Techniques for Architecture Design for Binary Arithmetic Decoder Engines Based on Bitstream Flow Analysis

Dieison Antonello Deprá, Sergio Bampi

Techniques for Architecture Design for Binary Arithmetic Decoder Engines Based on Bitstream Flow Analysis

Dieison Deprá, Sergio Bampi

PPGC – GME – Informatics Institute (II)
UFRGS – Federal University of Rio Grande do Sul
Porto Alegre – RS, Brazil
{dadepra, bampi.}@inf.ufrgs.br

Abstract. The design and implementation of a hardware accelerator dedicated to Binary Arithmetic Decoding Engine (BADE) is presented. This is the main module of the Context-Adaptive Binary Arithmetic Decoder (CABAD), as used in the H.264/AVC on-chip video decoders. We propose and implement a new approach for accelerating the decoding hardware of the significance map by providing the correct context for the regular hardware engine of the (CABAD). The design development was based on a large set of software experiments, which aimed at exploiting the characteristic behavior of the bitstream during decoding. The analysis gave new insights to propose a new hardware architecture to improve throughput of regular engines for significance map with low silicon area overhead. The proposed solution was described in VHDL and synthesized to standard cells in IBM 0.18 μm CMOS process. The results show that the developed architecture reaches 187 MHz with a non optimized physical synthesis.

Keywords: Hardware Dedicated Architectures for Decoding H.264/AVC Video Standard, Arithmetic Entropy Coding, CABAC, CABAD.

1 Introduction

The growing importance of high definition digital videos, mainly for real-time application, is calling for higher video compression efficiency to save storage space and transmission bandwidth [1]. The most advanced standard is the H.264/AVC, currently at the commercial state-of-the-art, defined by the ITUT/ISO/IEC [2]. This standard defines a great set of tools, which act in different domains of image representation to get higher compression ratios, roughly doubling the ratio obtained, by the MPEG-2 compressors [2]. The H.264/AVC introduces many innovations in the techniques used to explore the elimination of the redundancies found in digital video sequences.

The H.264/AVC standard specifies two alternative entropy methods: CAVLC (Context-Adaptive Variable Length Coding) and CABAC (Context-Adaptive Binary Arithmetic Coding) [1]. Both are based on the fact that the digital video sequences present non-stationary but predictable statistical behavior [3]. Moreover, this

statistical behavior is highly dependent on the type of content that is being processed and on the video capture technique [1, 2]. To address this issue, the H.264/AVC adopts an innovative approach that provides dynamic adaptive probabilities estimation, which is introduced in the CAVLC and CABAC [3] coding schemes.

The CABAC is the most important entropy encoding method defined by the H.264/AVC standard, allowing the H.264/AVC to reach 15% coding gain over CAVLC [2]. However, to obtain these coding gains a significant computational complexity is added in the coding hardware. Moreover, the coding algorithm is essentially sequential, as each step iteration produces only one bit and the next step depends on the values produced in the previous iterations [1]. The sequential nature of the CABAD leads to significant performance bottlenecks in the decoder. Many works found in the literature address these constraints trying to break data dependencies inherent to the nature of CABAC.

The goal of this work is to present a new hardware architecture to improve the throughput of the CABAD arithmetic engines. The architectural design aims to achieve a very efficient implementation, based on our experiments for a detailed bitstream flow analysis.

Next section presents an overview on context-adaptive binary arithmetic codec and the arithmetic engines are also detailed. Section 3 presents related works found in the literature. The bitstream flow analysis by simulation is discussed in Section 4. The architecture proposal is detailed in Section 5. The results of our architecture after synthesis are presented in Section 6. Section 7 the validation process applied in this case study are discussed. Finally, Section 8 addresses some conclusions and future work.

2 Context Adaptive Binary Arithmetic Codec Overview.

The context-adaptive binary arithmetic codec as defined by H.264/AVC standard is a framework for entropy encoding that transforms the value of a symbol in a word of code, with variable length near the theoretical limit of entropy [3]. It works with recursive interval divisions combined with context models that allow better coding efficiency [3]. Each subinterval represents a unique source symbol, and the size of the interval is proportional to that symbol probability of occurrence [3]. However, modeling occurrence probabilities of each symbol brings increasing computational complexity. One way to decrease this computational complexity is to use a binary alphabet [3].

The H.264/AVC standard, in its main and high profiles, supports the binary arithmetic coding/decoding, from the macroblock layer, to deal with information generated by the tools that act on the transform redundancies of the following kinds: spatial, temporal, and psycho-visual [2]. In the entropy methods of H.264/AVC standard the information that is arriving at the inputs of the encoding process or at the outgoing outputs in the decoding process are named Syntax Elements (SE) [3]. The SE is composed by the following information: i) type, used for codec control to determine the encoding process to be used; ii) and the value to be encoded based on the control information provided [2].

The decoding process is named CABAD and the encoding is referred to as CABAC. The encoding process receives at the input SEs with its type and value. Considering the SE type, a binarization method is applied to convert the SE value into a binary alphabet [2]. Then, the context model selects the appropriate context and sends it to the stage of arithmetic coding, responsible for generating the output bitstream and updates the context models. In Fig. 1 the integrated encoding and decoding dataflow is presented. Both encoding and decoding processes are composed by three steps that can be organized into four modules, which are described in the following subsections.

