befinitiv
Electronics, computer science & more

Latency analysis of the raspberry camera

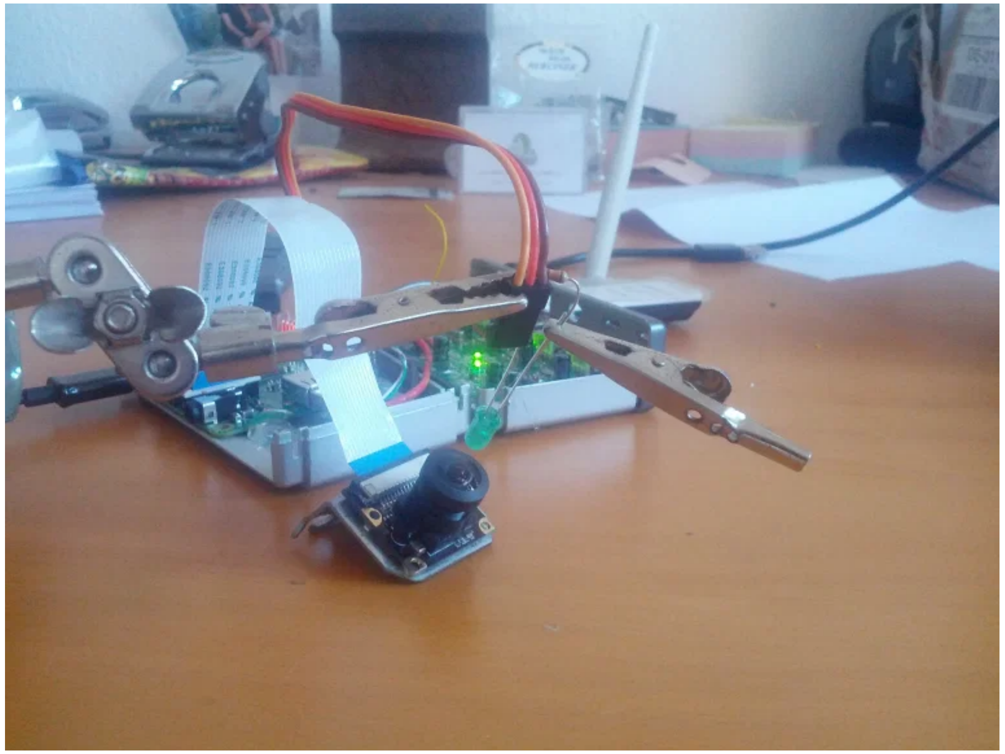
September 10, 2015
This post presents results of an analysis looking for the cause of latency in a Raspberry wifibroadcast FPV setup

—

The last bit a wifibroadcast FPV system would need to kill analog FPV would be better latency. The robustness of the transmission using wifibroadcast is already better, the image quality for sure, just the latency is a bit higher.
This post will not present a solution and will also not measure total system delay. It concentrates on the latencies in the TX PI.

Test setup

For measuring latency easily you need something to measure that is driven by the same clock as the measurement. I decided to go with a simple LED that is driven by a GPIO of the Raspberry. The LED is observed by the camera. This setup can be seen in the following Figure:



(https://befinitiv.files.wordpress.com/2015/09/img_20150909_185403.jpg)

I wrote a program that toggles the LED and outputs for each action a timestamp delivered by **gettimeofday() (http://linux.die.net/man/2/gettimeofday)**. This allows me know the time of the LEDs actions relative to the PIs clock.

Capture-to-h264 latency

The latency between the image capture to a h264 image comprises capture and compress latency. Since this both happens hidden by raspivid, it cannot be divided easily. And this is the number we more or less “have to live with” until Broadcom opens up the GPU drivers.

I measured the latency using a modified version of raspivid. The compression engine packs the h264 data into NAL units. Think of them as h264 images that are prefixed by a header (0x00000001) and then written image after image to form the video stream. The modification I added to raspivid was that each NAL unit received a timestamp upon arrival. This timestamp was written right before the NAL header into the h264 stream.

