IEICE TRANS. INF. & SYST., VOL.E95–D, NO.7 JULY 2012

PAPER

Reconfiguration Process Optimization of Dynamically Coarse Grain Reconfigurable Architecture for Multimedia Applications

Bo LIU[†], **Peng CAO**[†a)], ***Members***, **Min ZHU**[††], **Jun YANG**[†], **Leibo LIU**[††], **Shaojun WEI**[††], ***and*** **Longxing SHI**[†], ***Nonmembers***

SUMMARY This paper presents a novel architecture design to optimize the reconfiguration process of a coarse-grained reconfigurable architecture (CGRA) called Reconfigurable Multimedia System II (REMUS-II). In REMUS-II, the tasks in multi-media applications are divided into two parts: computing-intensive tasks and control-intensive tasks. Two Reconfigurable Processor Units (RPUs) for accelerating computing-intensive tasks and a Micro-Processor Unit (μPU) for accelerating control-intensive tasks are contained in REMUS-II. As a large-scale CGRA, REMUS-II can provide satisfying solutions in terms of both efficiency and flexibility. This feature makes REMUS-II well-suited for video processing, where higher flexibility requirements are posed and a lot of computation tasks are involved. To meet the high requirement of the dynamic reconfiguration performance for multimedia applications, the reconfiguration architecture of REMUS-II should be well designed. To optimize the reconfiguration architecture of REMUS-II, a hierarchical configuration storage structure and a 3-stage reconfiguration processing structure are proposed. Furthermore, several optimization methods for configuration reusing are also introduced, to further improve the performance of reconfiguration process. The optimization methods include two aspects: the multi-target reconfiguration method and the configuration caching strategies. Experimental results showed that, with the reconfiguration architecture proposed, the performance of reconfiguration process will be improved by 4 times. Based on RTL simulation, REMUS-II can support the 1080p@32 fps of H.264 HiP@Level4 and 1080p@40 fps High-level MPEG-2 stream decoding at the clock frequency of 200 MHz. The proposed REMUS-II system has been implemented on a TSMC 65 nm process. The die size is 23.7 mm^2 and the estimated on-chip dynamic power is 620 mW.

key words: *REMUS-II, coarse grain reconfigurable architecture, reconfiguration process, multimedia application*

1. Introduction

The existing computing architectures for multimedia applications can be generally classified into three types: general purpose processors (GPPs), application specific integrated circuits (ASICs) and reconfigurable architectures (RAs). Although solutions with GPPs, such as the ARM-like processors and the digital signal processors (DSPs), provide the most flexibility, they cannot fulfill the ever increasing requirements on performance, cost and power consumption for multimedia applications because of their sequential software execution. ASICs can provide best performance for specific applications, but it is not desirable for multimedia processing designs. The diversification and continuous evolution of multimedia standards and the seamless transition between various media processing algorithms pose higher flexibility requirements. However, ASICs always have fixed and limited functions performed by predefined modules, and that makes them infeasible to suit new design requirements or changes in standards. Besides, the long time-to-market delay and the rapidly growing non-recursive engineering (NRE) cost also make ASICs inflexible for multimedia processing solutions. Yet reconfigurable processors can provide satisfying solutions in terms of both efficiency and flexibility [1].

According to the granularity of the processing elements, RAs are divided into two groups: fine-grained RAs and coarse-grained RAs. In fine-grained RAs, such as FPGAs, the functionalities of the hardware are specified at bit-level. Therefore, fine-grained RAs are not dynamically reconfigurable because of the intractable and costly reconfiguration process, and that makes the fine-grained RAs not useful for implementing real-time multimedia applications. In contrast to fine-grained RAs, coarse-grained RAs (CGRAs) use word-length function units such as multipliers and arithmetic logic units. Because coarse granularity can benefit from a considerable reduction of reconfiguration time and configuration memory, the energy efficiency and area efficiency of CGRAs are much greater than those of the fine-grained ones. Therefore, CGRAs are more suitable for multimedia processing algorithms where data computation and transmission are always performed in word-level.

Among all video processing algorithms, MPEG-2 and H.264/AVC, which is also known as MPEG-4 Part 10, are two of the most commonly used video decoding standards. Especially, the new generation of the video coding standard H.264/AVC, which represents the latest evolution of video codecs, provides a much more efficient way in compressing the video with ratio about 50% data size as it used to be in older standard [2]. However, accompanied by the higher compression ratio and much better video quality, the computational complexity is significantly increased, which becomes even intractable for the High Definition (HD) resolution video decoding [3], [4]. Because of the colossal amounts of computing resources required for processing these multimedia applications, the scale of the reconfigurable units contained in CGRAs is always extremely large. Consequently, the reconfiguration process implementation for dynamically feeding such numerous reconfigurable units

Manuscript received December 22, 2011.
Manuscript revised March 15, 2012.