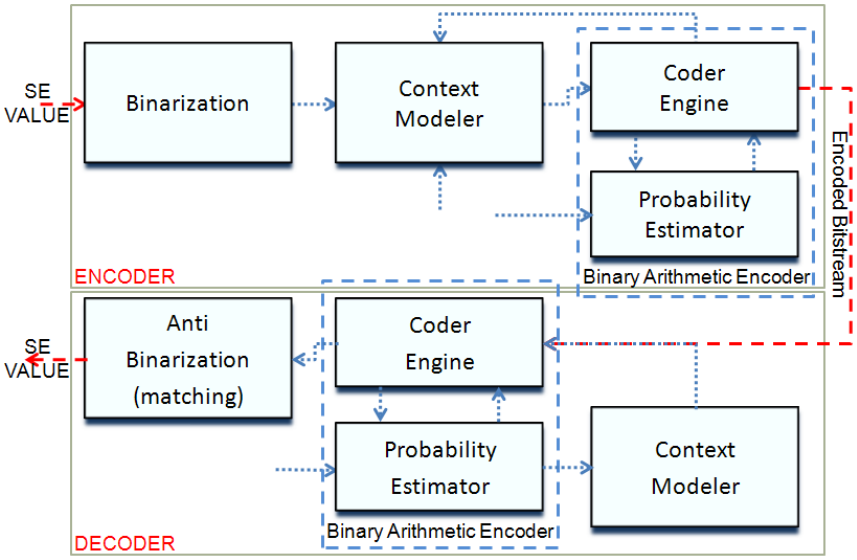


Fig. 1. CABAC and CABAD dataflow diagram with the three stages and four modules that compose the encoder and decoder.

As shown in Fig. 1 both encoding and decoding processes are composed by four modules can be organized into three stages that are described below:

Binarization/Anti-Binarization: The binarization process consists of mapping SE values for a unique sequence of bits that represents the original value. This mapping is done to reduce the symbols in the encoding alphabet, thus simplifying the amount of elements to be modeled and minimizing the costs of the context modeling and facilitating the task of arithmetical coding. Each bit, generated through this process, is denoted as "bin" and the set of all "bins" (bits) is named "binstring". From a total of seven binarization methods, four are fundamental: unary; truncated unary; fixed length; and exponential Golomb [2, 3].

Probability modeling: A context is a probabilistic model that represents a statistical distribution of a particular symbol on the basis of the review of the symbols previously processed and the probability of occurrence of the

current symbol. To adequately model all probabilities of occurrence of each symbol, CABAC defines 460 different contexts. Each bin of an SE can be associated with one or more contexts. During the encoding the probabilistic estimates must be kept updated to ensure the accuracy of the process. Each context model is composed by a pair of values, a 6-bit state value for the probability index (63 possible probability states), and a binary value for the most probable symbol "MPS". The state value is used as an index to the estimated probability value of the least probable symbol "LPS"[2, 3].

Binary Arithmetic Coder (BAC), or Decoder (BADE): It works based on the principle of recursive division of the interval of width R [3]. From the estimation of probability for LPS (pLPS) on a given range, two subintervals are obtained. The first is given by: rLPS = R * pLPS which is associated with LPS while the second (which is related to MPS) is given by: rMPS = R - rLPS. According to the encoded bin the rMPS or rLPS is chosen as new interval R. To simplify the computational complexity the value of R is quantized to 2 bits and the multiplication for rLPS values are pre-stored in a 64x4 fixed 2-D table indexed by the 6-bit state coming from context model and by the 2-bit quantized value of R. During binary arithmetic coding process two registers (range "R" and offset "O") are needed to keep the interval updates. The first one saves the current interval range while the second marks the lower bound within this interval (offset) [2, 3].

2.1 CABAD Algorithm Overview

The CABAD process involves a set of actions that occur below the slice layer. Fig. 2 shows the flow chart for these actions. For each new slice a new CABAD iteration happens. At the beginning of a slice a new context table is built from probability algorithm based on initial tables that depend on the slice type and of an index value (three possibilities) sent by the encoder. After that, CABAD initializes the variable CodlOffset getting the first nine bits reading from the encoded bitstream and the variable CodlRange si set to default value [2].

The CABAD decoding of macroblock layer of SE values are performed until an "End One Slice" (EOS) SE type is found. The first step in the SE decoding is the decision of its type and, based on this information it chooses an anti-binarization method [2]. After that, if the SE type is an EOS, then terminal decoding process is selected. Otherwise, for each bin of SE one of two other decoder processes, regular or bypass, must be chosen. For bins being decoded by regular process a context table address calculation must be done. The information retrieved from context table includes the MPS and its probability estimate index denoted by pState variable. The CABAD uses an offset fixed for each SE type combined with an increment defined by different possible forms, according to the SE type in conformance with [2] to generate context table addresses. For some SEs, obtaining increment index involves referring to SE from the left, top and current macroblock and, for others, the bin index is used for this purpose.

For bins that use the regular decoding process the CABAD obtains new rLPS from a look-up in a fixed pre-stored table indexed by pState and then one of four possible

values is selected by value of quantized CodlRange (CodlRange>>6) [2]. Then, new value of CodlRange is calculated and the comparison between CodlRange and CodlOffset define if MPS or LPS happens. After that, the context table must be updated with new values for MPS and pState which are obtained from a fixed table with state transition with different values for MPS or LPS occurrence. Next, the CodlRange and CodlOffset registers are available for the normalization process. In this case one or more bits of bitstream can be consumed [2]. Finally, one step of regular decoding process is finished; the contexts model and decoding environment register are updated. For other bins the bypass decoding process is applied. The bypass mode is simpler than the regular mode. Then, anti-binarization module is performed and the results of this operation determine if the binstring produced by the decoding environment matches with the method expected or not.

