computational demands of the multiplexing procedure.
- Program memory to store the software implementing the multiplexing functionality.
- Scratch pad memory to store data temporarily.
- Input buffers (Data Memory) to queue elementary streams.
- A transmission buffer (Data Memory) to queue the transport stream
- Internal bus connecting the CPU and the memory.

Being executed on a processing element, using memory, this function will also consume energy



At the level above, where abstract resources come into the picture, the multiplexing functional component can be mapped to:

- A communication channel resource representing the local bus for internal communication.
- A processing resource covering the computational needs of the multiplexing algorithm.
- A memory resource including the program memory, as well as the buffering needs between the multiplexer and other functions. The size of the buffers depends on the stream bit rate, by which the encoder fills the input buffer as well as by the network throughput.

5.3 Temporal aspects

The following timing parameters must be considered during task scheduling of the functional components taking part in the breeze delivery chain, in order to meet the end-to-end delay requirements:

- Initial multiplexing latency: multiplexing cannot start until elementary streams are available in the input buffer, which means that the encoding process finishing time will define the initial multiplexing latency the first time multiplexing needs to be executed. Once the whole packet delivery chain is up and running, the initial multiplexing latency has no importance. From then on, it will be the multiplexing start time that will specify the start of multiplexing execution. Initial multiplexing latency is measured from the first multiplexing request to the activation of the multiplexing functional component processing. This implies that multiplexing initial latency depends on the computational capacity of the CPU.
- Multiplexing start time: the multiplexing process starts after frame data are available in the input buffer, depending on the stream bit rate. Moreover, input buffer underflow will decrease multiplexing start time, while buffer overflow will increase it. Finally, the scheduler must guarantee that enough CPU is available to execute the multiplexing algorithm.
- Multiplexing finishing time: multiplexing finishing time is the sum of the start time and the latency of the multiplexing process. As such it strongly depends on the available CPU, but also on the buffer fillings, as imminent overflow of input buffer will increase multiplexing latency. CPU usage is the dominant factor, upon which the scheduler decides whether the processing unit can handle the execution of the multiplexing task, resulting in meeting or missing a deadline respectively.

5.3.1 Timing constraints for multiplexing

According to the MPEG-4 timing model, the end-to-end delay from the encoder input at the sending node to the decoder output at the receiving node must be constant. This end-to-end delay consists of the partial contributed delays of the functional chain, namely: the encoding process, the buffering, the multiplexing at the sending node, the transporting functional component, the demultiplexing, the decoder buffering and the decoding process. Under the above constraint, the receiver is free to modify the absolute values of all time stamps with a temporal offset, if it can handle the required additional buffering. Since packets must be delivered to the receiver before their decoding time, multiplexing should finish execution taking this under consideration together with the transporting latency. Multiplexing deadline depends on the stream bit rate.

Start time constraints

The earliest time at which multiplexing of a number of elementary stream units can begin is the time that is equal or later than the cumulative input time for the minimum amount of data needed by the selected transport stream format, which directly depends on the unit size, the TS format and the rate at which the mux input buffer is filled.

Finishing time constraints

The latest time at which multiplexing of a number of elementary stream units has to be completed is the earliest point in time at which any of the following finishing time conditions holds:

- The required transporting time of the transport frame unit(s) produced by the mux.
- The time at which the mux input buffer overflow occurs.

5.4 Example devices

Multiplexing can be done in the following devices.

- Media center
- PC-like central processor
- Laptop or tablet PC
- PDA
- Access Point, only if it is a special AP, able to perform MPEG specific adaptation of its behavior

6 Transporting

6.1 Function description

According to the deliverable D1b, a Transporting function instance may have as data source either a Multiplexing or another Transporting function instance, and as data sink either a De-multiplexing or again another Transporting function instance. The models of these two possible chains of usage of the Transporting function, which are closer to the usual design paradigm are depicted in Figure 6-1. According to this, the Transporting function instance part of the sending device is asynchronously triggered from the insertion of the first data block in the TX buffer, while from there on, either it is continuously active, as long as there are data in this buffer, running whenever the selected task scheduling policy determines, or it is asynchronously triggered from low-level events such as the end of a packet transmission or the reception of a feedback packet (ACK etc.). Moreover, in the receiving side, the nature of this function execution is always asynchronous, triggered by the reception of data bits from the medium.

