AN H.264/AVC DECODER FRONTEND TARGETING BROADCASTING DTV FOR SBTVD

Márlon A. Lorencetti, Letícia V. Guimarães, Altamiro A. Susin

Universidade Federal do Rio Grande do Sul

ABSTRACT

This work proposes an input interface to an H.264 video decoder targeting real broadcasting conditions. Switching on the decoder on an arbitrary moment of the input stream, for example because of power on or channel change, may lead to an incomplete set of data for the decoding process. Furthermore, signal loss or a global reset must be suitably undertaken by the decoder. From an error-free input stream, test conditions are generated by means of suppression of selected information. The interface receives the input signal as a sequence of NALUs (Network Abstraction Layer Unit), detects these units, communicates with the global control module and sends the RBSP (Raw Byte Sequence Payload) data to the input buffers of the entropy decoder modules. An elementary control was defined to process the protocol functions generated in the interface. The input interface and the control functions were implemented in software, following the co-design strategy adopted by the decoder development team. Some experiments were performed using this implementation, where it was possible to test some particular situations such as cleanup at power on.

1. INTRODUCTION

This work is part of the development and prototyping of a hardware H.264/AVC [1] video decoder for the SBTVD (Sistema Brasileiro de Televisão Digital) [2][3], the digital television system adopted in Brazil. A consortium of research laboratories in universities all over Brazil is developing all the coding and decoding IPs that are compliant with this system.

The existence of a data interface between the video decoder and the rest of the set-top box (or access terminal) is evident [3], and considering that different development groups are responsible for those tasks, there is the need to specify and simulate real conditions before the actual hardware implementation. The study of data transfer in this case leaded to a specification of signal requirements, which had to be put under test to verify whether it reached its purpose.

The input interface and the control functions were implemented in software, as a part of the co-design strategy being used by the H.264 decoder development group [4]. Therefore, the implemented software must follow the premises to attend the interface specification and be as modular as the proposed hardware, in order to aid the hardware project by acting as the first experimenting model to determine some parameters to be adopted in the development and validation.

In the section 2 of this work, are presented the protocol used in the bitstream of an H.264 coded video and the interface duties in the decoding process. The third section presents the proposed architecture and its working method. Section 4 shows the software experiments that were made in order to test the proposed scheme. Conclusions and future works are given in sections 5 and 6, respectively.

2. DECODER INPUT INTERFACE

2.1. Bitstream structure

The H.264 video bitstream comes encapsulated in NALUs, a sequence of byte aligned information that contains a heading byte. The first bit shall be equal to zero, the next 2 bits indicate whether the NALU class and the remaining 5 bits inform the NALU type [1]. The RBSP is the part of the NALU where the actual data is carried, which can be either image data, parameter sets or control actions. An access unit is a set of NALUs containing exactly one coded picture [1].

In order to detect the start of a new NALU, a start_code (0x000001) is sent in the bitstream. When this particular sequence needs to be sent as RBSP data, the encoder inserts a prevention byte (0x03), called emulation_prevention. This byte needs to be discarded when it comes in one of the following sequences: 0x0000**03**00, 0x0000**03**01, 0x0000**03**02, 0x0000**03**03. Additionally, an extra byte (0x00) called zero_byte shall be present before the start_code when the NALU is the first on its access unit or when the NALU contains a PPS (Picture Parameter Set) or SPS (Sequence Parameter Set) [1].

2.2. General interface features

The H.264 decoder input interface is usually seen as a buffer that stores the bits or bytes composing the NAL units sequence, also called video ES (Elementary Stream). These input data buffers are usually circular buffers, as shown in [5]. In this case, the detection and interpretation of the NAL units are allocated in the parser module, when in accord with this group, the parser module should be used to distribute the data to the corresponding processing modules in the video decoder.