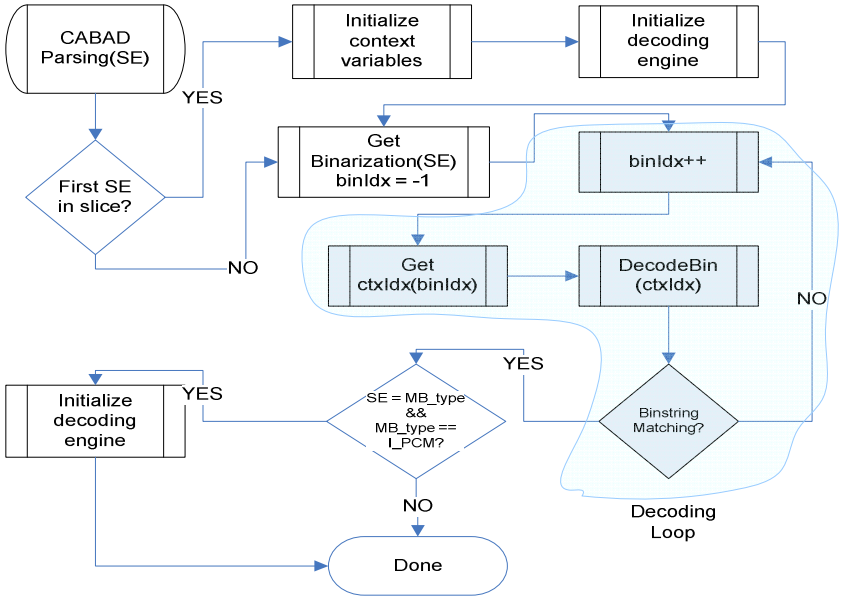


Fig. 2. CABAD algorithm flow diagram shows the action sequence released by CABAC Decoder to process each one SE inside of the slice layer.

3 Related Work

Techniques to reduce the latency and data dependency of CABAD have been widely discussed in the literature and they follow five basic approaches: pipeline; contexts pre-fetching and cache; elimination of renormalization loop; parallel decoding engines; and memory organization. The pipeline strategy is used in [4] to increase the bins/cycle rate. An alternative to solve the latency of renormalization process is presented in [5]. The speculative processing through the use of engines decoding

parallel is explored first in [6], then in [7] and [8]. High efficiency in the decoding process using pre-fetching and cache contexts is discussed in [6] and [9], respectively. Memory optimization and reorganization are addressed in [4].

The work of [8] presents optimizations in the arithmetic engine through the parallel execution in speculative mode and the adoption of leading zero anticipation that allows counting of consecutives zeros in CodlRange. These two approaches bring reductions in the delay in the critical path.

The hardware architecture proposed in [6] is based on analysis of the relationship between bins count for each SE type and the occurrence of each SE type in one macroblock. The usage rate imposed by each SE in each of the three decoder engines is a relevant aspect that is used to optimize the overall decoding process.

An evaluation of the data dependencies in the regular mode of decoder arithmetic engine is presented in [7]. In this study, the frequency of changes to CodlRange and CodlOffset registers are considered for cases where renormalization process happens combined with the observation of MPS or LPS decision.

Considering the various architectures proposed by different authors for the CABAD, a static characteristics analysis of constraints for decoding bitstream process was considered and some experiments were conducted by software simulations to extract the dynamic behavior of the decoder flow. This analysis is addressed in Section 4.

4 Bitstream Flow Analysis for Decoder Process

All our analyses were based on results obtained from statistical data collected by software routines that we introduced into the decoder module of the reference software (JM), version 10.2 [10]. To reach more representative data set in our analysis we decided to work with four different digital video resolutions, in YUV video format 4:2:0, more often used in the reports found in the literature: QCIF, CIF, D1 and HD1080p. Moreover, we evaluated the impact of the quantization parameters on the bitstream behavior. For our statistics we selected all 18 QCIF, 17 CIF, 18 D1 video sequences available in [11], and also seven additional HD1080p video sequences. The last are designated as: rush-hour; riverbed; blue-sky; tractor; sunflower; station2; pedestrian area. In total, there were 60 digital video sequences in this analysis, each with 200 frames. Fig. 3 show one frame of the HD1080p video sequences used in this case study in additional to QCIF, CIF and D1 listed in Table 1.

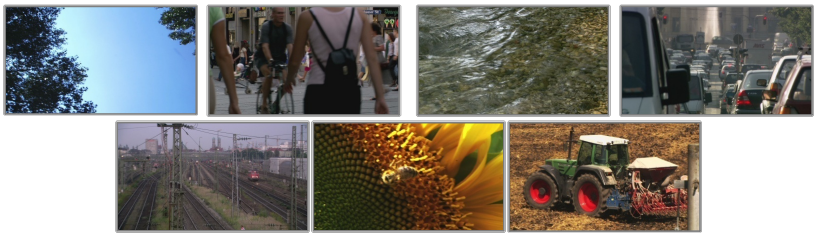


Fig. 3. Samples of HD1080 video sequence, that named: rush-hour; riverbed; blue-sky; tractor; sunflower; station2; pedestrian area, respectively.

Table 1. All video sequence used in this case study.

Video Sequences			
QCIF (176×144)	**CIF (352×288)**	**D1 (720×480)**	**HD1080 (1920×1080)**
Akiyo	Bridge-close	Abstract	Bluesky
Bridge-close	Bridge-far	Artant	Pedestrian
Bridge-far	Bus	Chips	Riverbed
Carphone	Coastguard	Concert	Rush-hour
Claire	Container	F1	Station2
Coastguard	Flower	Football	Sunflower
Container	Foreman	Ice	Tractor
Foreman	Hall	Leaves	
Grand-mother	High Way	Letters	
Hall	Mobile	Mobile	
Highway	Mother-daughter	Parkrun	
Miss-america	News	Rafting	
Mobile	Paris	Rugby	
Mother-daughter	Silent	Seawall	
News	Stefan	Suzie	
Salesman	Tempete	Tempete	
Silent	Waterfall	Toweres	
Suzie		Waterfall	

The encoding parameters employed for coding all sequences were: Profile IDC = 77, Level IDC = 40, SymbolMode = CABAC, GOP=IPBB and RDO=ON. Our experimental procedure was to perform, for all the video sequences, six different encoding processes, varying the parameters of quantization QPISlice and QPPSlice in pairs; namely the pairs were: 0:0, 6:0, 12:6, 18:12, 24:18, and 36:26, resulting in a total of 420 digital video sequences encoded. The decoding process was done for all encoded sequences using the JM v.10.2 decoder, to collect the statistics data and to obtain feedback for the validation process. The relations and statistical behavior were studied and synthesized, and they will be presented next.

