Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 197843, 16 pages
http://dx.doi.org/10.1155/2015/197843

Research Article

A Hybrid Scheme Based on Pipelining and Multitasking in Mobile Application Processors for Advanced Video Coding

Muhammad Asif,[1] Imtiaz A. Taj,[1] S. M. Ziauddin,[2] Maaz Bin Ahmad,[3] and M. Tahir[1]

[1]*Mohammad Ali Jinnah University, Islamabad, Pakistan*
[2]*Streaming Networks (Pvt.) Ltd., Islamabad, Pakistan*
[3]*Mohammad Ali Jinnah University, Karachi, Pakistan*

Correspondence should be addressed to Muhammad Asif; astz786@yahoo.com

Received 15 May 2015; Revised 22 September 2015; Accepted 4 October 2015

Academic Editor: Bormin Huang

One of the key requirements for mobile devices is to provide high-performance computing at lower power consumption. The processors used in these devices provide specific hardware resources to handle computationally intensive video processing and interactive graphical applications. Moreover, processors designed for low-power applications may introduce limitations on the availability and usage of resources, which present additional challenges to the system designers. Owing to the specific design of the JZ47x series of mobile application processors, a hybrid software-hardware implementation scheme for H.264/AVC encoder is proposed in this work. The proposed scheme distributes the encoding tasks among hardware and software modules. A series of optimization techniques are developed to speed up the memory access and data transferring among memories. Moreover, an efficient data reusage design is proposed for the deblock filter video processing unit to reduce the memory accesses. Furthermore, fine grained macroblock (MB) level parallelism is effectively exploited and a pipelined approach is proposed for efficient utilization of hardware processing cores. Finally, based on parallelism in the proposed design, encoding tasks are distributed between two processing cores. Experiments show that the hybrid encoder is 12 times faster than a highly optimized sequential encoder due to proposed techniques.

1. Introduction

In this era, portable multimedia devices with their corresponding media technologies play an important role in our daily life. These devices perform multimedia functionalities like video recording and playing, watching TV (mobile TV), videotelephony, and video streaming. Highly efficient video coding at lower power consumption is the fundamental requirement of these devices.

The H.264/AVC Advanced Video Coding standard (AVC) provides high-quality coding of video contents at very low bit rates relative to the former video compression standards [1–3]. The compression quality is achieved by adding a number of new compression techniques including spatial intraprediction, (4 × 4) integer transform, multiple reference frames, multiple block sizes for interframe coding, quarter pixel motion estimation (ME), deblock filtering, and content-adaptive arithmetic coding [4–6]. These additional features and functionalities improve the coding efficiency at the cost of a significant increase in computational complexity [3, 5, 7]. So it would be a challenging task to achieve real-time H.264 coding even on the recent available high power single processing units.

JZ47x is a series of low powered mobile application processors highly suitable for battery operated portable devices. These devices, for example, body worn video recorder, wireless video sensors, and video streamer, require long operation time. This series of processors introduces an innovative architecture to fulfill both high speed mobile computing and computationally intensive video coding requirements of portable multimedia devices. Particularly for H.264/AVC video coding, JZ47x processors (JZ4760, JZ4770, and JZ4780) are equipped with dedicated hardware processing units of some encoder blocks. They have two major cores, that is,

a Central Processing Unit (CPU) and a Video Processing Unit (VPU), where the VPU core controls the dedicated hardware processing units. They also have general-purpose direct memory access units (DMAs) for efficient data transfer and management during video coding.

Although JZ47x architecture has the ability to meet the computational requirements of H.264/AVC encoder, it places certain limitations on the system designer as well. The efficient implementation of H.264/AVC encoder for the JZ47x processors is a challenging task because some of the encoder blocks are implemented in hardware and others need to be implemented in software. As H.264 encoder has an integrated framework, where task dependency is large, the encoder tasks distribution, scheduling, and synchronization among hardware and software modules are a real test of designer skills. Moreover, it demands efficient memory management and simultaneous exploitation of all processing units for achieving real-time encoding.

This paper proposes an end-to-end software-hardware hybrid H.264/AVC encoder implementation scheme for JZ47x series of mobile application processors. The JZ4770 is taken as an example to implement the proposed scheme. The proposed scheme distributes the encoding tasks among hardware and software modules. The small size of on-chip memories and restrictions imposed by hardware processing units cause large data transferring among memories during the encoding process. To reduce this data transferring, several optimization techniques are proposed in this work. The hardware video processing units and DMAs need to be configured before each execution and take a prescribed amount of time to perform the assigned tasks. Because of inherent task and data dependencies in H.264/AVC encoder, the processor remains blocked during that time. This time is named as processing or polling time. It also causes the idling of hardware processing units. To reduce the polling and idling time of hardware processing modules, a pipelined design for encoder is proposed after exploiting fine grain macroblock (MB) level parallelism in which multiple macroblocks (MBs) are processed in parallel. Finally, a mechanism is developed to efficiently distribute the computational load among CPU and VPU cores to increase the encoding rate. The experimental results show that the incorporation of hardware processing units increases the encoding rate (from 4.48 fps to 23 fps) but the major advantage (from 23 fps to 58.70) is due to the proposed methodologies.

The main contributions of this paper are as follows:

(i) An end-to-end hybrid implementation of H.264/AVC encoder for JZ4770 mobile application processor is proposed that efficiently utilized all parallelization capabilities of this platform for high-performance video encoding at low-power consumption.

(ii) Memory efficient design for deblock filter video processing unit is proposed to increase the data reuse ratio and to reduce memory accesses.

(iii) Developed several optimization techniques are to speed up memory access and data transferring among memories during the encoding process.

(iv) A pipelined design for H.264 encoder is developed after exploiting fine grain MB level parallelism, where multiple MBs are processed in parallel.

(v) A mechanism is developed to concurrently utilize the CPU and VPU cores for encoding. This mechanism provides efficient task synchronization, scheduling, and load balancing between CPU and VPU cores.

The rest of the paper is organized as follows. Section 2 presents the literature review. Section 3 describes the JZ4770 platform architecture. The proposed scheme is described in Section 4. Encoder performance is evaluated in Section 5. Finally, the conclusion is drawn in Section 6.

2. Literature Review

Numerous schemes for implementing H.264/AVC encoders on different platforms have been proposed. Nguyen et al. [11] described the implementation of the H.264 baseline profile encoder for multimedia applications on Reconfigurable Multimedia System 2 (REMUSII). The implemented encoder achieved the coding speed of 30 fps for CIF (352 × 288) resolution video sequences. Sankaraiah et al. [10] proposed a parallelization method based on Group-Of-Pictures (GOP) for H.264 encoder. In this scheme, each GOP is encoded independently in a separate thread and the frames being referenced are included in the GOP. This resulted in reducing the encoding time by 5.6 to 10 times as compared to the standard sequential code implementation. However real-time encoding is difficult to implement using this technique. Moreover, this technique demands a significant amount of memory for storing all the frames. Asif et al. [12] exploited the macroblock (MB) level parallelism in H.264/AVC encoder and based upon this parallelism proposed a scheme to implement the encoder for multicore platform. The encoder is able to encode NTSC (720 × 480) and HD 720p (1280 × 720) resolutions at a frame rate of 72 fps and 32 fps, respectively. Aw et al. [13] described the optimizations techniques for implementing the H.264/AVC HD video encoding on TI TMS320DM648 DSP platform. The implemented encoder performed real-time encoding and streaming at 25 fps for 720p HD video. Lin and Yang [14] presented optimized implementation of H.264/AVC encoder on the TM320DM642 DSP platform. The encoder can encode VGA (640×480) and CIF resolution at a frame rate of 22.6 fps and more than 40 fps, respectively.

Schneider et al. [15] developed and optimized H.264 baseline profile encoder on TMS320DM642 DSP platform. In order to accelerate the encoding rate, Enhanced Direct Memory Access (EDMA) Controller, look-up tables, and intrinsic are used. The encoder could encode CIF resolution video sequences at a frame rate of 30 fps. For cell processor, Alvanos et al. [9] presented a way to implement fine grained task level parallelism of encoder. The runtime Tagged Procedure Calls (TPC) and a task-based programming model are used to parallelize the encoder. The implementated encoder achieved speedup between 4.7x and 8.6x on six synergistic processing elements (SPEs), compared to the serial version running on the power processing element (PPE). However, this implementation required significant programming effort

and had task management overhead. For real-time HD video coding, He et al. [16] exploited a decentralized pipelined parallel coding scheme using eight SPEs. The decentralized task creation decreased the task management overhead in this implementation. They used on-chip communication and multibuffering to transfer data among different encoder modules for efficient communication. The encoder is able to perform real-time HD encoding (1920 × 1080@31 fps) by using 8 SPEs (58 fps on 16 SPEs). Lu and Hang [17] and Schwalb et al. [18] presented adaptive ME algorithms for NVIDIA GPUs. These algorithms mainly focused on reduction of the computational complexity of ME and also tried to exploit potential parallelism. Moreover, these proposals mostly exploited data-level parallelism and focused on a single module of the prediction-loop of the H.264/AVC encoder.