The proposed interface module must receive and store the data coming from the demultiplexer, identify the NALUs and send the coded data to the processing

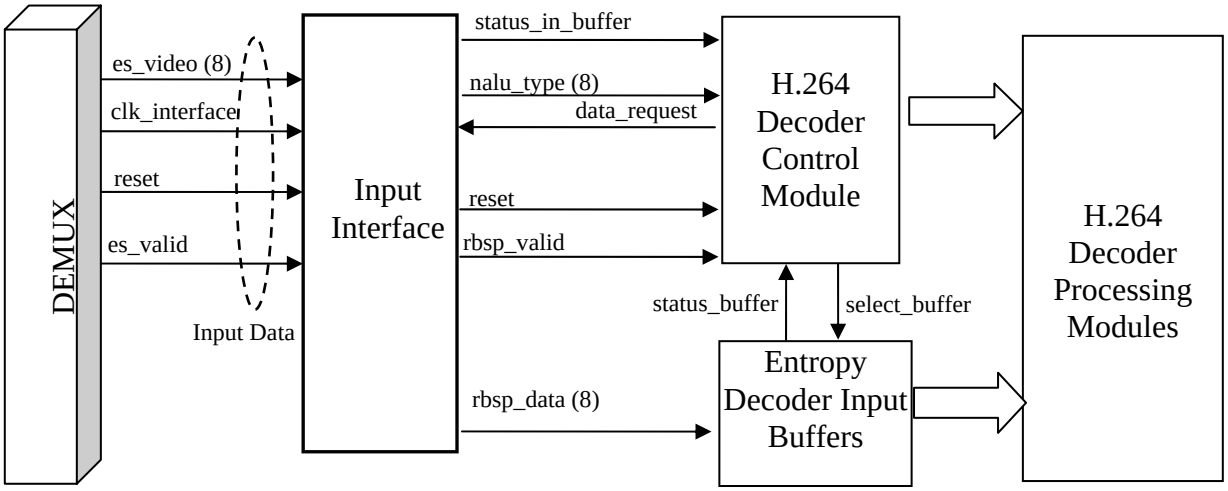

Fig. 1 – Proposed architecture

modules, while preventing that uncompleted or invalid data is stored in their input buffers, such as header and protocol signals.

The interface communicates with the global control, allowing a data flow control. The global control must monitor the status of the processing modules input buffers and inhibit reading/writing until the buffer is available. It must detect the signals informing a new NALU and the NALU type and recognize the required decoding conditions.

The contribution in this work is the interface design targeting broadcasting conditions. That is, the interface modules, and mostly the control modules, must be able to handle particular situations such as signal loss, output buffers overflow, input buffers underflow, a global reset, the lack of configuration parameters or reference image data. Considering this, the decoder must wait for the required decoding conditions, which shall happen after at most 5 seconds, as specified in [3]. These situations are usual in a broadcast system like the one where the final decoder will operate. Although, most of the related works found in the literature work with a regular input data reception, such as a video stored in memory or in a file [5][6]. In this kind of input, there is much higher communication reliability and all the required parameters are available in the beginning of the file.

Besides, other software implementations are not written with this specific purpose. For example, the JM Reference Software [7], the standard software for H.264 decoding, receives the bitstream as a file and is not able to decode that file if it does not begin with an NALU start_code (0x000001). In the same way, JM is not able to handle files where there is image data sent before the parameter sets.

3. PROPOSED ARCHITECTURE

In the proposed architecture, there is a control module that requests data and is informed by the interface modules when control actions are required. Signals are used to inform the detection of a new NAL unit, the type of NAL unit, the buffers status, and the existence of valid data.

After a reset, the control module requests a byte. The data incoming to the input buffer changes the signal *status_in_buffer* in the input interface data, allowing them to process the stored data. The reset values of the interface signals are shown in Tab. 1.

Tab. 1 – Architecture reset signal values

Signal	Reset Value	Source
reset	1	Demultiplexer
status_in_buffer	00 (empty)	Interface module
rbsp_data	XX (invalid)	Interface module
rbsp_valid	0	Interface module
nalu_type	00 (invalid)	Interface module
data_request	0	Control module

The data request from control module leads to the analysis of 4 bytes from the input buffer in the start_code/emulation_prevention detection module. The requesting process is shown in Tab. 2, where FIFO position filled with -1 represent invalid data; and status_fifo can be empty/half full/full, represented with values 0, 1 or 2, repectively. The status of the signal shown in Fig. 2 are represented in the columns of the table, where each line represents a cycle.

