CABAC Entropy Decoding Algorithm Implementation on FPGA For H.264

Rohan.A.Kandalkar and Prof(Mrs) Manisha.R.Ingle

***Abstract*- The demands for high quality, real-time performance and multi-format video support in consumer multimedia products are ever increasing. In future multimedia systems require efficient video coding algorithms and corresponding adaptive high-performance computational platforms. The H.264 video coding algorithm provide high enough compression efficiency to be utilized in these system and multimedia processor are able to provide the required adaptability. However, the algorithm complexity demands far more efficient computing platform. Context-based adaptive binary arithmetic coding (CABAC) is used in the H.264 video standard. This gives better compression efficiency. With better complexity Heterogeneous reconfigurable systems composed of multimedia processor and hardware accelerator, constitute the main part of algorithm. In this paper the hardware accelerator architecture for CABAC of main module and high profiles of H.264 are highlighted. An FPGA implementation of the CABAC entropy decoding process is used in co-operation with the decoding software on a Xilinx Spartan-6 platform[2].**

Keywords: **CABAC, FPGA, H.264, Spartan-6**

I. INTRODUCTION

Context based adaptive binary arithmetic coding (CABAC) used in H.264 is an extension of the binary arithmetic coding (BAC), where the coding offset value is dynamically adjusted based on the syntax element being encoded/decoded. While the coding efficiency of CABAC is superior to the conventional Huffman coding, the improvement comes with an increasing performance requirement; at least 3 GHz of computing power is required for the real-time decoding of a HD sequence if a general-purpose, yet high-speed, RISC machine processes the syntax parsing. As HD digital TV broadcasting coded in H.264. Main Profile and High Profile is being widely spread at the present, the necessity of a high-speed CABAC decoder is growing.

CABAC originates from the generic arithmetic coding to which a majority of researches have been devoted to remove the multiplication. Skew Coder, Q-Coder and other variants are the examples. Most of the works also have tried to approximate real-valued symbol probabilities to integer values. As a result, the generic arithmetic coding has evolved to CABAC. However, there have been relatively few works on the practical implementation of CABAC decoding. This approach is also found in which speeds up the decoding by using pointer chains to retrieve the context models for the subsequent bins and storing the intermediate results of context selection instead of the original reference macro-block data. However, there is no specific acceleration scheme except that the decoding is controlled by an optimized finite state machine (FSM). Moreover, it is reported that the performance is just enough to accommodate CIF 30 fps[4].

II. CABAC DECODER

The CABAC is a binary adaptive arithmetic coding composed of four components: syntax element decoding, context modelling, three decoding processes and anti-binarization as is shown in Figure 1.

A. Input bit stream

The input bit stream stored in a SDRAM memory which is associated to microblaze processor. A microblaze processor created using xilinx XPS tool. The XPS(xilinx platform studio) consist of EDK (embedded development kit) and SDK(software development kit), the actual hardware is generated in EDK and the software part is written in SDK. Firstly the image in JPEG/PNG/BMP format is converted into .txt format file with the help of MATLAB coding and this file stored first in PC and then with the help of RS232 it transfer to SDRAM memory of microblaze processor.

Rohan.A.Kandalkar and Prof(Mrs) Manisha.R.Ingle are with Maharashtra Institute of Technology, Pune, Email: rohankandalkar02@gmail.com, manisha.ingle@mitpune.edu.in