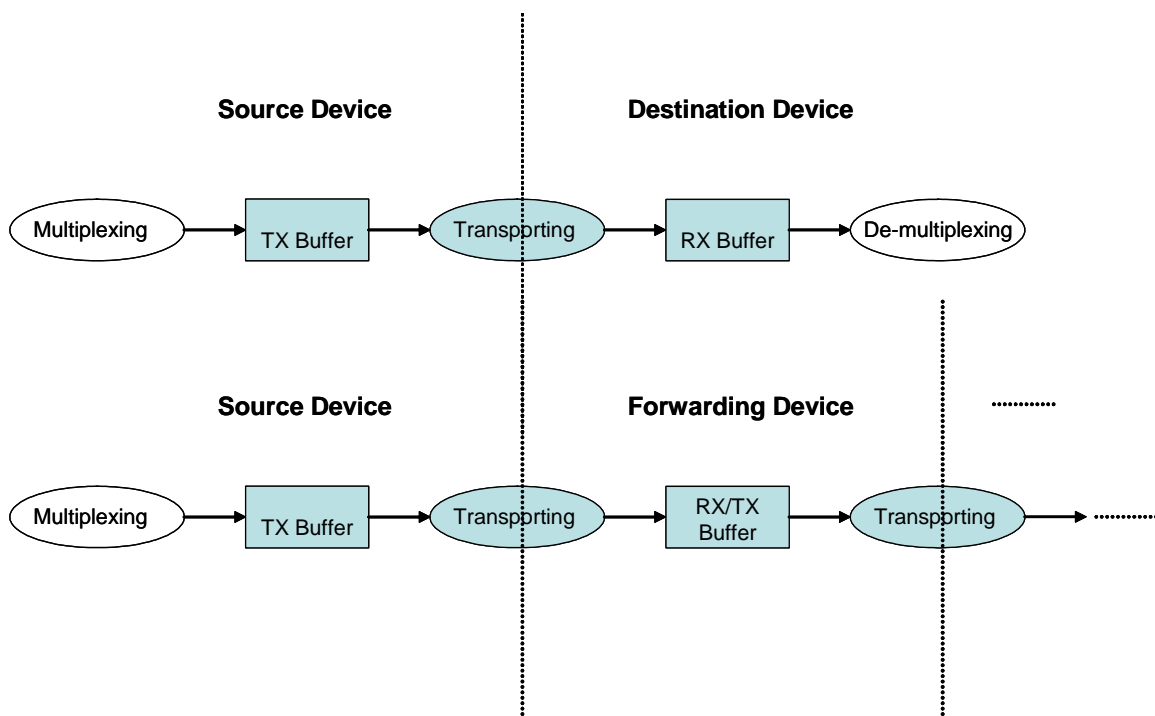


Figure 6-1: Transporting function usage models

6.2 Resources consumed

The Transporting function is actually the immediate consumer of all the identified resources in the system, that is, processing power, storage, bandwidth and energy. Its performance can be measured, at a first level of interest, mainly in terms of the offered throughput and end-to-end delay. In addition to these, the exercised PER at the function interface should be also considered as a performance / quality indicator when the transportation function implements an unreliable service (e.g. UDP) and the medium is not error-free. The resource consumption cost required for the transportation of a specific amount of user data in a specific throughput/delay/PER requested set-point can be analyzed as follows:

6.2.1 CPU

Processing power is needed for the implementation of the protocol processing and packet preparation code. Nevertheless, such a CPU load could be considered independent from the actual content and the semantics of the data to be transported with the same protocol stack, and only dependent from the requested traffic load (i.e. requested instant throughput at the function interface) and probably from the requested (variable) error correcting code strength, if implemented.

6.2.2 Memory

The maximum amount of data buffers needed is determined by the maximum anticipated delay of a data block, above which packets should be dropped (negatively affecting the PER) while the minimum amount of them is determined by the expected fluctuations in the relation of the input and output bitrate / throughput.

6.2.3 Bandwidth

The needed bandwidth is directly proportional to the requested throughput, as long as the medium is considered as error-free. Nevertheless, in the normal case where errors must be taken into account, the bandwidth consumed depends on the current BER of the channel and the selection of the error control function and its parameters. On the other hand, in the case of wireless transmission (i.e. 802.11) the available bandwidth may fluctuate depending on the signal level. If this available bandwidth is lower than the needed bandwidth, there will be buffer overflow on the sender's side and will result in the loss of information by the dropping of frames.

6.2.4 Energy

The main energy consuming components of the transporting function are the energy spent in CPU for the protocol processing code and the energy spent in the network interface during the transmission / reception of the data packets. In the former case, lowering the CPU frequency / voltage setting may reduce the energy spent in code execution – if this is allowed by the time budget assigned to the transportation function – while in the latter case, energy may be saved by proper selection of specific parameters as the transmission power and the number of retransmissions.

6.3 Temporal aspects

Figure 6-2 presents the temporal decomposition of the (initial) impact of the transporting function into the end-to-end timing of the stream data.

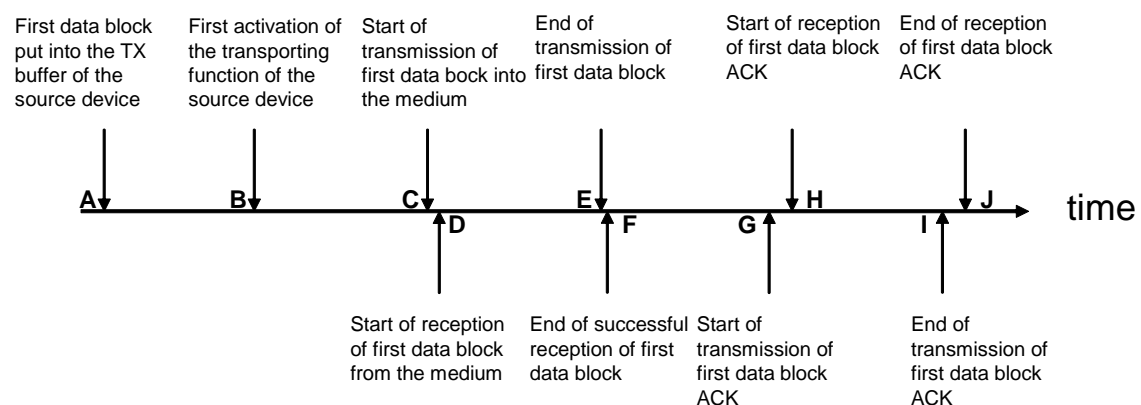


Figure 6-2: Initial end-to-end latency of Transporting function

According to this the following timing components are identified:

Start time constraints

- Transporting start time(s): The transporting function execution starts B-A time units after the first data block put into the TX buffer of the source device. Time B-A represents the latency local to the sending device, due to CPU scheduling aspects. After such a first asynchronous activation of the transporting function, subsequent activation points B come either after some CPU-related latency after point J in time, if the TX buffer remains non-empty, or after the repetition of the whole initial cycle (from point A), if the TX buffer empties. The latter depends on the stream bitrate at the input of the transporting function, and its relation with the bitrate in the medium and / or the consumption rate of the final demultiplexing sink.
- Transmission start time(s): The first data bit is placed in the medium at point C in time. The C-B time component represents the CPU related packet preparation procedures and the protocol related delays, mainly due to the MAC procedures. Since such MAC delays depend also on the load of the medium from neighbor devices, the C-B component is inversely proportional to the bandwidth portion at the MAC interface, which is available to the sending device.

Finishing time constraints

- Transmission finishing time: Point E can be directly calculated from the known bitrate at the PHY interface and the data block (i.e. LLC frame) size.
- Transporting finishing time: The termination of the transporting function invocation for a specific data block is considered either as the previously defined transmission finishing time E, if unacknowledged services are being used through all communication layers stack (i.e. multicasting / broadcasting), or as point J, which is defined as the time by which the acknowledgement of the data block is received by the source device. For the calculation of point J, the BER of the channel must be known, through which the needed number of retransmissions may be approximated.

6.3.1 Error management

At the level, two types of errors will be studied: The buffer overflow error described earlier and the transmission error that will eventually occur upon sending a packet from the transmitter to the receiver on the other side of the medium.

In the case of the buffer overflow, a congestion will occur either on the TX buffer of the source device or on the forwarding device. This congestion will lead to the miss of the deadline for the frames and will require the drop of particular frames. We will then need to handle a new dimensioning of the Breeze, This transition sequence is not covered in the BETSY study but should result in a new set of parameters from the multimedia source, compatible with the network capabilities.