Cheung et al. [19] and Azevedo et al. [20] proposed the application of scheduling techniques at the MB level for scalable H.264 video coding to assist embedded multicore devices. In these techniques, a specialized hardware for the optimized task submission or retrieval unit is used. However, the MB level implementation of the scheduling control makes this model rather expensive for the CPU + GPU platforms. Momcilovic et al. [21] proposed parallel dynamic model for hybrid GPU + CPU systems. In this model, entire interprediction loop of the encoder is parallelized on both the CPU and the GPU. To dynamically distribute the computational load among CPU and GPU, a computationally efficient model is also proposed. The presented model included both dependency aware task scheduling and load balancing algorithms. Momcilovic et al. [8] also presented an improved dynamic dependency aware scheduling algorithm for collaborative video encoding on hybrid GPU + CPU platforms. The scheme speeds up the encoder 2.5 times as compared to highly optimized GPU-only encoding.

In short, most of these schemes lack in describing the hybrid implementation of H.264/AVC encoder in which some blocks of encoder are implemented in hardware and the rest in software. As such, implementation requires efficient encoder tasks distribution, scheduling, and synchronization among hardware and software blocks. So there is a need to work in this direction to highlight the issues involved in such kind of implementation.

3. JZ4770 Architecture

The JZ4770 is an asynchronous multiprocessor (AMP) architecture. CPU and VPU are its major cores. The CPU core contains main processor (named as J1) which operates at a clock rate of 1000 MHz. On the other hand, VPU core has Auxiliary Processor (AUX) which operates at a clock rate of 500 MHz. Both J1 and AUX are MIPS (Microprocessor without Interlocked Pipeline Stages). A master slave model exists between J1 and AUX. The operating system kernel uses J1 while the users can program the AUX. J1 loads binary on AUX and commands it to start execution.

The video processing units for H.264 encoding are vector matrix arithmetic unit (VMAU), motion compensation and estimation engine (MCE), and deblock (DBLK). There are three types of on-chip memories including two tightly coupled shared memories TCSM0 (16 KB) and TCSM1 (48 KB) along with SRAM (scratch RAM 28 KB). Furthermore, DMA0, DMA1, and DMA2 are three general-purpose DMAs coupled with TCSM0, TCSM1, and SRAM, respectively [22, 23]. These DMAs support three different data transfer types, that is, byte, short, and integer.

The hardware video processing units and DMAs need to be configured before each execution and do not support multiple instances. They take a prescribed amount of time to perform the assigned tasks. The constraints of these hardware processing units are that they operate at MB level and take input/output data from/to on-chip memory. But the reference frames used by MCE module are allocated on main memory. The current and reference frames are allocated on main memory because of limited size of on-chip memories.

4. Proposed Scheme

4.1. H.264/AVC Porting to JZ4770. The sequential software (C code implementation) based x264 encoder is ported on JZ4770 platform. In this work, baseline profile of encoder is used due to its low complexity as compared to the main and extended profiles. To distribute the encoding tasks among software and hardware processing units, x264 encoder is transformed into a highly modular and flexible structure. In this structured encoder, all the functional modules are independent software modules. It allows an easy replacement of the software modules with their corresponding hardware processing units.

As shown in Figure 1, hardware processing units replace the demarcated functionalities. MCE module replaces motion estimation and compensation. The VMAU module replaces the software functions that perform intraprediction, residual calculation, DCT/IDCT, Q/IQ, and MB reconstruction. The DBLK module is used to remove deblocking artifacts from reconstructed frame. The software modules include intraprediction mode selection, reordering, Context-Adaptive Variable-Length Coding (CAVLC), edge extension, bit rate control, file reading or video capturing, and file writing. All these software modules execute main processor. The main processor is also responsible for the configuration of hardware modules before each execution.

4.2. H.264/AVC Encoder Data Dependencies and Profiling Analysis. Before optimization, parallel processing, and tasks mapping to speed up the H.264/AVC video encoder, an extensive analysis is required. This analysis encompasses computational requirements of various functional elements of the encoder and data dependencies among them.

In H.264/AVC encoding, MB is a basic processing unit of a frame. For encoding the frame, raw data of each MB is transferred from main memory to on-chip memory. The encoding mode of MB is selected either intra or inter. For intramode, the inputs to VMAU unit are raw MB data and intraprediction mode. For intermode, the inputs are raw MB data and motion compensated data. In intraencoding, the suitable intraprediction mode selection is done through software module. The VMAU is configured for intramode. The outputs of VMAU unit are quantized coefficients, coded

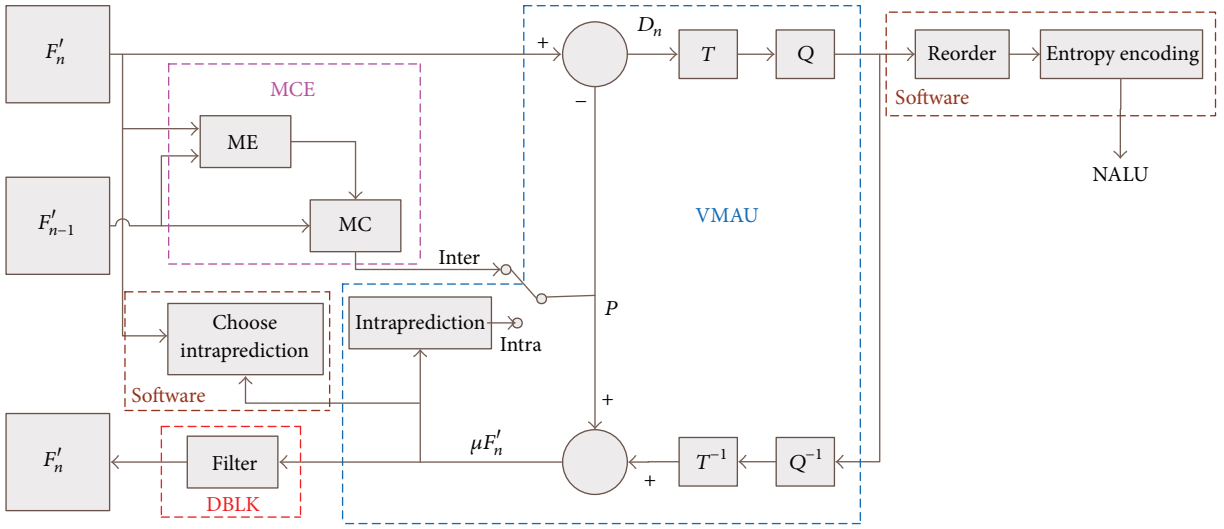

Figure 1: Hardware software codesign.

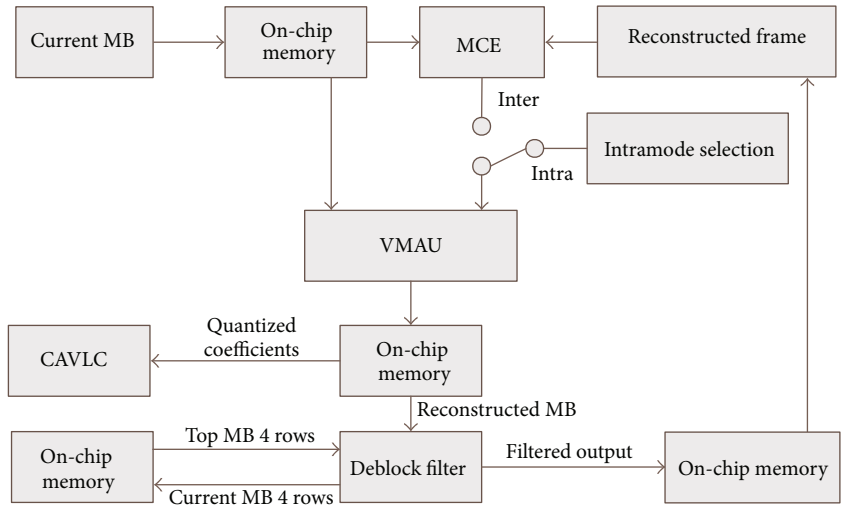

Figure 2: MB level processing flow of hybrid encoder.

block pattern (CBP), and reconstructed MB. Entropy coding encodes quantized coefficients and CBP, while the DBLK unit removes the blocking artifacts of reconstructed MB. The output of DBLK unit is transferred from on-chip memory to reconstructed frame on main memory. In interencoding, the MCE processing unit performs motion estimation. The outputs of MCE are motion vectors and motion compensated data. The VMAU is configured for intermode. The rest of processing is similar to intramode encoding.