I also wrote a program that was able to read my “timestamped” stream and convert it into single images that are attached with their corresponding timestamps.

For example, the LED toggling program gave me an output like this:

```
OFF     1441817299      404717
ON      1441817299      908483
OFF     1441817300      9102
ON      1441817300      509716
OFF     1441817300      610361
ON      1441817301      111039
OFF     1441817301      211695
ON      1441817301      717073
OFF     1441817301      817717
ON      1441817302      318342
OFF     1441817302      419034
ON      1441817302      919647
OFF     1441817303      20302
ON      1441817303      520965
OFF     1441817303      621692
ON      1441817304      122382
OFF     1441817304      223078
ON      1441817311      718719
OFF     1441817311      819685
ON      1441817312      320652
OFF     1441817312      421654
```

First column is the status of the LED, second column the seconds, third column the microseconds.

My h264 decoder then gave me something like this:

```
1441741267      946995  CNT: 0  Found nalu of size 950 (still 130063 bytes in buf)
1441741267      965983  CNT: 1  Found nalu of size 907 (still 129148 bytes in buf)
1441741267      983183  CNT: 2  Found nalu of size 1124 (still 128016 bytes in buf)
1441741268      3971    CNT: 3  Found nalu of size 1409 (still 126599 bytes in buf)
1441741268      27980   CNT: 4  Found nalu of size 3028 (still 123563 bytes in buf)
1441741268      51698   CNT: 5  Found nalu of size 7005 (still 116550 bytes in buf)
1441741268      68547   CNT: 6  Found nalu of size 9667 (still 106875 bytes in buf)
1441741268      89147   CNT: 7  Found nalu of size 10312 (still 96555 bytes in buf)
1441741268      109650  CNT: 8  Found nalu of size 19244 (still 77303 bytes in buf)
1441741268      138233  CNT: 9  Found nalu of size 19338 (still 57957 bytes in buf)
1441741268      160402  CNT: 10 Found nalu of size 31165 (still 26784 bytes in buf)
1441741268      172178  CNT: 11 Found nalu of size 19899 (still 6877 bytes in buf)
1441741268      195332  CNT: 12 Found nalu of size 25129 (still 105935 bytes in buf)
1441741268      213109  CNT: 13 Found nalu of size 24777 (still 81150 bytes in buf)
1441741268      236775  CNT: 14 Found nalu of size 24657 (still 56485 bytes in buf)
1441741268      259814  CNT: 15 Found nalu of size 24738 (still 31739 bytes in buf)
1441741268      274674  CNT: 16 Found nalu of size 24783 (still 6948 bytes in buf)
1441741268      300793  CNT: 17 Found nalu of size 24855 (still 106209 bytes in buf)
1441741268      314963  CNT: 18 Found nalu of size 18368 (still 87833 bytes in buf)
1441741268      339084  CNT: 19 Found nalu of size 17959 (still 69866 bytes in buf)
1441741268      365756  CNT: 20 Found nalu of size 17958 (still 51900 bytes in buf)
```

Where the first column represents the seconds and the second column represents the microseconds.

