

FA92 H.264

Nuvoton Technology Corp.



Example of Hierarchy of a Video Sequence

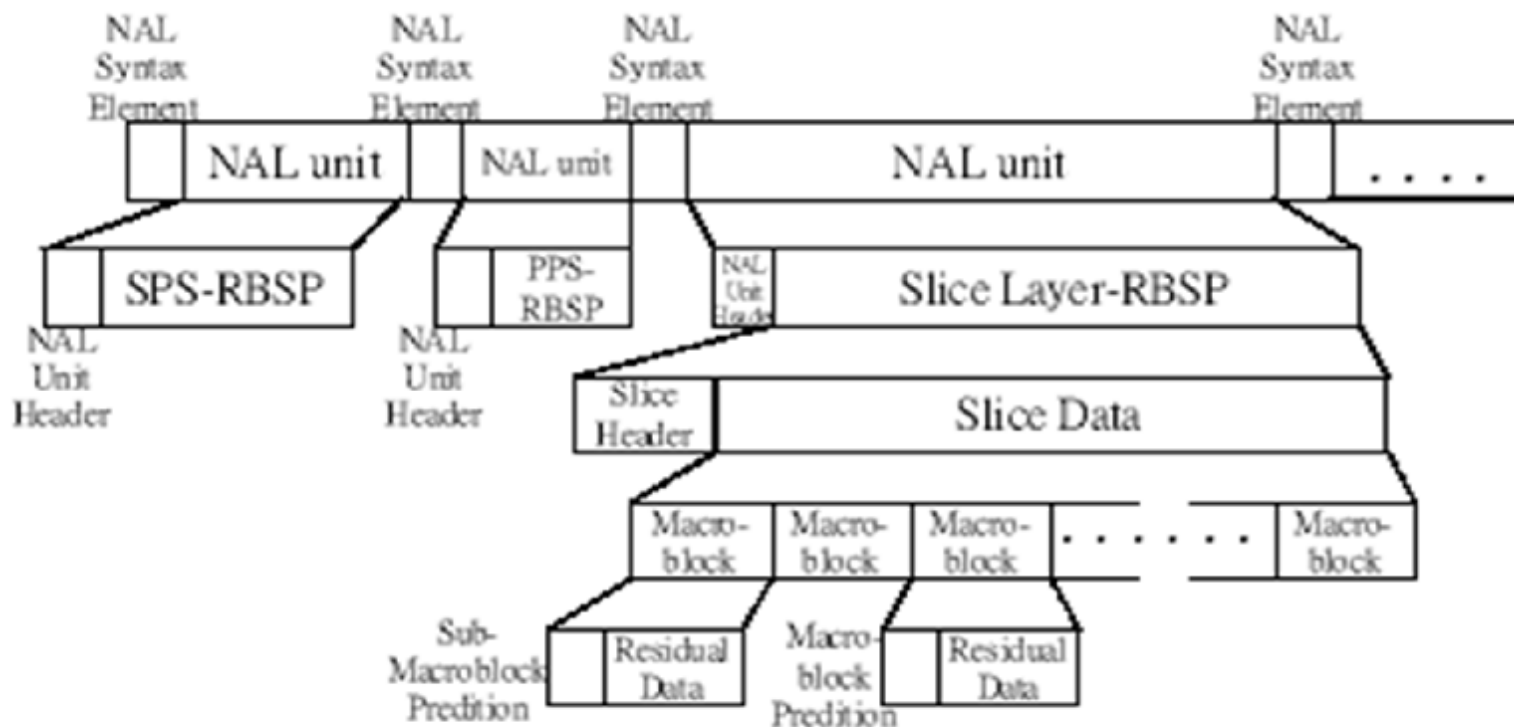


FIGURE 4. HIERARCHICAL STRUCTURE OF H.264 BIT STREAM

Sequence(pictures(slices(macroblocks(macroblock partitions(sub-macroblock partitions(blocks(samples)))))))

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
if(more_data_in_byte_stream()) {		
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
}		
}		