On the other side, errors occurring on the medium should not lead to a redefinition of the video quality (bandwidth requirements). But, depending on the type of transmission used (UDP or TCP), they will have or not an influence on timing. UDP does not require acknowledgement upon reception. In this case, obviously, there will be a degradation of the quality of the video displayed. But there will be no influence on end to end timing. For TCP, repetition will be needed as long as the packet is not received bounded by timeouts and repetition limits. Upon the non reception of G and after a given timeout, the sending process starts back to C.

6.3.2 Timing constraints for transporting

As the time budget assigned to the transportation of a specific data block (i.e. the time distance from a design-time transporting deadline) may not be constant, due to known execution time variations of previous functions in the end-to-end chain, and as the available bandwidth may not be constant, too, due to fluctuations of the channel BER and of the traffic load, it is important for the transporting function to avoid blind packet drops, since these may seriously affect the quality of the decoding result. This may be handled either by an increase in the channel bitrate or / and by an increase in the transmission power, in order to enhance the exercised BER, or finally by sending feedback to the source or sink functional components, in order for them to adapt their operation to the current conditions.

6.4 Example devices

In BETSY; transporting devices will be either an Ethernet card or a 802.11 access point. The first one being cabled will be much less sensible to medium fluctuation resulting in much less fluctuation of the link quality.

7 Delay buffering

7.1 Function description

The delay buffering function has the role of a big buffer. It has the responsibility of storing multimedia content for a given period of time while playing what has been recorded earlier. Delay buffering can be used to compensate jitter at the expense of latency when the buffer is used close to the display function for example but here, it will be used to allow the viewer to pause a multimedia stream and not lose part of the stream.

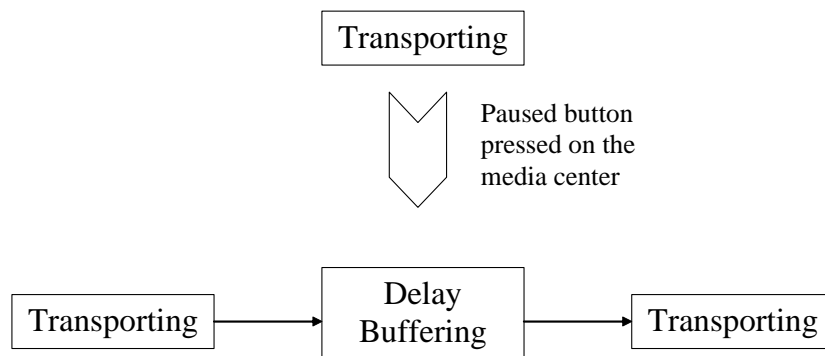


Figure 7-1: Delay buffering creation

As described in D1a, Delay buffering is a function that will be handled by the media center and is not the endpoint of a breeze. This function appears when the button Paused is pressed on the media center and disappear either when the delay is caught up (forced or using fast forward) or when the breeze disappears.

It is to be noted that the fact of introducing the Delay Buffering function will force the addition of one instance of the Transporting function in the Breeze.

7.2 Resources consumed

In terms of resources, CPU usage on the media center should stay very low when Delay Buffering is used. On the other hand, memory will be the main resource consumed time being transformed into memory, the more delay that will be introduced, the more memory that will be used. One of the side effects of this that differentiates buffers to the memory used in Delay buffering is that buffers are normally not shared between many Breezes (i.e. if a buffer is allocated, then it is fully available). On the other side, the memory used by Delay Buffering will be shared between many things: The storage of movies, data, one or more Delay Buffer) and its size is limited. If delay becomes too long and the hard disk gets full, then a decision will have to be made to empty the buffer, either by dropping the delay or by stopping the delay. In this function, no resource optimization seems possible by changing the quality of the multimedia stream included in the Breeze.

7.3 Temporal aspects

Delay buffering is a particular function in terms of end-to-end delay due to the fact that the delay introduced by the function itself (the user delay) is in the much higher order than the delay introduced by the ways to obtain this function (the system delay).

As noted in the first part of this chapter, when a Delay Buffering function is introduced in a breeze, a Transporting function is also added to the chain. This means that if the duration of the User Delay is very close to zero. The delay added to the original Breeze will be the delay of the Delay Buffering Function plus then delay of the Transporting function from the Media Center to the destination of the Breeze.

The latency introduced in the system delay will be limited to the time necessary to write and to read the information on the Hard Disk.

8 Demultiplexing

8.1 Function description

The Demultiplexing functional component takes a transport stream coming from a Transport function and takes it apart into one or more individual MPEG elementary streams, which then go to Decoding, or Transcoding functions, as shown in for the case of three elementary streams going to three decoders. In the realization of these functions, communication buffers are placed between the individual functions as shown in the Figure 8-1; a receiver buffer between Transporting and Demultiplexing and an output buffer between Demultiplexing and the subsequent decoder (or recorder or transcoder).