The 4 bytes FIFO reads a byte from the input buffer requesting data with *data_req1*=1 and waiting for the byte to be available with *byte_a*=1. This repeats until the FIFO is full. During normal operation, a byte is inserted in the tail of the FIFO while another is discarded from the head. Every byte insertion triggers the comparison searching for the start_code (0x00000001 or 0xXX000001) or the emulation_prevention (0xXX000003). If a start_code is found, the detector sends a signal to the NALU analyzer (*sc_detected*=1)

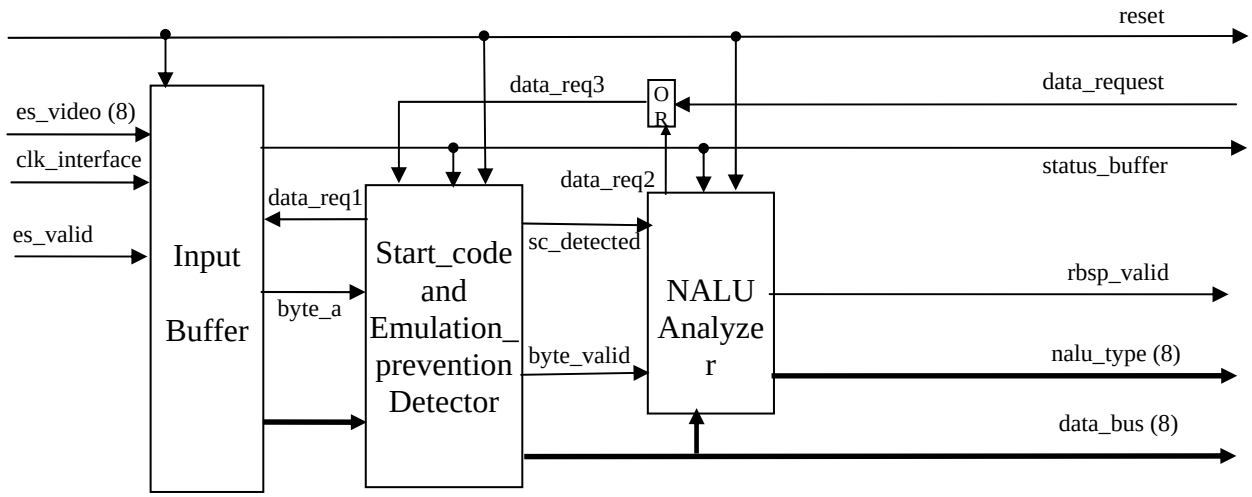

Fig. 2 – Proposed interface module

reading 3 or 4 new bytes from the input buffer. If it is an emulation_prevention, then the 0x03 byte is just ignored. Else, the RBSP byte at the head of the FIFO is sent to the data bus and the signal *rbsp_valid* is set to 1. The control module enables the corresponding module from the decoder to read the data bus.

When the NALU analyzer gets the *sc_detected* signal, it requests the header byte from the start_code/emulation_prevention module with *data_req2*=1, which is the next byte after the start_code. Reading the byte from the data bus with *byte_valid*=1, it decodes the NALU type and discards the header byte from the data bus requesting another byte. The NALU type is sent to the control module, which already has available the first RBSP byte in the data bus, with *rbsp_valid*=1.

The control module then uses the byte from the bus, stopping the data request with *data_request*=0, remaining in this state while the control module processes the current byte. The process restarts with a new data request, with *data_request*=1. The FIFO content throughout the input processing is shown in Tab. 2.

4. EXPERIMENTS AND RESULTS

Before the actual hardware implementation, the proposed architecture was implemented in software. This co-design strategy has been used by the video decoder development group. A Matlab routine was then written in order to validate this structure. This Matlab program performs the control actions and verifies the state of each signal per cycle, storing them on a file for further debugging. The routine was written in a structure that is similar to the proposed hardware, encapsulating every hardware module in a function and avoiding execution loops.

The input file is generated with the JM software encoder [7], setting the encoder to resend SPS and PPS in a certain period of frames. Then some bytes were removed from the beginning of the file in order to simulate a real broadcast, where the user will not necessarily turn on the decoder at the right time to get a new NALU with the required parameter sets. The data removal was made in such a way that the file does not always begins with a start_code. Alternatively, it begins with a NALU with a picture sent before the parameters needed to decode it.

The behavior of the interface and control modules was observed and captured into files. The sequence of control actions and the state of each signal matched the expected results, proving the proposed architecture a feasible effort to get the input data and control this part of the decoding process.