Figure 2 indicates the sequential dependencies between functional modules of the encoder; that is, the output data of one module corresponds to the input data of another subsequent module. Because of these dependencies, one hardware processing unit is used at a time and all the remaining units are idle. To assure the compliance of the whole set of strict dependencies imposed by the encoder, it can be observed that the exploitation of fine grained level parallelism may help to minimize the idling time of hardware processing units.

Figure 3 represents the breakdown of hybrid H.264/AVC encoder processing time with respect to the various functional modules. In order to evaluate the computational complexity of the encoder, NTSC resolution test video sequences including mobile, football, intros, garden, galleon, vtc1nw, and washdc are encoded. The encoding parameters are set as follows: MV search range is −32 to +32 pels and block size is 16 × 16, RD optimization is disabled, one reference frame is for motion estimation and compensation, motion estimation scheme is diamond search, MV resolution is 1/4 pel, GOP is chosen as 25 with structure IPPP, and four quantization parameters (QPs) are 24, 28, 32, and 36.

Figure 3 shows that data copying/transferring among memories is a dominant task taking about 30% of the total processing time. This data transferring is necessary because

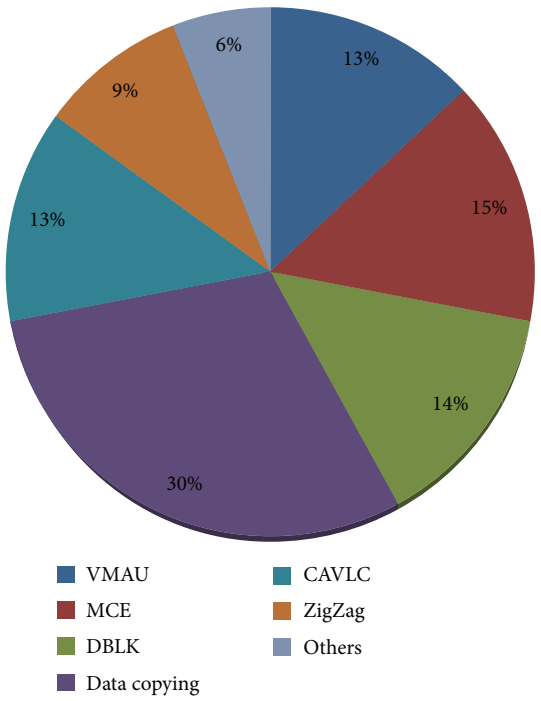

Figure 3: Time breakdown of hybrid encoder.

of the restrictions imposed by the hardware processing units and the limited sizes of on-chip memories. On the other hand, VMAU, MCE, and DBLK take significant portion of the overall processing time. The time taken by these hardware modules is their configuration and processing time. The CAVLC, Zigzag scanning, and others take the rest of the processing time.

Considering the above analysis, there is a need to exploit the parallelism in encoder for efficient utilization of hardware resources by reducing their idling and polling time. Moreover, efficient memory management is required to reduce the data transferring and copying operations during the encoding process.

4.3. Optimization Techniques for Memory Management and Data Transfer. From the profiling analysis presented in Section 4.2, it is clear that data transferring and accessing among memories are the most computationally intensive task. This section presents numerous optimization techniques to reduce the data transferring time and to increase data reuse ratio.

4.3.1. Memory Efficient Design for DBLK Module. The deblocking filtering of an MB needs pixel data from neighboring MBs, that is, top MB bottom edge (last four rows (4×16)) and left MB right edge (last four columns (16×4)). The memory constraint of the hardware DBLK unit is that the current MB and left MB right edge data must be adjacent in on-chip memory for both input and output as shown in Figure 4.

The outputs of DBLK unit are reconstructed filtered pixel data of current MB (16×16), top MB bottom edge (4×16),

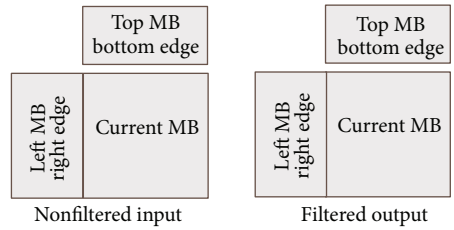

Figure 4: Input and output data placement of DBLK module.

left MB right edge (16×4), and nonfiltered current MB bottom edge (16×4). The pixels data of top MB bottom edge can be placed anywhere in on-chip memory. The left MB right edge and top MB bottom edge are usually copied from reconstructed frame on main memory to on-chip memory. Similarly, the output generated by DBLK module is copied from on-chip memory to main memory. This data transferring is necessary because reconstructed frame is stored on the main memory due to limited size of on-chip memory. In order to reduce these data copying operations, an efficient memory design for DBLK module is proposed as shown in Figure 5.

According to this design, the output of VMAU module is taken as the input to the DBLK module. This avoids data coping of current MB from main memory to on-chip memory. Moreover, the output of the DBLK module is stored adjacent to its input (current MB reconstructed pixel data from VMAU module) in such a way that the right edge of output overlaps with the left edge of its input as shown in Figure 5. This eliminates the need of copying the right edge of left MB, required in next iteration. The nonfiltered current MB bottom edge, output of DBLK, is stored in on-chip memory in such a way that the bottom edge of complete MB row of a frame is stored in contiguous memory. In this way, it can be reused as top MB bottom edge nonfiltered input of DBLK module for next MB row processing. Finally, only the filtered output is copied in reconstructed frame. The boundary cases, that is, top MB row, left MB column, and bottom MB row, have been handled as per requirement. This design resulted in increasing the data reusability and avoiding the excessive data copying operations.

4.3.2. Efficient Data Transfer. As mentioned in Section 3, hardware video processing units take input from and store output data to on-chip memories. Because of the limited size of on-chip memories, input and reconstructed frames are stored on main memory. For processing a frame, each MB data is copied to the on-chip memory. Similarly, the reconstructed MB needs to be transferred from on-chip memory to main memory. This requires multiple data transfer operations in which memory copy routines do involve CPU. This is a time consuming task as the CPU cannot execute any other operations during this time. The DMA is a handy tool to carry out these transfers without invoking the CPU.

In the proposed scheme, DMAs are used to speed up the data transferring among memories. Moreover, on-chip memory allocation for current and corresponding reconstructed

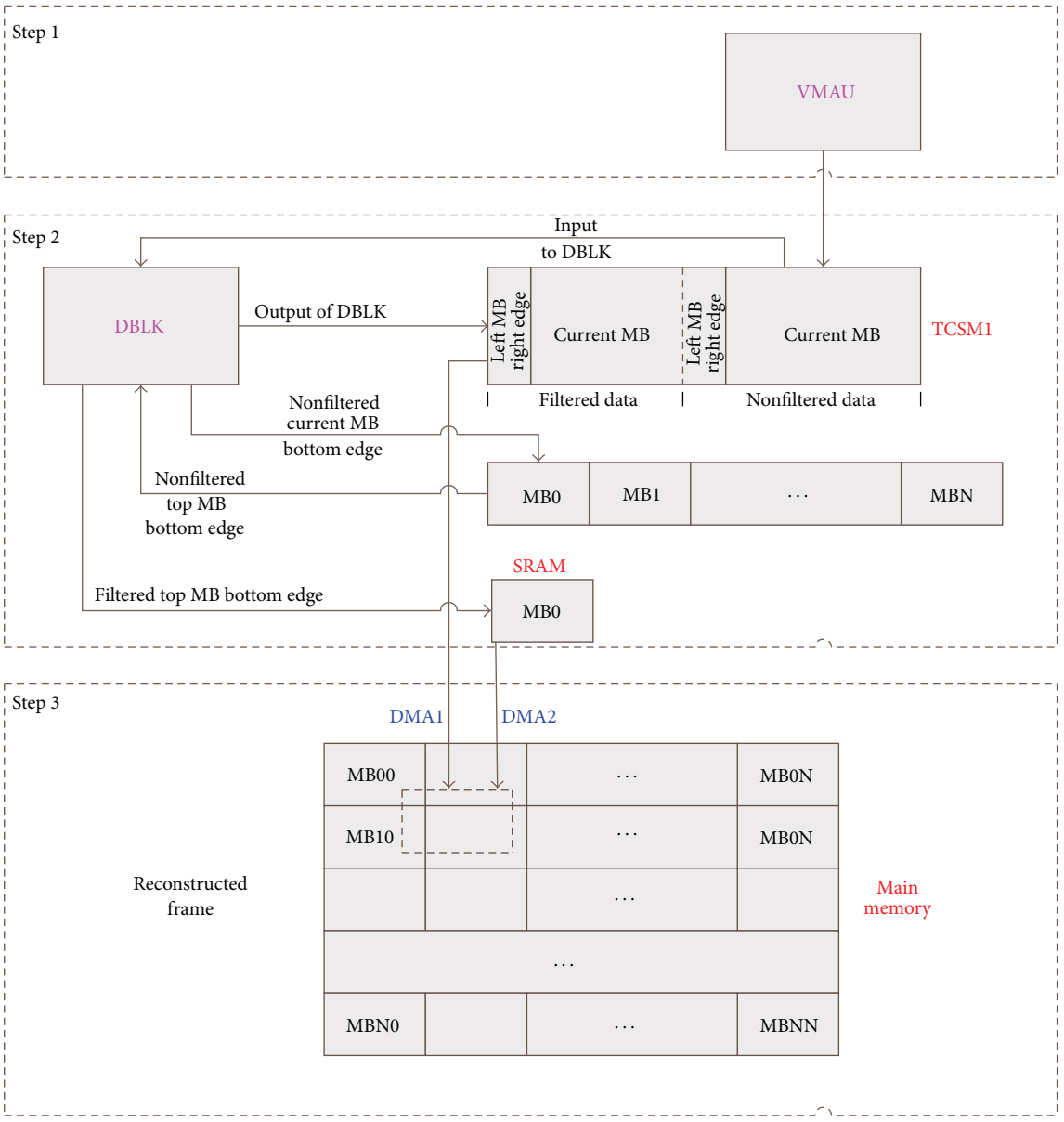

Figure 5: Design for DBLK module.

