GSoC2022: Adding UVC hardware timestamp support

Basic Information

name	**Y Yang**
IRC nick	yyang
Email address	yyangoO@outlook.com
Github user ID	@yyangoO
Location	Shanghai, China
Timezone I will be duration of GSoC	UTC+8
Academic background	Bachelor degree, master degree in reading of control science
Classes I've done	C, CPP, python, operate system, computer organization(arm), computer network, data structure

Goal

- I hope to add UVC hardware timestamp support for libcamera by the end of GSoC period.
- I expect to give each UVC device's request a more accurate sensor timestamp by using the SOF clock from UVC payload header data.

The plan

The work I've done

- I've built libcamera on ubuntu20.04lts with a USB webcam.
- I've read the linux kernel document and study how to use V4L2 to access UVC payload header data.
- I've read the UVC timestamp conversion function in linux kernel and figured out how it works.
- I've impleted a test application which can print each request's sensor timestamp and it's buffers' timestamp.

The background

- Currently, the timestamp in libcamera is the one that is sampled by the linux kernel when it completes the buffer, not the exact timestamp the camera sensor capture an image. Because of the time delay of signal transmission, the timestamp in libcamera is not accurate enough.

- PTS is the Presentation Time Stamp, which counts by using STC(Source Time Clock).
- SOF is an 11 bits incremental frame number of USB standard, when we use a UVC (USB Video Class) camera, the UVC device and the host share a same SOF signal. If we use SOF as the 'bridge', we can calculate the timestamp camera capture image.
- As the `uvc_video_clock_update` of linux kernel implementation, the PTS and SOF, SOF and host clock have a linear relationship. If we can get the linear function, we can get a more accurate timestamp.
- Linux kernel maintain an `uvc_clock` which has an array of sample clocks. So for the linear function $y = a \cdot x + b = \frac{(y_2 - y_1)}{(x_2 - x_1)} \cdot x + \frac{(y_1 \cdot x_2 - y_2 \cdot x_1)}{(x_2 - x_1)}$, we can calculate the a and b easily by using two sample clocks.

The idea

- Follow the implementation of linux kernel, we can maintain a data structure which contains some sample clocks.
- During each time we complete the buffer, we calculate the more accurate timestamp by calculating the linear function convert device clock to SOF, SOF to host clock. And update the data structure.

The implementation

1. Finish a test application which can print each request's sensor timestamp and it's buffers' timestamp, so that we can compare the origin timestamp and improved timestamp. Which could be:

```
[processing request]--> Request(0:C:0/1:0)
    > [request]--> SensorTimestamp 12345
    > [buffer]--> Sequence 0
    > [buffer]--> Timestamp 28088940751000
    > [buffer]--> BytesUsed 20775
```

2. Querying Capabilities of metadata capture interface in V4L2 and check the the support of UVC payload header data.
3. Access the UVC payload header data via V4L2 by using the `V4L2_META_FMT_UVC`. The UVC metadata block extracted from UVC packet headers and provided by the UVC driver through metadata video nodes. We can use the metadata interface in V4L2 to get the UVC payload header data.
4. Expose the port of PTS, SOF and STC from UVC payload header data in `V4L2VideoDevice`.
5. We can use the queue's FIFO feature to maintain a data structure in `UVCCameraData` which is a fixed size queue of sample clocks we need, like:

```
struct uvc_clock_sample_t
{
    u32 dev_stc;        // device STC clock
    u16 dev_sof;        // device SOF clock
    u16 host_sof;       // host SOF clock
    u64 host_time;      // host time clock
};
queue<uvc_clock_sample_t> sample_clock;
```

6. Add a function in `UVCCameraData` so that we can use the sample clocks and PTS to convert the timestamp of device clock to host clock, like:

```
u64 UVCCameraData::getSensorTimeStamp();
```

7. Finally, when we complete a buffer of UVC camera in `UVCCameraData::bufferReady`, we update a more accurate timestamp, like:

```
void UVCCameraData::bufferReady(FrameBuffer *buffer)
{
    Request *request = buffer->request();

    request->metadata().set(controls::SensorTimestamp, getSensorTimeStamp());

    pipe()->completeBuffer(request, buffer);
    pipe()->completeRequest(request);
}
```

8. Also, we need to pay attention to dealing with the rollover of PTS and SOF.

The timeline

- **Week 1**:
  - Get familiar with libcamera.
  - Get familiar with UVC payload header.
  - Get familiar with V4L2's metadata.
- **Week 2**:
  - Implement a test application which can print each UVC camera request's sensor timestamp and it's buffers' timestamp, so that we can compare the origin timestamp and improved timestamp.
- **Week 3**:
  - Add the UVC payload header data capability checking.
  - Add V4L2 accessing fucntion for `V4L2VideoDevice`.
- **Week 4**:
  - Expose the port of PTS, SOF and STC in `V4L2VideoDevice` for `UVCCameraData`.

    - Test the port.
- **Week 5**:
    - Implement the data structure which store the sample clock.
    - Test the function.
- **Week 6**:
    - Implement the function in `UVCCameraData` which is used to update the sample clocks data structure.
    - Test the function.
- **Week 7**:
    - Implement the timestamp convertion function's framework in `UVCCameraData`.
    - Test the function.
- **Week 8**:
    - Finish the timestamp convertion function in `UVCCameraData` and deal with the rollover of PTS and SOF.
- **Week 9**:
    - Implement the sensor timestamp's update in `V4L2VideoDevice`.
    - Test the function.
- **Week 10**:
    - Test the whole project and debug.
- **Week 11**:
    - Perfect the project.
    - Send in a PR.
- **Week 12**:
    - Wirte documents and do the final work.

Technical background

Computer vision program with openCV

- I've wrote a computer vision program with openCV during my college for RoboMaster competition. In tis program I've used V4L2 to read camera instead of openCV's API, objects detection by using traditional computer vision algorithm, Kallman filter and so on.
- csu-rm-vision
- use v4l2 to use camera

Vulkan android surface render for ncnn

- I've wrote a PR for ncnn ncnn for the Vulkan surface render. I finished a Vulkan pipeline and shader for the image's convert and render.
- And for the test of PR, I use NDK to operate camera of android device.
- add android surface render

- cam-ncnn-win

reference

- libcamrea
- linux kernel: `uvc_clock`
- linux kernel: `uvc_video_clock_host_sof`
- linux kernel: `uvc_video_clock_update`
- linux kernel V4L2 document
- USB video class document