[†]The authors are with National ASIC system Engineering Research Center, Southeast University, Nanjing, China.
[††]The authors are with Institute of Microelectronics, Tsinghua University, Beijing, China.
a) E-mail: caopeng@seu.edu.cn

DOI: 10.1587/transinf.E95.D.1858

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers

with configuration data in time becomes critical, and therefore poses essential requirements for high-efficient reconfiguration process.

In this paper, we present a novel reconfiguration architecture implementation approach for improving the reconfiguring performance of a large-scale CGRA called Reconfigurable Multimedia System II (REMUS-II). In comparison with the original version REMUS-I [5], [6], REMUS-II primarily focuses on improving the performance of the reconfiguration process as well as the data access process. The major work of REMUS-I focuses on the reconfigurable array design. However, the performance evaluation of REMUS-I is based on a high-level simulator called SimREMUS [5], without considering the actual characteristics of reconfiguration process and data access process. If the performance degradation due to the effects of reconfiguration process is not taken into account, the performance evaluation of the entire system is unable to be given precisely. Although the reconfigurable arrays of REMUS-I are able to process the H.264 HiP decoding application in evaluation, the reconfiguration architecture design of REMUS-I cannot meet the high requirement of the dynamic reconfiguration performance for multimedia applications. This paper mainly focuses on the reconfiguration process optimization of REMUS-II. Firstly, we propose a hierarchical configuration storage sub-system as well as a 3-stage pipelined reconfiguration processing structure, to provide well organized reconfiguration structure and guarantee the required reconfiguration performance for multimedia algorithm processing. Besides, we also introduce a multi-target reconfiguration method and several caching strategies to dynamically detect and exploit the temporal locality of the configuration data for reusing purpose. Experimental results show that, with the proposed optimization approach, the performance of reconfiguration process will be improved by 4 times. Based on RTL simulation, REMUS-II can support the 1080p@32 fps of H.264 HiP@Level4 and 1080p@40 fps High-level MPEG-2 stream decoding at the clock frequency of 200 MHz. The results also show that REMUS-II achieves higher processing performance and consumes less power in comparison with ADRES [7], [8] and XPP-III [9]–[12], which are the two most famous state-of-the-art CGRAs. The proposed REMUS-II system has been implemented on a TSMC 65 nm process. The die size is 23.7 mm^2 and the estimated dynamic power is 620 mW.

This paper is organized as follows. Section 2 presents related works in this field. Next, we describe the REMUS-II and its reconfiguration architecture in Sect. 3. In Sect. 4, we present the implementation details of the optimization for improving the reconfiguration process, including the hierarchical configuration storage architecture, the 3-stage pipelined reconfiguration processing structure, and the prefetching and caching strategy for configuration reusing. Finally, implementation results are reported in Sect. 5 and conclusions are presented in Sect. 6.

2. Related Works

From the architectural view, CGRAs can be classified into two groups: 1D-linear-array based architectures and 2D-mesh-array based architectures [13]. The 1D-linear-array based CGRAs, such as RaPiD [14] and Piperench [15], are only efficient for computations that can be organized as a linear flow. Compared with 1D-linear-array based CGRAs, 2D-mesh-array based CGRAs are more suitable for dealing with block-based applications which are very common in multimedia processing. Among published 2D-mesh-array based CGRAs which can realize the H.264 decoding application, ADRES [7], [8], XPP-III [9]–[12], ReMAP [16], MORA [17] and MorphoSys [18] are the typical representatives. However, ReMAP, MORA and MorphoSys mainly focused on the acceleration of some critical tasks of H.264, such as IDCT and deblocking, but no information were given about the performance of the whole H.264 decoding process. Besides, neither ADRES nor XPP-III can support the 1920×1080@30 fps real-time decoding of H.264 High-Profile (HiP) streams. ADRES could only realize H.264 1280×720@30 fps video decoding at a working frequency of 300 MHz [8]. XPP-III, which is a popular commercial reconfigurable media processor, can support H.264 Baseline-Profile (BP) 1920×1080@24 fps videos at a working frequency of 450 MHz [10], [11]. Furthermore, as a result of the great working frequency of 450 MHz, the dynamic power of XPP-III is as high as 3,420 mW [12]. In order to support the 1920×1080@30 fps real-time decoding of H.264 HiP streams at a low working frequency, the scale of the CGRAs should be large enough to contain more reconfigurable units for processing more operations in parallel [19]. Therefore, the reconfiguration architecture becomes a crucial part of the large-scale CGRA design for applications where high performance is required.