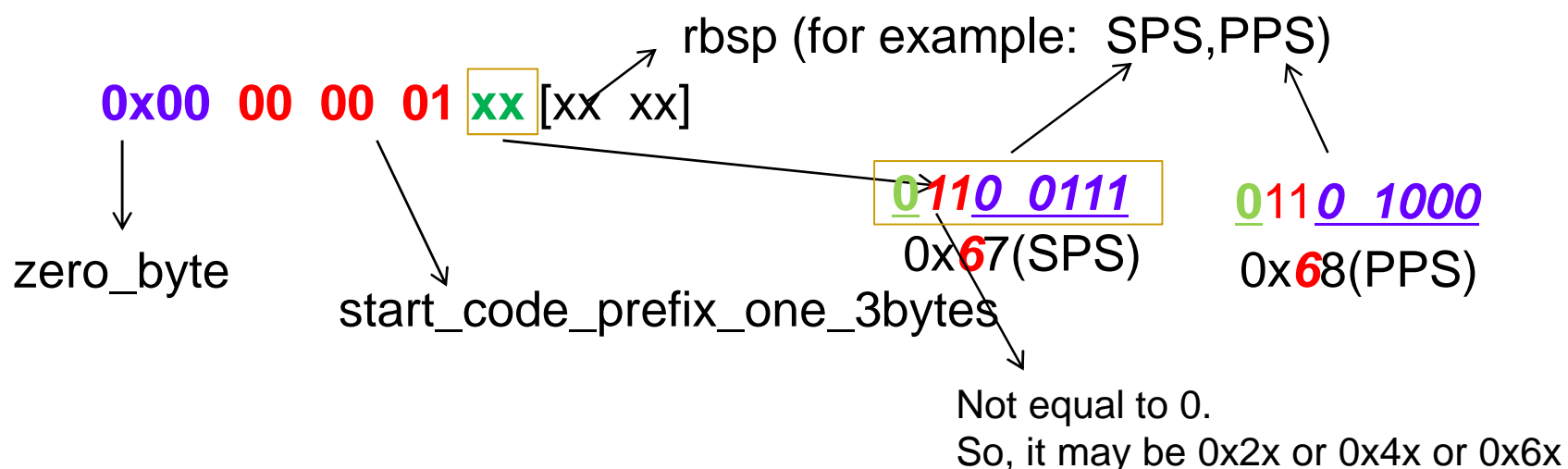


Table 7-1 – NAL unit type codes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp()	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp()	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp()	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp()	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp()	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp()	5
7	Sequence parameter set seq_parameter_set_rbsp()	0
8	Picture parameter set pic_parameter_set_rbsp()	1
9	Access unit delimiter access_unit_delimiter_rbsp()	6
10	End of sequence end_of_seq_rbsp()	7
11	End of stream end_of_stream_rbsp()	8
12	Filler data filler_data_rbsp()	9
13..23	Reserved	
24..31	Unspecified	

P

I

An IDR frame is a special type of I-frame in H.264. An IDR frame specifies that no frame after the IDR frame can reference any frame before it. This makes seeking the H.264 file easier and more responsive in the player

SPS (Sequence Parameter Set)

```
seq_parameter_set_rbs() {  
    profile_idc  
    ...  
    level_idc  
    seq_parameter_set_id    //0~31  
    ...  
    pic_width_in_mbs_minus1  
    pic_height_in_map_units_minus1  
    ...  
}
```

PPS (Picture Parameter Set)

```
pic_parameter_set_rbsp() {  
  pic_parameter_set_id // 0~255  
  seq_parameter_set_id // 0~31  
  ...  
  pic_init_qp_minus26      // -26 to +25  
  pic_init_qs_minus26  
  ...  
}
```

Encoder	Decoder
Baseline profile 3.1	Baseline profile 3.1
144x80 to 1920x1088 in step of 16 720P @25fps	144x64 to 1920x1088 720P @30fps

