Complexity Analysis of H.264 Decoder for FPGA Design

Tuomas Lindroth, Nastooh Avessta, Jukka Teuhola and Tiberiu Seceleanu
Dept. of Information Technology, Communication Systems Laboratory
University of Turku
Turku, Finland
E-mail: {tuomas.lindroth, nastooh.avessta, jukka.teuhola, tiberiu.seceleanu}@utu.fi

Abstract— A major challenge in the design of any real time system is the proper selection of implementation and platform alternatives. In this paper, a suitable FPGA-based design of the H.264 decoder is presented. Since H.264 standard only specifies the syntax and semantics of the video stream and not the video codec itself, the selection process may be directed based upon the temporal complexity of different parts of the decoder. Here, we present the process flow of these parts using basic algebraic operators. The analysis of the required logic elements to implement the decoder, on various platforms, is presented.

H.264 decoder; decoding block; baseline profile

I. INTRODUCTION

H.264, also known as MPEG-4 part 10 or Advanced Video Coding (AVC) [1], is a recent video compression technique, which is a successor to H.263 and MPEG-2 standards. Like the earlier video coding standards, H.264 does not specify a video encoder. Instead, it specifies the syntax of a coded bit stream, the semantics of these syntax elements and the process by which the syntax elements may be decoded [1]. In this paper, we concentrate on the decoder.

H.264 defines seven *profiles*, each supporting a set of coding functions and specifying what is required of an encoder and decoder. The baseline profile is the basic profile, the simplest and, at least partly, similar to MPEG-2 and H.263 [2]. It is mainly intended for videoconferencing and wireless communications. Here we concentrate on the baseline profile, for which we analyze the logic element requirements needed in an FPGA implementation.

To achieve efficient development, hardware requirements have to be reliably identified. Temporal complexity analysis offers a possibility to envision a roadmap without delving into the details of the implementation. As H.264 codec design includes both the encoder and the decoder, for seven different profiles, it is imperative to readily assess the design specification for different alternatives.

The organization of this paper is as follows. Section II provides an overview of the H.264 decoder and the detail description of the complexity analysis. In Section III a resource estimation model is presented and analyzed. Finally, Section IV summarizes the results.

II. DESIGN PROBLEM

The design problem is to identify the amount of memory and logic needed to implement an H.264 decoder. As shown in Figure 1, the decoder consists of nine, basically independent, blocks that can be considered separately. The compressed bit stream from the NAL (Network Abstraction Layer), after processing, is available as F'$_n$(reconstructed) block.

We use the terms luma and chroma rather than the terms luminance and chrominance in order to avoid the implication of use of linear light transfer characteristics that is often associated with the latter terms [2].

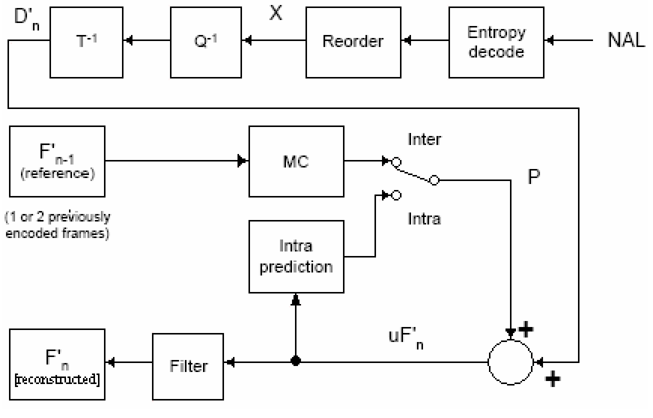

Figure 1 *H.264 decoder* [1]

Each block is analyzed by determining the maximum number of basic algebraic blocks needed to perform that operation, for an 8 bit input. The word ROM represents all the constant values in memory within a certain block, e.g. coefficients. The connotation RAM is used to denote the changing values, e.g. inputs and outputs. We have used the style of 'YxX' to represent a matrix size of Y rows and X columns.

Entropy decoding block includes two different decoding schemes: CAVLC (context-adaptive variable length coding) for decoding residual data [3] and Exp-Golomb Coding for other coded units [1]. CAVLC implementation is quite complicated but doesn't require many logic elements; therefore it is simplified to look-up tables. Exp-Golomb is shown in Figure 2.

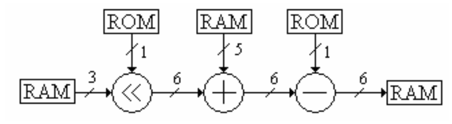

Figure 2 ***Exp-Golomb Process Flow***

Reorder block can be implemented in various ways. We chose to use a look-up table, which only requires storage elements.

