

System-on-Chip Design Methodology for a Statistical Coder

Thinh M. Le, X.H. Tian, B.L. Ho, J. Nankoo, Y. Lian
Department of Electrical and Computer Engineering
National University of Singapore
Email: elelmt@nus.edu.sg

Abstract—In this paper, we propose a system-on-chip software hardware co-design methodology for a statistical coder. We use the Context Adaptive Binary Arithmetic Coder (CABAC) used in the Main profile of the H.264/AVC video coding standard as a design example. The design methodology first involves performance and complexity analyses of the existing CABAC reference software, and thus the top-level CABAC software hardware architecture can be conceptualized. The design is aimed to strike a balance between software modules and hardware modules based on design constraints. Verification is performed by comparing the compressed bit stream generated by the reference CABAC SW (without any HW assisted circuitries), with that output by the top-level CABAC architecture (with HW assisted circuitries). Standard video test sequences have been used for verification purpose. The CABAC architecture is then put within the system-on-chip frame work where system bus and its signals, input/output FIFO buffers, debug structures, reset circuit, etc. are designed into. Compared to existing statistical coders, this design is aimed for significant coding time saving by balancing timing between software modules and hardware modules, is well verified with standard video test sequences, and is reusable as an IP in a SoC environment.

Index Term—CABAC, design methodology, statistical coder, SoC.

I. INTRODUCTION

There have been reports on the entropy coders and their hardware implementations in the literature. The terms entropy and statistic can be used interchangeably. In the video compression research area, many have reported entropy coder/decoder designs for video codec. Di *et al.* [1] proposed a pipelined architecture for *Context-based Adaptive Variable Length Coding* (CAVLC) decoder running at a maximum operating frequency of 125MHz. It is not mentioned if the decoder was tested with standard test sequences. Osorio *et al.* [2] proposed a CABAC compression system - a portion of which performs both the context index calculation and binarization steps of less frequently coded syntax elements in software to improve coding efficiency of hardware arithmetic coder. An operating frequency of more than 200MHz was suggested. *System-on-Chip* (SoC) implementation was mentioned as a forward-looking statement but not implemented. Chen *et al.* [3] proposed a hardware CABAC

decoder with buffer for storing the syntax element contents of 24 neighboring MBs, and intended to implement this design as an IP block.

In this paper, we look at the top down approach of designing a generic statistical coder as an *intellectual property* (IP) block in SoC environment. In general, there are 3 types of coders in data compression: spatial coder to compress data in the spatial domain, temporal coder to reduce redundancy in the temporal domain. The third type of coders in data compression is the entropy coder which compresses the spatially and temporally compressed data/symbols further based on their statistics. Classical statistical or entropy coders involve one stage in which the more probable symbols are coded with shorter bit patterns, and the less probable symbols are coded with longer bit patterns. Bit patterns stored in tables are associated with the statistics of the symbols in study. The resulting compressed bit stream is supposedly less than the pre-coded bit stream. The compression ratio of the pre-coded (spatially/temporally compressed but statistically uncompressed) bit stream over the statistically compressed bit stream determines the efficiency of that coder. Huffman [4] and Arithmetic [5] coders are among these types. Recently, there emerges another more advanced entropy coder with context adaptation. There are two stages: context management and context coding. Context management involves the conversions of events, coefficients, and parameters (such as those produced by the preceding spatial/temporal stages of a video coder) into binary symbols. Context coding adopts any of the aforementioned one-stage entropy coder to provide additional compression. Statistical coder is non-parallel in nature. As an interpretation of Amdahl's law [6], the parallel portion of the codes can be vectorized and distributed to other HW functional blocks, whereas the non-parallel portion of the codes cannot be distributed and is considered as the bottleneck of the entire process. If the effect of bottleneck can be minimized, compression time can be reduced tremendously.

For that reason, we propose a design methodology for a statistical intellectual property (IP) coder under SoC environment. SoC-based IP design has been gaining popularity due to its performance (time saving and power aware), reusability (portable to different designs using the same SoC platform), and maintainability (enhancement and debugging capability). We use the *Context Adaptive Binary Arithmetic Coder* (CABAC) used in the Main profile of the H.264/AVC video coding standard as a design example. The remaining sections of the paper are organized as follows. The design methodology and its design flow are proposed in section II. The overview of CABAC and its performance and complexity analyses are presented in Section III. The HW/SW partitioning is discussed in section IV, while the HW design of CABAC, and its SoC interface are presented in section V and VI. The test strategies are presented in Section VII, followed by the conclusions in section VIII.