The reconfiguration architecture plays a very important role in CGRAs, especially in large-scale CGRAs, since the numerous reconfigurable units contained require being dynamically fed with configuration data in time. The reconfiguration architecture design usually includes two aspects: the reconfiguring organization structure and the prefetching and reusing mechanism adopted for improving reconfiguration process.

Basically, hierarchical memory structures for configuration storage are very widely used in CGRAs. MorphoSys uses a simple three-level configuration storage hierarchy [18]: the external SDRAM is the global shared memory; an on-chip memory called the context memory (CM) stores all the configuration data; a group of registers, which are placed inside the reconfigurable array and called the context registers, hold the current active configuration data. Besides, the architecture of CM is customized for optimizing the transmission between CM and the context registers. However, all configuration data should be stored in CM, and the content of CM is organized by a static method. The flexibility and applicability of such a system will be

limited by the specified CM, due to its fixed memory size. It is inappropriate to place all of the configuration data in an on-chip memory. The reconfiguration architecture designs of MORA and ReMAP are very similar to MorphoSys. The major difference is that the configuration data could be dynamically transferred from external memory to reconfigurable array [16], [17]. However, the prefetching and reusing strategies are not mentioned in their reconfiguration architecture designs. Since the scale of the reconfigurable array contained in the CGRAs mentioned above is very fixed and small (not more than 8×8), the reconfiguration performance required can be easily solved. However, the system performance will also be limited by the small and fixed array scale. For the multimedia applications, the much more computations involved, the much high processing performance is required. Therefore, the CGRAs with small array scale could never be used to solve the multimedia applications with high requirements of system performance.

Unlike the CGRAs with a small array scale, both ADRES and XPP-III provide a flexible architecture design, where the reconfigurable arrays can be easily scaled up. Therefore, in ADRES and XPP-III, many considerations have been spent on the optimization of the reconfiguration process, besides the hierarchical structure design for configuration storage. In ADRES, a specified VLIW processor is placed between the reconfigurable array and the external memory. This VLIW processor is responsible for supervising and scheduling the reconfiguration process of the reconfigurable arrays. Besides, there is also a register file called the Conf. RAM placed inside the reconfigurable array. With this Conf. RAM, configuration data transferred from the VLIW processor can be buffered and reused. In XPP-III, there are three kinds of arrays: the array of ALU-PAEs, the array of RAM-PAEs and the array of FNC-PAEs [20]. The ALU-PAEs are responsible for computing-intensive operations, and the RAM-PAEs are responsible for the on-chip data buffering and reusing. The array of FNC-PAEs, which is tightly coupled with the array of ALU-PAEs, is responsible for control-intensive operations, including the supervision of the reconfiguration process of all ALU-PAEs.

As the array scale of REMUS-II is much greater than all the other CGRAs mentioned above, REMUS-II can obtain higher processing parallelisms, and thus a higher processing performance, too. To ensure the higher parallelisms and system performance, the reconfiguration architecture design of REMUS-II should be carefully considered. In this paper, the reconfiguration process of REMUS-II is optimized to meet the high requirement of the dynamic reconfiguration performance for multimedia applications.

3. Architecture of REMUS-II

3.1 Top-Level Architecture of REMUS-II

In comparison with the original version [5], [6], REMUS-II primarily focuses on improving the performance of the reconfiguration process as well as the data access process.

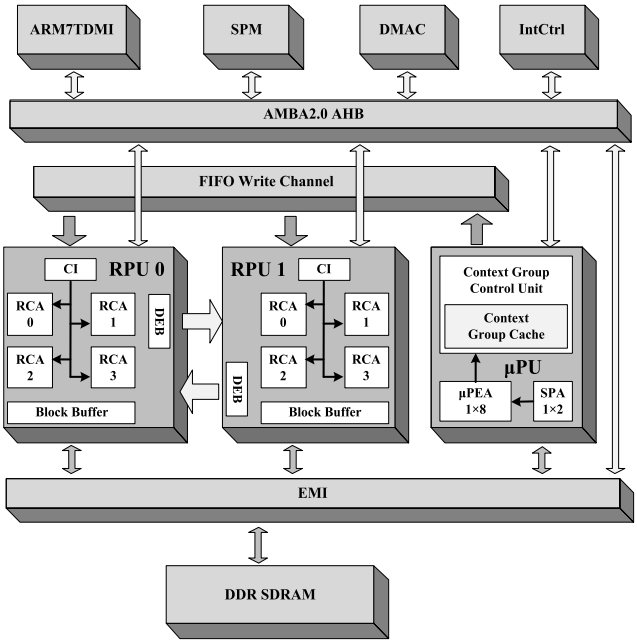


Fig. 1 Top level architecture of REMUS-II.