Table A-1 – Level limits

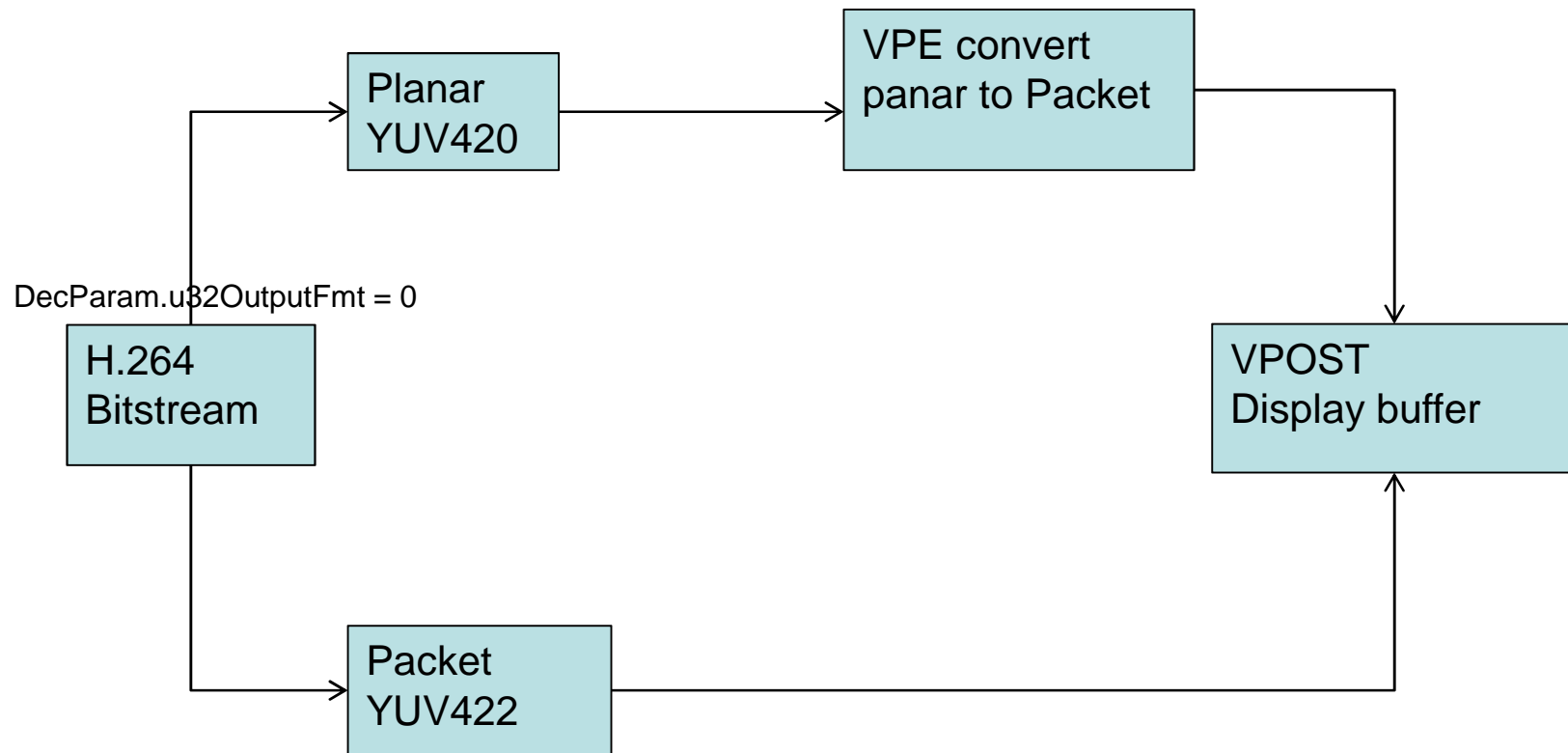
Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDPB (1024 bytes)	Max video bit rate MaxBR (1000 bits/s or 1200 bits/s)	Max CPB size MaxCPB (1000 bits or 1200 bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
3	40 500	1 620	3 037.5	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[-512,+511.75]	4	16

H.264 DECODER

Limitation

- └ Decoder does not support
 - └ disable_deblocking_filter_idc=2 (range : 0 ~ 2, slice_header)
 - └ frame_cropping_flag (SPS level)
 - └ I_PCM (mb_type at macroblock_layer)
 - └ ASO (arbitrary slice order) or FMO (flexible macroblock ordering)(PPS)