One of the problems of CABAD is to determine the actual throughput needed for the decoding process to occur in real-time. This happens because the length of codeword generated by CABAD is variable and may change significantly between

iterations, since the coding method is context-adaptive. Furthermore, for some SE types it can be difficult to determine the binstring length and the SE sequence, as they vary according to the slice type and the macroblock type. However, H.264/AVC standard in its level 4.0 defines the upper-limit bit/rate at 20Mbps. We analyzed the bit count at the input and output of the CABAD, before quantization, and the ratio between them varies between 1.3 and 2.1 times. Then, we can consider that, in the worst case, the architecture has to process at nearly 42Mbps, to reach throughput enough for real-time decoding at 30 frames per second in the 1080 x 1920 format (1080p).

The Binary Arithmetic Decoder Engines (BADE) are the CABAD kernels. They are responsible for regenerating the binstring since of the bitstream and internal variables. Each bin is produced by one of three BADE kinds. Considering that the decoding process is done bin by bin, it requires high performance because inside this module resides the CABAD critical path. The BADE basic organization is shown in Fig. 4.

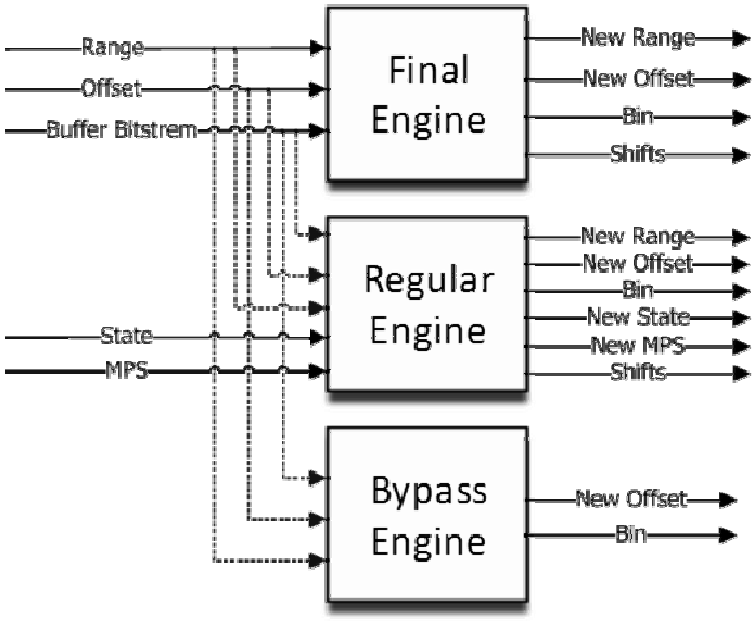


Fig. 4. The three kinds of Binary Arithmetic Engines present into CABAC core, its organization and they connection with the internal registers.

The H.264/AVC standard defines which type of BADE engine each SE must use. Moreover, part of binstring of one SE type can be produced by one BADE type while another part can be produced by other one. The regular engine is the most complex BADE block and is used on most SEs. The bypass engine is only used by the suffix part of motion vector differential (MVD) and transforms coefficient (COEF). Additionally, the signal bits of COEF have to be treated in the bypass engines. During

software profiling, the BADE engine utilization by each SE type was observed to determine the better strategy for the dedicated architecture design.

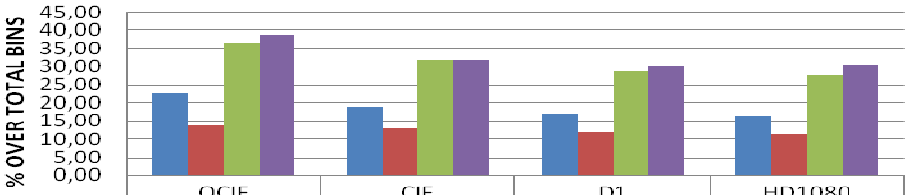
By analyzing the bins count occurrence in the bitstream we observed that just four SE types (coded_block_flag, coeff_level, sig_coeff_flag and last_sig_flag) account for more than 93% of all bins, in average, for all types of macroblocks. Thus, a deeper analysis of the behavior of these SE types was performed to improve the gain in the BADE. In the first study we investigated the bins distribution for different SE types in each one macroblock types. In the Table 2 the results obtained are summarized.

Table 2. Distribution of bins by different SE types for each macroblock type.

	Information	Code Block Flag (%)	Sig & Last Flags (%)	Coefficient Levels (%)	Other SEs (%)
I MB	Total occurs	2.32	60.38	34.30	3.00
	Bins generated	0.67	17.62	80.00	2.31
	Regular Process	0.86	22.31	74.68	2.15
	Bypass Process	0	0	100	0
P MB	Total occurs	2.51	58.50	37.07	1.92
	Bins generated	1.47	34.21	62.05	2.27
	Regular Process	1.76	40.95	54.90	2.39
	Bypass Process	0	0	98.33	1.67
B MB	Total occurs	2.61	57.58	38.56	1.25
	Bins generated	1.80	39.75	46.29	12.16
	Regular Process	2.17	47.98	37.46	12.39
	Bypass Process	0	0	98.97	1.03
Average	Total occurs	2.55	58.07	37.65	1.73
	Bins generated	1.54	35.14	61.10	2.22
	Regular Process	1.86	42.41	53.30	2.43
	Bypass Process	0	0	98.78	1.22

The results show that there are, on the average, seven significant coefficient flag (SE_SIG) and five least significant coefficient flag (SE_LAS) for each 4x4 residual blocks. The utilization of arithmetic engines shows that regular engines produce 80.8% of bins count while the bypass produced 19.2% of them. Another interesting fact is that many bins produced by regular and bypass is generated in a consecutive way, 84.92% and 29.35%, respectively.