The top level architecture of REMUS-II is shown in Fig. 1. It consists of a system controller implemented with ARM7TDMI, a scratch-pad memory (SPM) with a size of $32\,\mathrm{K} \times 32$ bits, two Reconfigurable Processor Units (RPUs) for speeding up computing-intensive tasks, a Micro-Processor Unit (μPU) for speeding up control-intensive tasks, and several assistant modules, including an Interrupt Controller (IntCtl), a Direct Memory Access Controller (DMAC), and an External Memory Interface (EMI) with a data I/O width of 64 bits. All of the modules are AMBA2.0-AHB-compatible and connected to the 32-bit AHB Bus module used as the system bus. The two RPUs can exchange data through a specific bus with a size of 256 bits. Besides, there is also a specific 512-bit FIFO Write Channel implemented to provide a super-high bandwidth simplex communication path between the μPU and the two RPUs. In this proposed SoC design, the entire configuration data for all target multimedia applications and the massive computation data such as the reference frames is stored in the external main memory, which is a double data rate (DDR) SDRAM chip.

The reconfiguration and scheduling process of the two RPUs are supervised by the μPU. There are three pipelined operations performed on μPU for the reconfiguration and scheduling process: ① the configuration flow is parsed and the corresponding configuration data is located; ② if the configuration data is cached in μPU, then they are directly sent to RPUs; ③ if not, an extra pre-fetching operation which is responsible for pre-loading corresponding configuration data from external memory, will be carried out between the former two operations.

The decoding algorithm including both H.264 and MPEG-2 on the slice layer can be divided into two

parts: entropy decoding tasks and MB (macro-block) decoding tasks [6]. The entropy decoding tasks, including the Context-Adaptive Binary Arithmetic Decoding (CABAD) and the Context-Adaptive Variable Length Decoding (CAVLD), are control-intensive tasks and thus are mapped on to the μPU. For H.264, MB decoding is composed of three kernel sub-algorithms: IDCT, Intra Prediction/MC and deblocking. MB decoding tasks are computing-intensive tasks, and therefore are mapped on to the two RPUs.

With the supervision of μPU, the two RPUs can work in either ping-pong mode or parallel mode, which are adopted for H.264 decoding and MPEG-2 decoding, respectively. The details will be described in Sect. 4.

3.2 Reconfigurable Processor Unit

Figure 1 depicts that each RPU contains four fundamental Reconfigurable Arrays (RCAs), a Configuration Interface (CI), a Block Buffer, and a Data Exchange Buffer (DEB). RCA is a powerful dynamic reconfigurable system consisting of an 8×8 Processing Element Array (PEA) and several related modules including the register file and the local memory. CI is responsible for receiving and buffering configuration data sent from the μPU. DEB is designed to exchange data between RPUs, while the Block Buffer is responsible for the data transfers between RPUs and other modules. REMUS-II provides a convenient way for the bidirectional data transfer between different RPUs. When some data are required to be transferred from one RPU to the other, the previous RPU firstly writes these data into its DEB, and then informs the other RPU to read them out.

The configuration data sent to RPU is always regarded as a configuration package with a fixed data size. When a certain configuration package is sent to RPU, the CI will firstly parse the identification head (*Id_head*) of the configuration package. The *Id_head* is made up of the first 64 bits of configuration package. In the *Id_head*, there are a 16-bit synchronization code (*Sync_code*) and a 10-bit source-identification code (*Src_code*), as shown in Fig. 2. The *Src_code* is used to indicate a new configuration package. Besides, there is also a 6-bit target-identification code (*Tar_code*) for specifying the target RCAs to which the configuration package will be sent, as shown in Fig. 2. After being parsed by the CI, the rest bits of the configuration package except the *Id_head* (also defined as the *Conf_context*) will be sent to the corresponding RCAs.

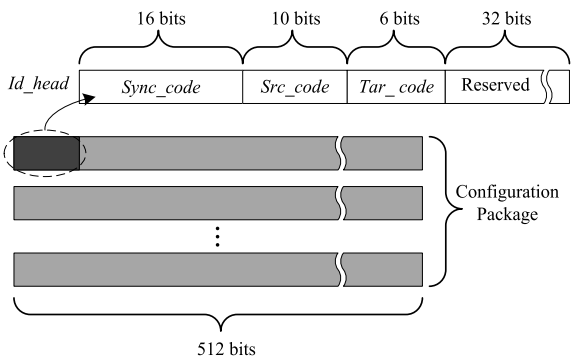


Fig. 2 Structure of the configuration package.