Decode to display



VideoIn MB data for H.264 enc

MB n, Y																MB n+1, Y															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Buffer

MB n						MB n+1						MB n+2					
0	1	2	253	254	255	0	1	2	253	254	255	0	1	2

MB n, U in equal mode

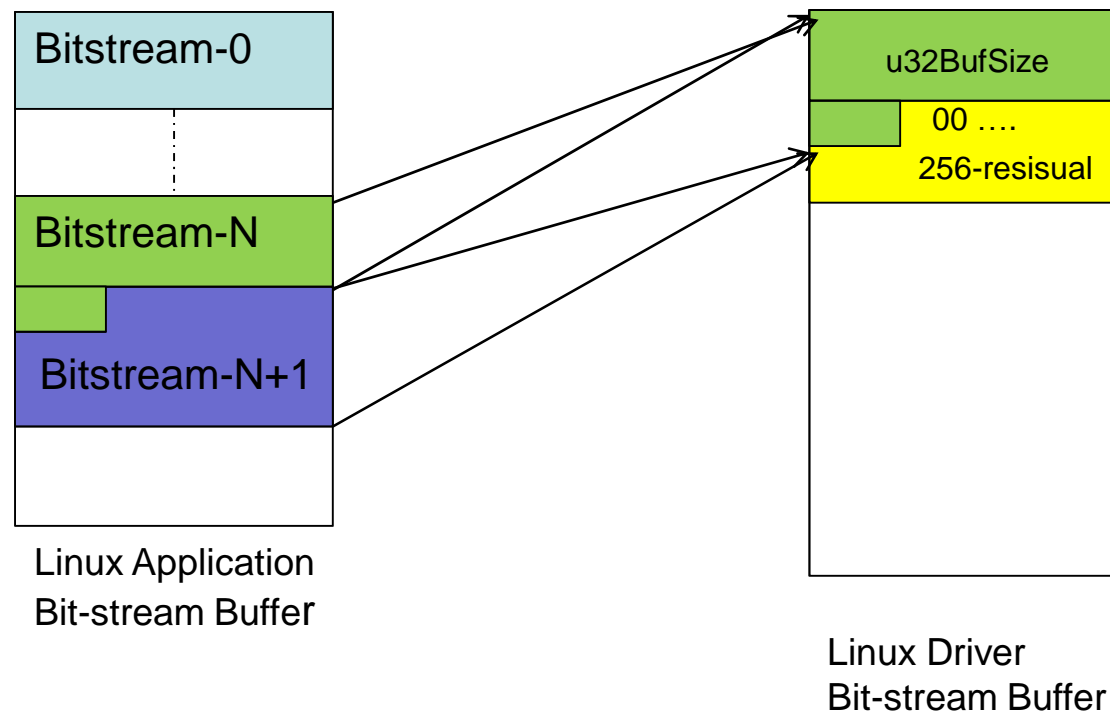
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

MB n, U

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

MB n										MB n+1										MB n+2																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672

Linux Decoder Bitstream maintain



Rate Control

- Rate control is controlled by application
 - H264RateControlInit(...,initq,) function initialize the related parameter à if **initq=0**, all the quant of rate control is “0”. It means fixed quant=0.
 - H264RateControlUpdate(**h264_ratec**, ...) calculate next Quant value based on current Quant value and bitstream size
 - Next Bitstream use **h264_ratec.rtn_quant** as Quant value for current bitstream encoding.

H264RateControlUpdate(...)

```
{ calculate Quant for next frame  
  by current Quant and bitstream size }
```

H264RateControlInit(...)

```
{ set Quant = 0;  
  set bitstream_size =0;  
  Call FAVC_IOCTL_MODE_INIT}
```

Favc_encode(...)

```
{ call FAVC_IOCTL_ENCODE_FRAME  
  H264RateControlUpdate(...);  
}
```