Besides, some experiments were made in the PRH.264 software decoder developed by the decoder team [4]. In this C written software, some structures were created. One of them implements a FIFO, with some auxiliary functions to clear, read or write from the FIFO, allocate and free its memory. This structure will be extensively used in the interface modules and in the processing blocks, to simulate their input buffers. The other structure implements the control signals indicating the states of the buffers or the detection of protocol and parameters. Some functions use this structure to implement the control module, checking and changing the values of the signals.

5. CONCLUSIONS

This paper presented an interface and control method to be implemented in the hardware video decoder for the SBTVD. The experiments showed that this architecture meets its requirements, detecting the NAL protocol and sending a raw data payload to the processing modules. Also, the frequency of the data arrival from the demultiplexer is considerably smaller than the frequency of these modules, assuring that the actions performed here will not disturb the data flow.

6. FUTURE WORKS

The control actions must be expanded to cover all the modules of the decoder, performing a correct data transfer between modules, and assuring the synchronization over the decoding process. In the PRH.264 software, the NALU decoder shall recognize the control messages from the units EoS (End of Sequence), EoStream (End of Stream) and FD (Filler Data). Besides, the obvious sequel of this work is its implementation in hardware with an HDL description, simulation and FPGA prototyping.

7. REFERENCES

[1] Video Coding Experts Group, "ITU-T Recommendation H.264 (03/05): Advanced video coding for generic audiovisual services", *International Telecommunication Union*, 2005.

[2] "ABNT NBR 15602-1 Televisão Digital Terrestre – Codficação de vídeo, áudio e multiplexação", *ABNT*, Rio de Janeiro-RJ, 2007.

[3] "ABNT NBR 15604 Televisão Digital Terrestre – Receptores", *ABNT*, Rio de Janeiro-RJ, 2007.

[4] M.A. Lorencetti, W.T. Staehler, A.A. Susin, "Incremental Hardware Development from Modular Mixed C-VHDL Simulation", 8th Students Forum on Microelectronics SForum'08, Gramado-RS, 2008.

[5] K. Xu, C.S. Choy, "Low-power H.264/AVC Baseline Decoder for Portable Applications", 2007 International Symposium on Low Power Electronics and Design, Portland, USA, 2007.

[6] M. Brown, K.W. Hsu, "A Novel 5.46mW H.264/AVC Video Stream Parser IC", SOC Conference, *IEEE International*, 2008.

[7] "H.264 Reference Software", http://iphom.hhi.de/suehring/tml/, 2009.

Tab. 2 – Signals vs cycles

cycle	reset	FIFO				status_fifo	data_request	data_req1	data_req2	sc_detected	rbsp_valid	nalu_type
0	1	-1	-1	-1	-1	0	0	0	0	0	0	0
1	0	0	-1	-1	-1	1	1	0	0	0	0	0
2	0	0	0	-1	-1	1	1	0	0	0	0	0
3	0	0	0	0	-1	1	1	0	0	0	0	0
4	0	1	0	0	0	2	1	0	0	0	0	0
5	0	-1	-1	-1	-1	0	1	1	0	1	0	0
5	0	105	-1	-1	-1	1	1	0	0	1	0	0
6	0	16	105	-1	-1	1	1	0	0	1	0	0
7	0	0	16	105	-1	1	1	0	0	1	0	0
8	0	0	0	16	105	2	1	0	0	1	0	0
9	0	0	0	0	16	2	1	0	0	0	1	9
10	0	1	0	0	0	2	1	0	0	0	1	9
10	0	1	0	0	0	2	0	0	1	0	0	9
11	0	-1	-1	-1	-1	0	1	1	0	1	0	9
11	0	103	-1	-1	-1	1	1	0	0	1	0	9
12	0	66	103	-1	-1	1	1	0	0	1	0	9
13	0	0	66	103	-1	1	1	0	0	1	0	9
14	0	40	0	66	103	2	1	0	0	1	0	9
15	0	231	40	0	66	2	1	0	0	0	1	7
16	0	64	231	40	0	2	1	0	0	0	1	7
17	0	176	64	231	40	2	1	0	0	0	1	7
18	0	75	176	64	231	2	1	0	0	0	1	7
19	0	32	75	176	64	2	1	0	0	0	1	7
20	0	0	32	75	176	2	1	0	0	0	1	7