II. DESIGN METHODOLOGY

We propose a design methodology for a SoC statistical coder which will be realized as a soft IP block. A soft IP block is a logic synthesizable RTL design. We do not attempt to produce a hard IP block where physical synthesis has been done and GDSII ready. As seen in Figure 1, there are 8 steps going from performance and complexity analyses to constraining the design with functional timing requirements. It is crucial to assess the complexity of software to be mapped onto the supposedly application specific top-level architecture. When the complexity is low, decision can be made to run the software using the existing microprocessor. If the complexity is high, it is better to identify the bottlenecks.

After complexity analysis is performed, system specifications are written to address not only user's needs but also plans to minimize the effects of bottlenecks. In step 3, system HW/SW partitioning is carried out. The portion of reference code - which can effectively be realized into HW - is replaced by *HW preferred modules*, while the remaining portion is left to be run in SW on the existing microprocessor. The decision by which HW/SW are partitioned is based on timing constraints such as the rate at which the symbols arrived at the interface of the top-level architecture. All *SW preferred modules* are referred to as one single *SW non-IP block*, while all hardware preferred modules are referred to as one single *HW IP block*. In step 4, HW IP block is designed accordingly, followed by HW IP block verification in step 5. Verification is performed by comparing the compressed bit stream generated by the reference SW (without any HW assisted circuitries), with that output by the top-level architecture. It is important to be certain that design errors are caught at this step. In step 6, SoC design features will be designed into the HW IP block. Such features are system bus and its signals, input and output FIFO buffers, debug structures, reset signal, etc. Step 7 involves the co-simulation of both HW IP block and SW non-IP block to find ways to minimize latencies between HW IP block and SW non-IP block, and thereby, reducing

processing time of the top-level architecture. A detailed discussion of test strategies and timing evaluation is provided in Section VII. Step 8 is to constrain the design with timing requirements such as the rate at which the top-level architecture has to be clocked at. If timing requirement is not met, the process goes back to step 3 to partition HW/SW again and to redesign HW in step 4.

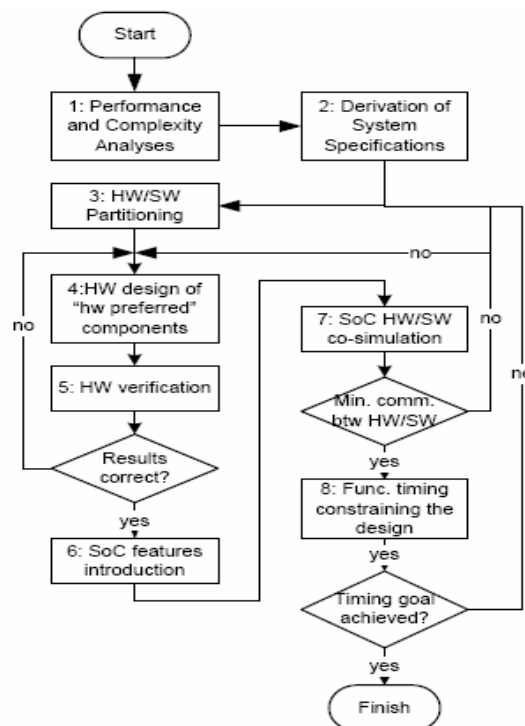


Figure 1: Design flow of a SoC statistical coder

III. OVERVIEW OF CABAC AND ITS PERFORMANCE AND COMPLEXITY ANALYSES

The emerging ITU-T/ISO/IEC standard H.264/AVC [7] for video compression aims at enhanced coding efficiency and network friendliness over a wide range of transmission systems. While the basic framework is similar to the motion compensated hybrid scheme of previous video coding standards, additional tools improve the compression efficiency. Such tools include multiple reference frames, $\frac{1}{4}$ pel accuracy motion estimation, and CABAC [8].

For the design of CABAC coder, the main parts are binarizer, context modeler, and binary arithmetic coder. The binarization process maps all non-binary valued *syntax elements* (SE) into binary sequences called bin strings. The context modeler is used to select a probability model from a pre-defined set of probability models for each bin in the bin string when the *decision coding mode* is used. There are 4 types of context modeling. The first type selects a probability model by considering the syntax values of its neighbors for the current syntax element. The second type considers the values of prior coded bins in the bin string for its selection. The last two types

are only used by quantized transform coefficients. One of them selects a probability model based on the position of the to-be-encoded quantized transform coefficient in the scanning path, while the other performs the selection by evaluating a count of the encoded levels with respect to a given threshold level. When in *bypass coding mode*, no context modeling is performed and a fixed probability model is used instead. The binary arithmetic coder performs arithmetic coding of each bin value using the selected probability model. The principle of arithmetic coding is based on recursive sub-division of interval length. The codeword is given by the minimum bit precision that uniquely identify the final interval. After each encoded bin, the probability model is updated with new statistics based on the encoded bin value.