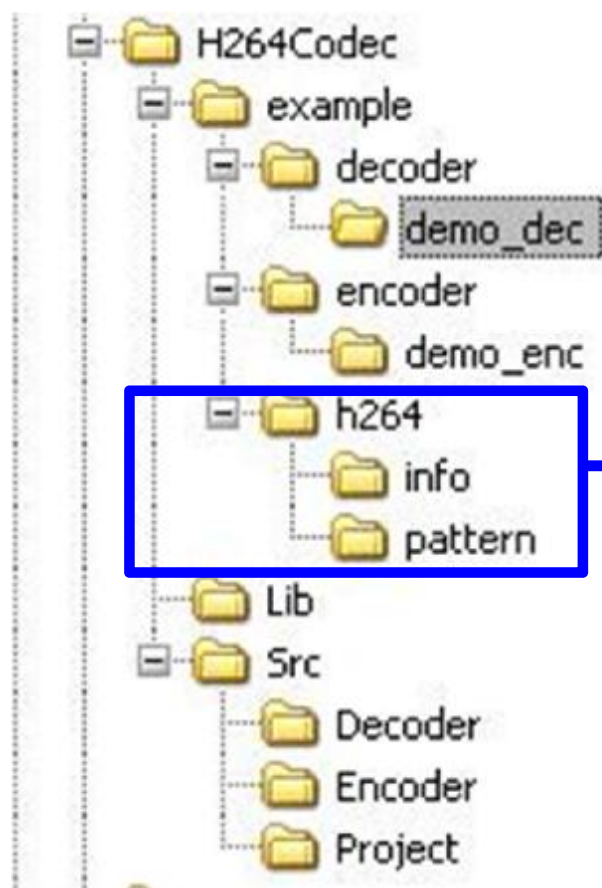
APP

Driver

h264_encoder_encode(...)

```
{ update Quant for PARM4}
```

non-OS Library



Copy h264 folder to SD card
as the test source pattern

Non-OS Example

- Decode Sample

- Source pattern :

- ◆ H.264 bitstream are in **SDCard\h264\pattern** folder
 - ◆ H.264 bitstream length are in **SDCard\h264\info** folder
 - ⌞ The file name of H264 bitstream length is same as bitstream, but extension is .txt
 - ⌞ For example, bitstream.264 in h264\pattern, and bitstream.txt in h264\info folder
 - ◆ all *.264 or *.jsv bitstream in C:\h264\pattern folder will be decoded

- Decoded image is converted by VPE to show on panel (located at the right-upper coner)

- **Change Linked VPOST library if it is different panel.**

Non-OS Sample ..

- Encode Sample

- Source pattern :

- ◆ H.264 bitstream are in
SDCard\h264\pattern\foreman_qcif_2d.yuv folder

- Output :

- ◆ Encoded bitstream **encQcif.264** at SDCard\h264
 - ◆ Encoded bitstream length for each frame
Qcifframe_info.txt

Decode example

Decoded 2nd file



Decoded 1st file



Linux configure

- Menuconfig
 - Device Drivers à Misc devices

```
.config - Linux Kernel v2.6.35.4 Configuration

                                Misc devices
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are
hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
to exit, <?> for Help, </> for Search.  Legend: [*] built-in [ ] excluded <M> module < >
module capable

--- Misc devices
< >  Analog Devices Digital Potentiometers
< >  Integrated Circuits ICS932S401
< >  Enclosure Services
< >  Intersil ISL29003 ambient light sensor
< >  Taos TSL2550 ambient light sensor
< >  Dallas DS1682 Total Elapsed Time Recorder with Alarm
< >  Silicon Labs C2 port support (EXPERIMENTAL) --->
      EEPROM support --->
< >  Intel Wireless MultiCom Top Driver
[*]  W55FA92 H.264 Codec support
      Enabled Codec Driver (H.264 Decoder / Encoder Both) --->
      Max Frame Size (1280 x 720(720P)) --->
(2)  Max Encoder Instance (min.=1 and max.=2)
(1)  Max Decode Reference Frame Num (min.=1 and max.=16)
(0xB72000) Frame buffer size

<Select>  < Exit >  < Help >
```