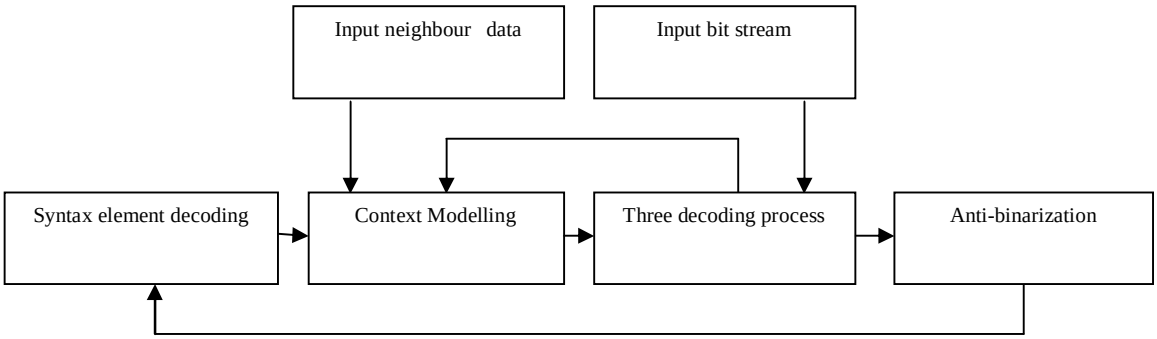

Figure-1: CABAC Decoder Block Diagram

CABAC DECODER FLOW

Figure-2 shows the flow chart of CABAC decoding. At the beginning of new slice, it build the context table from the initial table. Then initialize codlOffset and codlRange by using the first 2 bytes of the bit stream. After initializing the context table, codlOffset and codlRange, go on to decode a macroblock. In the macroblock (MB) layer, the CABAC decoder first performs syntax element decoding. It should determine on which syntax element (SE) to decode. Secondly it does context modeling. It calculates the context value by referring to the syntax elements of the left, top, and current macroblocks. Then, it performs one of these three decoding process: regular decoding, bypass decoding, terminal decoding. In the regular decoding process, the decoder will update codlRange by looking up the rangeTabLPS table and update the corresponding entry in the context table by looking up the transIdxLPS table and the transIdxMPS table. It also updates codlOffset by reading from the bit stream. After the regular decoding process, the decoder may either perform the bypass decoding process or go on to decode the next syntax element. In the terminal decoding process, if the decoding result is "1", the decoder goes to decode the next slice otherwise, it needs to decode the next macroblock. After the regular or bypass decoding process, the CABAC decoder performs an anti-binarization process that translates the decoding result into the syntax element value.