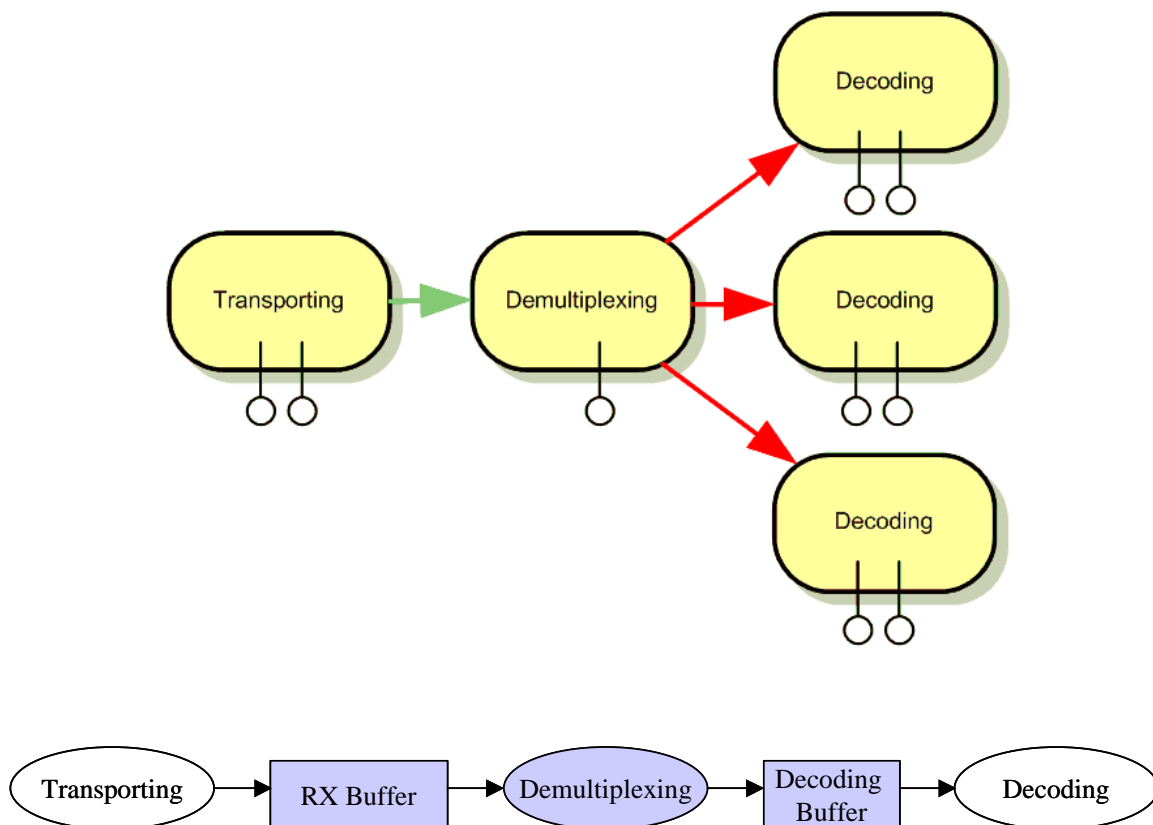


Figure 8-1. Demultiplexing function usage models

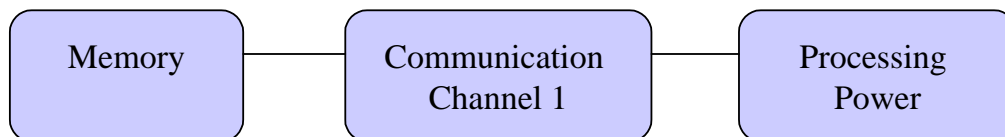
As demultiplexing functionality is complementary to the multiplexing functional component, all multiplexing rules described in Section 5.1 hold. This functionality is positioned in the Delivery Layer of MPEG-4 and takes care of deinterleaving of the synchronized, multiplexed stream into individual elementary streams according to their index and time stamp and places them in the right order to the decoding buffer for further processing.

8.2 Resources consumed

The Demultiplexing function is usually realized by a processing unit, often a CPU, the type of which will depend on the device. This CPU is likely to be shared with the Transporting function for the higher layers of the protocol stack. Subsequent functions may also be on the same CPU, or on other, dedicated physical components. The input and output streams of the Demultiplexer are buffered in one or several memories that are available to the processing unit.

Based on the afore-mentioned functionality, concrete resources for supporting demultiplexing are identified to be:

- CPU to handle computational demands of the demultiplexing procedure. The CPU demands for demultiplexing the MPEG transport stream into individual elementary streams are relatively small. They are not subject to change of parameters.
- Program memory to store the software implementing the demultiplexing functionality.
- Scratch pad memory to store data temporarily.
- Input buffer (Data Memory) to queue transport streams and output buffers to the decoders, recorders or transcoders. Larger buffers and loosened timing constraints may relax the real-time constraints on demultiplexing.
- Internal bus connecting the CPU and the memory.
- Being executed on a processing element, using memory, this function will consume energy. This resource usage is not subject to control by changing parameters of the Demultiplexing function.



At the level above, where abstract resources come into the picture, the multiplexing functional component can be mapped to:

- A communication channel resource representing the local bus for internal communication.
- A processing resource covering the computational needs of the demultiplexing algorithm.
- A memory resource including the program memory, as well as the buffering needs of the demultiplexer. The buffer size depends on the network throughput.

8.3 Temporal aspects

Accordingly to multiplexing, the following timing parameters must be considered during task scheduling of the functional components taking under consideration end-to-end delay requirements:

- Initial demultiplexing latency: demultiplexing cannot start until elementary streams are available in the receiver buffer, which means that the transporting process finishing time will define the initial demultiplexing latency the first time demultiplexing needs to be executed. Initial demultiplexing latency is measured from the first demultiplexing request to

the activation of the demultiplexing functional component processing. This implies that demultiplexing initial latency depends on the computational capacity of the CPU.

- Demultiplexing start time: the demultiplexing process starts after frame data are available in the receiver buffer, depending on the stream bitrate and the network latency. Moreover, receiver buffer underflow will decrease demultiplexing start time, while buffer overflow will increase it. Finally, the scheduler must guarantee that enough CPU is available to execute the demultiplexing algorithm.
- Demultiplexing finishing time: demultiplexing finishing time is the sum of the start time and the latency of the demultiplexing process. As such it strongly depends on the available CPU, but also on the buffer fillings, as imminent overflow of receiver buffer will increase demultiplexing latency. This implies that network conditions also affect the demultiplexing finishing time. CPU usage is the dominant factor, upon which the scheduler decides whether the processing unit can handle the execution of the demultiplexing task, resulting in meeting or missing a deadline respectively.

8.3.1 Timing constraints for demultiplexing

According to the MPEG-4 end-to-end delay constant analyzed in section 5.3.1, the receiver is free to modify the absolute values of all time stamps with a temporal offset, if it can handle the required additional buffering but it must guarantee that the temporal difference between the time stamps must be preserved for real-time performance purposes. Since the decoder buffer size is fixed and defined in the Decoder Configuration Descriptor, the Decoder buffer fill rate must be equal to the maximum stream bit rate and therefore, the demultiplexing deadline depends on the stream bit rate.

Moreover, since packets must be delivered to the decoder buffer before their decoding time, demultiplexing should finish execution under this constraint.

Start time constraints

The earliest time at which de-multiplexing of a transport stream unit can begin is the time that is equal or later than the cumulative input time for that unit, which directly depends on the unit size and the rate at which the receive buffer is filled.

Finishing time constraints

The latest time at which de-multiplexing of a transport stream unit has to be completed is the earliest point in time at which any of the following finishing time conditions holds:

- The required decoding time of the frame which the TS unit belongs to, divided by the number of the TS units needed to deliver that frame.
- The time at which the input buffer overflow occurs.

8.4 Example devices

Demultiplexing can be done in the following devices.

- TV screen with set top box
- embedded processor and memory
- Media center

- PC-like central processor
- Laptop or tablet PC
- PDA
- Access Point. Only if it is a special AP, able to perform MPEG specific adaptation of its behavior

9 Decoding

9.1 Function description

In its simplest form, playing out an MPEG video stream requires three activities: input, decoding, and display, which are separated by an *input* buffer and a set of *frame* buffers, see Figure 9-1.