Linux application



→ Decoder source pattern

- Linux Driver source code is at
W55FA92\Tags\Linux-2.6.35.4_fa92\drivers\misc\codec264 folder
- Linux application
 - decoder
 - ◆ Decode all
 - ◆ H.264 bitstream are in ..\pattern folder
 - ◆ H.264 bitstream length are in ..\info folder
 - encoder_vin :
 - ◆ Encode QVGA bitstream from VideoIn
 - ◆ Encoded encQVGA.264 bitstream is at
application\h264codec folder
 - ◆ Encode bitstream length file encQVGA.txt is at
application\h264codec folder

Tool

- FA92 Linux H264 Buf Calculate.xls :
 - Calculate the H.264 Encoder & Decoder Buffer requirement for each resolution.
- Find264Length.exe
 - Find each frame bitstream length in H.264 bitstream
- N2F_H2642D.exe
 - Convert the Planar YUV420 format to Planar YUV420 MB format.
- YUVviewer.exe
 - View the Planar YUV420 format Video

FA92 Linux H264 Buf Calculate

Encoder	Decimal	hexadecimal	Linux Configure File	Decimal	Hexadecimal
Image Width	1280	500	Encoder + Decoder total Buffer size in Linux Config file	16617472	FD9000
Image Height	720	2D0			
Driver Instance	2				
Bitstream Buffer	1384448	152000			
Reconstruct Buffer	2768896	2A4000	Note : Input B2, B3, B4 and B15 for calculation		
SysInfo Buffer	230400	38400			
DMA buffer	624	270			
Encoder Only Total Buffer	8773632	85E000			
Decoder					
Image Width	1280	500			
ImageHeight	720	2D0			
Reference Number	1				
Decoded Bitstream Buffer	1384448	152000			
Decoded Buffer	2764800	2A3000			
INTRA predict size	2560	A00			
MB inform size	2560	A00			
Output buffer	3686400	384000			
Decoder Only Total Buffer	7843840	77B000			

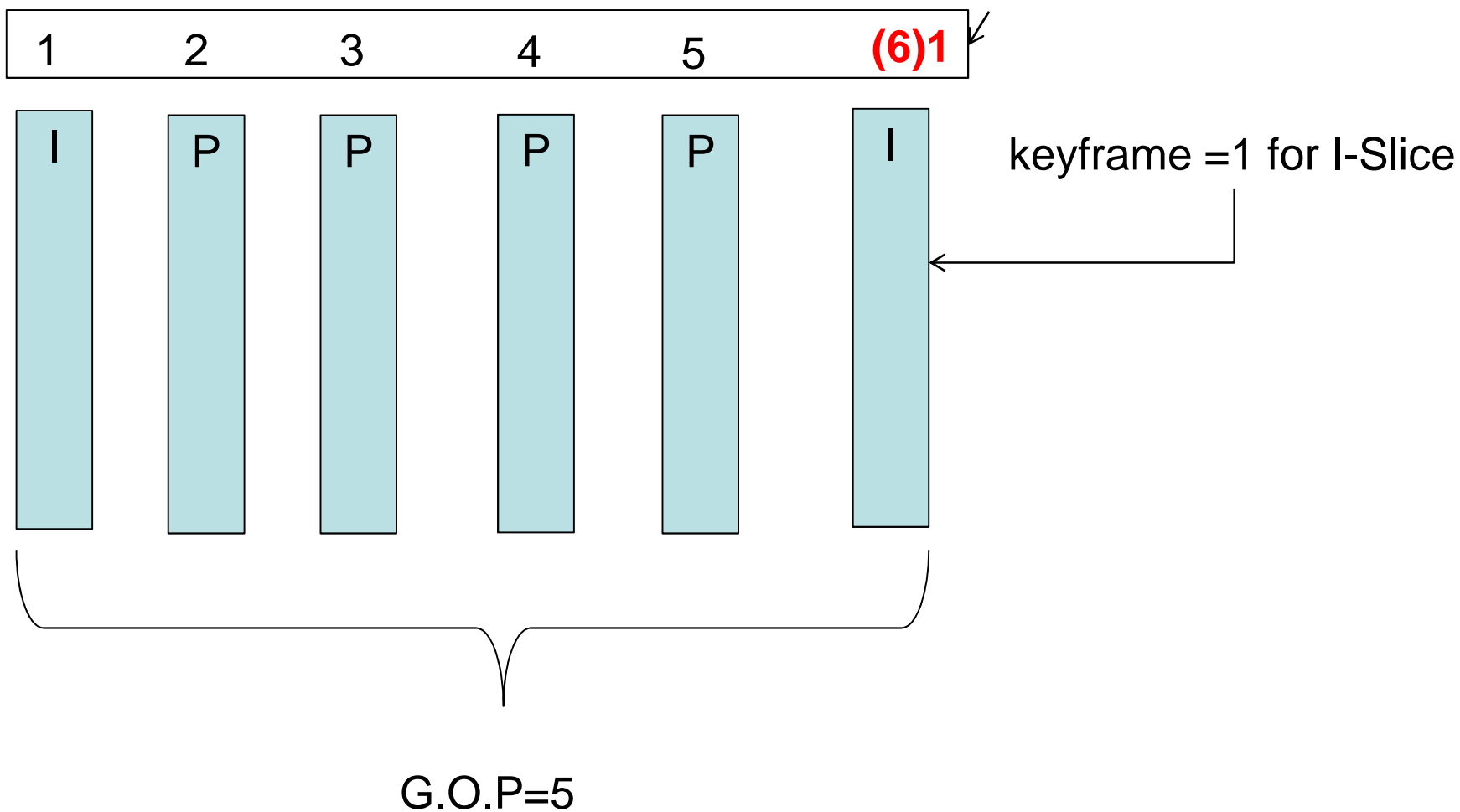
Encoder Init

```
// 1 : force the encoder to create all I Frame
// 0 : forces the encoder to crate a P frame
// -1 : let the encoder decode (based on the u3IPIntervale)
enc_param.intra = -1;
enc_param.u32IPInterval = 30; // G.O.P.
enc_param.u32BitRate = 64*1000; // unit : bps
enc_param.fFrameRate = 30; // vui.num_nuits_in_tick
// 1: force encoder to output sps+pps
// 0: force the encoder to output sps+pps on any I frmae
// -1 : (default) only output sps+pps on first IDR frame
enc_param.ssp_output = -1;
```