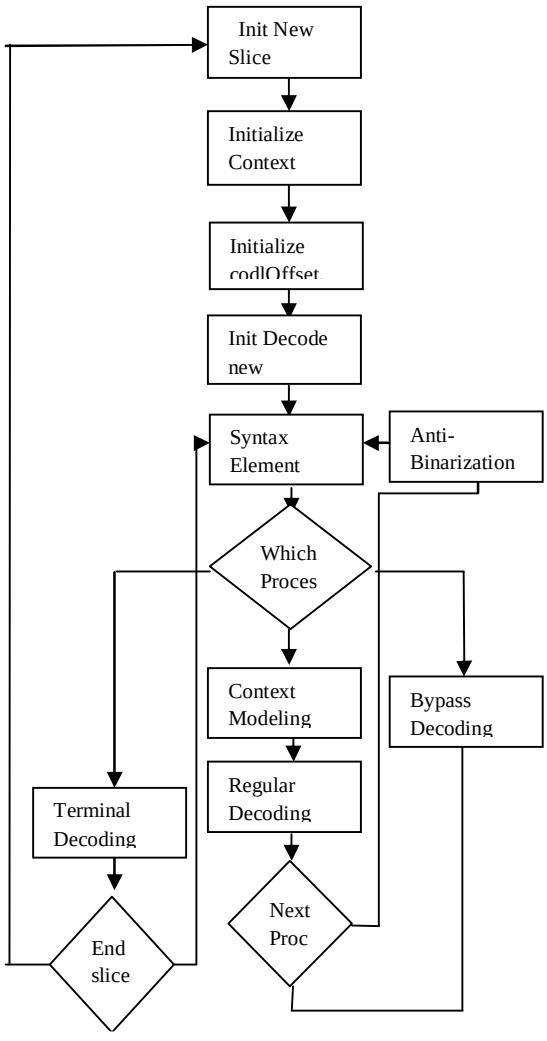

Figure:-2 CABAC Decoder Flow

B. Syantax element

The H.264/AVC standard defines this meaningful information as syntax element (SE). It also defines three variables: codlOffset, codlRange and MPS. The CABAC decoder determines the value of the output bit according to the values of codlOffset and codlRange. The H.264/AVC standard defines for the CABAC decoder five tables: context table, initial table, rangeTabLPS table, transIdxLPS table, and transIdx MPS table. The context table is constructed from the initial table and is indexed by the variable context. Each entry of the context table contains two variables: pStateIdx and MPS. The bigger

pStateIdx is the more probable the output bit equals to MPS. At the beginning of decoding a macroblock, it must determine which syntax element to decode. In this section, describe the syntax element decoding flow. Firstly divide the syntax element into control information SEs and coefficient information SEs. Figure-3 and Figure-4 show their decoding flows, respectively

The control information SE decoding flow is divided into three sub-flows based on the slice type. For an I slice, the decoder first decodes the Mb_type. If the Mb_type is intra_4x4 it decodes the intra 4x4_pre_mode and rem_intra_pre_mode otherwise decodes the Chroma_pre_mode directly.

