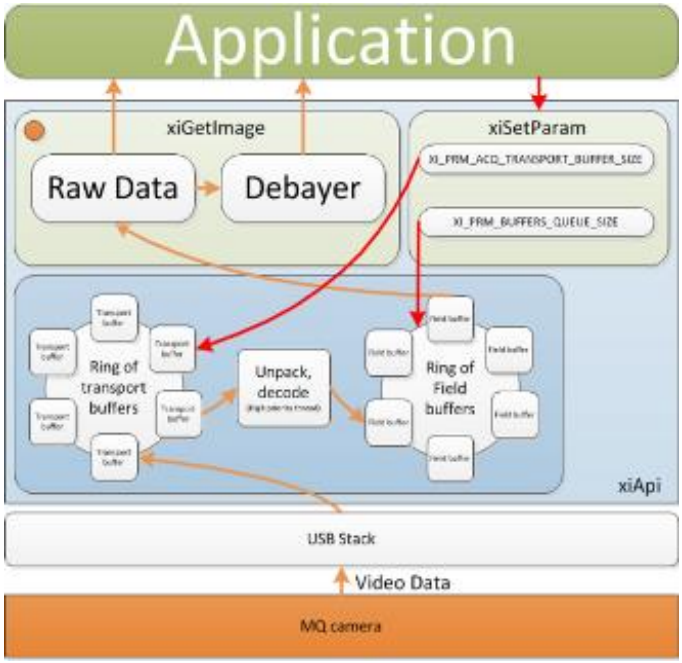

Lost Frames and How Much Data to Buffer Inside the Camera?

There is a common belief that a bigger memory buffer located inside the camera and capable of storing several image frames ensures that application will never lose frames.

This application note elaborates on the subject of lost frames, its reasons and remedy.

We will address this question from the perspective of relevant camera interface, namely CameraLink, FireWire, USB, GigE, CoaXPress, PCI Express and Thunderbolt.

Reasons

What are the reasons for the lost frames? It is all about **bandwidth, latencies** and **buffer handling**.

Bandwidth and latencies are mutually dependent, but always have one or more underlying root causes. These can be divided into hardware and software related.

On the **bandwidth** and **hardware** side:

1. Bandwidth limit

If the available to the camera bandwidth on the interface is lower than the amount of data to be delivered, such setup will always lose frames.

2. Latency

If available bandwidth is equal to the required, such system again will lose frames since it cannot tolerate any latency in servicing request from the hardware interface.

3. Reliability margin

If available bandwidth has a margin above the required one, such system can reliably work without lost frames. The question how big shall be this margin and whether it is related to the camera buffer size will be addressed below.

Hardware

The **bandwidth** is provided by the PC host side and utilized by the camera. Bandwidth can be limited by the following **hardware** reasons:

1. Communication link

Loss of transport packets* because of the high error rate or unreliable communication link.

This problem must be addressed by troubleshooting the reason for the loss of transport packets. Connectionless protocols like UDP used by GigE cameras require application level to track whether all packets have arrived and request resending of a packet, or a whole frame, if a packet did not arrive within certain time.

For transaction based protocols, like **USB bulk transfer, packet cannot be lost**. There is **100% reliable protocol** of packet delivery by design of the USB 3.0 interface.

This protocol is implemented at the USB 3.0 transaction layer. If delivery of a packed failed after several retries, then both host and camera are aware about that failure and may attempt to resend this packet, and/or report an error to the API.

Reasons for not delivered packet can be signal integrity of SuperSpeed signals, firmware issues, EM immunity, hardware design robustness like board layout, proper power rails decoupling, high speed signal impedance, etc.

These aspects are valid both for the camera and host sides, and last but by far not least is the cable. Cable performance significantly influences reliability of the communication by attenuating high speed signals and delivering sufficient power to the camera.

2. Other devices

There are other devices sitting on the same bus and thus competing for the bandwidth and other shared resources. This can be resolved by placing each camera on individual xHCI controller.

3. Firmware

Camera firmware is not capable of streaming data at the maximum speed supported by the host, and thus reduces bandwidth by introducing latencies in committing transport packets.

PC hardware by itself is not imposing latencies.

Software

Software contributes to **latencies** to handle request of supplying empty buffers to receive new camera data. These latencies in the PC software (like USB protocol stack and camera drivers, the buffer handling routines at API level, etc.) are effectively reducing the available bandwidth on one side.

1. Interfaces

Latency of servicing interrupts from hardware interface cards.

Camera interfaces, like FireWire, USB, GigE, are using respective host adapters. These adapters have different requirements for maximum latency to service interrupts, from which the most important and time critical is request to get descriptors for delivery of new data. The latency depends from the performance of the computer, operating system, software drivers, protocols and software stacks.

Camera interfaces like PCI Express or Thunderbolt are not using intermediate buffers or bus protocols. These two interfaces have substantially lesser software components and by design have much shorter latencies.

2. Applications

Application latencies – if application is in the loop of processing data from each frame and cannot complete this task within required timeslot. This can happen if algorithm for processing is not time deterministic by design. Another reason can be that underlying operating system cannot provide required amount of CPU resources within processing timeslot.

Buffer handling

At the end of the day the software aspect converges to the buffer handling method. There are two typical implementations of buffer handling:

1. Allocation

Application allocates image buffers in computer memory, commits these buffers to camera queue, and waits when a buffer will get out of the queue with image data. Then application must process data and again commit “empty” buffer. The problem happens if application is “slow” and cannot process all incoming frames in time.

If application needs to acquire and process data in bursts, then application allocates as many buffers as needed for 2 (or more) consecutive bursts and commits all of them to a camera. This will ensure that all data are delivered and application has spare time between bursts to complete processing, and then commit again all processed buffers.

2. API

Camera API allocates a circular queue of buffers sequentially filling it with image data coming from a camera. When application wants to get new image data, it is either put on hold until first buffer is filled with image data, or returns immediately if there is already a buffer with image data. API keeps track of which image was last delivered to the application and which image is currently filling with image data. API detects “overrun” situation when application is too slow and whole circular queue is already filled with data.

Summary

So, to wrap-up all previous considerations:

1. For connection less protocols (UDP in GigE) **it is vital to have large image buffer** inside the camera to be able to resend the packet or frame if it is lost.

2. For protocols which ensure packet resending on transaction layer like **USB 3.0, large image buffer inside the camera is not providing any benefit**. What is required is a FIFO to tolerate jitter on bus availability.

3. To build a reliable high speed application **both bandwidth and latencies** must be analyzed and troubleshoot.
If they are not, then sooner or later camera will lose frames, regardless whether it has or has not a large image buffer inside.

Buffer handling method also must be aligned with the application requirements.

XIMEA API, drivers and camera firmware are tuned to utilize all bandwidth from supported host controllers and have lowest possible latencies.