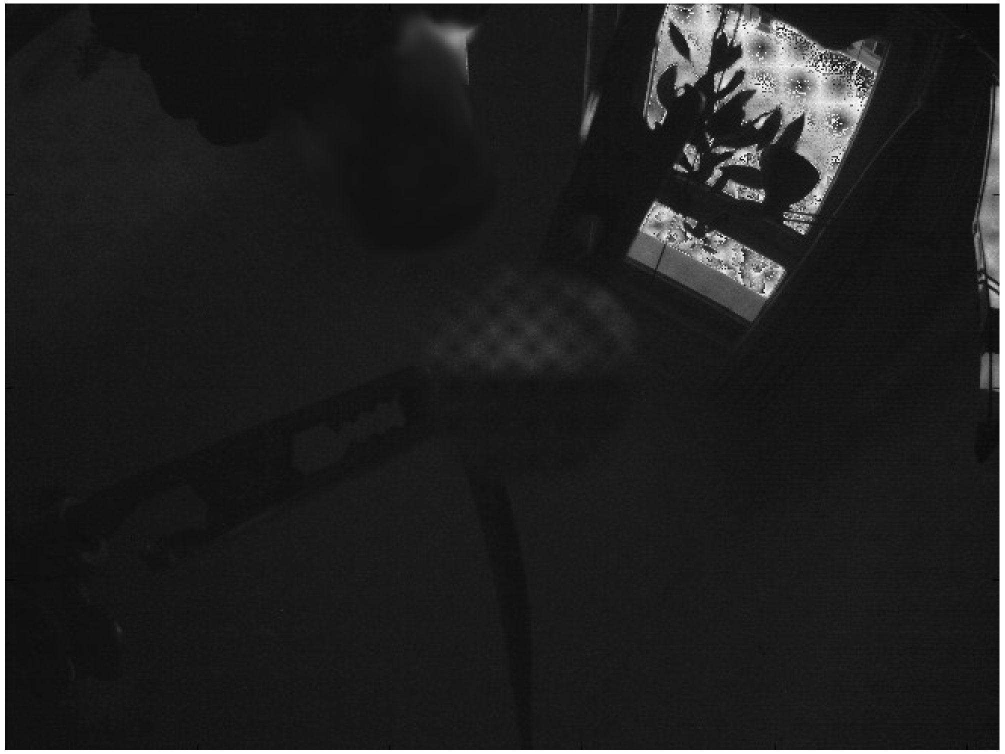
Since the LED was running at 2Hz and the camera at 48Hz I could directly relate the LED event to a specific video frame just by looking at the images (-> is the LED on or off?). This gave me two timestaps, the first of the LED event and the second of the capture of it.

The delay I got out of these was always in the range between **55ms and 75ms**. The variation of 20ms makes sense since this is roughly our frame time. Depending on whether I captured the LED at the beginning (longer delay) or at the end of the exposure (shorter delay) the times vary.

Camera settings were: 48FPS, -g 24, -b 6000000, -profile high

Capture latency

I was wondering: Does the 55ms minimum latency come mostly from compression or from capturing? I looked through ways to capture directly and found some nice hack here: **https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=109137 (https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=109137)**.
Here I did the same trick with the LED and was even a bit more lucky to capture this shot:



(https://befinitiv.files.wordpress.com/2015/09/led.jpg)

Notice that the LED is half-on half-off? This is due to the rolling shutter effect of the sensor. By accident I captured the LED right "in the middle" where it turned off. So the capture time of this shot is known to be exactly when the LED switched state (so here we do not have the 55-75ms jitter as in the case above). The delay between event and capture in this case was **38ms**. Unfortunately, the camera runs in this hacky mode at 5Mpixel. So this is not exactly the 720p FPV scenario.

Compression latency

To validate my findings from above I also timestamped the hello_encode program included in Raspian. This gave me a compression latency of **10ms for a single 720p frame**.

Conclusion

Although my different measurements are not completely comparable to each other I now have a rather clear view on the latencies:

```
Capture: 40ms
Compression: 10ms
FEC-Encoding+Transmission+Reception+FEC-Decoding+Display: Remaining ~50-100ms (to be confirmed)
```

One thing I did notice on my experiments with raspivid: It uses fwrite to output the h264 stream. Since this is usually buffered I noticed sometimes about 6KiB being stuck in the buffering. Now that size if far away from a whole frame size so it won't cause one frame being stuck inside the pipeline. But nonetheless, it will probably cause a small delay.

Another thing I noticed is that the NALU header (0x00000001) is written by raspivid at the beginning of each new h264 frame. Since the decoder needs to wait until the next header to know the end of the frame, a latency of one frame is created (unnecessarily).
Maybe a variant of wifibroadcast that is directly integrated into raspivid would make sense. This could treat whole h264 frames as an atomic block, divide the frame into packages and transmit them. Right now the blocking is done at fixed intervals. This leads to the problem that additional data is stuck in the pipeline due to a partly filled block. I still need some time to think it over but there could be some potential here.