Inverse quantization has three different variations: one for residual, one for luma DC coefficients and one for chroma DC coefficients. The 4x4 luma DC coefficient inverse quantization is used when the macroblock is encoded in 16x16 intra prediction mode [1]. In DC coefficient inverse quantization we have used value 51 for QP (quantization parameter). This is its maximum value and is used in order to assess the worst case scenario, in terms of operational complexity. Note that matrix operations (Hadamard transform) are true matrix multiplications and not element-wise. These three variations are shown in Figure 3, Figure 4 and Figure 5

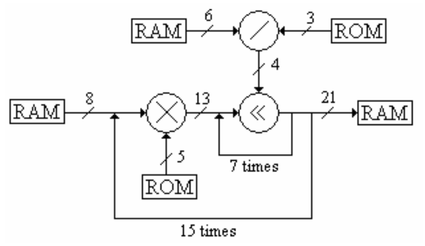

Figure 3 ***4x4 Residual Inverse Quantization***

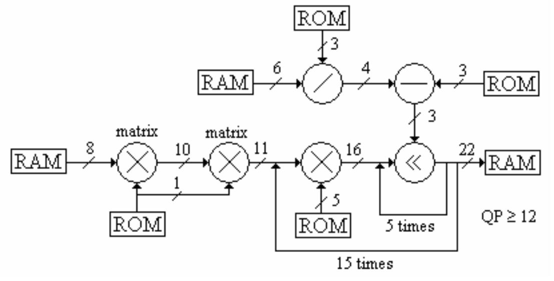

Figure 4 ***4x4 Luma DC Coefficient Inverse Quantization***

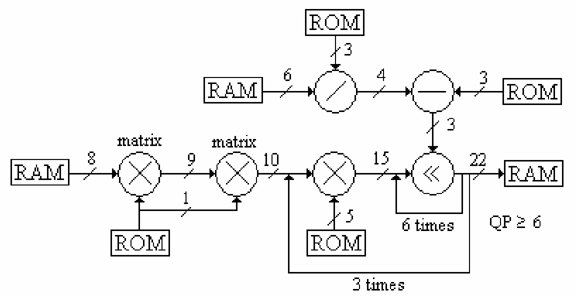

Figure 5 ***2x2 Chroma DC Coefficient Inverse Quantization***

Inverse transform is the same for all of the different inverse quantization modes. The rescaled 2x2 chroma coefficients are first replaced in their respective 4x4 blocks, which are then transformed [1]. The Inverse transform process flow is shown in Figure 6.

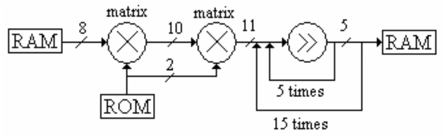

Figure 6 ***4x4 Inverse Transform***

Intra prediction has its highest operational complexity, when 16x16 luma prediction mode 3 and 8x8 chroma prediction mode 3 are used [1]. These modes are called *plane*. (Also 4x4 luma prediction modes, total of 9 elements, are possible.) The different implementations of all of these modes (without possible initialization) are shown in Figure 7, Figure 8 and Figure 9.

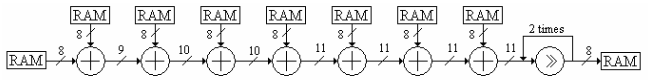

Figure 7 ***4x4 Luma Prediction Mode 2***

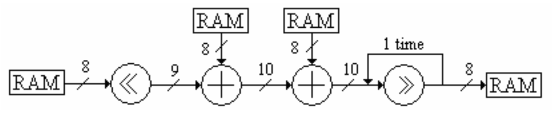

Figure 8 ***4x4 Luma Prediction Modes 3–8***

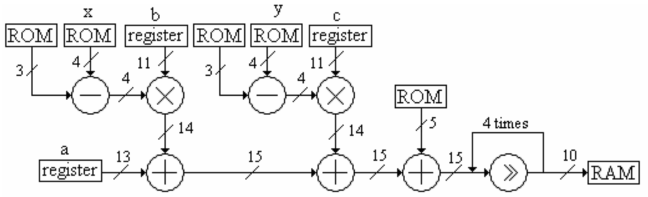

Figure 9 ***16x16 Luma and 8x8 Chroma Prediction Mode 3***

MC (motion compensation) consists of motion vector prediction and interpolation. Motion vector calculation is quite simple in the terms of logic elements; therefore it is excluded. *Inter prediction* (interpolation) uses quarter-pel samples for luma and eight-pel samples for chroma [1]. Luma samples are calculated in two phases: 1) the half-pel samples, by using six-tap FIR filter, and 2) the final values, by linear interpolation of half-pel samples. These processes are shown in Figure 10, Figure 11 and Figure 12.