Using the PIN tool [9], our performance and complexity analyses show that with the use of reference software JM 9.3 of CABAC H.264/AVC encoder, a consistent bit-rate saving of 4-11% can be achieved, where a higher gain is obtained at lower bit-rates. However, the algorithm in CABAC reference SW is more computational intensive compared to that of *Universal Variable Length Coder* (UVLC) used in the Baseline profile, partly due to the additional instructions executed for binarization and context modeling. The higher computational demand required by CABAC is reflected in the video encoder when *rate-distortion optimization* (RDO) is employed. With RDO, the computational complexity of CABAC increases by more than two orders of magnitudes. This is because each macroblock is now entropy coded for each of the 9 possible prediction modes so as to compute the corresponding rate. Compared to the UVLC, the computational complexity of CABAC is higher by up to 55%, while its data transfer rate is higher by up to 74%, when RDO is used. The high computation and large data transfer rate make it difficult for CABAC reference SW to meet the requirements of real-time embedded system.

Further complexity analysis shows that out of the 99% instructions executed by the CABAC for rate computation, 68% are used for encoding the macroblock in Intra 4x4 mode, while the remaining 31% are used for the other Inter modes and Intra 16x16 modes. Note that for each macroblock, the coding state is reset to its initial state prior to encoding the macroblock with the next prediction mode. As such, parallelizing techniques can be applied in hardware design to perform encoding Intra 4x4 mode separately from the other modes to achieve complexity reduction since sequential processing is not needed. As such, this research involves the design and development of a reusable IP block for efficient compression and power-aware CABAC implementation.

IV. HW/SW PARTITIONING OF CABAC CODER

Accurate partitioning of HW preferred blocks from the CABAC reference SW encoder is the first step of HW/SW co-design. All the communication channels connecting to the top-level CABAC architecture and all I/O parameters need to be determined. Then the SW/HW interface of CABAC can be

designed in the SW encoder. The interface packs input parameters of CABAC to the format which is compatible with the system bus specifications and can be easily parsed by the CABAC HW IP block. In the co-design procedure of CABAC, over 20 types of input parameters used in HW IP block have been identified, which can be classified to four categories. The first category is the slice header information such as slice type and *quantization parameter* (QP), and coding mode control signals such as RDO control information. The second category are MB parameters, such as MB type, intra prediction mode, reference frame index, motion vector difference, and the residual related data. The third category is EOS flag, which is processed at the end of each MB, and the last category is context index increment (ctxIdxInc), where a part of it is calculated by in the SW non-IP block while the remaining in HW IP block. All elements typed classification tag, sub-typed classification tag, and parameter value are packed into 32-bit packets and sent to system bus interface of the top-level CABAC architecture for verification.

The number of context index increment (ctxIdxInc) values for each SE type were profiled over several video coding tests, the resulting statistics indicate that over 60% of the ctxIdxInc values belong to the syntax element (SE) types for quantized transform coefficients such as significant coefficient flag, last significant coefficient flag, and coefficient of absolute level_minus1. The ctxIdxInc values of these three SE types are calculated in context modeler, because the calculation can be easily implemented in HW without referencing SE values in the neighboring coded blocks. Additionally, over 60% percent of transmission cost of ctxIdxInc values can be saved.

It is evident from the reference software JM 9.3 of H.264/AVC encoder that many types of context information in the neighboring MBs are needed during the calculation of ctxIdxInc value of the current coding MB. The larger the picture size, the more the information are required to store. To implement all ctxIdxInc calculations in hardware means that all context SE information needed should be stored in hardware buffers or transmitted into HW IP block through the system bus. The efficiency of hardware will be significantly influenced by the data transmission delay through system bus or the costs of continuous read and write operations on the context MB buffers. In comparison, all these context information can be easily accessed from the coding buffers in software implementation, and the cost of calculation using existing processor is much lower. Therefore, all the context related ctxIdxInc calculations are performed by SW using existing microprocessor, while all the other functions of CABAC reference SW is targeted for HW realization.