From → Uncategorized

32 Comments

1. **extra permalink**
   Hello,

   I got similar results few weeks ago. Measurement was implemented using Dominator HD goggles + as display connected to raspberry's HDMI. Measurements with oscilloscope showed that smallest latency is around 40-42 ms with 30 ms jitter. This is far away form 20 ms latency on analog FPV (http://www.grigaitis.eu/?p=1026).

   Here is hardware limitation of Raspberry Camera. Raspberry camera has two LVDS lines to transmit data. One line at 800Mps. I am not sure (going to check) but it seems that one of them is used. If use two lines may be possible get data faster.

   But anyway there is not so much room to make it faster. Maybe it is possible to modify raspivid to force cut frame into smaller frames and send to compression engine peace by peace when not waiting for entire frame to arrive and send it trough wifi ASAP.

   For example, we have 30 fps, thus time to get one frame should be around 1sek/30fps=33 ms. Thus if we cut one frame 1280×720 by factor 4 we get 16 smaller frames size of 320 x180. Thus from camera frames are sent by rows, then there will be need to wait only 180 lines of 4 frames and it would take 8.25 ms. then those 4 frames are sent to compression and to wifi. When other frames are downloaded from camera then first frames may be already compressed and prepared to send. By this method is possible to reduce overall latency, but size of stream may increase because of split frames.
   It seems quite hard work but it may help reduce latency by 15-20 ms.

   Reply
   - **Bryce permalink**
     That is great work you did there!

     I had a similar idea with breaking apart the frame into sub-frames and sending that. As far as I could find out, the GPU that does the compression doesn't look it would support that. And the code on there is closed source. Either someone on the broadcom/PI side would need to add that feature or use a different chip (which there doesn't seem to be a lot of options at this point)

     Reply
2. **Jolyon permalink**
   Befinitiv, thank you for creating and developing this project!! Looks like you are making progress finding the main causes of latency. Looking forward to reading your future findings!

   Reply
3. **rav permalink**
   Thanks for your efforts and all your work, it's really great!

Because we can actually observe 48fps, reading the image from the camera takes at most ~20ms.
If you get this raw data after 38ms, there must be one frame stuck in the input. Unfortunately we can't look into the firmware and change this behavior.

Your hooking function is very neat. Even if the improvement is small, moving the output of the encoder through a pipe to the tx-program still takes more time and every millisecond counts.

I've been trying to accomplish this by modifying raspivid and tx to use shared memory. This way one can implement additional logic in raspivid and for example drop frames, if they are generated faster than tx can transmit.
Same for rx and hello_video: If for some reason frames are being received faster than you can display, drop all but the latest. Removing the pipe should reduce latency, too.
Right now I've gotten the transmitter to work, but I still need to do it on tx side.

There is still room for improvements in the way the NALU header is transferred.

In most cases simply adding 0x00000001 to the end of the buffer from the encoder and removing it in the next callback will work, but it can also create errors.

You need to modify raspivid, to get it done right.
The buffer contains flags, which describe the data and they can be looked up in mmal_buffer.h

These seem to be the most important:

MMAL_BUFFER_HEADER_FLAG_CONFIG:

The very first units contain configuration data which is needed by the receiver to initialize the decoder. These units are only a few bytes long and should be stored somewhere and retransmitted regularly.

They could be added to incomplete packets for example. This way even if you start your receiver after the stream has been initialized, it will know how to decode it.

MMAL_BUFFER_HEADER_FLAG_FRAME_END:
If this flag is set, the nalu header should be added to the end of the buffer. In the next callback the header at the beginning of the buffer can be safely removed.

MMAL_BUFFER_HEADER_FLAG_KEYFRAME:
Keyframes are important, so one could set a higher retransmission rate for these.

Reply

- **befinitiv permalink**

  Hi rav

  Interesting that you also hack on raspivid 🙂

  In my tests I observed that the SPS/PPS units come always before each keyframe.
  I also found that the fwrites are always called with a complete NALU with 0x00000001 at the beginning. Have you seen occurrences where the NALU is split into several fwrite calls?
  In my tests adding 0x00000001 to the end of fwrite did introduce some problems that I have not yet analyzed. This might indicate a splitted NALU (although I never observed it).