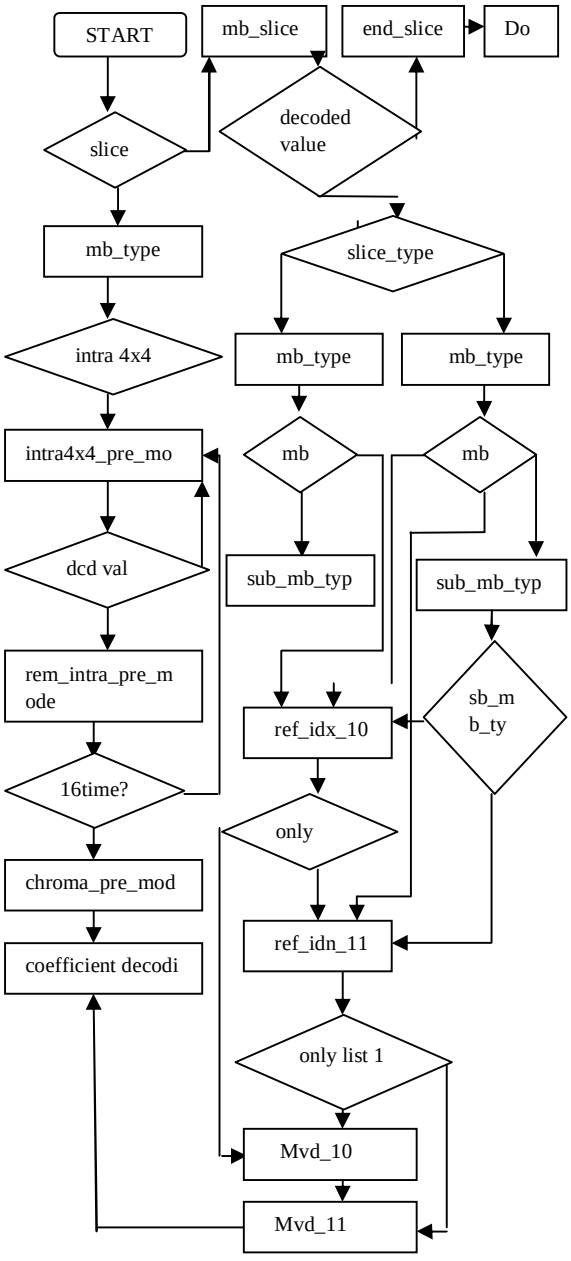

Figure-3: The decoding flow for control information SE

After decoding the Chroma_pre_mode performs the coefficient information SE decoding flow. When decoding a P or B slice, the decoder will first decode the Mb_skip_flag. If the decoding result is "1", it can skip the macroblock; otherwise, it goes on to decode the Mb_type and the Sub_mb_type. After decoding Sub_mb_type, the decoder knows about the partitioning type of the currently decoded macroblock. Then, it can start decoding Ref_Idx and Mvd. If the slice type is P, the decoder only needs to decode Ref_idx_l0 and Mvd_l0 otherwise, it must get the prediction mode of the B slice according to Mb_type and Sub_mb_type. If the prediction mode is bi-prediction, it must decode Ref_idx_l0, Ref_idx_l1, Mvd_l0 and Mvd_l1. The decoder only needs to decode Ref_idx_l0 and Mvd_l0 when the prediction mode is forward; otherwise, it decodes Ref_idx_l1 and Mvd_11 for backward prediction.

At the beginning of coefficient SE decoding, the decoder decode Coded_block_pattern,Mb_qp_delta and Coded_block_flag.If Coded_block_flag is zero, the decoder can skip this 4x4 block. After decoding Coded_block_flag, the decoder decodes Significant_coeff_flag and Last_significant_flag.

From the decoding results, it knows the number of coefficients in the current 4x4 block and then starts to decode Coeff_abs_level_minus1.

C. Input neighbour data

A pre-defined set of past symbols called context data. Length of neighbour data is similar to the length of original input data.

D. Context modelling

Before the regular decoding process, the CABAC decoder must calculate the context value ranging from 0 to 398. With the context value as index, it can get pStateIdx and MPS from context table. The pStateIdx range from 0 to 63.The context value is calculated as the following two equations.

Context value = ctxIdxOffset + ctxIdxInc .....(1)

Context value = ctxIdxOffset + ctxIdxInc+ ctxIdxInc.....(2)