3.3 Micro (μ) Processor Unit

The μPU contains a 1×8 micro-processor element array (μPEA), a 1×2 bit-stream processor array (SPA), and a specific cache called Context Group Control Unit (CGCU), as shown in Fig. 3 (a). The SPA, is a reconfigurable bit-stream decoder, and consists of two bit-stream processor elements (SPEs) in order to process data fields efficiently. The μPEA is responsible for carrying out control-intensive tasks including the supervision of the reconfiguration process.

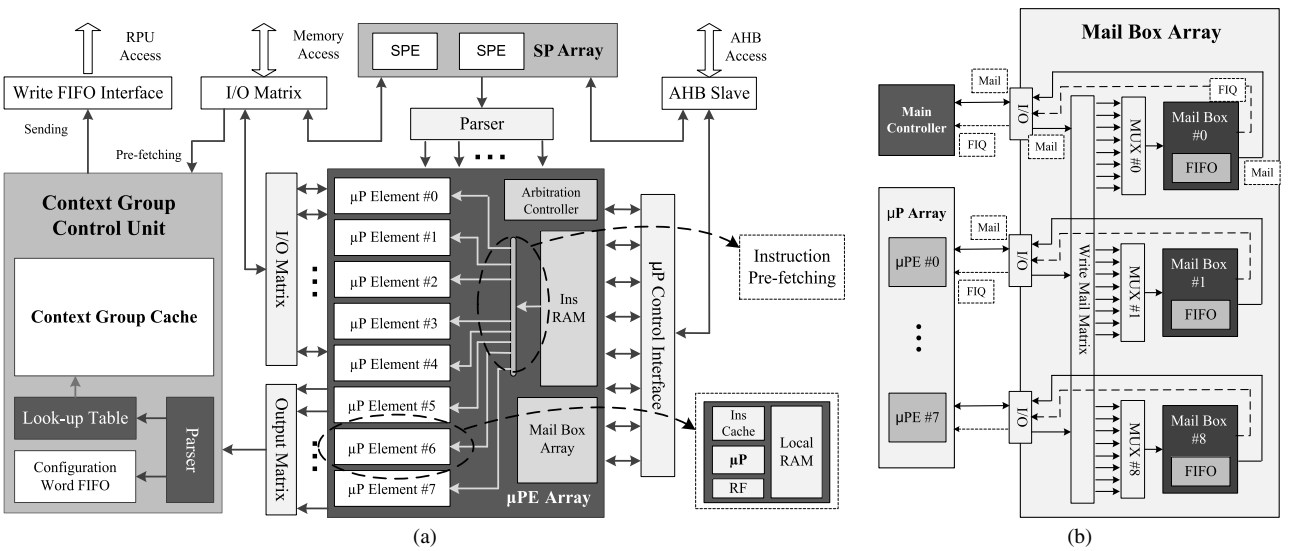


Fig. 3 (a) Architecture of the μP unit, (b) Structure of the mail communication.

The CGCU is responsible for the configuration data pre-loading and distributing, including three kinds of sub-operations: pre-loading configuration data from external memory, caching configuration data in μPU, and sending configuration data to RPUs.

The RSIC processor ARM7TDMI, which is well-known for its outstanding power efficiency [21], is chosen to implement these micro-processors (μPs). Each micro-processor element consists of a μP, a register file for setting parameters and checking status, an 8 K bytes local memory, and a tiny instruction cache. In order to improve memory utilization, all μPEs share a 32 K-byte instruction memory called Ins RAM. Furthermore, a tiny but high-efficiency cache with the 256-bit I/O data width is implemented between each μPE and the shared Ins RAM to improve the performance of instruction loading process, as shown in Fig. 3 (a). The instruction loading process is also regarded as the self-reconfiguration process of μPEA, which is different from the reconfiguration process of the RPUs. In addition, the communication between one μPE and another μPE, and the communication between the system controller and the μPEs, are both implemented by the Mail Box Array which is composed of nine mail boxes, as shown in Fig. 3 (b). Each of the system main controller and the eight μPEs has an associated mail box. Communication messages are sent and received through these mail boxes. Each mail box provides an output interrupt signal which is connected to its associated μPE or system controller. These interrupt signals are finally connected to the ARM7TDMIs as the ARM fast interrupt request (FIQ) input signal [21]. By monitoring these FIQ signals, the μPEs and the system controller can observe whether the associated mail boxes are empty or not.

4. Reconfiguration Process Optimization

The maximum number of system clock-cycles for processing each MB is defined as T_{mb} and calculated in Eq. (1), where F_{sys} is the system working frequency, N_{mb} is the number of MBs per frame, and fps is the frames per second. Besides, the calculating result should be rounded down to the nearest integer. According to Eq. (1), T_{mb} should be limited to 816 clock-cycles to support the 1080p@30 fps stream decoding at the clock frequency of 200 MHz.

$$T_{mb} = \left\lfloor F_{sys}/(N_{mb} \times fps) \right\rfloor \quad (1)$$

The mapping sketch of the parallel working mode (e.g. for MPEG-2 decoding) and the ping-pong working mode (e.g. for H.264 decoding) are shown in Fig. 4 (a) and Fig. 4 (b), respectively. The total system overhead for H.264 and MPEG-2 decoding mainly consist of three parts: the entropy decoding tasks, the MB decoding tasks and the reconfiguration processing tasks. These three tasks are performed as three pipelined steps as follows.