MB is done in such a way that all DMAs (DMA0, DMA1, and DMA2) can be performed in parallel for transferring data among memories. The memory for current MB is allocated on TCSM0 while its corresponding reconstructed filtered pixel data and left MB right edge are stored on TCSM1. The SRAM is used for storing reconstructed filtered pixels data of top MB bottom edge. Such memory allocation helps in performing all DMAs in parallel to speedup of the data transferring. DMA0 is responsible for transferring current MB data from main memory to on-chip memory TCSM0 for encoding. The reconstructed MB is transferred back from on-chip memories TCSM1 and SRAM to main memory through DMA1 and DMA2, respectively. The data transfer size of DMA0 for an MB is 384 bytes. The DMA1 and DMA2 transfer 288 bytes and 96 bytes of data in each execution, respectively. For an MB encoding, 384 bytes of data is transferred from main memory to on-chip memory and vice versa. The DMAs incorporation in the design speeds up the data transferring during the encoding process.

4.3.3. Algorithmic Optimization Using SIMD Instructions. Zigzag scanning and edge extension algorithms belong to the software part of hybrid encoder because their support is not available in hardware. These algorithms involve multiple memory access and data copying operations that significantly contribute in computational cost. This section describes the optimization of these algorithms using SIMD instructions to speed up the encoding. The SIMD instructions enable multiple arithmetic or logic operations to be executed simultaneously. Ingenic processor JZ4770 provides 60 SIMD

Table 1: % gain through optimization techniques.

Encoder modules	% Gain
Data transfer	27%
DBLK design	32%
Edge extension and Zigzag	35%

instructions for the multimedia codec optimization [24]. These instructions operate on 32 bits registers.

Zigzag scanning is performed on quantized coefficients before entropy encoding. The 16 bits quantized coefficients are accessed from on-chip memory. In this work, 32 bits load and store operations are used to make quantized coefficients loading and storing faster. The SIMD instructions are also used to arrange two coefficients simultaneously instead of one. Algorithm 1(a) shows the Zigzag scanning algorithm optimization using SIMD instructions (S32LDD, S32SFL, and S32STD), in which quantized coefficients are loaded from on-chip memory to dedicated MXU registers (xr1-xr8) using S32LDD instruction. After that, S32SFL is performed on these registers in pairs to adjust the positions of coefficients according to Zigzag order. Then, final reordered coefficients are stored to main memory from dedicated MXU registers (xr1, xr10, xr5, xr7, xr6, xr4, xr9, and xr8) using S32STD instruction. The S32SFL (r1, r2, r3, r4, 3) is packing instruction. This instruction packs the two most-significant halfwords (16 bits) from the arguments r2 and r3 into r1. The halfword from r2 is packed into the most-significant halfword of r1; the halfword from r3 is packed into the least-significant halfword of r1. Similarly, it also packs two least-significant halfwords from the arguments r2 and r3 into r4.

For performing motion estimation, the reference frame is padded all around. This edge extension may be several pixels wide, depending on the motion estimation mode. The upper and lower edges are generated by copying the first and last rows of the frame into the border or edges. Similarly, left and right edges are generated by copying the first and last columns of the frame into the border. In this work, 32 bits loading and storing operations are used instead of 8 bits operations. The use of SIMD instructions helps to extend four pixels in parallel. Algorithm 1(b) shows the top edge extension using SIMD instructions (S32LDD and S32STD).

Table 1 lists the achieved speedup gain through each optimization technique.

4.4. Encoder Pipeline Design. Considering the data dependency analysis made in Section 4.2, it is observed that the heterogeneous structure of the H.264/AVC encoder has inherent sequential modular data dependency; that is, output of one module becomes input of its subsequent module. Because of that modular dependency, one hardware processing unit is used at a time and all of the other units remain idle. Moreover, the polling of hardware processing units and DMAs take significant part of encoding time. These factors not only decrease the throughput of hardware processing unit but also have adverse effect on encoder performance. For efficient utilization of all hardware processing units and reducing polling time, a pipeline design for H.264 encoder has been proposed after exploiting fine grained parallelism at MB level. Exploiting MB level parallelism requires satisfying the data dependencies between MBs. For a given MB, intraprediction, interprediction, deblock filtering, and entropy encoding need data from neighboring (left, top, top left, and top right) MBs. This data includes pixel data, prediction mode, motion vectors, and number of nonzero coefficients. The simplest way to resolve all these dependencies is to process MBs in scan order.

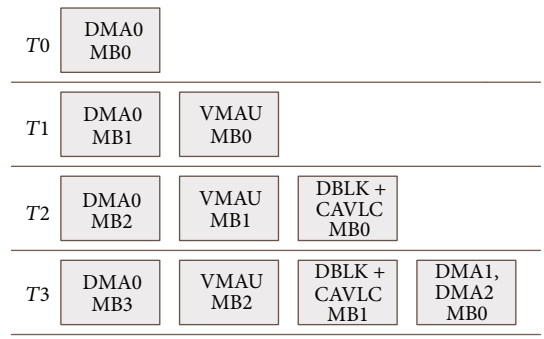

Figure 6: Four-staged pipeline for intra-MBs.

In proposed design, an MB encoding process is divided into several stages and each stage is executed on a separate processing core. After that these processing cores are pipelined at MB level through which multiple MBs are processed simultaneously in scan order. For processing intra- and inter-MB, a pipeline design is described below.

Intra-MBs. The encoding task for intra-MB is performed in four major pipeline stages. These stages are DMA0 stage (it transfers raw MB data from main memory to on-chip memory), VMAU stage, DBLK + CAVLC stage (this stage performs deblock filtering and entropy encoding in a same time slice), and DMA1 stage (it transfers filtered reconstructed MB data from on-chip memory to main memory). Each intra-MB goes through the final DMA1 stage after four-MB cycle time latency. This four-stage pipeline for intramode is shown in Figure 6.

Inter-MBs. The encoding task for inter-MB is performed in five major pipeline stages. These stages are DMA0 stage, MCE stage (it performs motion estimation), VMAU stage, DBLK + CAVLC stage, and DMA1 stage. The MB0 goes through the DMA1 stage after five-MB cycle time latency. This five-stage pipeline for intermode is shown in Figure 7.

Algorithm 2 presents the pseudocodes of proposed encoder pipelined design, in which hardware modules for processing current MB are configured before polling them for previous MB. The usage of hardware modules in this way reduces the polling time.