The occurrence of bins related to the SEs of the significance map (SE_SIG and SE_LAS) also deserve emphasis, since together they represent between 27% and 36% of all bins processed by the CABAD. Moreover, they have special interest for decoding engines since they usually occur in sequence, i.e. each SE_SIG is followed by a SE_LAS. However, this does not occur when the value of SE SE_SIG is zero, in this case the next SE decoded should be another SE SE_LAS. Fig. 5 illustrates the relationship between bins occurrence of the significance map for each of the resolutions discussed, highlighting the percentage difference occurrences between SE_SIG and SE_LAS.



	QCIF	CIF	D1	HD1080
SE_SIG	22,57	18,91	16,89	16,31
SE_LAS	13,82	12,90	11,82	11,35
Total	36,39	31,81	28,71	27,66
Diff (%)	38,78	31,80	29,99	30,44

Fig. 5. Bin occurrences for Significance Map SE in each resolution used.

5 Design Architecture

The proposed architecture development was based on observations made on the behavior of the bitstream for many coding scenarios and on few previous works found in the literature. The bitstream flow analyses have shown that for some specific situations using an approach with specialized processing can provide throughput gains in the decoding process. As presented in section 2, 3 and 4, the exploration of the parallel speculative execution of BADE engines is a good alternative to reach greater throughput without excessive area increase.

Our design is based on the work presented in [12] which applies multiple parallel engines for speculative execution. In this work we include few extensions mainly in the regular branch. The new arrangements in the regular engine interconnections aim at exploring characteristics behavior of the SE_SIG and for the SE_LAS kinds of the syntax element to reach high throughput in the significance map decoding.

From the work presented by Yu and He in [6] a significant part of the new proposed architecture for CABAD makes use of two regular engines for decoding a variable number of bins per cycle. Depending on each implementation, the context modeling can provide one or two context models for regular engines branch, thus varying the efficiency of the decoding. But, for special situations this approach may not improve efficiency because according to the decision of the first regular engine the second bin for each one of these engines needs to use a different context.

The H.264/AVC standard defines that each 4x4 coefficient block should refer to one significance map [2]. This map set is composed by two types of SE (SE_SIG and SE_LAS) which should occur in a specific order. The significance map is generated according to the process order and the coefficients value. The process for generation of the significance map for a 4x4 coefficient block example is shown in Fig. 6. The Index line shows the index or the values in zig-zag scan order while the Value line shows each one coefficients value for a 4x4 example block. The lines with Flag SIG and Flag LAST show the composition of the significance map for the 4x4 example block.

	Coefficients in Zig-Zag Order															
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	24	2	0	-1	1	1	0	0	0	3	0	0	0	0	0	0
Flag SIG.	1	1	0	1	1	1	0	0	0	1						
Flag LAST	0	0		0	0	0				1						

Fig. 6. Significance Map generation for a 4x4 coefficients block.

As shown in Fig. 6 for each SE_SIG with value equal to one there is one SE_LAS, but when the SE_SIG is equal to zero then the SE_LAS element does not occur. Based on the results analysis, summarized in the Fig. 5, it is possible to identify that this mismatching between SE_SIG and SE_LAS pair happens, in average, for roughly 30% of the cases in the HD1080 video sequences that we tested. This fact opens the opportunity to explore decoding optimizations, specifically as to when one specialized process to supply the correct context for BADE can be used.

The proposed architecture employs multiple engines instances for the case of variable number of bins per cycle, and also adopts specialized mechanism for context selection. Our design basic structure is shown in Fig. 7.

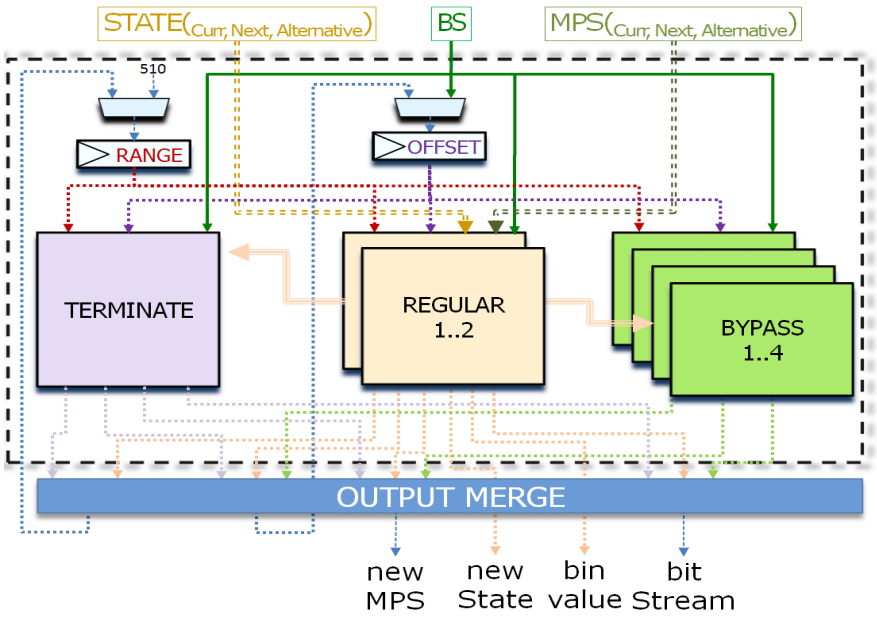


Fig. 7. BADE core with arrangement.