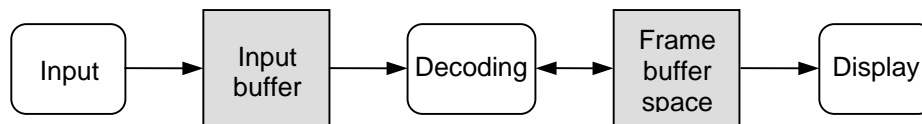


Figure 9-1: MPEG processing model

The input function (task) directly responds to the incoming stream. It places encoded video stream in the input buffer at a certain rate, expressed in bits per second, the *bit rate*, BR . In the simple case, the input activity is very regular, and only determined by the fixed, constant bit rate. In a more general case, the input may be of a more bursty character due to an irregular source (e.g., the Internet), or due to a varying multiplex in the transport stream.

Decoding means transforming an MPEG-4 Elementary stream into raw digital information that can be rendered on a screen. The decoding task extracts the video data from the input buffer at a specific *frame rate*, FR . Some common frame rate values are e.g., 24 fps, 25 fps and 30 fps. Decoded frames are then put in the frame buffers.

The display task is IO bounded, and often performed by a dedicated co-processor. It is driven by the refresh rate of the screen, the *display rate*, DR . The display task, once started, must always find a frame to be displayed.

9.2 Resources consumed

The performance of the decoding process depends directly on available processing power, the size and number of available buffers and the energy left.

9.2.1 CPU

Software decoding of high-quality video streams requires extensive processing power due to complex coding techniques. The difficulty is that MPEG decoding does not consume a constant amount of processing, since a video stream contains different frame types. In general, B frames that exhibit the highest compression ratio usually require resource-intensive compression techniques. Besides, potential wide variation between scenes (e.g., talking heads versus action), will put varied demands on required processing power at different points in time. If the processor cannot work fast enough to meet the time constraints, the decoder has to speed up, either by quality reduction, i.e., reducing the load by using a downgraded decoding algorithm or frame skipping, i.e., not decoding all frames.

9.2.2 Memory

As mentioned above, decoding requires an input buffer and frame buffers. The input buffer serves several purposes. First, it has to compensate for the irregular data size for different frames. This irregularity is bounded, and the bounding is encoded in the stream, in the form of a parameter called *VBV buffer size*. Second, the input buffer has to compensate for varying decoding times, which are not foreseen by the encoder. Therefore, this compensation cannot be bounded a priori. Third, a realistic decoder retrieves the data from the input buffer according to its processing. The resulting non-zero retrieval time relaxes the buffer requirement, but can also not be bounded a priori. Therefore, the input buffer size is essentially a design choice, closely related to the initial decoding latency and the desired end-to-end latency.

The frame buffers serve a dual purpose. They serve as reference buffers for the decoder and as input buffers for the display task, or output buffer for the decoding task. It is possible that a certain frame buffer is used in both capacities at the same time. This makes frame buffer management somewhat more complicated than input buffer management. The number of required frame buffers for decoding depends on the stream structure. For example, if the stream contains two or more B frames in sequence, the minimum number of frame buffers needed is four: two for the reference frames, one for the B frame being displayed, one for the B frame being decoded. The use of four frame buffers allows a certain irregularity in the delivery of output frames by the decoder, directly influencing the start times and the finishing times for decoding of frames. For example, if we have a video stream structure with two B frames between each pair of reference frames, the second B frame can be decoded in the same frame period as the first B frame is being displayed, since they are using different frame buffers.

9.2.3 Energy

For portable battery-powered devices, the energy is a critical scarce resource which consumption needs to be controlled. Similar as processing power, the energy consumption for decoding a video frame will vary between different frames, since the complexity of frame decoding and the time remaining to meet the next deadline varies dynamically. This implies that energy consumption for frame decoding can selectively be reduced by reducing processing speed when timing constraints can be easily be fulfilled.

9.3 Temporal aspects

Video and audio, as well as stream processing in general, have throughput requirements and real-time deadlines. For example, decoding a 25 fps video stream requires periodically a newly decoded frame every 40 ms. This implies there are 40 milliseconds per picture available to read the picture from the disc, decode its contents and display it on the screen. Delays in this process may result in severe video quality degradation of the played stream.

9.3.1 Latency

Once we start to play out a video stream, the *end-to-end latency* is fixed and it is measured from the arrival of the first bit at the input task to the display of the first pixel or line on the screen. If this latency is not fixed, the system cannot work correctly over time

The end-to-end latency is the sum of the *decoding latency*, and the *display latency*, see Figure 9-2. The decoding latency and the display latency are not necessarily fixed.

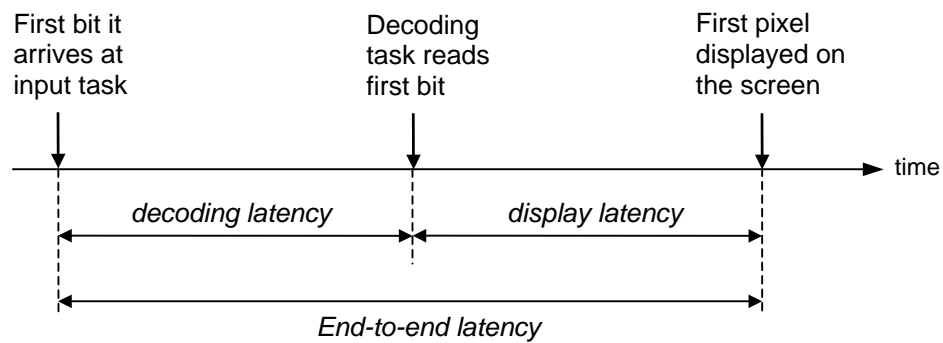


Figure 9-2: End-to-end latency for MPEG payout

The initial decoding latency is measured from the arrival of the first bit at the input task to the reading of the first bit of the first frame, after the header, by the decoder. The initial display latency is measured from the reading of the first bit of the first frame, after the header, by the decoder, to the display of the first pixel on the screen. If the decoding task is strictly periodic, the decoding and display latencies are constant. If the decoder is asynchronous, i.e., if its activity is determined by the buffer fillings, both latencies can vary due to different decoding times for frames. The latency variation allowed is a design decision, based on the maximum allowed end-to-end latency, and the available buffer space.