To show the advantage of the proposed pipelined design, profiling analysis is carried out before and after exploiting the MB level parallelism. The time taken by each module in hybrid encoder before pipelined design is shown in Figure 8. Table 2 shows that the polling and configuration of hardware video processing units take 51% and 20% of total processing time, respectively.

```
(a)
// Optimized Implementation for Zig-Zag Scanning
 {
/*
  For Input
  1 2 3 4
  5 6 7 8
  9 10 11 12
  13 14 15 16
 Output after Zig-Zag scannig is
 1 2   5 9   6 3   4 7   10 13   14 11   8 12   15 16
 */
 /* load quantized coefficients to MXU registers */
 S32LDD(xr1, dct, 0x0); // xr1 = 2 1
 S32LDD(xr2, dct, 0x4); // xr2 = 4 3
 S32LDD(xr3, dct, 0x8); // xr3 = 6 5
 S32LDD(xr4, dct, 0xc); // xr4 = 8 7
 S32LDD(xr5, dct, 0x10); // xr5 = 10 9
 S32LDD(xr6, dct, 0x14); // xr6 = 12 11
 S32LDD(xr7, dct, 0x18); // xr7 = 14 13
 S32LDD(xr8, dct, 0x1c); // xr8 = 16 15
 /* adjust positions of coefficients according to Zig-Zag order */
 S32SFL(xr9, xr6, xr4, xr10, 3); // xr6 = 12 11   xr4 = 8 7   xr9 = 12 8   xr10 = 11 7
 S32SFL(xr4, xr10, xr7, xr6, 3); // xr10 = 11 7   xr7 = 14 13   xr4 = 11 14   xr6 = 7 13
 S32SFL(xr7, xr6, xr2, xr10, 3); // xr6 = 7 13   xr2 = 4 3   xr7 = 7 4   xr10 = 13 3
 S32SFL(xr6, xr10, xr5, xr2, 3); // xr10 = 13 3   xr5 = 10 9   xr6 = 13 10   xr2 = 3 9
 S32SFL(xr5, xr2, xr3, xr10, 3); // xr2 = 3 9   xr6 = 6 5   xr5 = 3 6   xr10 = 9 5
 /* reordered coefficients are stored to main memory from MXU registers */
 S32STD(xr1, level, 0x0); // xr1 = 2 1
 S32STD(xr10, level, 0x4); // xr10 = 9 5
 S32STD(xr5, level, 0x8); // xr5 = 3 6
 S32STD(xr7, level, 0xc); // xr7 = 7 4
 S32STD(xr6, level, 0x10); // xr6 = 13 10
 S32STD(xr4, level, 0x14); // xr4 = 11 14
 S32STD(xr9, level, 0x18); // xr9 = 12 8
 S32STD(xr8, level, 0x1c); // xr8 = 16 15
(b)
 // Optimized Implementation for top edge extension of luma component
 for (j = 0; j < (Lwidth/16); j++)
 {
    S32LDD(xr1, ExtndDataUP_Inp, 0x0); // loading data
    S32LDD(xr2, ExtndDataUP_Inp, 0x4);
    S32LDD(xr3, ExtndDataUP_Inp, 0x8);
    S32LDD(xr4, ExtndDataUP_Inp, 0xc);
    for (i = 0; i < 16; i++)
    {
       S32STD(xr1, ExtndDataUP_Out, 0x0); // edge extension
       S32STD(xr2, ExtndDataUP_Out, 0x4);
       S32STD(xr3, ExtndDataUP_Out, 0x8);
       S32STD(xr4, ExtndDataUP_Out, 0xc);
       ExtndDataUP_Out+ = 4;
    }
    ExtndDataUP_Inp+ = 64;
 }
```

Algorithm 1: (a) SIMD implementation of Zigzag scanning and (b) SIMD implementation of edge extension.

```
while(mbno < total_mbs+n)
 {    // where n = no. of pipeline stages - 1
// DMA0 Stage
   If(mbno < total_mbs)
      Configure DAM0 ( ) // for current MB
      If(mbno > 0)
           Poll DMA0( ) // for previous MB
      End If
      exeute DMA0( ) // for current MB
      If(last MB)
        Poll DMA0( ) // for current MB
      End If
   End If
// MCE Stage
   If(slice is P_TYPE)
     mbno_mce = mbno-1;
     If(mbno_mce >= 0 && mbno_mce < total_mbs)
       If(mbno_mce > 0)
         If(current MB is inter type)
           Configure MCE( ) // for current MB
         End If
         If(prvious MB is inter type)
           poll MCE( ) // for previous MB
         End If
       End If
       If(current MB is inter type)
         exeute MCE( ) // for current MB
         If(last MB)
           Poll MCE( ) // for current MB
         End If
       End If
     End If
   End If
// VMAU Stage
   mbno_vmau = mbno-2;
   If(mbno_vmau >= 0 && mbno_vmau < total_mbs)
      Configur VMAU( ) // for current MB
      If(mbno_vmau > 0)
        poll VMAU( ) // for previous MB
      End If
      exeute VMAU( ) // for current MB
      If(last MB)
        poll VMAU( ) // for current MB
      End If
   End If
// DBLK Stage
   mbno_dblk = mbno-3;
   If(mbno_dblk >= 0 && mbno_dblk < total_mbs)
      Configur DBLK( ) // for current MB
      If(mbno_dblk > 0)
           Poll DBLK( ) // for previous MB
      End If
      exeute DBLK( ) // for current MB
      Entropy encoding( ) // for current MB
      If(last MB)
        poll DBLK( ) // for current MB
      Endif
   End If
```

Algorithm 2: Continued.

```
// DMA12 Stage
   mbno_dma12 = mbno-4;
   If(mbno_dma12 >= 0 && mbno_dma12 < total_mbs)
      Configur DMA12( ) // for current MB
      If(mbno_dma12 > 0)
         Poll DMA12( ) // for previous MB
      End If
      exeute DMA12( ) // for current MB
      If(last MB)
         poll DMA12( ) // for current MB
      End If
   End If
   mbno++;
}
```

Algorithm 2: Pseudocode of proposed encoder pipelined design.

Table 2: % of total processing time taken by HW processing units before pipelined design.

HW processing units	Configuration time	Polling time	Total time
VMAU	2%	12%	14%
MCE	1%	13%	14%
DBLK	7%	12%	19%
DMA0	3%	7%	10%
DMA1, DMA2	7%	7%	14%
Total	20%	51%	71%

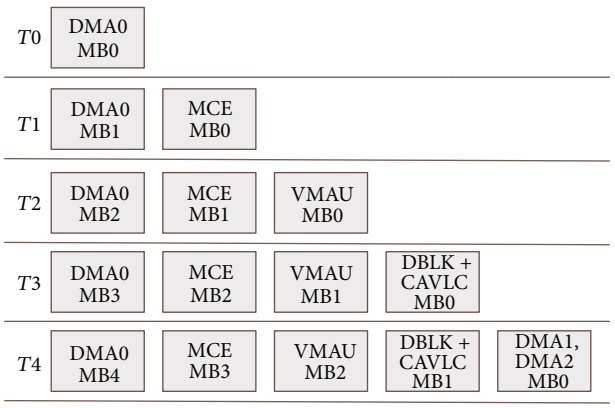

Figure 7: Five-staged pipeline for inter-MBs.

After pipelined design, the distribution of processing time among different encoding blocks is shown in Figure 9. Table 3 shows that polling time reduces from 51% to 6%.

4.5. Encoder Partitioning between CPU and VPU Cores. Even though the pipeline design significantly shortens the encoding time, the encoder overall performance still heavily depends on distribution of encoding tasks between CPU and VPU. The profiling analysis made after pipeline design demonstrates that hardware modules configuration and processing take 40%, while CAVLC and Zigzag scanning consume 47% of the overall processing time. So the potential advantage is to distribute these computationally demanding modules among CPU and VPU cores. As in proposed pipelined design, entropy encoding (CAVLC and Zigzag) and deblock filtering (DBLK) modules both are at the same stage. There is no data dependency between them and they can be parallelized. In order to take the advantage of this parallelism, encoder tasks are distributed between CPU and VPU cores.

Figure 10 shows the distribution of encoder between CPU and VPU cores. This distribution is made by considering

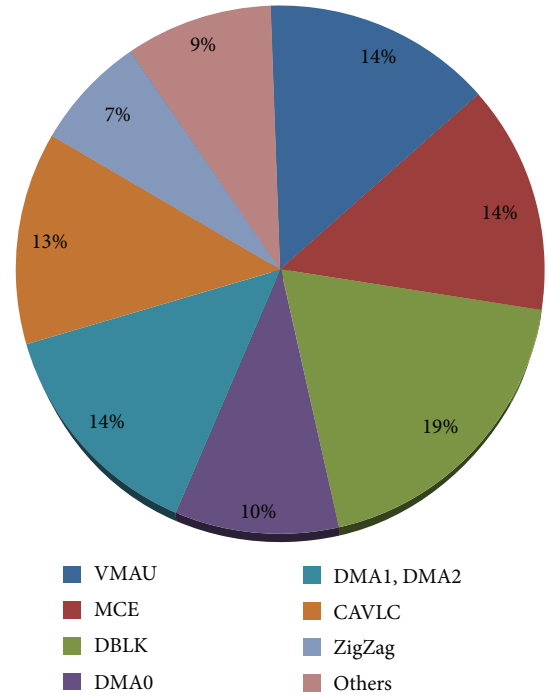

Figure 8: Time breakdown before pipeline.

Table 3: % of total processing time taken by HW processing units after pipelined design.