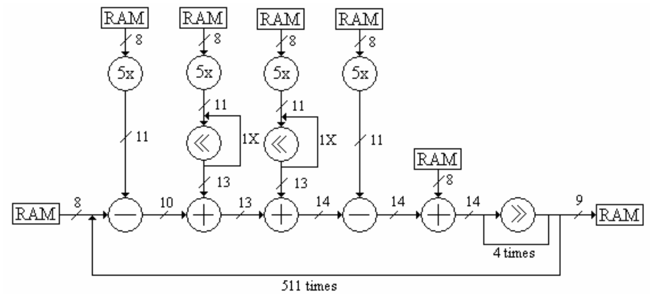

Figure 10 ***Half-pel Interpolation (Luma)***

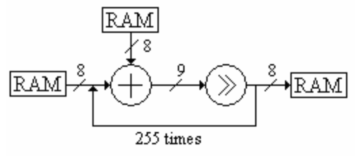

Figure 11 ***Quarter-pel Interpolation (Luma)***

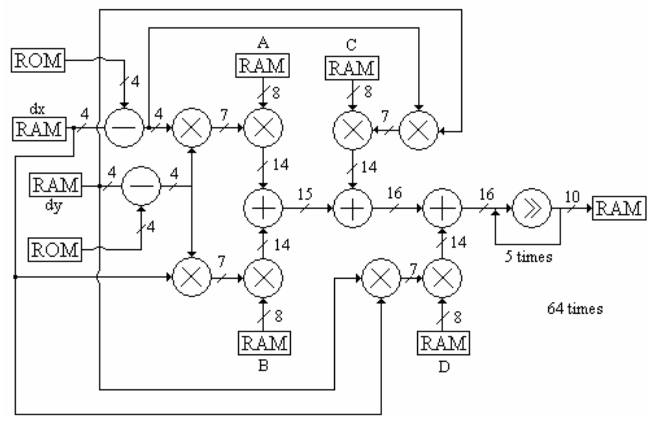

Figure 12 ***Eight-pel Interpolation (Chroma)***

Deblocking filter has its worst case scenario when Bs (boundary strength) is 4. Then 4 five-tap and 2 four-tap filtering are needed for luma and 2 three-tap filtering for chroma [4]. These are shown in Figure 13 to Figure 16.

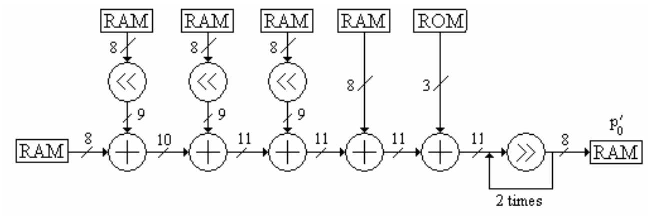

Figure 13 ***Deblocking Filter (Luma), Five-tap, Output p'$_0$***

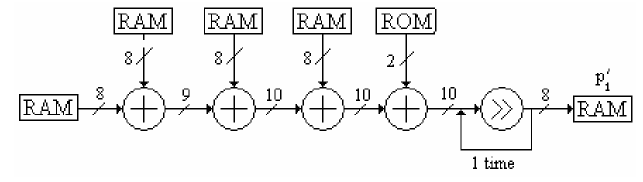

Figure 14 ***Deblocking Filter (Luma), Four-tap, Output p'$_1$***

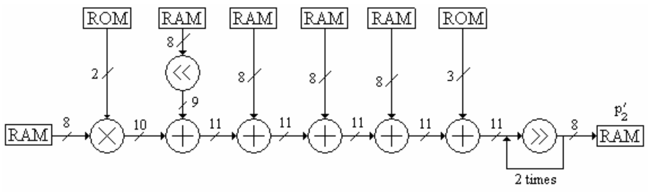

Figure 15 ***Deblocking Filter (Luma), Five-tap, Output p'$_2$***

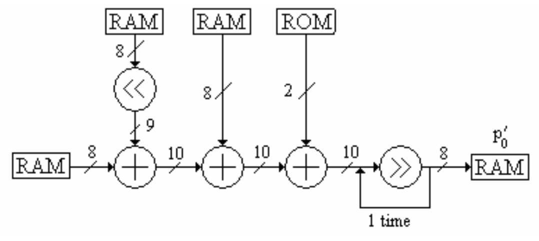

Figure 16 ***Deblocking Filter (Chroma), Three-tap***

III. RESOURCE ESTIMATION MODEL

We have used Altera Quartus II tools to estimate the required logic elements of an FPGA. As product families we have used Stratix, Stratix II and Cyclone II. Stratix II uses ALUTs (adaptive look-up table), which are approximately the same as 1.25 logic elements (LE) [5]. Thus all ALUTs in simulation results are multiplied by this factor to estimate the number of corresponding LEs. Memory bits are calculated by evaluating the storage requirements of processes and their respective input and output.