Step1. The entropy decoding tasks are processed by SPA. The processing results of this step consist of two parts: the residual entropy and the input options of the system control flow (also called configuration parameters in this paper). The configuration parameters are then sent to the μPEA to drive the reconfiguration processing step. In addition, the residual entropy will be finally loaded into RPU0 as the source input data of IDCT.

Step2. The reconfiguration processing tasks are performed by μPEA. The reconfiguration tasks processed in this step are driven by the continuously input configuration parameters produced in the former step. In REMUS-II, there are eight μP elements comprised in μPEA, and each μP element can process one reconfiguration task once per time. Each reconfiguration task is responsible for the reconfiguration process of one MB. The reconfiguration task includes two parts: firstly, μPEA parses the control flow and locates the configuration data; and then, CGCU prepares the configuration data and sends them to RPUs. In μPEA, eight reconfiguration tasks for eight sequential MBs can be processed in a parallel way, as shown in Fig. 4. However, the configuration data transferring tasks performed in CGCU can only be scheduled in a sequential way. Therefore, there are timing gaps (equal to T_{mb}) between the neighboring reconfiguration tasks, as shown in Fig. 4. Since the timing gaps have an enormous impact on reconfiguration performance, further optimization methods for this problem will be introduced afterwards.

Step3. The MB decoding tasks are processed by the two RPUs. This step is driven by the incoming configuration data generated in the Step2. Once a RPU is reconfigured, then corresponding source input data will be loaded and computed by the RPU. The final output results of each MB will be stored into external memory by the RPU, after the completion of a computation. In addition, the temporal calculation variables only effective within the current MB decoding tasks usually are buffered in RPUs for efficient memory access purpose.

For MPEG-2 decoding, the parallel mode is chosen and the two RPUs work in a complete parallel way, since the MB decoding task can be processed within T_{mb} by each RPU. As shown in Fig. 4 (a), the two RPUs perform two MB decoding tasks concurrently and are mapped with same configurations in this mode. Therefore, the reconfiguration overhead can be reduced with a technology called multi-target reconfiguration. With the proposed technology, a configuration package can be sent to more than one RPU at once, so the transmission of configuration data can be reduced, and this will result in performance promotion of the reconfiguration process. The details of the multi-target reconfiguration technology will be illustrated in Sect. 4.1.

For H.264 decoding, the ping-pong mode is chosen and the two RPUs work in pipelined way, because the MB decoding task is too complicated to be processed within T_{mb} by one RPU. In this mode, the task which requires high processing performance will be divided into two data-dependent sub-tasks, and each sub-task will be then allocated into one RPU. Therefore, this task can be processed in pipeline by the two RPUs. As shown in Fig. 4 (b), the two RPUs are scheduled as follows: RPU0 performs MC and IDCT (the sub-task1), and RPU1 performs deblocking

