WEDNESDAY, NOVEMBER 16, 2011

Exploring H.264. Part 2: H.264 Bitstream format

In previous topic I wrote about color models and how pixel color data can be stored. Now it is time to go deeper in H.264 bitstream format and find those pixels data.

Obviously, the decoder operates with a sequence of bits received in a specific format. The binary stream is structured and divided into packets. On the upper level, there is separation of the stream on NAL-packets, and the stream has approximately the following form:

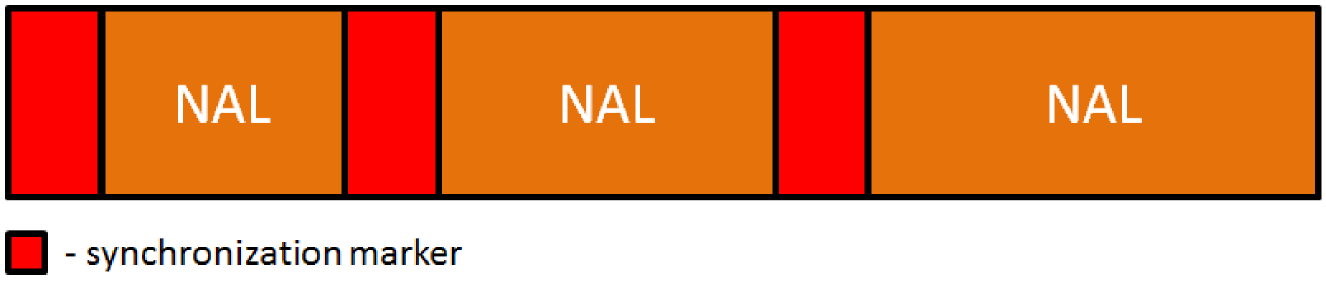

Figure 1. Stream separation on NAL-packets

The abbreviation **NAL** stands for Network Abstraction Layer. The packet structure is shown in Figure 2.
The first byte of a NAL-packet is a header that contains information about the type of packet. All the possible packet types are described in Table 1.

Table 1. NAL types

Type	Definition

0	Undefined
1	Slice layer without partitioning non IDR
2	Slice data partition A layer
3	Slice data partition B layer
4	Slice data partition C layer
5	Slice layer without partitioning IDR
6	Additional information (SEI)
7	Sequence parameter set
8	Picture parameter set
9	Access unit delimiter
10	End of sequence
11	End of stream
12	Filler data
13..23	Reserved
24..31	Undefined

NAL-type defines what data structure is represented by current NAL-packet. It can be slice, or parameter set, or filler and so on.