The simulation models are created by using the basic operations shown in the process flow figures in Section II, excluding the inverse transform, where we have used an optimized 1D DCT transform VHDL code by [6].

The matrix multiplications shown in the inverse quantization are broken down to simple multiplications and additions. Luma and chroma values are added together in the final result to form a complete overall picture of the required logic elements.

Table III presents the specifications of the FPGAs obtained from Altera Quartus II that were considered in the resource estimation model. Table I presents the results, for a block-parallel implementation. We are using the worst case scenario, so intra prediction, inter prediction and filter use block sizes of 16x16 for luma and 8x8 for chroma, which are called macroblocks. The corresponding worst case sizes for the other four blocks are 4x4 for luma and 2x2 for chroma.

Table II outlines the logic element and memory requirements for the serial implementation, where values calculated are for one 1x1 block.

Table I ***Results for Parallel Implementation***

FPGA / Block	Logic elements			Memory bits
	Stratix	Stratix II	Cyclone II	
Entropy decode	16	9	14	2600
Reorder	1	1	1	180
Inverse quantization	1199	1151	1163	1150
Inverse transform	4664	4152	4672	500
Intra prediction	58560	37440	56640	4000
Inter prediction	69504	62464	67776	9900
Filter	46848	48832	45056	46000
Total	**180792**	**154049**	**175322**	**64330**

Table II ***Results for Serial Implementation***

FPGA / Block	Logic elements			Memory bits
	Stratix	Stratix II	Cyclone II	
Entropy decode	16	9	14	2600
Reorder	1	1	1	180
Inverse quantization	1199	1151	1163	1150
Inverse transform	583	519	584	500
Intra prediction	183	117	177	100
Inter prediction	612	505	591	150
Filter	402	410	385	400
total	**2996**	**2712**	**2915**	**5080**

Table III FPGAs Used in the Simulation Model

FPGA	Logic elements	Memory bits
1 – Stratix (EP1S10F484C5)	10570	920448
2 – Stratix II (EP2S15F484C3)	15600	419328
3 – Cyclone II (EP2C5T144C6)	4608	119808

The above description contained only the main low-level logic element requirements. The high-level control should be added in a complete implementation.

It is noted that a fully parallel implementation of the H.264 decoder will not fit in any of the considered FPGAs. Nonetheless, as shown in Table II and Table III, a partially serial implementation will suitably fit in all the considered FPGAs. However, the degree of concurrency is highly dependent on specific design requirements and constraints, and as such is not the concern of this study.

As a comparison, [7] implements H.264 Baseline Encoder Core (without filter) using 129000 logic gates and [8] H.264 Baseline Video Decoder IP Core with 150000 gates. Both of these logic element requirements are less than the upper bound we defined in Table I.

IV. CONCLUSION

In this paper we have shown that at early stages of design, it is possible to obtain upper bounds on design specifications. As our guideline, we have chosen the worst case scenario, in terms of temporal complexity, to identify the design limits.

Future work will attempt to identify the optimum design specifications, based on project specific objectives and constraints.

REFERENCES

[1] I.E.G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia", John Wiley & Sons, Ltd, pp. 159–207, 2003.

[2] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC) Joint Video Team (JVT), Doc. JVT-G050r1, May. 2003.

[3] G.Bjøntegaard and K.Lillevold, "Context-adaptive VLC (CVLC) coding of coefficients", *3rd Meeting of the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG* Fairfax, VA, Doc. JVT-C028rl, May 6–10, 2002.

[4] P. List, A. Joch, J. Lainema, G. Bjøntegaard and M. Karczewicz, "Adaptive Deblocking Filter", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003

[5] Altera Corporation, "Stratix II Device Family Data Sheet", pp. 1-2, 2005. http://www.altera.com/literature/hb/stx2/stx2_sii5v1_01.pdf

[6] T. Sherif, "VLSI Design and Implementation of Different DCT Architectures for Image Compression", B.Sc. Graduation Project, Electronics and Communications Engineering department, Ain Shams University, Cairo, Egypt, 2000.

[7] V. Liguori and K. Wong, "Designing A Real-Time HDTV 1080p Baseline H.264/AVC Encoder Core", DesignCon 2006

[8] 4i2i Communication Ltd., "H.264 / MPEG-4 Part 10 Baseline Video Decoder IP Core", Rev 1.0, December 2005, pp.1-7. http://www.4i2i.com/downloads/H264BaselineIPDecoderCore.pdf