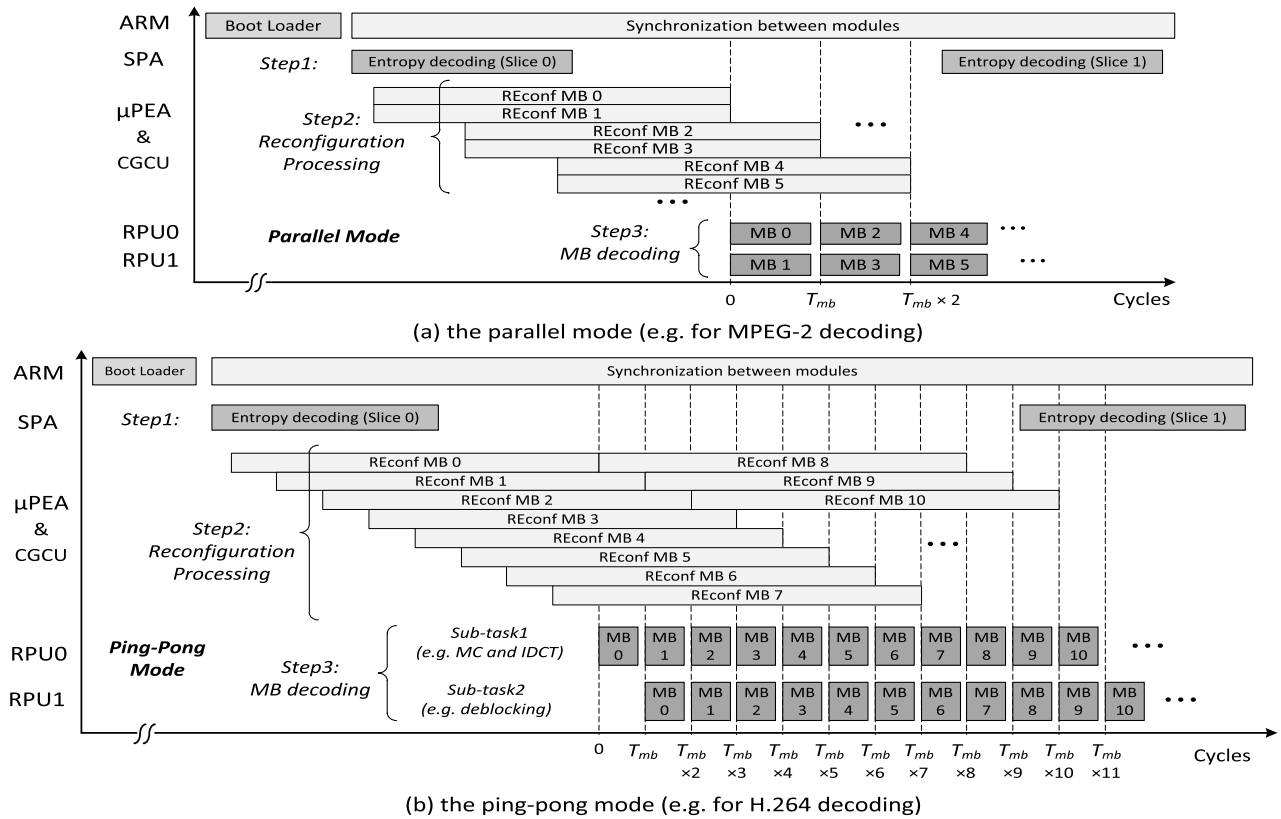


Fig. 4 Mapping sketch of video decoding on REMUS-II.

(the sub-task2). As a result, each sub-task can be easily processed in one time-slot (the time-slot is equal to T_{mb}). Besides, the calculation results of sub-task1 are generated by RPU0 and sent to RPU1 as the input of sub-task2 through the DEB.

Because of this pipelined scheduling approach, totally three MBs can be processed in the pipeline, and therefore one MB can be then decoded in three time-slots. In order to fulfill the performance requirements, each step should be restricted in one time-slot. Among the three steps, to restrict the reconfiguration processing step in one time-slot is the major bottleneck problem.

The requirements of the optimization can be denoted by the performance gain of the reconfiguration process. The required performance gain of the reconfiguration process is defined as G and calculated in Eq. (2), where $N_{cluster}$ represents the average bits of the configuration data required by the two RPUs for processing one MB (also defined as a configuration cluster), B_{DDR} represents the bandwidth of external memory and T_{mb} represents the maximum number of system clock-cycles for each MB.

$$G = \frac{N_{cluster}/B_{DDR}}{T_{mb}} \tag{2}$$

Supposing B_{DDR} is about 64 bits/cycle, the required performance gain of reconfiguration process for H.264 and MPEG-2 1080p@30 fps stream decoding are shown respectively in Table 1. For H.264, the two RPUs totally require

Table 1 Requirements of the reconfiguration process optimization.

Application	H.264	MPEG-2
$N_{cluster}$	200Kbits	120Kbits
G	3.9	2.3

200 K bits configuration data for processing the MB decoding. As a result, almost 3200 clock-cycles will be required for loading a reconfiguration cluster. Therefore, the performance of reconfiguration process should be improved about four times. For MPEG-2, the two RPUs are usually mapped with a same configuration setting, so the size of a configuration cluster is only about 120 K bits. The performance of reconfiguration process for MPEG-2 is required to be improved over two times.

The reconfiguration process is going to be optimized from the following aspects: Firstly, a hierarchical memory structure for configuration storage is implemented to mitigate the performance gap between on-chip processors and off-chip memory. Secondly, the reconfiguration process is further scheduled in a 3-stage pipeline mode to loosen the restricted time requirement, and corresponding optimization methods are introduced in each stage. Finally, a supervising strategy is proposed for the configuration cache to achieve a high performance on configuration pre-fetching and distributing. Additionally, in the following Sect. 4.1, the multi-mapping feature of the reconfiguration process is illustrated, and the multi-target reconfiguration technology is

then introduced to improve the performance of reconfiguration process.