  I guess that you refer to the hooking-function in the "low_lat_raspivid_hook" branch of wifibroadcast? If not, you might take a look into that branch, could be interesting for you.

  Best regards,
  befinitiv.

  Reply

4. **rav permalink**

   Yes, I was refering to "low_lat_raspivid_hook". I've made modifications in encoder_buffer_callback right before the fwrite call.

   Strange, I only receive 2 SPS/PPS units after initialization and after that no more. I always have to restart raspivid to get rx going, if it wasn't started.

   From time to time there are calls with short data and no FRAME_END flag. The next packet will not have the NALU header in this case and if you add one, the frame will become corrupted.
   Right now I'm adding a header to the end of the data, if the flag is set and store this information. In the next callback I remove the header, if there is one and I already added it in the previous frame. Looking good so far.

   Actually I would prefer even shorter packages, like "extra" pointed out. But changing encoder_output->buffer_size does not have any effect at all.

   Reply

   - **befinitiv permalink**

     Did you use the -ih flag of raspivid? This switches between the SPS/PPS behavior you are seeing and I am seeing.

Your observations concerning FRAME_END flag would explain what I am seeing with my "dumb" NALU header replacement. I will try to hook also into mmal_buffer_header_mem_lock to get a pointer to the buffer struct.

Reply

- **rav permalink**
  You were right, I forgot the -ih flag. But I don't like it right now, because I found some better settings.

  I've discovered that setting the intra refresh type to cyclicrows yields a much better overall datarate without any peaks at the keyframes. On top of that, if packetloss occures, only small parts of the screen get disturbed, instead of the whole bottom of the screen. Because only small parts of the frame are beeing updated, the intra refresh rate should be increased a lot. 5-10 works best for me.
  The -ih flag will now constantly add configuration frames, because each frame is a keyframe. This causes lots of distortions in the image, so I do not use it right now. It seems not to work right anyway, because hello_video won't start, if it is launched later on.
  The parameter for raspivid is "-if cyclicrows".
  I've managed to get everything to work with shared memory, but I'm not sure, if it helped or if there is anything else to do.
  Right now I'm at constant 120ms-160ms: https://youtu.be/FN-0GrjlcpI

  I've also played around with annotations, which could be used to integrate an OSD into the video stream.
  Just add "-a 65535" to your raspivid parameters for a surprising demonstration!
  "raspicamcontrol_set_annotate" can be used to display any text right in the middle of the screen.

- **befinitiv permalink**
  Very interesting. I did not know about the cyclic keyframes. I will try it out as soon as I find the time to do so!

- **rav permalink**
  These are the parameters I'm using right now:
  tx:
  ```
  ./tx -b 8 -r 4 -f 1024 wlan0
  ```

  raspivid:
  ```
  ./raspivid -t 0 -n -fps 49 -w 1280 -h 720 -b 6000000 -g 5 -ex sports -pf high -if cyclicrows
  ```

  I'm also recommending this site to measure latency:
  http://www.testufo.com/#test=frameskipping&horiz=10&vert=5

  Each square represents "1 / fps" milliseconds. If you point your raspicam at it and display the output of your camera right next to it and take an image with another camera, you can easily calculate the latency. I like it better than comparing some blurry numbers on stopwatches.

5. **Calle s permalink**
   Great work you are doing here this might be a fixed issue for you.

   Annyway a tip is to use ffplay to replay the video as it is coming i tested several players and found it to be the best one.