As Fig. 7 shows, the three kinds of engines present in CABAD are organized in one hierarchical arrangement, namely: one terminate engine, two regular engines and

four bypass engines. The BADE block (in Fig. 7) receives three context pairs (STATE and MPS) and the bitstream buffer (BS). According to the SE kind, one of its engines is used. The Regular and Bypass engine instances are organized into two distinct branches. Inside the Regular branch two bins can be produced in one cycle while in the Bypass branch at most four bins can be produced in a single cycle. The Regular engine is more complex than the other engines, and contains the critical path of this module. We used optimizations to reduce the delay of this block.

Initially an operations reordering is made by the regular engine, as the approach presented in [7]. This results in two parallel paths inside the regular engine, one to treat the occurrence of the MPS and another for the LPS. Another important aspect is the access to static memories to retrieve information about the next state (access the MPS_TABLE, and the LPS_TABLE) and the rLPS estimate probabilities (RLPS_TABLE). These memories are addressed by pState, which is provided from the context model stored in the context memory. The fact that these memories are inside the regular engine affects the critical path. Furthermore, when we concatenate two regular engines, two accesses to these memories in the same cycle is required. To solve this problem we apply an approach similar to that adopted in [13], in which the memories are concatenated and combine the information about current, the next MPS states, the next LPS states and rLPS estimate probabilities. Thus, we can obtain all information needed to decode two bins that reference the same context with just one access to the static memory.

Finally, we applied the first one detect (FOD) strategy to solve the renormalization problem in an approach similar to [5]. The special approach used to resolve the renormalization allows it to save between 2 to 8 cycles, because the loop is eliminated and the renormalization always happens in only one cycle. To reduce the FOD delay, the FOD is broken in two segments, one for the low interval part and another for the high interval part, as illustrated by Fig. 8. Then, adding just one multiplexer we can select the renormalization part between the low and high ranges. To finish, the Range first bit is used to choose between new and old register values for the renormalization process.

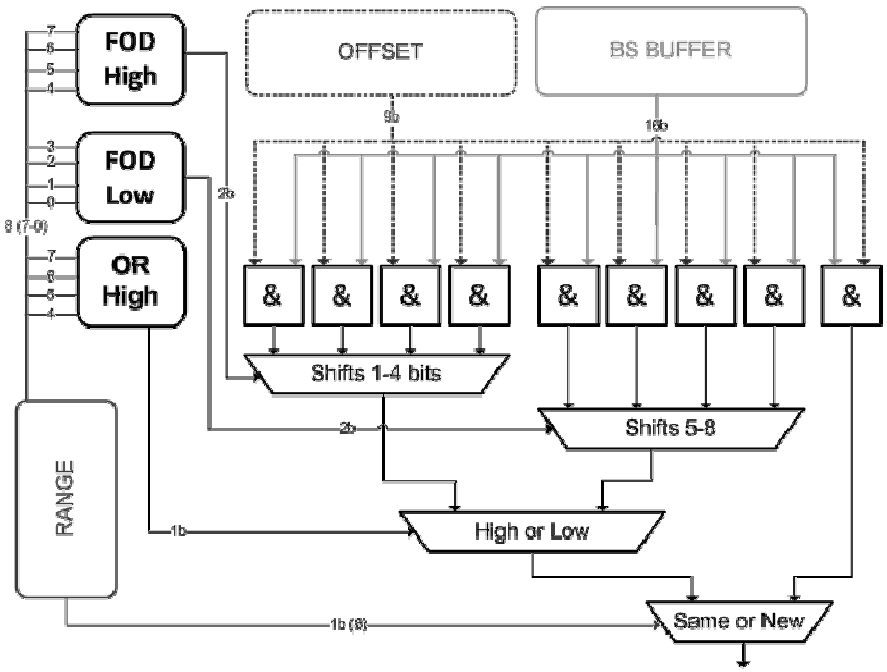


Fig. 8. Renormalization block with FOD accelarates.

Considering that the regular engine is responsible for most of the bins produced by the CABAD, it seems a good alternative to increase the parallelism level in this engine, instantiating additional regular engines. Meanwhile, the regular engine is in the BADE critical path due to its long combinational logic depth (that includes adders, comparators, ROM and renormalization). Thus, it is not advantageous to use a larger number of regular engines concatenated because this would cause performance degradation for all other CABAD stages and the throughput may not be satisfactory. Moreover, our analysis has shown that the Regular engines are underutilized because the context modeling cannot be efficient for all situations, especially for significance map decoding.

A new interconnection approach, namely of SP_SIGMAP, for regular engines was developed to improve throughput in the regular branch. The BADE block can receive three context pairs from context modeling. These context pairs can be used in significance map decoding to explore the characteristic behavior of these SEs. The first regular engine receives one context to decode one SE_SIG, while the second regular engine receives two contexts, being one to decode one SE_LAS and other to decode the next SE_SIG. If the first regular engine result was MPS then the second regular engine receives the second context else the third context will be delivered to the second regular engine. The interconnections for regular branch engines are shown in Fig. 9.

The next section analyzes the results obtained by our architectures when processing digital video test sequences, the same utilized in the simulation analysis discussed in section 4.

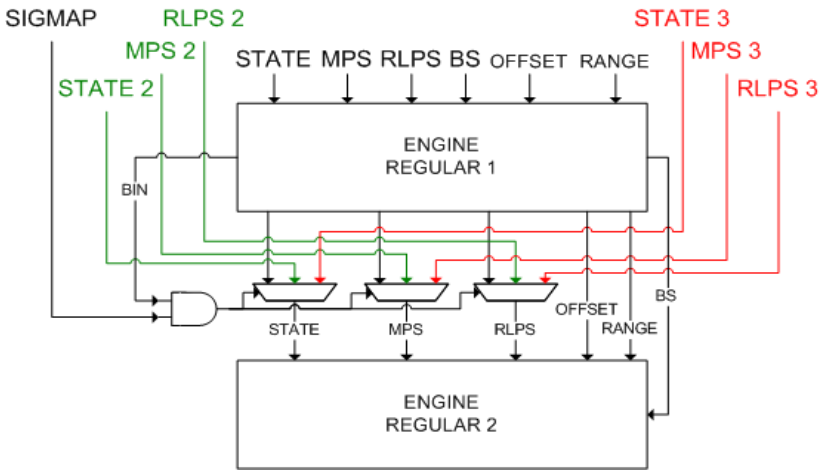


Fig. 9. Regular engines interconections inside the Regular Branch.