The top level CABAC architecture (without its SW non-IP block) is shown in Figure 2. It consists of two parts: the HW preferred modules (ASM controller, CABAC core, and context FIFO buffer) grouped into the HW IP core, and the SoC modules (FIFO IN Buffer, FIFO OUT Buffer, Wishbone system bus Interface) at the boundary. The whole system is triggered by a system clock, and initialized by a reset signal. The design details of these two parts are discussed in the

following section.

V. STRUCTURE OF CABAC HW IP BLOCK

The HW CABAC IP block implements CABAC entropy coding in most situations, including frame coding, field coding, *Rate Distortion Optimization* (RDO) and MB level adaptive frame/field (MBAFF) coding. As shown in Figure 2, the CABAC IP block consists of three modules: an *algorithmic state machine* (ASM) controller, a CABAC core, and a context index increment FIFO buffer (context FIFO).

A. ASM controller

The role of ASM controller is to read and parse 32-bit input data packets, and transmit the data together with additional classification tag to the CABAC coder. Except the 3-bit header of data element type, the information inside the packet can be slice header information, RDO control flag, MBAFF flag, EOS flag, or 23-bit package of one SE data consisting of the followings: 5 bits of SE type, 2 sign bits, 2 SE values of 16 bits, or 4 bits of *ctxIdxInc* values. In the case of SE, five types of input symbols have been identified; They are non-residual symbols, residual symbols, intra prediction mode, skip flag and motion vector difference. Instead of using duplicate hardware logic to process different symbols, the same hardware structure is adopted to process the SEs of the same type. Each SE in the ASM controller is packed into an output packet, consisting of SE type, value, sign, and a sub-category tag. The tag is used to differentiate the sub-categories of one SE, such as motion vector differences in the x and y directions, and residual SEs of different context block category [7]. The ASM controller consists of 10 states (Figure 3). Residual data, non-residual data, and EOS are processed in three different paths, because the SEs - including coded block flag, significance map, and last significance map - are generated from (run, level) pairs of residual within the ASM controller. In comparison, non-residual data are packed in the ASM controller and sent out directly. The ASM controller must parse new slice header information, and reinitialize CABAC context models if EOS value is true. The three processing paths of FSM controller are shown in Figure 3. RDO_processing state generates and sends RDO control signal to CABAC core.

B. The CABAC core

The design of the CABAC core is shown in Figure 4. The core consists of a binarizer, a small FIFO buffer, a context modeler, a binary arithmetic coder including decision coder engine and bypass coder engine, and bit stuffing/packing block. The decision coder engine is composed of the arithmetic coding core and the renormalizer.

The binarizer parses the input packets from the ASM controller, and converts them into SE values to binary decision string according to the SE type. A look-up table is used to process the SE values such as MB types, while bit shifter scheme is adopted to implement unary or truncated unary