   i run it with
   datain | ~/bin/ffplay -fflags nobuffer -i –

   Reply
6. **Roberto Sale permalink**
   Hi. I was using a PC with Ubuntu as receiver, but after upgrading tx and rx with the new FEC version, copying the .mcs1 version of the firmware to /lib/firmware and changing the start script to ./tx -b 8 -r 4 -f 1024 wlan0, my mplayer don't want to play any video, but the link is working, I can see the raw output in my console.
   What did that change? My startup script is:
   tx:
   ifconfig wlan0 down
   iw dev wlan0 set monitor otherbss fcsfail
   ifconfig wlan0 up
   iwconfig wlan0 channel 1
   raspivid -t 0 -w 1296 -h 730 -fps 49 -b 3000000 -g 5 -pf high -if cyclicrows -o – | /home/pi/fpv/wifibroadcast/tx -b 8 -r 4 -f 1024 wlan0 &

   my rx is:

   sudo ifconfig wlan2 down
   sudo iw dev wlan2 set monitor otherbss fcsfail
   sudo ifconfig wlan2 up
   sudo iwconfig wlan2 channel 1

   What can be the problem?
   Thanks a lot!

   Reply
   - **befinitiv permalink**
     Could you also please post the rx command?

     Reply
7. **Roberto Sale permalink**

It works now! The omission of the -ih flag broke my video, but with that works again.
I was hoping that with the FEC the latency get down, but it still the same, 166 ms.
Thanks a lot!

Reply

8. **tomat permalink**
hi
just want to say that your twitter translation got broken
I see last twit on june. Could you fix it up?

Reply

9. **Thomas permalink**
Hi,

awesome work you're into! Lovin' it! Please let me ask probably a dumb question: Your solution already supports receiver diversity which is great!… do you think it could be usefull to also add TX-Diversity? You could broadcast data using two or more sticks on the copter using different WLAN channels. That would multiply the bandwdth and allow for even better picture quality and reliability. One could also think of establishing an uplink channel using 2 WLAN sticks in vice versa RX/TX configuration. This would allow for transmittng mission data (e.g. MavLink) and RC-control inputs to the copter via one single solution. Could you implement some of those ideas? I love to hear from you — Thomas, TBH.. FPV Team TheBlindHawks

Reply

10. **dmitry permalink**
Have you tried disconnecting "camera_video_port" from "encoder_input_port" to check latency of the raw video stream? Maybe connection between those ports introduces some buffering/delays? Or maybe "camera_preview_port" is faster than "camera_video_port"? It just does not make any sense to have 40ms latency with 48FPS just at acquisition stage, there is buffering somewhere.

Reply

  - **befinitiv permalink**
  True, there is definitely some buffering taking place there. But currently I am not motivated to dig that deep into the camera stuff of the raspberry. Those experiments are time consuming and I am too afraid to bump into the closed-source "wall" (always a pleasure to work with Broadcom hardware…) without achieving any results.

  But if you have some new infos, please post it here so that we all can benefit 🙂

  Reply

11. **dmitry permalink**

Also have you tried sending some dummy packets through tx-rx chain, to estimate what is the latency for data input-FEC-modulation-transmission-demodulation-FEC-output path? Without any video at all.

Reply

- **befinitiv permalink**
  I've benchmarked FEC before integrating it into wifibroadcast. I don't recall but the delay was not huge (below 10ms). But I agree, this test would be interesting.

  On the other hand, minimizing the wifibroadcast latency is simple: Disable FEC packets and make block size small and you should be below 3ms (I used the ping RTT of normal wifi to estimate that upper boundary).

  Reply
  - **rav permalink**
    When fec is enabled, data packets are not sent directly, but in blocks. When a frame is finished, the block is not ready to be sent and it will only be sent, when new data from the next frame arrives. This means, that fec introduces up to one frame delay.

    We actually want to transmit high quality video, so it makes sense to be sending with a high data rate and finish transmitting one frame shortly before the next one is ready, which would make the delay insignificant.

  - **befinitiv permalink**
    wifibroadcast was also using before the addition of FEC. However, as long as the block size is well below the size of a frame, it makes no difference. The reason is, that also the output of raspivid needs to be "pushed out" by the following frame. So the one frame delay is already caused there and wifibroadcast makes things only marginally worse. If you are interested I implemented an alternative version of wifibroadcast that uses dynamic block lengths. The tx immediately sends out what it has and the rx immediately forwards it (if received correctly). Due to the effect mentioned at the beginning the gain is very little. That's why I've dropped the development on it.