6 Experimental Results

The developed architectures were described in VHDL and synthesized to 0.18 um CMOS standard cells based on the IBM cell library using the Cadence RTL compiler. The Modelsim tool, version 6.01a, was used during the simulation and architectural validation process. The architecture development presents a new arrangement for binary arithmetic decoders of CABAD that is able to generate up to 4 bins per cycle, in the best case. The utilization of four decoding bypass engines inside the BADE increases the hardware resources required, while providing more efficiency compared to the BADE architecture with just two decoder bypass engines. Table 3 shows the hardware synthesis results for the architecture proposed. It compares the solutions with two and four decoder bypass engines in the architecture and our design.

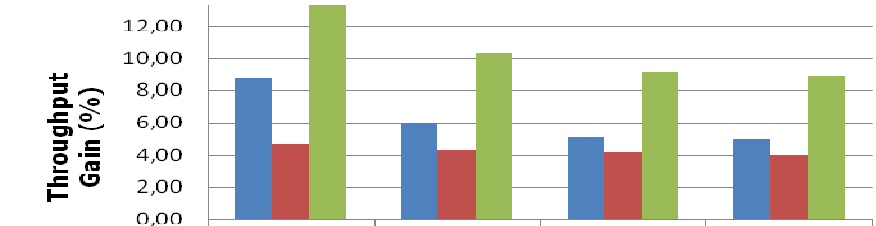
Table 3. Distribution of bins by different SE types for each macroblock type.

Information	**Architectures for multi bin BADE engines**			
	[6][1]	**[12]**	**Our Proposal**	**Differences (percent)**
Gates	3671	3928	4022	94 (**+2.4%**)
Max. Frequency	191.86	190.25	187	-3,25 (**-1.7%**)
Max. Bins/Cycle	3(2R1B)	4(4B)	4(4B)	0

[1] Our implementation of the author´s proposal.

The results in Table 3 indicate that the increase in the hardware costs is around 2.4% for our design and the maximum frequency decreases 1.7%, both when compared to the design proposed in [12]. A large number of test-benches were

developed and run to evaluate the performance of our architecture with data extracted from the reference software during the decoding process of the 440 digital video sequences listed in Section 4. For these video test-benches we observed that the two main approaches adopted can improve the throughput when compared to previous works presented in [6] and [12]. So, the potential gain for the four Bypass engines (4 BYPASS) and for the specialized context selection for significance map in regular engines (SP SIGMAP) were analyzed for each different resolution of the video sequences tested. The results of these analyses are shown in Fig. 10.



	QCIF	CIF	D1	HD1080
SP SIGMAP	8,75	6,01	5,06	4,97
4 BYPASS	4,68	4,34	4,13	3,99
Both Combined	13,43	10,35	9,19	8,96

Fig. 10. Performance analysis to four classic resolutions.

The data presented in Fig. 10 shows that the SP SIGMAP approach can improve the throughput from 4.97% for HD1080 up to 8.75% for QCIF video sequence resolutions. Furthermore, the 4 BYPASS approach offers additional gain from 3.99% up to 4.68% for HD1080 and QCIF, respectively. The proposed design adopts both approaches, and when compared to [6] which does not use neither of these techniques, it reaches 8.96% to 13.43% throughput gains. When compared to [12] the proposed design shows the throughput gains indicated in the SP SIGMAP line of Fig. 10.

The strategy to evaluate the performance of our hardware was also employed to validate our design: in these simulations we compared the outputs generated by our architecture to the results generated by the JM10.2 decoding module [10]. To this end, we introduced extra code (routines) in this software to save the inputs and the outputs of the BADE engines for later comparison with the hardware simulations. This strategy was used for extensive architecture validation.

Validation Process

In the development cycle of integrated circuits, the validation process can reach 70% of the design time. This information indicates the challenge of this process. The approach used in this work to minimize this time was to make a hierarchical and incremental validation. In this approach, several validation steps were made according to the complexity and the abstraction level of the developed blocks.

In the first step, the blocks of lower abstraction level were validated as standalone block. This was accomplished by generating the intermediate data from the specifications given in the H.264/AVC standard. These stimuli were used in each of the blocks and the verification was done by comparing the waveforms in the simulator to the functional definition of that block.

In the second step, the blocks were grouped according to their function and the validation was done for the entire group. In this step a software implementation of the norm was used to produce the input stimuli and the expected output. The software model used to generate the stimuli and the expected results was based on the reference software of the H.264/AVC (Surking, 2009). Modifications were done in this software to get the right data for the hardware validation. Fig. 11 illustrates the data extraction process for validation.

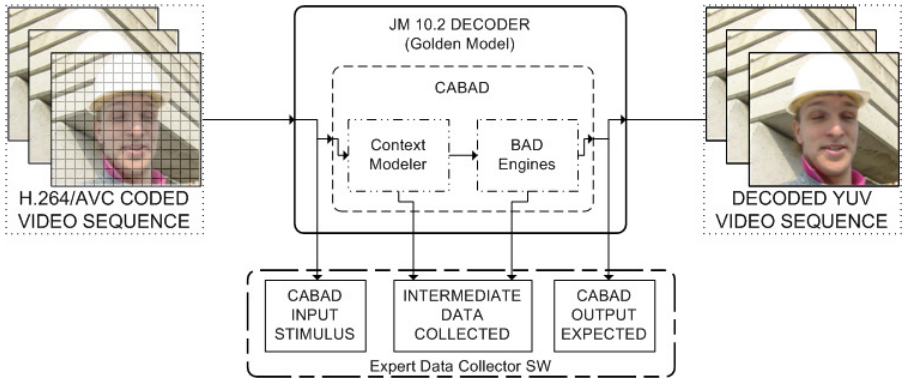


Fig. 11. Data extraction process for functional validation of the individual blocks and the complete architecture.