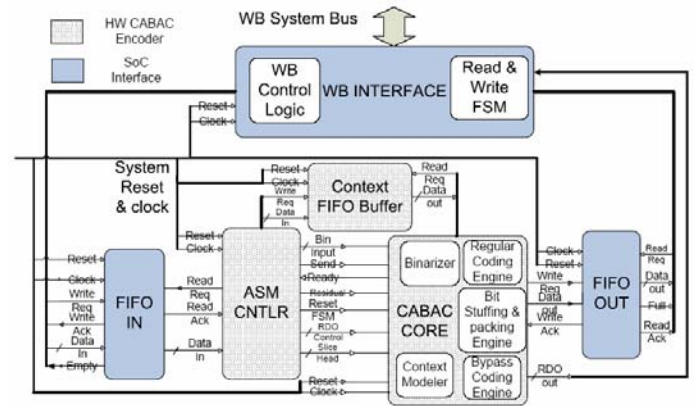


Figure 2: Top-level architecture of the CABAC statistical coder

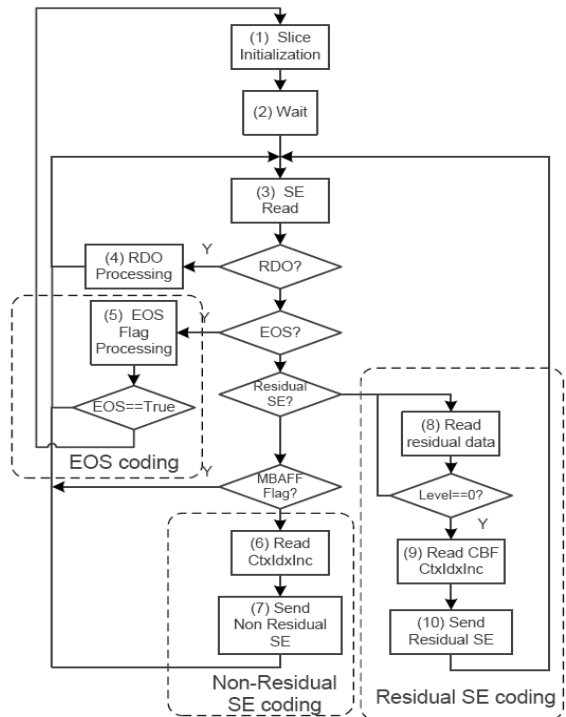


Figure 3: The ASM controller

binarization more efficiently. Both the length and value of each binary decision string are provided to the binary arithmetic coder. For some SE types, such as motion vector difference, a suffix decision string may also be generated, which is processed after the prefix string. Because the time to generate a decision string is much less than that of coding the string, a small FIFO buffer is built between the binarizer and the subsequent coding blocks in order to remove the time that the arithmetic coder and context modeler wait for the next decision string. Exponential Golomb coding that generates suffix bin strings of absolute level and motion vector difference is done in a separate coding module to max coding time of binarizer.

The context modeler implements the initialization of probability state index table and MPS value table, and calculation of context offset of each binary decision string. After fetching *ctxIdxInc* value from context FIFO or

calculating it in the context modeler, context index is calculated and context model of each binary decision is selected from the lookup tables. The update of context model is also implemented in this block. Counting the number of `ctxIdxInc` values of each SE type in several video coding tests. In the RDO coding mode, loading and storing of the whole context models are implemented by data copying between context tables and their backups.

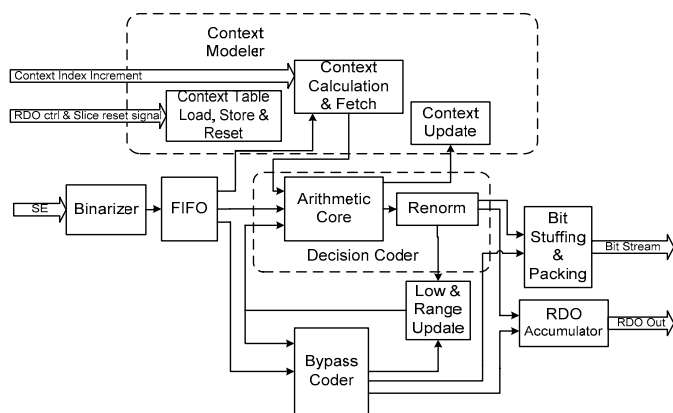


Figure 4: Partially parallel coding structure of CABAC core

The arithmetic coding core and renormalizer of decision coder are time consuming blocks. It is not practical to use multiple processing units to accelerate the coding speed because of the data dependency of Low and Range values of current coding bin on the previous bin coding results. In the current design, although these two blocks process data in a sequential fashion, the context modeler is designed to process data in parallel with the renormalizer (as illustrated in Figure. 4). First, the bin decision with corresponding context model is coded in the arithmetic core. The updated Low and Range are then processed in the renormalizer. Concurrently, the context modeler updates the context information of the current context model. If the context offset and `ctxIdxInc` value of the next binary decision are available, the next context index is calculated and the next context model is selected. With this partially parallel scheme in binary arithmetic coder and the FIFO buffer of binary decision strings, the maximum coding time of one binary decision can be reduced to the time of Range LPS fetching, arithmetic coding, and renormalization.

The output bits from bypass coder and renormalizer of decision coder are packed into a 32-bit stack of the bit stuffing and packing block. When the stack is full, the 32-bit packet is written to the FIFO OUT buffer. After coding of each slice, the output bit stream is stuffed with bit 0 to achieve integer-byte length if it is not so. Bit stuffing mechanism is implemented by bit-shift and bit-and operations, and controlled by monitoring of the status of a 3-bit counter and the EOS flag. After the coding of the last MB in a slice, data bytes left in the stack are sent to the FIFO OUT buffer if the stack is not empty. Additional tag information is output from CABAC coder to indicate the number of bytes left to be sent out. In RDO coding, the coding lengths of decision bins from bypass coder and

decision coder are accumulated and outputted when one RDO mode coding finishes.