HW processing units	Configuration time	Polling time	Total time
VMAU	9%	1%	10%
MCE	3%	2%	5%
DBLK	12%	1%	13%
DMA0	3%	1%	4%
DMA1, DMA2	7%	1%	8%
Total	34%	6%	40%

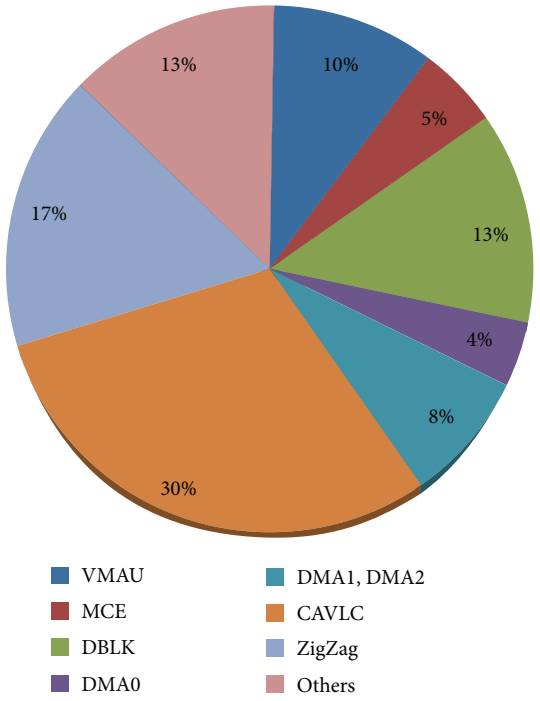

Figure 9: Time breakdown after pipeline.

the parallel processing, load balancing between the cores, and scalability. The tasks of encoding performed by CPU core are the initialization of the encoder and hardware processing units, video capturing, frame level parameter setting and controlling, Zigzag scanning, motion vector prediction, entropy encoding, bit rate control, edge extension, and file writing. The VPU core is responsible for configuration and controlling of hardware modules, MB level scheduling of hardware processing units, encoding parameters selection, and data transferring between main and on-chip memories. The encoder tasks distribution between CPU and VPU resulted in efficient utilization of all parallelization capabilities of this platform for high-performance video encoding.

4.6. Scheduling Strategy and Encoder Flow. In the proposed design, CPU and VPU cores are working concurrently and they are pipelined at MB level. They used on-chip memories for exchanging data and/or messages. Ten buffers are

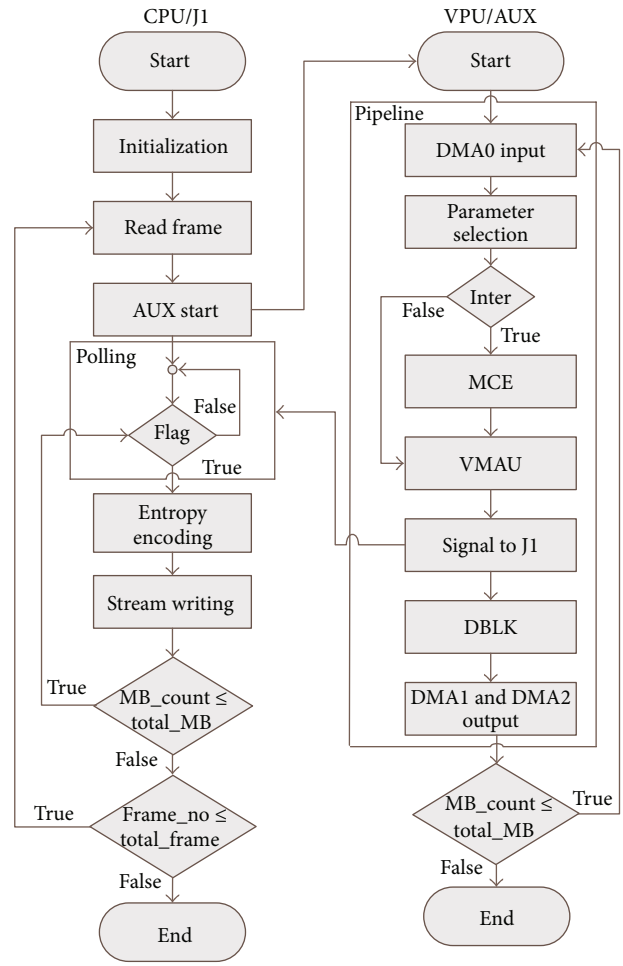

Figure 10: Proposed encoder flowchart.

allocated and used in a ping pong manner for sending and receiving frame and bitstream data between the cores. Figure 10 shows the flowchart of the proposed H.264 encoder. The encoding process is described as follows.

(1) The raw data of the frame is read and preprocessed by CPU core.

(2) After CPU sets the frame level parameters, it starts the VPU and waits for a completion signal.

(3) The VPU core processes MBs using hardware processing units as described in Section 4.4.

(4) After completion, the VPU signals CPU and sends MB parameter structures. This structure contains motion vectors, residual quantized coefficients, CBP, and other parameters required to encode.

(5) The CPU performs entropy encoding, writes bit stream, control the rate, and receive data structure from VPU.

(6) The VPU core stops after all MBs in a frame have been processed.

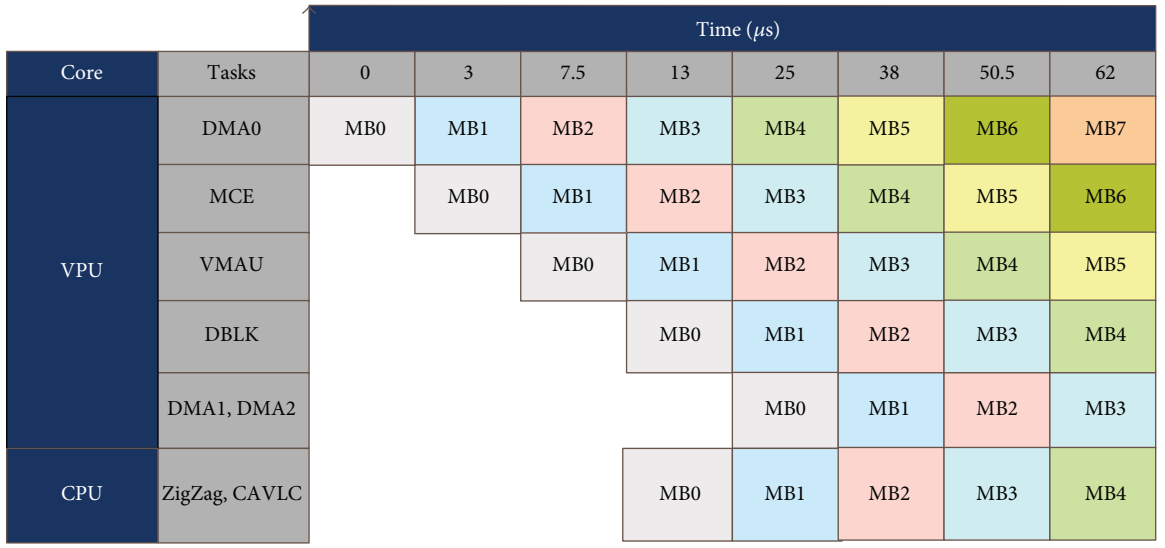

Figure 11: Timeline for multicore pipelined encoder.

(7) The above procedure continues until the encoding of all candidate frames.

The timeline of proposed multicore pipelined encoder is shown in Figure 11. It clearly demonstrates the exploitation of fine-grained MB level parallelism and all processing capabilities of JZ4770 platform. When pipeline is full, five MBs are processed in parallel.

It is important to note that the flexible design of the proposed scheme allows the integration of any fast mode selection and motion estimation algorithms. The motion estimator can be easily integrated at the place of MCE module and mode selection algorithm can be incorporated in the parameter selection module. Moreover, in case of CABAC entropy encoding, CABAC takes the place of CAVLC.

4.7. Comparison of the Proposed Architecture with Recent Schemes. Sankaraiah's [10] method is based on Group-Of-Pictures (GOP) in which each GOP is encoded independently in a separate thread and the frames being referenced are included in the GOP. This method required a lot of memory for storing all the frames, and therefore this technique maps well to multicore architectures that fall into general-purpose processors (GPP) categories. In addition, parallelization at the GOP-level results in a very high latency that cannot be tolerated in some applications. Therefore, this scheme is not well suited for multicore architectures, in which the memory is shared by all the processors, because of cache pollution.

Alvanos et al. [9] divide the MB encoding process in three phases including analyzing and encoding, entropy encoding, and deblocking. After that, each phase is executed as a separate task. In order to resolve the dependency, Alvanos made use of 2D-wavefront parallelism technique [25] in which all macroblocks are issued in an antidiagonal based manner and wait before issuing the next antidiagonal. However, most of the time consuming tasks analysis and encoding are assigned to the main Synergistic Processing Element (SPE). Therefore, the scheme cannot be implemented for real-time processing on low-power CPUs like one used in this study.

Momcilovic [8] approach simultaneously distributes the motion estimation and interpolation modules on CPU and GPU. While rest of the encoder modules are processed sequentially on either CPU or GPU, this scheme requires very high power CPUs and GPUs so it is not an embedded solution.