The data extraction process for the production of input stimuli and the results for comparison were done using the same standard video sequences of the section 4. Actually these stimuli were produced at the same time the data for static and dynamic analysis were produced. This approach allowed us to significantly reduce the time spent, once we had to process all the video sequences only once. It also made the data used for analysis and validation consistent with each other.

The second step followed the flow showed in Fig. 12. Inside a test-bench file, the input stimuli were injected into the validating block (Design Under Test - DUT). The outputs of the DUT were stored for later comparison to the expected outputs.

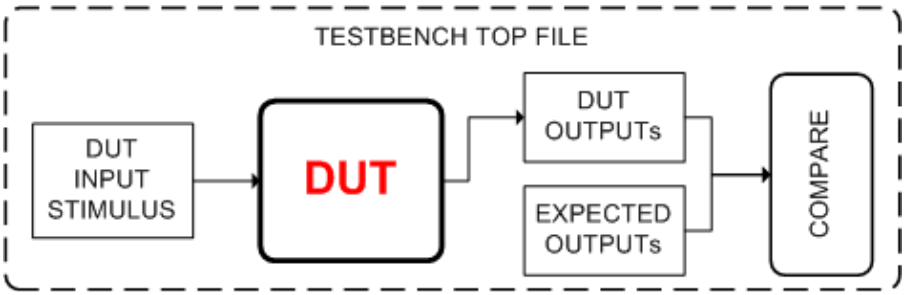


Fig. 12. Processo de extração de dados para a validação funcional dos blocos individuais e da arquitetura completa.

8 Conclusions and Future Work

This work presented a novel dedicated hardware architecture for the BADE of the CABAD block that supports the decoding of up to four bins per cycle. The architectural decisions were supported by a detailed analysis of the bitstream flow generated by a software video decoder. The results show that, with a hardware cost increase of just 2.4%, we obtain 5% efficiency gain in the utilization rate of the BADE module. The analysis of the bitstream flow shows that it is possible to explore the dynamic behavior of CABAD algorithms to develop novel hardware solutions.

The next step in this development will be to integrate this BADE module inside the CABAD top-level hardware architecture and to evaluate performance and throughput of the entire H.264/AVC decoding hardware with the same digital video sequence inputs. Given that in our simulation experiments we used a limited length for the search area for the motion vector calculations, one needs to analyze the behavior of the bitstream flow when the search area for motion estimation is increased.

Acknowledgment

The authors gratefully acknowledge the Brazilian R&D agencies, CNPq and CAPES, for financial support.

References

1. Wiegand, Thomas; Sullivan, G. Bjøntegaard, G., Luthra, A.: "Overview of the H.264/AVC Video Coding Standard". In: IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, pp. 560-576, Nº 7, July, (2003)
2. "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14 496-10 AVC". In: Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-TSG16/Q.6, March (2003)

3. Marpe, D., Schwarz, H., Wiegand, T.: "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard". In: IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, Nº 7, July (2003)
4. Yang, Y-C., Lin, C-C., Chang, H-C., Su, C-L., Guo, J-I.: "A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding With Look Ahead Parsing". In: (ICME) Multimedia and Expo, 2006 IEEE International Conference on, pp. 357-360, July (2006)
5. Eeckhaut, H., Christiaens, M., Stroobandt. D., Noolet, V.: "Optimizing the critical loop in the H.264/AVC CABAC decoder". In: Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on. December (2006)
6. Yu, W., He, Y.: "A High Performance CABAC Decoding Architecture". In: IEEE Transactions on Consumer Electronics, Vol. 51, pp. 1352-1359, No. 4, November (2005)
7. Kim, C-H., Park, I-C.: "High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction". In: Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on. May (2006)
8. Bingbo, L., Ding, Z., Jian, F., Lianghao, W., Ming, Z.: "A high-performance VLSI architecture for CABAC decoding in H.264/AVC". In: ASICON '07. 7th International Conference on, pp. 790-793, October (2007)
9. Zhang, P., Gao, W., Xie, D., Wu, D.: "High-Performance CABAC Engine for H.264/AVC High Definition Real-Time Decoding". In: Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers. International Conference on. pp. 1-2. Las Vegas, NV, USA. January (2007)
10. Suhring, K.: "H.264/AVC Reference Software". In: Fraunhofer Heinrich-Hertz-Institute. Available in: http://iphome.hhi.de/suehring/tml/download/. [Accessed: March 2008]
11. Reisslein, M.: "YUV Video Sequences". Video Traces Research Group. Available in: http://trace.eas.asu.edu/yuv/index.html. Accessed: March (2008)
12. Depra, D. A., Rosa, V. S., Bampi, S.: "A novel hardware architecture design for binary arithmetic decoder engines based on bitstream flow analysis". SBCCI 2008:.. In: Proceedings of the 21st Annual Symposium on Integrated Circuits and Systems Design. SBCCI '08. International Symposium on. pp. 239-244. September (2008)
13. Mei-Hua, X., Yu-Lan, C., Feng, R., Zhang-Jin, C.: "Optimizing Design and FPGA Implementation for CABAC Decoder". In: High Density packaging and Microsystem Integration, 2007. HDP '07. International Symposium on. pp. 1-5. June (2007).