9.3.2 Buffer underflow and overflow

Since the decoder can be asynchronous, there is a risk of buffer overflow and underflow, which could result in severe visual artefacts. If the decoder is too slow, the decoding latency gets increased, leading to missed decoding deadlines. On the other hand, if the decoder is too fast, the display latency gets decreased, which effects the decoding process in such way that the decoder is blocked until the input task catches up. Hence, the decoding latency must be kept within a certain range. This is illustrated in Figure 9-3.

Input overflow is more serious than the underflow. In some cases, the input can be delayed, e.g., in case of a DVD player. In other cases, the input task cannot be blocked, especially in case of a broadcast input, where the input buffer must be made large enough to accommodate at least the variation that is allowed by the frame buffers. Thus, input overflow must be prevented.

There are three measures that contribute to preventing overflow: judicious choice of end-to-end latency and input buffer size, speeding up the processing by allocating more processing resources, and preventive load reduction, e.g., by decreasing bit rate of the stream, using degraded decoding algorithm and frame skipping. Which methods should be used depends on the situation. If the network bandwidth is limited, the streaming server can replace the current stream with a lower bit rate alternative. If the processing power on the display device is an issue, then frame skipping or downgraded decoding algorithm that uses less CPU power can be used.

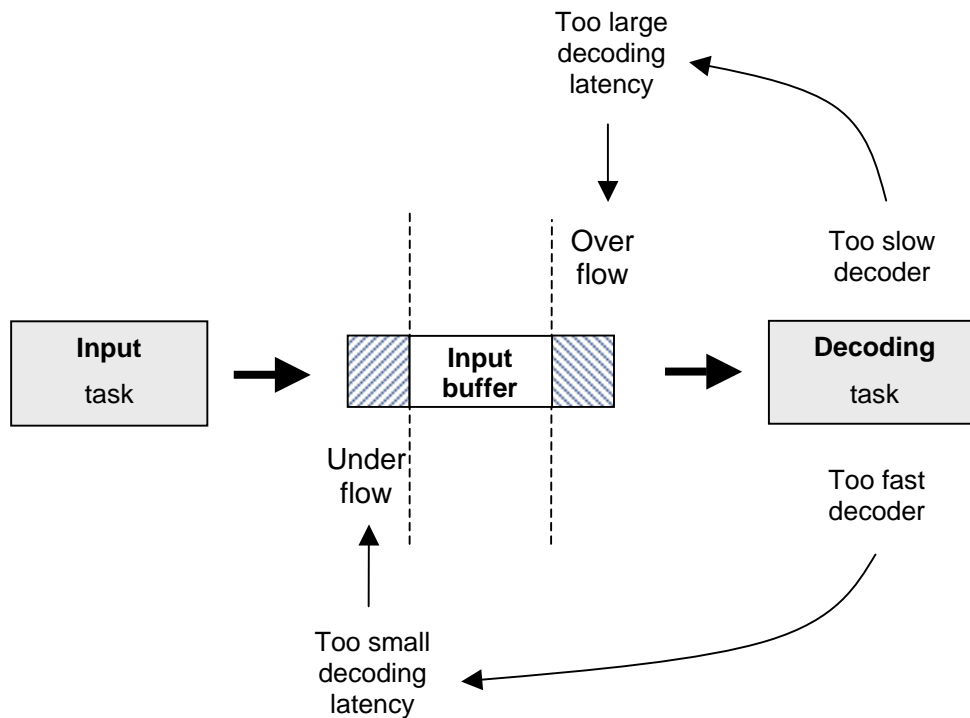


Figure 9-3: Input buffer overflow and underflow

9.3.3 Timing constraints for decoding

Timing constraints for an MPEG video decoder stem from roughly three sources:

- Video stream constraints – in particular frame ordering and their dependencies, poses mostly relative constraints. For example, in order to decode a B frame, its reference frames need to be decoded first. Besides, the backward reference frame must be transmitted and decoded before the B frame, meaning that it will have earlier decoding start-time but later display time than the B frame.
- Display rate constraints – related to the refresh rate of the screen, defines mostly absolute constraints. They depend on hardware characteristics, which in turn define when a picture should be ready to be displayed. Consumer TV sets typically have refresh rates of 50, 60, or 100Hz, computer screens may have more diverse values.
- Resource and synchronization constraints – incurred by the frame buffers. The number and handling of frame buffers depends on hardware and architecture design, i.e., the constraints will be implementation dependent. Therefore we do not include specific constraints, which would change with design decisions.

Start time constraints

The earliest time at which decoding a frame f can begin is the earliest point in time at which all of the following start time conditions hold:

- Frame header parsed and analyzed, i.e., the time it takes to parse the frame header and extract relevant information needed for frame decoding. This time is platform dependent.
- For B and P frames: the decoding finishing time of the forward/backward reference frame.

- Frame data available in input buffer, i.e., the time that is equal or later than the cumulative input time for the frame, which directly depends on the frame size and the rate at which the buffer is filled.
- Free frame buffer available. This is always naturally true for reference frames: they require at least two buffers, one for the current frame and one for the previous reference frame it references to. When a new reference frame is being decoded, at most one of them is needed for reference. As a consequence, for reference frames, this constraint becomes true one frame period earlier than it would for B frames.

The last two constraints are necessary for unblocked video stream processing.

Finishing time constraints

The latest time at which decoding a frame has to be completed is the earliest point in time at which any of the following finishing time conditions holds:

- Required display time of the frame. For example, If we have a TV set displaying a digital broadcast stream the input frame rate is equal to the display frame rate: 50 - 60 Hz, depending on the region. Other input streams may have different frame rates, and other displays may have different display rates, i.e., the display rate is a multiple of the frame rate. In both cases, we need to match the frame rate with the display rate to obtain realistic finishing time constraints (deadlines) for decoding.
- Imminent overflow of input buffer, i.e., is the time at which the input buffer overflow occurs. By a judicious choice of input buffer size, as outlined above, this constraint will always be met. Should the completion constraint be missed, though, data loss at the input buffer will occur, with the risk of having to recapture the stream, which will take at least the until the next sequence header.

Decoding deadlines

If we assume that frame rate of the stream is equal to the display rate of the rendering device, $FR=DR$, the deadline of decoding frame n is given by acceptance time of frame 1 by renderer plus $(n-1) / FR$. If not, we need to match FR with the DR to obtain correct deadlines. In the later case, we can use the timing constraints presented above to derive start times and deadlines for frame decoding. Simply, we set the earliest start time and the deadline for decoding a frame to be the most strict start time and finishing time constraint for that frame.