C. Context index increment FIFO buffer (context FIFO)

The context FIFO buffer stores the `ctxIdxInc` values that are generated by the software. Each `ctxIdxInc` value is referenced together with context offset to select proper context model for one binary decision. In the case of residual SE coding, only the `ctxIdxInc` value of coded block flag is sent to context FIFO, as the values of the other three SE types are calculated in CABAC coder. Accordingly, a small FIFO of 32 x 4 bits is large enough to accommodate all `ctxIdxInc` values of one SE. The context FIFO is empty after coding of each non-residual SE or after coding of the coded block flag of residual coding SEs. Therefore, the structure of context FIFO is simplified with no consideration on buffer full or empty situations. The FIFO is triggered by the clock, write and read request signals. Design optimization methods, such as skipping the process of `ctxIdxInc` value in the ASM controller and writing `ctxIdxInc` value directly to context FIFO, or transferring of multiple `ctxIdxInc` values (4 bits each) in one data packet to further reduce the communication load and enhance CABAC processing speed.

VI. SoC FEATURES OF THE CABAC CODER

SoC interface of the CABAC coder consists of three components (Figure. 2): input FIFO (first-in first-out) buffer, output FIFO buffer, and *Wishbone* (WB) system bus interface. Debug structures can also be inserted at a later time. Input data are first transmitted to the WB system bus interface and written into the input FIFO buffer. The ASM controller reads packets from the input FIFO, and parses them according to the header information. Common syntax elements (SE) or end of slice (EOS) flag are formatted in the ASM controller, and sent to CABAC coder directly. Context index increment (`ctxIdxInc`) values are written into the context FIFO buffer. The CABAC core receives input data from ASM controller, and encodes them through decision coding or bypass coding route. In the decision coding route, the `ctxIdxInc` values are generated by the context modeler or input from context FIFO. The output bits are also packed into a 32-bit packet in the bit stuffing and packing engine and written to the output FIFO buffer. Output coding data are then read by WB interface and sent to the bit stream buffer in the RAM. RDO coding length is sent directly from CABAC core to the WB interface with no buffering.

A. Input and output FIFO buffers

The input FIFO is designed to buffer large number of data packets received from WB interface before the CABAC core can process them. The output FIFO stores data packets output by the CABAC core before they are sent out by the WB interface. The read or write request to the FIFO is processed at the positive edge of the clock, and acknowledgement signal is set high when the operation completes. To reduce FIFO access time, concurrent read and write operations are supported in the FIFO structure when the buffer is neither full nor empty. In the

case of input FIFO empty or output FIFO full, the CABAC core will pause until new data are written to the input FIFO or coding results are read out from the output FIFO. No data will be lost in these special cases.

B. The Wishbone bus interface

The Wishbone System-on-Chip (SoC) Interconnection Architecture [10] for Portable IP Cores is a flexible system bus design for use with other semiconductor IP cores. The IP core with WB interface can be easily integrated into a SoC system through WB bus, and its design can be reused in different applications. At the top-level CABAC architecture, a WB master interface is built. Because it is more convenient for a master to control data transmission on the bus, bus holding time is reduced during CABAC data transmission, and conflicts on the system bus are also reduced.

Different control strategies can be applied in the WB master to control the data written to the input FIFO, and read from the output FIFO. The aim is to reduce the idle time of CABAC core to the minimum. The current control strategy is to fetch a block of data packets from the slave and write the packets to the input FIFO when the input FIFO is nearly empty, and read a block of data packets out of output FIFO and write the packets to slave when the output FIFO is nearly full. The control logic is implemented by a Read and Write FSM. Single read and single write WB bus operations are implemented according the WB specifications. The hand-shaking mechanism of WB bus ensures the data transmission safety. Block read or block write operations can also be implemented to enhance data communication rates on the bus with more buffer space and control logic at the WB interface.

C. Prospects of a video encoder

System level HW/SW co-design of H.264/AVC video encoder is planned at stage 2, and implemented after stage 6 (Figure 1). The prospective system scheme of SoC video encoder consists of an on-chip processor, HW preferred blocks, Wishbone (WB) system bus, WB bridge, DMA, and RAM and ROM. Computation intensive functional blocks, such as CABAC, motion estimation, etc. are implemented as hardware processing blocks. In this report, we only discuss the design of the CABAC architecture. Other complex control algorithms, such as the rate-distortion optimization, are run on the on-chip processor, by running the instruction program stored in the RAM. Each hardware block in the scheme is connected to the WB system bus through WB interfaces. The WB bridge is the arbitrator deciding which WB master can use the system bus. The corresponding bus connection between the master and slave is built up in the bridge. To reduce the computation burden of processor, large amount of data transmission between HW blocks and RAM may be controlled by a DMA. System software programs are stored in the ROM and loaded to RAM when the system boots.