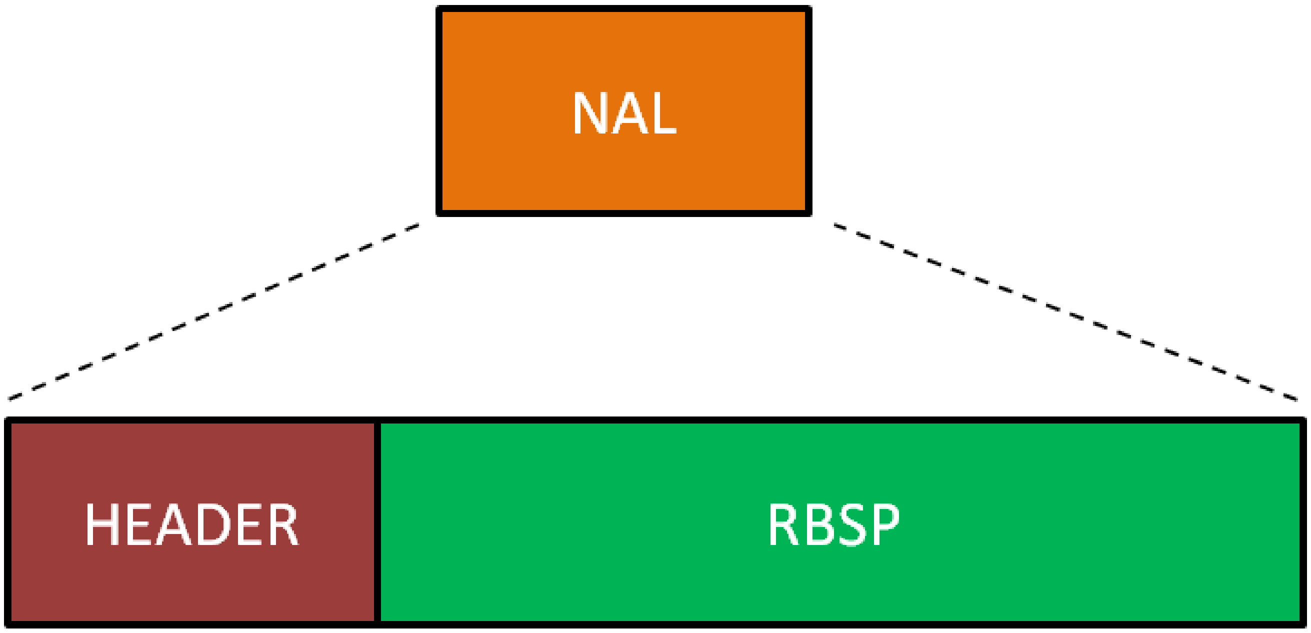

Figure 2. NAL-packet structure

As can be seen from the figure, the payload of NAL-packet identified as **RBSP** (Raw Byte Sequence Payload). RBSP describes a row of bits specified order of SODB (String Of Data Bits).
So RBSP contains SODB. According to the ITU-T specification if SODB empty (zero bits in length), RBSP is also empty. The first byte of RBSP (most significant, far left) contains the eight bits SODB; next byte of RBSP shall

contain the following eight SODB and so on, until there is less than eight bits SODB. This is followed by a stop-bits and equalizing bit (Figure 3)