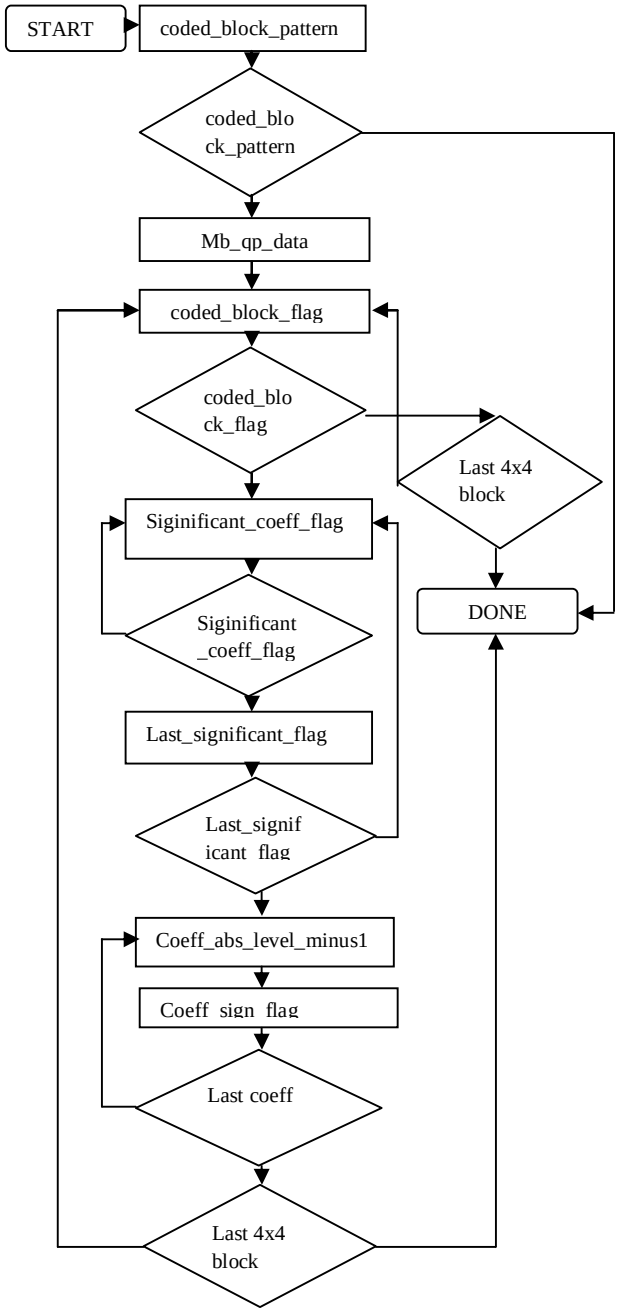

Figure-4: The decoding flow of coefficient SE

Eqn (2) is used to calculate the context value for the SE Coded_block_flag, the SE Significant_coeff_flag, the SE Last_significant_flag and the SE Coeff_abs_level_minus1, while Eqn (1) is used for the others.

The decoder can get the value of ctxIdxOffset according to the currently decoded syntax element and the value of ctxIdxInc by referring two neighbouring syntax element or previously decoded syntax element. When the decoder is calculating context value using Eqn (2), it gets the value of ctxCatOffset by looking up the ctxCatOffset table with ctxBlockCat as index. The value of ctxBlockCat is determined by macroblock type and the 4x4 coefficient transform block.

E. The decoding process

The CABAC decoder defines three decoding processes: regular decoding process, bypass decoding process and terminal decoding process. The terminal decoding process is only used to decode End_slice_flag. If the appearance possibility of MPS equals to the appearance possibility of LPS, the decoder will use the bypass decoding process otherwise, it will use the regular decoding process. Both the regular decoding process and the terminal decoding process consist of two sub-processes: arithmetic decoding and renormalization.

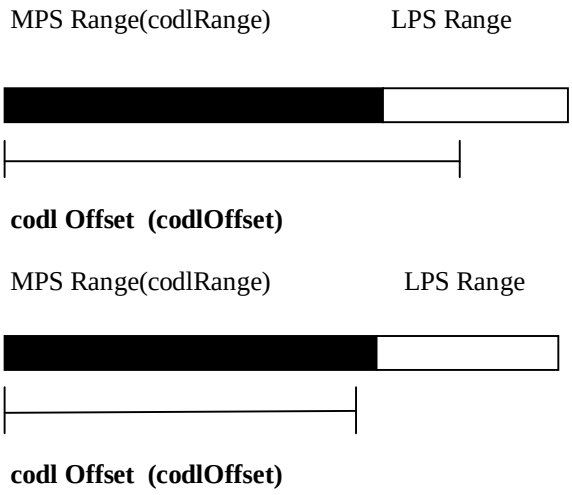

Figure-5: Principal of regular decoding process

The regular decoding process is a binary adaptive arithmetic coding. When code_offset (codlOffset) is larger than MPS_range (codlrange), it outputs LPS and updates code_offset by minus MPS_range, and P(MPS) (pstateIdx) by looking up the transIdxLPS table. When code_offset (codlOffset) is less than MPS_range (codlrange).