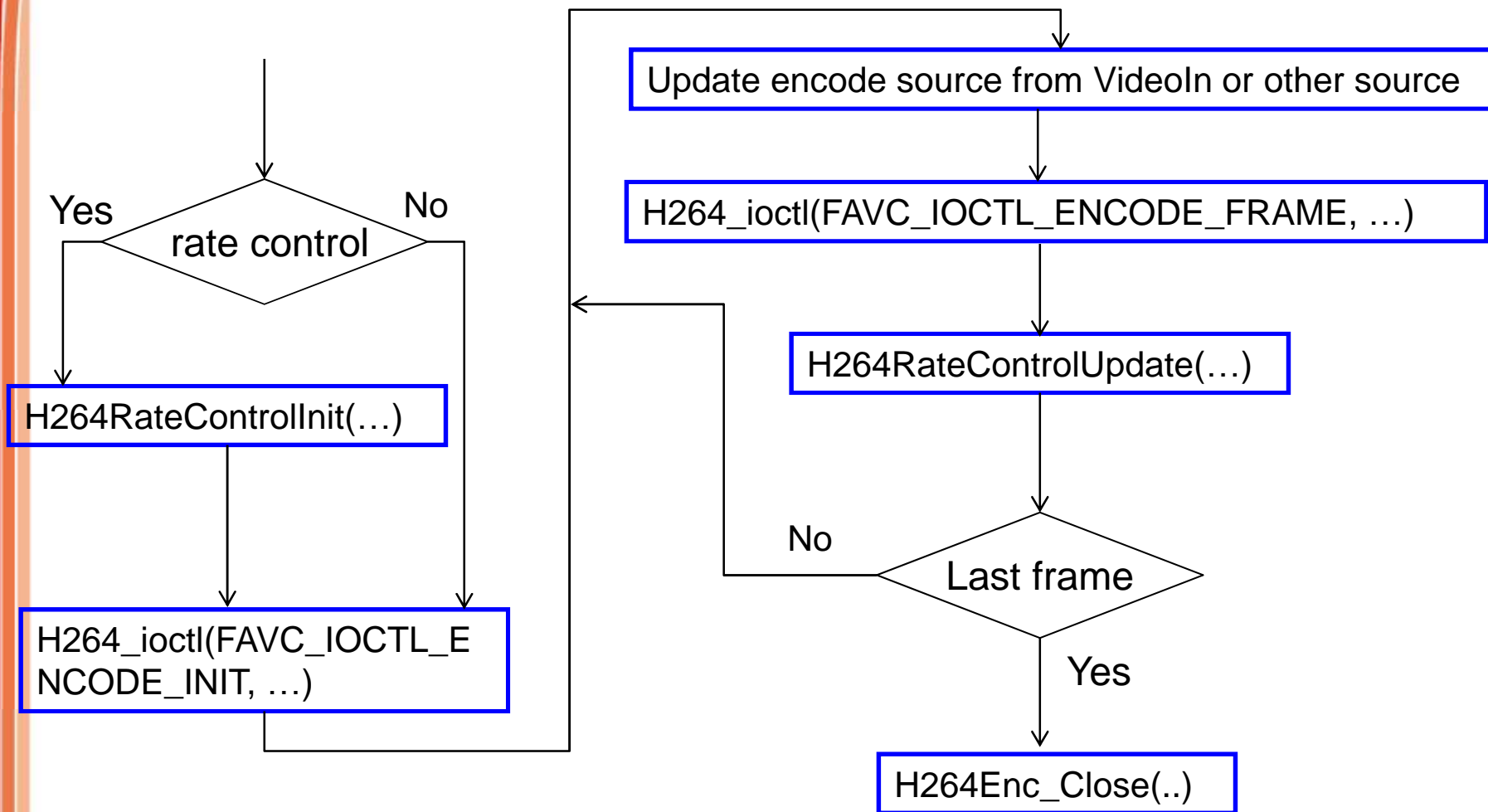
H264_ioctl(FAVC_IOCTL_ENCODE_INIT, & enc_param) //non_OS