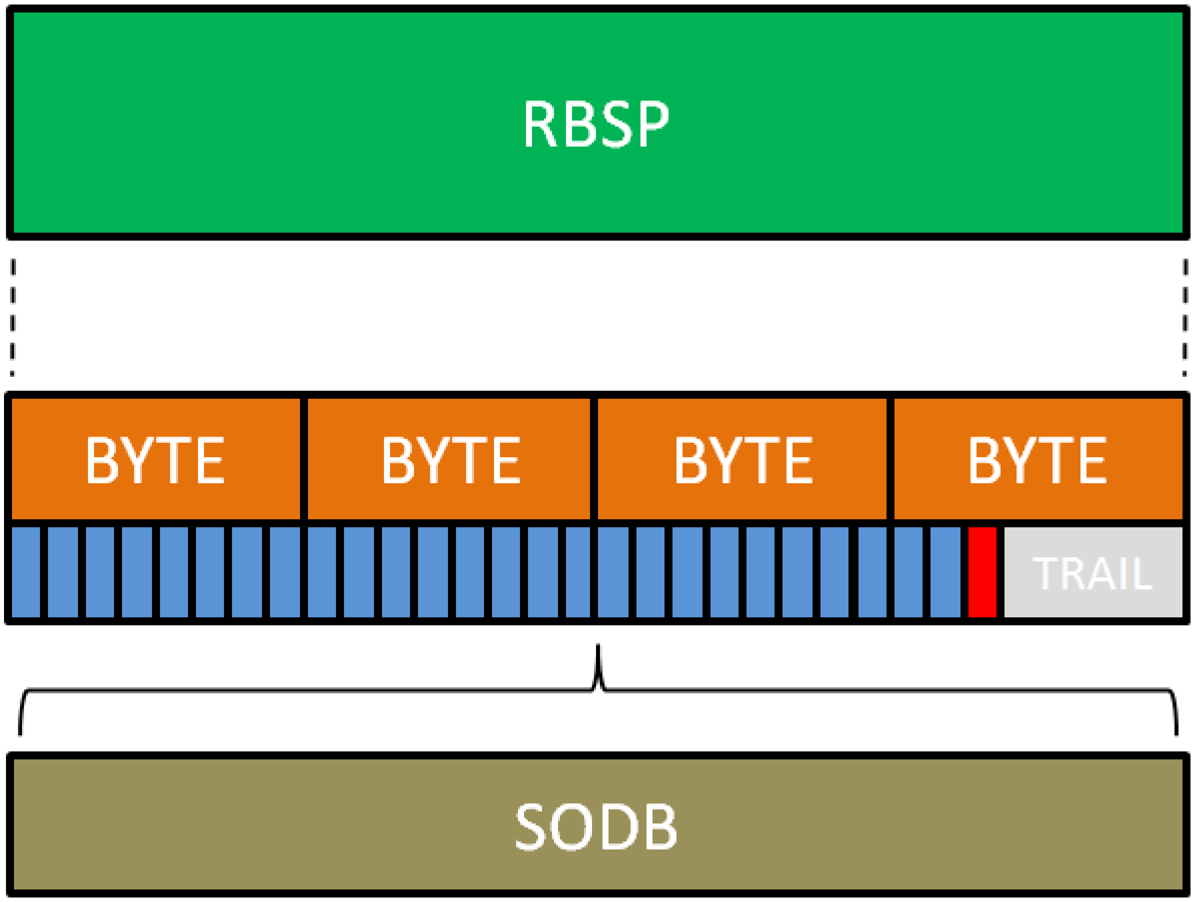

Figure 3. Raw Byte Sequence Payload (RBSP)

Now let’s look closer to our bitstream:

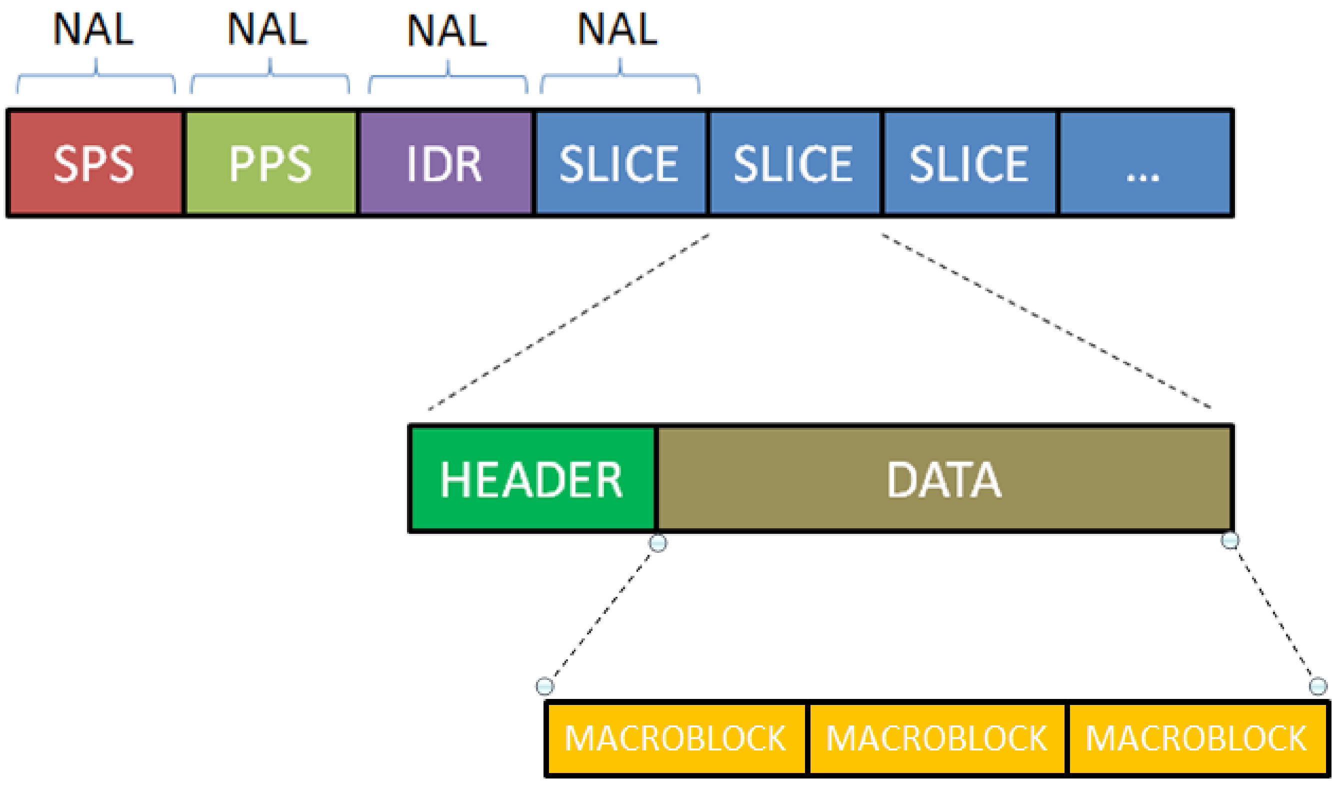

Figure 4. Detailed H.264 stream

Any coded image contains slices, which in turn are divided into **macroblocks**. Most often, one encoded image corresponds to one slice. Also, one image can have multiple slices. The slices are divided into the following types:

Table 2. Slice types

Type	**Description**
0	P-slice. Consists of P-macroblocks (each macro block is predicted using one reference frame) and / or I-macroblocks.
1	B-slice. Consists of B-macroblocks (each macroblock is predicted using one or two reference frames) and / or I-macroblocks.
2	I-slice. Contains only I-macroblocks. Each macroblock is predicted from previously coded blocks of the same slice.
3	SP-slice. Consists of P and / or I-macroblocks and lets you switch between encoded streams.
4	SI-slice. It consists of a special type of SI-macroblocks and lets you switch between encoded streams.
5	P-slice.
6	B-slice.
7	I-slice.
8	SP-slice.
9	SI-slice.

Looks like table 2 contains some redundant data, But that is not true: types 5 - 9 mean that all other slices of the current image will be the same type.

As you noticed every slice consists of header and data. Slice header contains the information about the type of slice, the type of macroblocks in the slice, number of the slice frame. Also in the header contains information about the reference frame settings and quantification parameters. And finally the slice data – macroblocks. This is where our pixels are hiding.

Macroblocks are the main carriers of information, because they contain sets of luminance and chrominance components corresponding to individual pixels. Without going into details it can be concluded that the video decoding is ultimately reduced to the search and retrieval of macroblocks out of a bit stream with subsequent restoration of pixels colors with help of luminance and chrominance components. This is how single macroblock looks like:

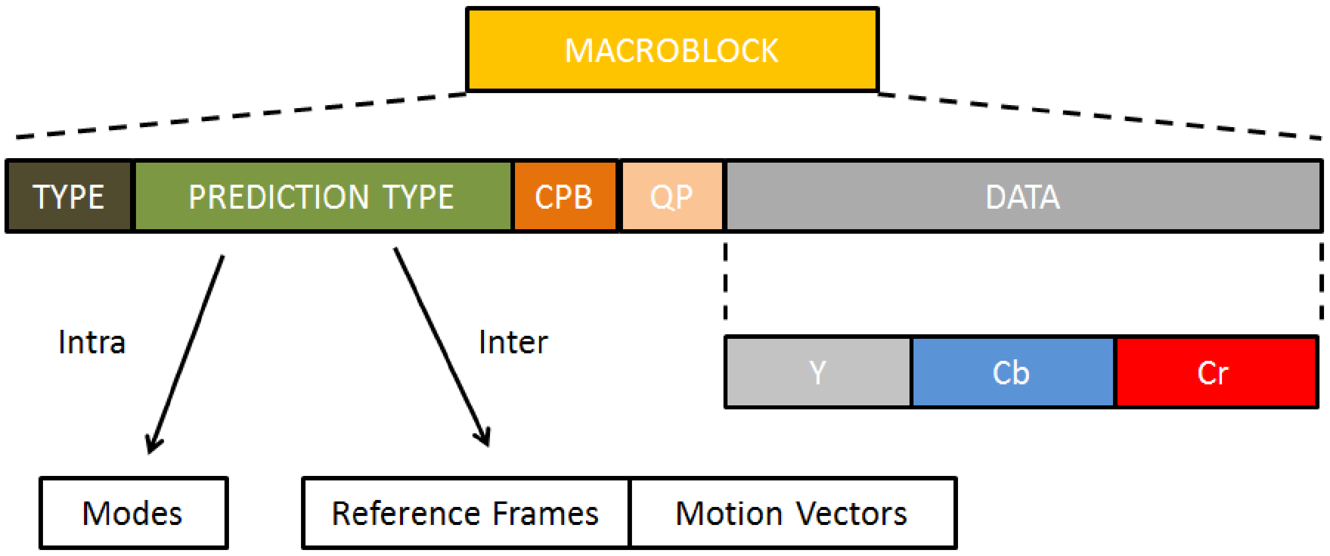

Figure 5. Macroblock

Here we have macroblock type, **prediction type** (which is the subject of the next article), Coded Block Pattern, Quantization Parameter (if we have CPB) and finally – data: the sets of **luminance** and **chrominance** components.

That is all for now. Next H.264 topic will be definitely dedicated to the macroblock prediction.
I hope you enjoyed this article. Feel free to comment or ask any questions. Good luck.

8 comments:

Danang Tri Massandy July 30, 2012 at 10:49 AM
thanks for the explanation, it means a lot!
btw, can you give me tips for parsing slice header to get slice type and how to get slice data?

Denis M. August 2, 2012 at 2:22 AM
I'm glad this information helped you in your research. H264 research is really challenging and provides a lot of fun :)

I can't give you some specific tips at this point, but you can always dig Annex B of H264 standard. And there are a lot of custom parser implementations all over the internet.

Here are some useful links for you:
http://h264bitstream.sourceforge.net/

http://cekirdek.pardus.org.tr/~ismail/ffmpeg-docs/h264_8c-source.html

http://www.elecard.com/en/products/professional/analysis/stream-analyzer.html - this is not open source, but helps in h264 bitstream research

http://aragorn.pb.bialystok.pl/~marekpk/papers/parfieniuk.09.ujt.final.pdf - some info about decoders

Good luck, and feel free to ask questions :)

Edwin Solórzano July 3, 2013 at 8:17 AM
Hi Denis,
Thank you for this great post!

Could you tell me how the NAL header changes if we use SVC?

Denis M. July 6, 2013 at 3:33 AM
There are a bunch of changes and i'm not very familiar with this part of h264 specification. You can check out the 6190 rfc (http://tools.ietf.org/html/rfc6190) and we can discuss it.

Mor July 9, 2013 at 6:01 AM
Hi Denis,
very good explanation.
How do I extract only I frame data from bit-stream?
What will be the start point and end point for I frame in h264 bit-stream?

Regards,
Mor

Denis M. July 9, 2013 at 10:16 AM
Hey, Mor
thats a big question and i'm afraid i cant give you the exact answer like "start at 0x.... and stop at 0x...." :)

To get the frame you need to find all the frame slices (one or more) so basically in your case you have to parse the bitstream, fetch the I-slices, check the frame number in slice header and decode the slice.

I understand, this is not a good answer and i feel really sorry for that. I'm not working with video decoding at the moment and it's been a while. But still i'll try to help.

Buy the way, there is a good tool for analysing h264 stream - H264Visa. It will show all the necessary information about the file.

Mor July 9, 2013 at 10:16 PM
Thank you Denis for information :) It will help!

Bhaskar July 10, 2013 at 4:59 AM

Thanks for the information..Denis. But how do I go about finding the bit-size of a particular macroblock in H.264 (CABAC) bitstream (like those of H264Visa etc.)? I am using FFmpeg. I posted a question at http://stackoverflow.com/questions/17520626/how-to-get-the-exact-macroblock-bit-sizes-from-h-264-cabac-bitstream. Any encouraging thoughts?