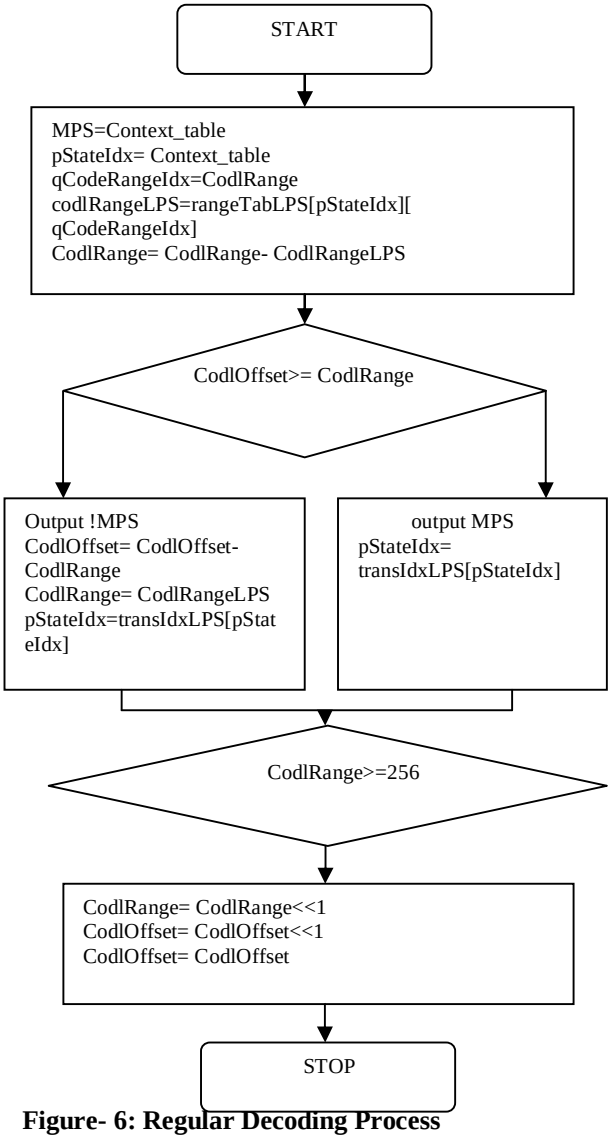

Figure- 6: Regular Decoding Process

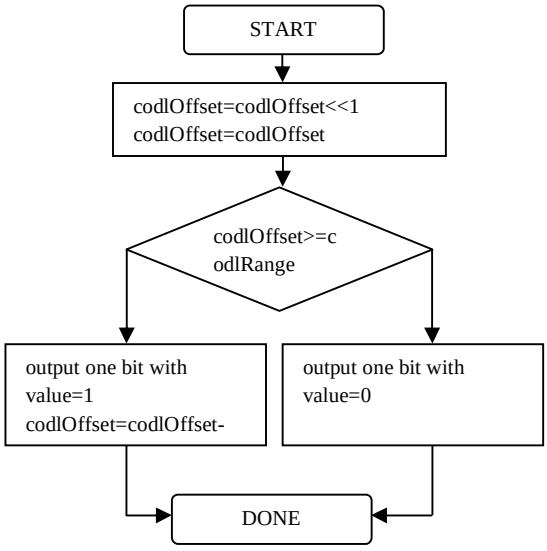

Figure-7: Bypass Decoding Process

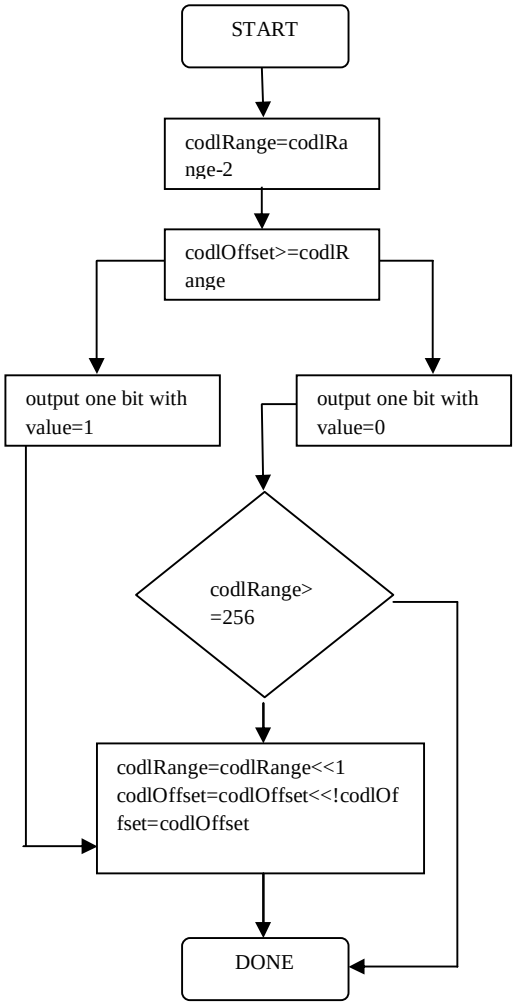

Figure-8: Terminal Decoding Process

F. Anti-binarization

There are five different coding techniques are unary code method, truncated unary code method, kth order Exp-Golomb code method, table mapping code method and fixed-length code method.