9.4 Example devices

MPEG decoding increasingly moves from dedicated hardware to software, for reasons of cost, rapid upgradeability, and configurability. As a consequence, all physical devices with a processing unit will be able to perform decoding operations. Examples include laptops, tablets, PDAs, media centers, TVs, etc.

However, the decoding of high-quality streams is quite computationally expensive at the same time as video processing applications are more and more deployed in small embedded systems that traditionally exhibit limited processing and network resources, e.g. set top boxes, mobile phones or PDAs. Hence, for some combinations of streams and devices, the decoding timing constraints will not automatically be met. While being more flexible, software solutions are more irregular, since video processing will compete for the CPU with other applications in the system. Besides, cost-effective software media processing requires high average resource utilization, leading to instability upon worst-case resource demands. Consequently, we need

methods for decreasing the load required by media applications in resource constrained systems.

10 Rendering

10.1 Function description

The rendering function is the last step of a video/audio chain and its main function is to get the pixels of a video frame from memory to the screen (or display) or the audio frames to the loudspeakers. In this section we focus on the video rendering function. The rendering step is typically done in hardware by a dedicated co-processor. Besides the main function of feeding the screen with pixels it can also provide integrated upscaling functionality. For example a QCIF video frame is scaled up to 1024x756. Or SD to HD. The reason to integrate the scaling in the rendering process is to save memory traffic. Most renderers handle one or more video streams and compose the final image before sending it to the screen. Each stream is handled in a rendering layer. The user interface, subtitles, video, picture in picture are normally handled in different layers.

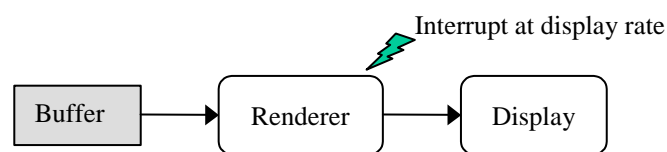


Figure 10-1 Rendering chain

Figure 10-1 shows the rendering function. When the display interrupt enables the renderer it gets the data from the input buffer and transfers it directly to the display. The input buffer is a dual buffer so that even if the next frame does not arrive in time the renderer still has the previous one to display (tolerance mechanism). There is a buffer available per rendering layer.

10.2 Resources consumed

10.2.1 Processor

Rendering is typically done in dedicated hardware synchronized with the display. This hardware block is not shared in time. However, it can handle one or more video and graphics layers. In this case the resource can be shared among different processing chains.

10.2.2 Memory

The renderer requires memory to store at least one video frame per rendering layer. For example, a full resolution high definition video frame is $1920 \times 1080 \times 4 \approx 8.2$ MB. If the renderer handles 5 streams (dual buffer) in the worst case it has to store 82MB. However not all layers handle high definition, only a couple do, therefore more realistic memory requirements are in the order of 20MB.

10.2.3 Communication bandwidth

The renderer can operate in different modes. It can simply render, or compose video layers, or scale. The bus bandwidth and energy requirements for each operation can vary depending on the mode. At this late stage of the video/graphics processing chain the amount of data transmitted is typically huge, for example, full resolution high definition video frame is 8.2 MB @ 60 fps ≈ 497 MB/sec per rendering layer.

10.2.4 Energy

The rendering function energy consumption is very constant due to its regular behavior: it takes a pixel from the buffer and places it on the screen. The rate at which the data has to be displayed and the bus utilization are the main factors that can affect the energy consumed.

10.3 Temporal aspects

The renderer is typically a periodic pull component. Upon activation it gets an element from the input buffer and transfers it to the screen at the display rate. It has hard real-time requirements due to its fine grain synchronization with the display and the huge amount of data required. Tight synchronization implies that every 20msec (progressive video 50fps) or 16ms (video 60fps) a new frame has to be ready in the input buffer or an artifact will occur. The huge amount of data makes it unfeasible (for the moment) to provide buffers that hold more than one frame, which means no execution time variation are allowed: each pixel has to be delivered at the exact time.

Since the renderer is the sink function of the chain, it typically determines the end-to-end latency $1/f$, where f is the frame rate.

The deadline for the renderer is also $1/f$ since a frame has to be provided at the speed expected by the screen.

The execution time of the renderer is also $1/f$.

10.4 Example devices

Quality video composition processor (QVCP) of the Nexperia system on chip PNX5000. The QVCP unit combines two planes of display data from different sources before output. It supports either two video planes or one video and one graphics plane, such as video from a TV tuner and OSD graphics. Together with the on-chip 2D engine and the memory-based scaler, the QVCP enables the PNX5000 to deliver world-class picture quality optimized for LCD TV applications

11 Temporal aspects table

The objective of this chapter is to give a comprehensive overview of temporal aspects for each functional component and identify the set of parameters that influence the timing of the components. Those are explained in details in previous chapters, where we identify the timing constraints of all functional components and show their impact on end-to-end timing.

FC	Temporal aspects	Depends on	Comment
Capturing	capturing start times	Quick Capture Interface (XScale), FIFO, polling or DMA to buffer	<i>I/O bounded, 640x480 * 16bit/pixel = 600KBytes * 25fps = 15MBytes/s</i>
		user interaction first, then capture frame rate	<i>CMOS Image sensor generates stream of data, per frame or continuously, rates between 15-30fps</i>
	capturing stop times	user interaction	<i>like start, independent operation</i>
	capturing deadlines	capturing frame rate	<i>periodic hard deadline, but softened by FIFO for interrupt-driven polling or even DMA to memory buffers</i>
	initial capturing latency	0.5-1.0 us (15MBytes/s) irrelevant??	<i>FIFO has 8-16 bytes depth</i>
	capturing latency	capturing frame buffers	
Encoding	encoding start times	available CPU	<i>encoding cannot start until frame data is available in the input buffer, which depends on capturing process</i>
		buffer fillings (NW BW)	<i>encoding can be postponed if output buffer is not being freed fast (low available NW BW)</i>
		frame buffers	<i>free frame buffer must be available</i>
	encoding finishing times	available CPU	
		buffer fillings	<i>imminent overflow of input buffer will postpone the finishing times</i>