VII. TEST STRATEGIES

Different test strategies were used at different stages of the

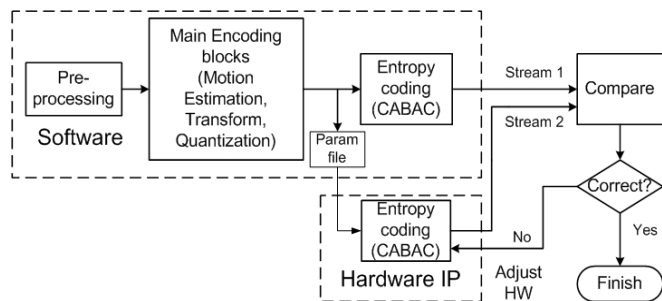


Figure 5: Verification of the HW IP block

TABLE I:
ENCODER CONFIGURATION SETTINGS USED FOR THE TESTS

Tools	1	2	3	4	5	6	7	8	9
Hadamard	0	0	0	0	0	1	1	1	1
Search Range	16	16	16	16	16	16	8	32	16
Ref. Frame	1	1	1	3	5	5	5	5	5
Inter Mode	1	4	7	7	7	7	7	7	7

design flow to i) verify HW functionality (step 5, Figure 1), ii) ensure minimum communication latency between HW IP block and SW non-IP block (step 7, Figure 1), and iii) ensure satisfaction of overall timing requirement at the top-level CABAC architecture (step 8, Figure 1).

Figure 5 depicts the verification of CABAC HW IP block. Two bit streams are generated and compared against each other. Stream 1 is generated as output of the CABAC SW coder using the reference SW without any hardware assisted circuitries. The input to the CABAC SW is redirected to a parameter file which is then used to test the CABAC HW as if the HW has been incorporated into the entire video codec. Stream 2 is generated as a result of the CABAC HW IP block. Any differences in the bit streams would indicate a design error in the CABAC HW IP block.

Test sequences *Foreman* and *Coastguard* for both QCIF and CIF formats were used. The tests were carried out using a sequence of 15 frames, with quantization parameter (QP) values between 24 and 36. A *Group of Pictures* (GOP) consists of *IPBPB...PB*. Configurations 1-9 have been used and are given in Table I. For each of these configurations, Frame, field, RDO, and MBAFF coding formats are tested. The output bit-streams obtained from the SW coder and HW IP block were identical. This validates that our HW IP block is functionally identical to the SW CABAC coder.

For the SoC HW/SW co-simulation step, we compared the timing performance between our top-level CABAC architecture and the CABAC reference SW. This is done by measuring the time taken by each to perform entropy coding of the syntax elements for the same test sequence. Let $t_{top-levelCABAC-architecture}$ denote the time taken by our top-level CABAC architecture to perform entropy coding, and $t_{referenceSW}$ denote the time taken by the CABAC reference SW, the objective is to achieve significant time-saving as shown in

expression (1).

$$t_{top-levelCABACarchitecture} \ll t_{referenceSW} \quad (1)$$

We further assume that the operating frequencies (or periods) of the existing processor and the top-level CABAC architecture are similar. Therefore, the expression for numbers of cycles corresponding to expression (1) can be derived:

$$c_{top-levelCABACarchitecture} \ll c_{referenceSW} \quad (2)$$

Since the top-level CABAC architecture uses a HW/SW co-design of both HW IP block and SW non-IP block, it is further necessary to consider the inter-block latency. Therefore:

$$c_{top-levelCABACarchitecture} = c_{HWIPBlock} + c_{SWnon-IPBlock} + c_{inter-block-latency} \quad (3)$$

In Eq.(3), when CABAC reference SW is used, the $c_{SWnon-IPBlock}$ is in fact $c_{referenceSW}$, and $c_{HWIPBlock}$ and $c_{inter-block-latency}$ become zero. When a large portion of CABAC reference SW is implemented by HW, the corresponding $c_{HWIPBlock}$ should reduce $c_{referenceSW}$ to a much larger proportion, while introducing $c_{inter-block-latency}$. Also, in Eq.(3), the remaining portion of $c_{SWnon-IPBlock}$ cannot be reduced because no change is made when it is running on the same existing processor. The $c_{HWIPBlock}$ can be reduced by applying better parallelism and better technology, while the $c_{inter-block-latency}$ can be minimized by better buffering and better partitioning.