5. Performance Evaluation and Discussion

To evaluate the performance of the proposed scheme, two different frame resolutions, NTSC (720 × 480) and 720p HD (1280 × 720), are considered. All test sequences are in 4 : 2 : 0 format and have 300 frames each. The NTSC resolution test sequences are mobile, football, intros, garden, galleon, vtc1nw, and washdc. Stockholm, Shields, Parkrun, and Mobcal are considered HD 720p resolution test sequences. The video coding parameters are set as follows: MV search range is −32 to +32 pels and block size is 16 × 16, RD optimization is enabled with metric sum of absolute difference (SAD), rate control method is single pass constant quantizer (QP Mode), one reference frame is for motion estimation and compensation, motion estimation scheme is diamond search, MV resolution is 1/4 pel, and GOP is chosen as 25 with structure IPPP. The RD optimization technique improves the RD performance but decreases the encoding speed. The use of computationally intensive Psychovisual Optimization metric improves the RD performance and reduces the encoding speed. On the other hand, simpler metrics like SAD and sum of absolute transform difference (SATD) increase the encoding speed at the cost of degradation in RD performance.

5.1. Encoder Performance at Different Implementation Stages. To demonstrate the advantages of proposed scheme, results are computed at three separate stages. The first stage is based on the x264 sequential software implementation [26]

Table 4: NTSC results at different stages (Qp = 24).

Sequences	Stage 1 (fps)	Stage 2 (fps)	Stage 3 (fps)
Garden	3.73	18.75	44.50
Football	3.65	19.74	53.86
Intros	4.02	24.90	64.62
Mobile	3.55	18.96	48.78
Galleon	4.89	24.50	65.99
Vtc1nw	6.70	28.27	69.38
Washdc	4.79	26.17	63.80
Average	4.48	23.04	58.70

Table 5: 720p results at different stages (QP = 26).

Sequences	Stage 1 (fps)	Stage 2 (fps)	Stage 3 (fps)
Stockholm	1.53	9.30	28.73
Shields	1.47	8.85	27.68
Parkrun	1.33	7.76	18.99
Mobcal	1.52	8.32	25.91
Average	1.46	8.55	25.32

executing on CPU core. The second stage belongs to Section 4.1 in which video processing units are incorporated. Finally, stage three introduces the numerous optimization methods, pipeline design, and parallel processing. Tables 4 and 5 depict the results at different implementation stages for NTSC and 720p resolutions, respectively.

The average encoding rate for NTSC resolution at first stage is 4.48 fps. After the second stage, the encoding rate increased to 23 fps. Finally, the average encoding rate is increased up to 58.70 fps after the third stage. The results demonstrate that the addition of hardware processing units increases the encoding rate from 4.48 fps to 23 fps and the proposed optimization techniques increase the encoding rate from 23 fps to 58.70 fps. Similarly, the encoding rate for 720p resolution is increased from 1.46 fps to 25.32 fps. In this case, encoding rate is increased from 1.46 fps to 8.55 fps due to addition of hardware processing units and from 8.55 fps to 25.32 fps because of the proposed optimization techniques. It can be concluded that although the use of hardware processing units increased the encoding rate, major advantage is due to their efficient utilization through proposed optimization methodologies.

Table 6 shows the impact of each optimization method on the total encoding speedup. It demonstrates that the encoder partitioning between CPU and VPU cores contributes most significantly, that is, 45.71%, to speed up the encoding process. The reasons behind such significant encoding gain are parallel processing and reduction of hardware configuration time. As in the proposed pipeline design, entropy encoding and deblock filtering (DBLK) modules both are at the same pipeline stage. This stage takes most of the encoding time and becomes a bottleneck for the rest of the pipeline stages. After encoder partitioning, these two modules are parallelized at MB level and overall encoding load is distributed between two processing cores. This resulted in speedup of the encoding process. Moreover, all hardware processing units are shifted to VPU core that can configure them using physical address of on-chip memories directly. Such configuration of the hardware processing units reduced the overall encoding time.

Table 6: Optimization method versus contribution to encoding speedup.

Optimization method	Speedup
Memory management and data transfer	20%
Encoder pipeline design	34.29%
Encoder partitioning between CPU and VPU cores	45.71%

5.2. Comparison with Sequential (CPU Only) Implementation. A group of experiments is carried out on these test sequences with four quantization parameters, that is, QP = 24, 28, 32, and 36. The performance comparison of the implemented encoder with highly optimized sequential x264 encoder [26] is presented in Table 7 for NTSC test sequences. Table 7 illustrates that the proposed scheme shows a consistent gain in encoding rate for all sequences ranging from 58.70 fps at QP = 24 to 72.10 fps at QP = 36.

For HD 720p resolution test sequences, the quantization parameters set are 26, 30 and 34. Table 8 shows the frame rate comparison of the proposed encoder with sequential software implementation [26]. For most of the 720p test sequences, real-time encoding is achieved. The average increase in encoding rate is ranging from 25.32 fps at QP = 26 to 32.04 fps at QP = 34.

The experimental results demonstrate that, for both resolutions NTSC and 720p, the proposed scheme achieved a speedup of more than 12x when compared with highly optimized sequential implementation on CPU core without loss in PSNR.

Although the proposed solution is based on x264 baseline profile, it can also be used directly for main or high profile with B frames and CABAC entropy coding by replacing CAVLC module with CABAC implemented in software and assigned to the CPU. However, the CABAC implementation may result in increase of encoding time. For HD 720p and higher resolutions, video sequences real-time encoding may not be possible.

5.3. Comparison with State-of-the-Art Implementation Schemes. Table 9 compares the proposed scheme with previous works in terms of speedup gain. The comparison of the proposed scheme is done with three preceding works, Momcilovic et al. [8], Alvanos et al. [9], and Sankaraiah et al. [10]. It can be seen in Table 9 that the proposed scheme outperformed the other techniques in terms of speedup gain.

5.4. Energy Saved. Energy saving is another important metric to evaluate the performance of any technique for mobile devices. As the energy consumption is one of the most critical issues in case of mobile application processors, the implementation scheme must cater for this important issue. The energy consumption for sequential (CPU only) and proposed implementation is computed as follows: applied

Table 7: Performance comparison for NTSC resolutions.

QP	24		28		32		36	
Sequence	Sequence fps	Proposed fps	Sequence fps	Proposed fps	Sequence fps	Proposed fps	Sequence fps	Proposed fps
Garden	3.73	44.50	4.04	58.48	4.31	61.56	4.59	70.85
Football	3.65	53.86	4.00	54.38	4.41	59.37	4.78	71.75
Intros	4.02	64.62	5.47	67.82	6.34	70.73	6.72	72.23
Mobile	3.55	48.78	3.74	58.36	3.96	63.50	4.33	70.62
Galleon	4.89	65.99	5.51	70.15	6.24	72.29	7.01	74.63
Vtc1nw	6.70	69.38	7.42	71.11	7.75	72.50	7.97	74.22
Washdc	4.79	63.80	5.75	66.57	6.72	68.07	7.38	70.40
Average	4.48	58.70	5.13	63.83	5.68	66.86	6.11	72.10

Table 8: Performance comparison for 720p resolutions.

Qp	26		30		34	
Sequence	Sequential fps	Proposed fps	Sequential fps	Proposed fps	Sequential fps	Proposed fps
Stockholm	1.53	28.73	1.87	30.42	2.32	32.41
Shields	1.47	27.68	1.80	30.17	2.17	33.77
Parkrun	1.33	18.99	1.44	23.78	1.66	28.34
Mobcal	1.52	25.91	1.79	31.65	2.17	33.67
Average	1.46	25.32	1.72	29.00	2.08	32.04

Table 9: Performance comparison with other techniques.

Techniques	Speedup gain
Momcilovic et al. [8]	2.5x
Alvanos et al. [9]	4.7x–8.6x
Sankaraiah et al. [10]	5.6x–10x
Proposed	More than 12x

Table 10: Power consumption comparison.

Implementation	Average Instantaneous current mA	Instantaneous power consumed W
Sequential	55	0.28
Proposed	62	0.31

voltage is 5 volt, steady state current (standby mode) is 180 mA, QP is 24, and the number of iterations is 5. Average system current in the case of sequential and the proposed implementation is 235 mA and 242 mA, respectively.

The power consumption comparison of proposed implementation with sequential implementation is given in Table 10. Although instantaneous power consumed by the proposed scheme is higher, the overall power consumed to encode 300 frames is much lower than the sequential software implementation. Tables 11 and 12 provide the comparison of energy consumed to encode 300 frames of NTSC and 720p resolutions, respectively. The proposed technique saves about 4.90 mWh and 58.28 mWh energy to encode 300 frames of NTSC and 720p resolutions, respectively.

Table 11: Energy consumption comparison (NTSC).