III. TESTING AND RESULTS

A. Processor

This diagram shows the graphical view of processor with the SDRAM memory . To generate this design the EDK & SDK tools are used to store an input in binary data in SDRAM, when processor request on PLB bus to access binary data after that it gets the repeat arbiter request form PLB bus and then it starts receiving binary data which is

stored in memory.

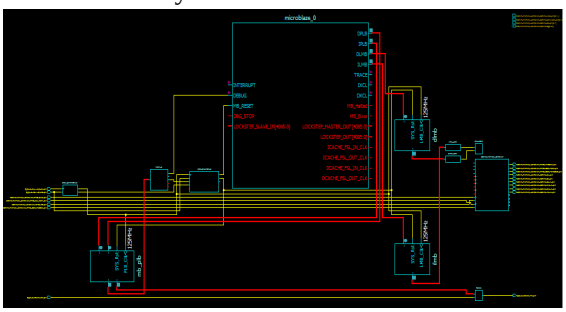

Figure-1: Generated processor design using XPS tool.

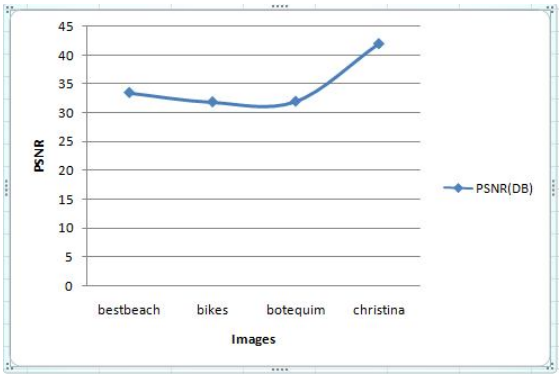

Figure-2: PSNR calculation of different images

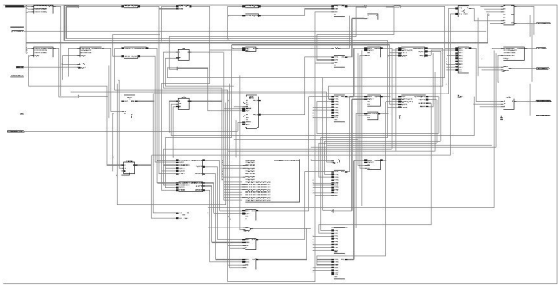

Figure-3: The synthesized architecture for the CABAC decoder

IV. CONCLUSION

The co-design of software and hardware is used to accelerate the decoding speed of design. Here XPS tool is used to store the input image in physical memory which is present on hardware. The design of embedded hardware which improves decoding performance it implies that the application will not be executed by a high performance computer, but rather on an embedded system which has to deal with limited area, speed and power.

V. REFERENCES

[1] Won-jinkim, koelcho and ki-seokchung " **Multi-Thread syntax element portioning forparallel entropy decoding**" Department of Electronic and Computer Engineering International conference, Hayang university, seoul, korea (2011).

[2] Martinus Johannes Pieter Berkho "**Analysis and Implementation of the H.264 CABAC entropy decoding engine**" ,Computer Engineering Mekelweg 4,Thesis Master of Science, The Netherlands (2010).

[3] Eeckhaut, H."**Optimizing the critical loop in the h.264/avc cabac decoder**" IEEE International Conference on FPT 2006, December, Department of electronic and information systems, Ghent University, Belgium (2006)

[4] Heiko Schwarz, and Thomas Wiegand "**Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard"**, Fraunhofer-Institute for Telecommunications –IEEE Transection on circuits and systems for video technology Heinrich-Hertz Institute (HHI), Berlin, Germany (2003).