or

ioctl(favc_enc_fd,FAVC_IOCTL_ENCODE_INIT, &enc_param) // Linux



Encode Flow



Yes

Decoder Init

Defined in favc-avcodec.h

```
typedef struct
{
    UINT32 u32API_version;           // API version (0x00010000)
    UINT32 u32MaxWidth;              // Not used now
    UINT32 u32MaxHeight;             // Not used now
    UINT32 u32FrameBufferWidth;      // if (u32FrameBufferWidth != -1), decoded image width is cropped with u32FrameBufferWidth
                                     // if (u32FrameBufferWidth == -1), decoded image width is continued on memory
    UINT32 u32FrameBufferHeight;     // if (u32FrameBufferHeight != -1), decoded image height is cropped with u32FrameBufferHeight
                                     // if (u32FrameBufferHeight == -1), decoded image height is continued on memory
    UINT32 u32Pkt_size;              // Current decoding bitstream length ( the exact bitstream length for one frame)
    UINT8 *pu8Pkt_buf;               // Current decoding bitstream buffer address (application ready bitstream here)
    UINT32 pu8Display_addr[3];       // Buffer address for decoded data
    UINT32 got_picture;              // 0 -> Decoding has something error. 1 -> decoding is OK in current bitstream
    UINT8 *pu8BitStream_phy;         // physical address. buffer for bitstream (allocated and used by library only)
    UINT32 u32OutputFmt;             // Decoded output format, 0-> Planar YUV420 format, 1-> Packet YUV422 format
    UINT32 crop_x;                   // pixel unit: crop x start point at decoded-frame (not supported now)
    UINT32 crop_y;                   // pixel unit: crop y start point at decoded-frame (not supported now)

    FAVC_DEC_RESULT tResult;         // Return decoding result by library
} FAVC_DEC_PARAM;
```

H264_ioctl(FAVC_IOCTL_DECODE_INIT, & dec_param)

Decode Flow