The coding cycles of CABAC core are summarized in Figure 6. An average of 5.5 cycles is needed to encode one bin in decision coding and 2 cycles in bypass coding. The $c_{inter-block-latency}$ only contributes a few cycles to the total coding time which is the inter-block delay from processor to the CABAC core, and it is ignored.

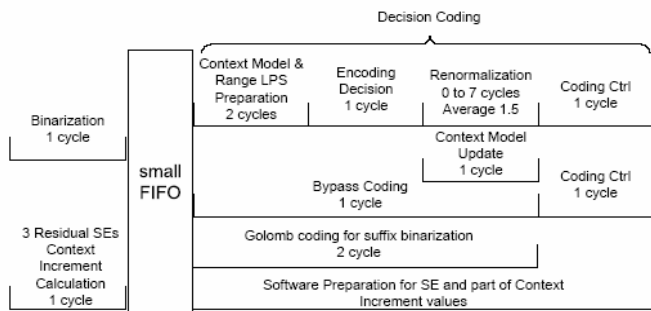


Figure 6: Coding cycles of CABAC core

Our test results of one QCIF intra frame show that $c_{referenceSW}$ is 5,201,560, $c_{SWnon-IPBlock}$ is 570,026, and $c_{HWIPBlock}$ is 145,987. It satisfies expressions (2) and (1), and proves the efficiency achieved by implementing major work of CABAC in HW instead of SW. Software optimization methodology can also be explored to further reduce the coding cost of the SW non-IP block and thus the top-level CABAC architecture.

The third test is on the overall timing requirement of the top-level CABAC architecture. This can be seen as the rate at which the statistical coder is expected to process the coming

syntax elements. This test is critical for real-time encoding of video. At the time of this writing, we intend to relax this constraint and focus our effort in HW verification and SoC HW/SW co-simulation.

VIII. CONCLUSIONS

In this paper, we propose a SoC software / hardware co-design methodology for a statistical coder. We emphasize on the complexity analysis by which bottlenecks can be identified and its effects can be minimized beforehand. The HW verification stage ensures error-free design even before the introductions of SoC features. SoC HW/SW co-simulation further enables the design to achieve minimum communications between the HW IP block and the SW non-IP block. To apply this design flow to top-level CABAC architecture, we are able to define and refine the HW IP block with minimum design time. Compared to existing statistical coders, this design is aimed for significant coding time saving by balancing between software modules and hardware modules. It is well verified with standard video test sequences, and is reusable as an IP in a SoC environment. The HW CABAC IP is simulated in ModelSim and synthesized by Design Compiler of Synopsys targeted toward the ST 0.13 μ m standard cell library. Synthesis results show that CABAC coding core can work at a clock frequency of over 300MHz. Processing speed over 55 Mbps can be achieved, which is suitable for QCIF RDO coding. If RDO mode is turned off, and extra processing power can be used for CABAC HW IP, it is suitable for HDTV-scaled video coding.

REFERENCES

- [1] W. Di, G. Wen, M.Z. Hu, and Z.Z.Ji, "A VLSI Architecture Design of CAVLC Decoder", *IEEE Proceeding ASIC*, vol. 2, pp. 962-965, Oct. 2003.
- [2] R. R. Osorio and J. D. Bruguera, "Arithmetic Coding Architecture for H.264/AVC CABAC Compression System," in *Proceedings of the EUROMICRO Systems on Digital System Design*, 2004.
- [3] J.W. Chen, C. R. Chang and Y. L. Lin, "A Hardware Accelerator for Context-based Adaptive Binary Arithmetic Decoding in H.264/AVC," in *IEEE International Symposium on Circuits and Systems*, Page(s):4525 – 4528, Vol. 5, May 2005.
- [4] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", *IEEE Proc IRE*, vol. 40, pp. 1098-1101, 1952.
- [5] I. H. Witten, R. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression", *Communications of the ACM*, pp. 520-541, Jun. 1987.
- [6] G. M. Amdahl, "Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability," *Proc. AFIPS Conf.* pp.483-485, Reston, VA, 1967.
- [7] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," *Joint Video Team*, Mar. 2003.
- [8] D. Marpe, H. Schwarz and T. Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard," *IEEE Trans Circuits Syst. Video Technol.*, vol. 13, no. 7, Jul. 2003.
- [9] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser and G. Lowney, "Pin: Building Customized Program Analysis Tool with Dynamic Instrumentation," *PLDI*, Apr. 2005.
- [10] *Wishbone specifications: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, Revision: B.3, Sep. 2002.