FC	Temporal aspects	Depends on	Comment	
	encoding deadlines	frame rate (FR)	<i>The higher the network latency the lower encoding latency should be, in particular for real-time bidirectional applications</i>	
		constraints on network latency		
	initial encoding latency	BitRate	<i>Initial enc. latency is measured from the arrival of the first bit at the input task to the reading of the first bit of the first video frame by the encoder</i>	
		Coprocessor for capturing		
	encoding latency	CPU		
		encoder output buffer	<i>output buffer overflow will lead to increase the encoding latency (this is related to available NW BW)</i>	
		decoder input buffer	<i>decoder input buffer overflow will lead to decrease encoding latency, buffer underflow will increase it. Normally encoder should avoid decoder buffer overflow</i>	
		BR (Qp), encoding mode	<i>Parameters such as QP, resilience, coding mode (I,P,B) impact the encoding complexity</i>	
	Multiplexing	initial multiplexing latency	CPU	
		multiplexing start times	available CPU input buffers stream bitrate (BR)	
multiplexing finishing times		available CPU buffer fillings		
multiplexing deadlines		stream bitrate (BR)		

FC	Temporal aspects	Depends on	Comment
Transporting	initial transporting latency	CPU	<i>Initial transporting latency is measured from the first transportation request to the activation of the transport FC processing</i>
		available bandwidth	
	transporting start times	CPU available at sending time	<i>CPU Needs to be available to prepare and send the frame</i>
		Stream bitrate (BR)	<i>Transporting starts after frame data are available in the input buffer, which depends on BR</i>
		Output buffer full	<i>A full transport frame needs to be ready to be sent</i>
		Input buffers	
	transmission start times	802.X	<i>Represents mainly the queuing delays due to MAC or higher layers procedures</i>
		CPU	<i>For the packet preparation and network/transport layer protocol computations</i>
	transmission finishing times	802.X	<i>Transmission speed, L2 frame size</i>
	transporting finishing times	802.X CPU buffer fillings CPU available at receiving side Communication protocol (Ack Needed)	<i>Transmission speed, L2 Frame size, BER and number of retransmissions</i>
	function latency	Input buffer available	
	transporting deadlines	stream bitrate (BR)	<i>If the stream bit rate is higher than the bandwidth, dropping will be needed</i>

FC	Temporal aspects	Depends on	Comment
Delay Buffering	Start time	Available space on memory	<i>It is to be noted that the delay selected by the user will be of of a much bigger order than the delay due to any resources.</i>
		CPU available	
		Input buffer availability	<i>One way of looking at the timing impact of Delay Buffering is to look at its impact on storing and retrieving</i>
		User action	
	Finish time	User action	
	Deadlines	Available output bandwidth	
	Initial Latency	Memory Latency	
	Function latency	Memory Latency User action	
Decoding	Decoding start times	steam bitrate (BR) video frame size (FS)	<i>decoding cannot start until frame data is available in the input buffer, which depends on BR and FS</i>
		input buffer	<i>Input buffer underflow will decrease decoding latency, while buffer overflow will increase it</i>
		frame buffers	<i>free frame buffer must be available</i>
	decoding finishing times	available CPU	
		quality parameters	<i>for scalable video decoder</i>
		resolution	QCIF, CIF, SDTV, HDTV
		video frame size	
		buffer fillings	<i>iminent overflow of input buffer will pospone the finishing times</i>
	decoding deadlines	frame rate (FR)	<i>If we assume that $FR=DR$, the deadline of decoding frame n is given by acceptance time of frame 1 by renderer plus $(n-1)/FR$. If not, we need to match FR with the DR to obtain correct deadlines.</i>
		display rate (DR)	

FC	Temporal aspects	Depends on	Comment	
	initial decoding latency	BR NW bandwidth	<i>Initial dec. latency is measured from the arrival of the first bit at the input task to the reading of the first bit of the first video frame by the decoder</i>	
Rendering	display start times	CoProcessor	<i>IO bounded, performed on dedicated co-processor</i>	
		display rate	<i>display device dependent</i>	
	display finishing times	CoProcessor		
	display deadlines	display rate		
	initial display latency	BR NW bandwidth CPU buffer fillings		<i>The initial display latency is measured from the reading of the first bit of the first frame by the decoder to the display of the first pixel on the screen.</i>
		CoProcessor		
display latency	Frame buffers		<i>Frame buffer underflow will lead to decrease of the display latency, while buffer overflow will increase it</i>	

Table 11-1: Temporal Aspects Table

12 Conclusion

The transport of a video stream from a producer to a number of consumers can be decomposed into a number of functional components, such as encoding, transmitting, decoding etc. The parameters of each functional component have an impact on the end-to-end quality. Resource usage depends on these, interdependent parameter settings.

At the same time, video and audio, as well as stream processing in general, have throughput requirements and real-time deadlines.

Hence, for some combinations of streams and devices, the timing constraints on stream processing within a functional component will not automatically be met. Furthermore, in order to fulfill end-to-end timing constraints on a global basis, devices and resources have to meet specific constraints locally.

In this deliverable, we analyzed the identified functional components for their impact on end-to-end timing and look into trade-offs between resources imposed by the function of the component and how those trade-offs influence the timing of performed operations.

For each component we identify parameters and constraints that influence the timing of the functions performed by the component, such as display times of frames, latencies, jitter and reactions to changes imposed by the system status, e.g., resource availability, user input or quality change. Proposed timing constraints are then analyzed for their influence on end-to-end timing of the delivery chain of a breeze.

Together with D1b and D3a, D2a completes the initial “inventory work” for BETSY, i.e., defining the scope of devices and resources

References

- [1] Micron Technology Inc, *MT9V111 1/4" SOC VGA Digital Image Sensor*, Data Sheet
- [2] Intel Inc, *Intel QuickCapture Technology for the Intel PXA27x Processor Family*, White Paper
- [3] ST Microelectronics, *VS6502 VGA Color CMOS Image Sensor Module*, Data Sheet
- [4] http://www.semiconductors.philips.com/acrobat_download/literature/9397/75012588.pdf
- [5] Harmke de Groot et al., *Deliverable D1a, Scenario and use cases*, BETSY project, November 2004
- [6] Harmke de Groot et al., *Deliverable D1b, Description of high-level architecture and definition of components and functional entities*, BETSY project, March 2005
- [7] J.-D. Decotignie, M. Sénéclauze et al, *Deliverable D3a, Parameter and resource requirements equivalence classes*, BETSY project, June 2005
- [8] BETSY project, *Description of Work*, May 2004
- [9] S. Saponara, C. Blanch, K. Denolf, J. Bormans, *The JVT Advanced Video Coding Standard:Complexity and Performance Analysis on a Tool-by-Tool Basis*, IEEE Workshop Packet Video (PV'03), Nantes, France, April 2003
- [10] Ming-Ting Sun, Amy R.Reibman, *Compressed Video over Networks*.