12. **dmitry permalink**
    Actually I kind of tested it without any transmission at all by raspivid …. -o – | hello_video.bin. So I found that -pf baseline gives you better latencies than -pf main or -pf high. I did not get exact numbers yet, will try to measure today, but it looks promising. And I found that raspberry HW accelerated decoder is by far the best compared to any PC alternatives, at least I could not find any decoder with less that 0.5 sec latency. BTW I am thinking reviving my DVB-T setup I left because of 0.4s latency minimum, I will try to connect my dvb-t USB dongle and use hello_video.bin to decode it. Maybe I could get the same numbers as with wi-fi.

    Reply
    - **befinitiv permalink**
      Did you try gstreamer? On my machine this "feels" even a bit faster than the RPI. Other people also use it with success.

Reply

- **rav permalink**
  The decoder used by Raspivid introduces 2-3 frames delay. I've used the annotation function to create a short video of 3 frames, where each frame contains only the corresponding number and experimented with this video. Frame 0 is first seen, after frame 3 has been fed into the decoder, so there is some buffering going on. One can force the decoder to display the previous frame frame received by setting a flag saying that the stream has ended. Unfortunately this causes graphical problems and you have to restart the decoder…

13. **dmitry permalink**
    No gstreamer has the same lag as mplayer. I will try it again though, maybe I should play with the settings. Is there some special h264 decoder i should use? How to force gstreamer to use PC GPU for decoding?

    Reply
14. **dmitry permalink**
    BTW am I right thinking that if we put raspivid into passive state and then hit Enter, so if we measure the time between keypress and first callback with data we will measure our CSI acquisition delay? Or it is not so simple?

    Reply
15. **mnu permalink**
    in another project i noticed that the video latency is a lot less when i use this raspicamsrc gstreamer-plugin (https://github.com/cxp1991/raspicamsrc-raw) instead of a raspivid + pipe combination. sniffing into the code it seems they handle buffers by themselves.

    a gstreamer sink for udp-packet-injection would be cool. for now sadly above my programming skills…

    Reply
16. **sonium permalink**
    Hi, I switched on some latency related options in raspivid and I believe that this reduced the glass-to-glass latency for me by about 15ms +/- 5ms

    https://github.com/sonium0/userland/commit/a7afc2d11ab18960fd69adddc5979470f0d65b44

    Please let me know if this makes sense and/or someone can verify.

    Reply
    - **befinitiv permalink**
      Very interesting. I did not know about this. How did you measure the latency?

      Reply

17. **Nicolas permalink**
    Here is an interesting quote by one of the Raspi Engineers:

    ———————————————–
    jeanleflambeur wrote:
    – the encoder for I420 doesn't emit any output buffers if the MMAL_PARAMETER_VIDEO_ENCODE_H264_LOW_LATENCY parameter is set. It works with opaque. No errors are generated.

    [6by9 Raspi Engineer:]
    LOW_LATENCY mode is not a mode intended for general use. There was a specific use case for it where the source could feed the image in a stripe at a time, and the encoder would take the data as it was available. There were a large number of limitations to using it, but it fulfilled the purpose. This is the downside of having released the full MMAL headers without sanitising first – people see interesting looking parameters and tweak. At that point it is user beware!
    ———————————————–

    (taken from here: https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=79825)

    Maybe it's really worth looking into that some more? That looks like H.264 slices support, no?

    Oh, and I've stumbled upon something interesting (although probably way too much work to implement in wifibroadcast):

    A TDMA implementation for WiFi with Atheros Chipsets (unfortunately FreeBSD):
    Click to access FreeBSD_TDMA-20090921.pdf

    That should make bi-directional communication for control and whatever other data perfectly reliable and feasible, no more problems with collisions.

    Reply

    - **Nicolas permalink**
      Replying to myself 😉

      I've seen this in the Raspberry firmware changelog recently:
      firmware: MMAL: Support MMAL_ENCODING_xxx_SLICE formats

      Couldn't find any info on it, but maybe this could be used to reduce latency some more? As I understand it, h264 slices are what rav wrote about on Sep 15 2015 in his comment here (?)

      Reply