Sequences	Sequential mWh	Proposed mWh	Energy saved mWh
Garden	6.14	0.58	5.56
Football	6.28	0.48	5.80
Intros	5.70	0.40	5.30
Mobile	6.45	0.53	5.92
Galleon	4.68	0.39	4.30
Vtc1nw	3.42	0.37	3.05
Washdc	4.78	0.40	4.38
Average	5.35	0.45	4.90

Table 12: Energy consumption comparison (720p).

Sequences	Sequential mWh	Proposed mWh	Energy saved mWh
Stockholm	61.56	3.36	58.21
Shields	62.50	3.54	58.96
Parkrun	64.45	5.67	58.78
Mobcal	60.66	4.04	56.62
Average	62.26	3.99	58.28

5.5. Scalability of the Proposed Scheme for High Efficiency Video Coding (HEVC). The proposed scheme is scalable and

it can be easily modified for High Efficiency Video Coding (HEVC) standard. HEVC retains the basic hybrid coding architecture of H.264/AVC. Moreover, the basic tool set of video encoding in HEVC codec is quiet similar to H.264/AVC coding standard, for example, intra- and interprediction, motion estimation and compensation, transformation, quantization, entropy encoding, and deblocking filtering. However, HEVC uses more adaptive quadtree structure based on a coding tree unit (CTU) instead of a macroblock (MB) and several improvements have been made in tool set to improve its compression performance and throughput speed (particularly for parallel-processing architectures).

For HEVC, the basic encoding flow and building blocks of the proposed scheme remain the same as shown in Figure 10. However, the following modifications are required inside the building blocks.

(1) In HEVC, the basic processing unit is CTU instead of MB. So the main encoding loop will be over CTUs.

(2) Both the processing cores (CPU and VPU) and hardware processing units will be pipelined at CTU level.

(3) The DBLK stage in proposed design will perform both deblocking and sample-adaptive offset (SAO) filtering.

(4) In HEVC, CAVLC will be replaced by CABAC.

(5) Memory requirement will increase because of large size of metadata structures in HEVC.

6. Conclusion

This paper proposes an end-to-end software-hardware hybrid implementation scheme of H.264/AVC encoder for JZ47x series of processors. The proposed scheme not only distributes the encoding tasks between hardware and software modules but also performs synchronization between different cores and the hardware accelerator blocks. The idling and polling time of hardware processing units is also reduced by exploiting fine grained parallelism at MB level. The possibility of asynchronously processing on the CPU and VPU is also effectively exploited to efficiently distribute the computational load among these processing cores. This resulted in increasing the encoding rate and reducing power consumption. The implemented encoder can encode NTSC and HD 720p video sequences at 72 fps and 32 fps, respectively. This is approximately more than 12 times the encoding rate of the highly optimized sequential encoder. The performance improvements techniques demonstrated in this work can also be applied to other families of processors. It may be noted that the proposed scheme is scalable and it can be easily modified for High Efficiency Video Coding (HEVC) standard.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] I. Werda, T. Dammak, T. Grandpierre, M. A. B. Ayed, and N. Masmoudi, "Real-time H.264/AVC baseline decoder implementation on TMS320C6416," *Journal of Real-Time Image Processing*, vol. 7, no. 4, pp. 215–232, 2012.

[2] K. Babionitakis, G. Doumenis, G. Georgakarakos et al., "A real-time H.264/AVC VLSI encoder architecture," *Journal of Real-Time Image Processing*, vol. 3, no. 1-2, pp. 43–59, 2008.

[3] Z. Wei, K. L. Tang, and K. N. Ngan, "Implementation of H.264 on mobile device," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 3, pp. 1109–1116, 2007.

[4] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 688–703, 2003.

[5] Q. Tang and P. Nasiopoulos, "Efficient motion re-estimation with rate-distortion optimization for MPEG-2 to H.264/AVC transcoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 2, pp. 262–274, 2010.

[6] B. La, M. Eom, and Y. Choe, "Dominant edge direction based fast intra mode decision in the H.264/AVC encoder," *Journal of Zhejiang University: Science A*, vol. 10, no. 6, pp. 767–777, 2009.

[7] M. Asif, M. Farooq, and I. A. Taj, "Optimized implementation of motion compensation for H.264 decoder," in *Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT '10)*, pp. 216–221, IEEE, Seoul, Republic of Korea, December 2010.

[8] S. Momcilovic, N. Roma, and L. Sousa, "Exploiting task and data parallelism for advanced video coding on hybrid CPU + GPU platforms," *Journal of Real-Time Image Processing*, pp. 1–17, 2013.

[9] M. Alvanos, G. Tzenakis, D. S. Nikolopoulos, and A. Bilas, "Task-based parallel H.264 video encoding for explicit communication architectures," in *Proceedings of the 11th International Conference on Embedded Computer Systems (SAMOS '11)*, pp. 217–224, IEEE, Samos, Greece, July 2011.

[10] S. Sankaraiah, H. S. Lam, C. Eswaran, and J. Abdullah, "Gop level parallelism on h.264 video encoder for multicore architecture," in *Proceedings of the International Conference on Circuits, System and Simulation (ICCSS '11)*, vol. 7, pp. 127–133, IPCSIT, May 2011.

[11] H. K. Nguyen, P. Cao, X.-X. Wang et al., "Hardware software co-design of H.264 baseline encoder on coarse-grained dynamically reconfigurable computing system-on-chip," *IEICE Transactions on Information and Systems*, vol. E96-D, no. 3, pp. 601–615, 2013.

[12] M. Asif, S. Majeed, I. A. Taj, M. Bin Ahmed, and S. M. Ziauddin, "Exploiting MB level parallelism in H.264/AVC encoder for multi-core platform," in *Proceedings of the IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA '14)*, pp. 125–130, Doha, Qatar, November 2014.

[13] K. S. Aw, S. M. Goh, K. H. Goh, T. K. Chiew, and J. Y. Tham, "Real-time HD video encoding on DSP," in *Proceedings of the 5th IEEE Conference on Industrial Electronics and Applications (ICIEA '10)*, pp. 1992–1995, IEEE, Taichung, Taiwan, June 2010.

[14] D.-T. Lin and C.-Y. Yang, "H.264/AVC video encoder realization and acceleration on TI DM642 DSP," in *Advances in Image and Video Technology: Third Pacific Rim Symposium, PSIVT 2009, Tokyo, Japan, January 13–16, 2009. Proceedings*, vol. 5414 of *Lecture Notes in Computer Science*, pp. 910–920, Springer, Berlin, Germany, 2009.

[15] D. Schneider, M. Jeub, Z. Jun, and S. Li, "Advanced H.264/AVC encoder optimizations on a TMS320DM642 digital signal processor," in *Proceedings of the 16th International Conference on Digital Signal Processing (DSP '09)*, pp. 1–14, July 2009.

[16] X. He, X. Fang, C. Wang, and S. Goto, "Parallel HD encoding on CELL," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '09)*, pp. 1065–1068, IEEE, Taipei, Taiwan, May 2009.

[17] C.-T. Lu and H.-M. Hang, "Multiview encoder parallelized fast search realization on NVIDIA CUDA," in *Proceedings of the IEEE Visual Communications and Image Processing (VCIP '11)*, pp. 1–4, Tainan, Taiwan, November 2011.

[18] M. Schwalb, R. Ewerth, and B. Freisleben, "Fast motion estimation on graphics hardware for h.264 video encoding," *IEEE Transactions on Multimedia*, vol. 11, no. 1, pp. 1–10, 2009.

[19] N.-M. Cheung, X. Fan, O. Au, and M.-C. Kung, "Video coding on multicore graphics processors," *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 79–89, 2010.

[20] A. Azevedo, B. Juurlink, C. Meenderinck et al., "A highly scalable parallel implementation of H.264," *Transactions on High-Performance Embedded Architectures and Compilers (HiPEAC)*, vol. 4, no. 2, pp. 111–134, 2011.

[21] S. Momcilovic, N. Roma, and L. Sousa, "Multi-level parallelization of advanced video coding on hybrid CPU/GPU platform," in *Proceedings of the 10th International Workshop on Algorithms*, pp. 165–174, Rhodes Islands, Greece, August 2012.

[22] Ingenic Semiconductor, *JZ4770 VPU Programming Manual*, Ingenic Semiconductor, Beijing, China, 2011.

[23] Ingenic Semiconductor Co, *JZ4770 Data Sheet*, Ingenic Semiconductor Co, 2011.

[24] Ingenic Semiconductor, *MXU Instruction Usage Guide*, Ingenic Semiconductor, Beijing, China, 2011.

[25] E. B. Van Der Tol, E. G. T. Jaspers, and R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," in *Image and Video Communications and Processing*, vol. 5022 of *Proceedings of SPIE*, pp. 707–718, Santa Clara, Calif, USA, January 2003.

[26] x264, http://www.videolan.org/developers/x264.html.