4.1 The Hierarchical Configuration Storage Sub-System

Taking the power and area consideration into account, the memory size of integrated SRAMs should be limited to a tolerably small amount, since the silicon area occupancy and power consumption increases significantly when the memory capacity rises. As mentioned in Sect. 2, the small and fixed memory capacity will restrict the probable size of configuration data, and this will seriously hamper the flexibility and applicability of target CGRAs. As a result, the whole configuration data is located in external memory rather than in integrated SRAMs.

In our proposed REMUS-II, the configuration data for all target applications are located in an external memory which is implemented by a DDR SDRAM chip. DDR SDRAM chips have been commonly used as the external main memory in video processing systems, since they provide high capacity data storage at commodity prices. Since DDR SDRAMs are organized in linear rows, data reading from a different row will cause extra SDRAM operations (i.e. row pre-charge, and row active), and thus requires much more time than reading from a same row as the previous access [22]. Therefore, the technique of data pre-fetching with consecutive memory access and the technique of data buffering should be introduced to reduce external memory access latency.

In order to mitigate the performance gap between on-chip processors (RPUs and μPU) and off-chip memory, a configuration storage hierarchy is implemented to deal with the issues posed by moving configuration data from external memory to computational processors. There are three storage levels presented in REMUS-II. The first level is DDR SDRAM which provides a global shared storage of configuration data needed for the complete configuration flow of target applications. The second level is the on-chip configuration data cache, which is used as a temporal repository of the configuration data which will be used recently. The third level is the register files in each RCA where configuration data is set to be active. The working process of this storage hierarchy is shown as follows.

Firstly, the configuration data is automatically pre-fetched from external DDR memory, and then cached by a specific cache structure called CGCU implemented in μPU. In order to improve memory access efficiency, the DDR access is always implemented in a long-burst-length pattern by EMI.

Afterwards, the μPU dynamically sends the configuration data to the appropriate RPU. The configuration data is then received and buffered by CI, which is shared by four RCAs in the present RPU.

Finally, the configuration data is parsed by CI and then mapped to the appropriate RCA when previous task is accomplished.

Since the RCAs in one RPU are very likely to be mapped with same configuration settings, one configuration package may be sent to two or more RCAs at once, and this is called the configuration multi-mapping. In order to support the multi-mapping feature, FIFO Write Channel and the *Tar_code* contained in *Id_head* are designed with the support of multi-target reconfiguration (more than one RPU or RCA at once). The most two significant bits of *Tar_code* indicate the target RPUs: '01' and '10' represent RPU0 and RPU1 respectively, and '11' indicates the current configuration package is sent to both RPU0 and RPU1. The least four significant bits of *Tar_code* indicate the target RCAs: each bit (bit0 ~ bit3) represents a RCA (RCA0 ~ RCA3) respectively. Therefore, one configuration package can be sent to more than one RCA at once by setting the *Tar_code*. With the multi-target reconfiguration technology there will be a visible decrease on the number of configuration packages transferred from μPU to RPUs.

Considering the decrease in transmission of configuration data by the multi-target reconfiguration technology, the average clock-cycles required for transferring a configuration cluster is then calculated in Eq. (3) (defined as T'). The parameter T represents the clock-cycles required for transferring a configuration cluster without the multi-target reconfiguration technology. The parameter P is the equivalent ratio of T' to T, and P represents the decrease ratio of the transmission with multi-target reconfiguration technology. Much less the parameter P is, much less the transmission of configuration data is. The computational formula for parameter P is shown in Eq. (4). The parameter P_{dupi} (i is 0~7) represents the average probability of i-duplicate reconfiguring cases, and the total sum of P_{dupi} is 1, as shown in Eq. (5). The i-duplicate reconfiguring case indicates the present configuration package will be sent to $(i+1)$ RCAs. Additionally, the 0-duplicate case is also called the non-duplicate reconfiguring case, where the configuration package is only sent to a certain RCA. By profiling and computing the value of these P_{dupi}, the parameter P can be calculated by Eq. (4).

$$T' = P \times T \tag{3}$$

$$P = \frac{\sum_{i=0}^{7}((8-i) \times P_{dupi})}{8} \tag{4}$$

$$1 = \sum_{i=0}^{7} P_{dupi} \tag{5}$$

In the CI of each RPU, there is also a tiny cache memory called Context Kernel Cache, which is used for the configuration package buffering and reusing. Figure 5 depicts the architecture of the CI. It includes three sub-modules: the Input FIFO receives the incoming configuration packages; the Context Kernel Cache is responsible for configuration package buffering and reusing; the Configuration Buffer assigns the configuration package to corresponding RCAs. The Context Kernel Cache consists of three sub-modules including a cache controller, a cache memory (CacheMEM) and a tag register file (TagRF), as shown in Fig. 5. The

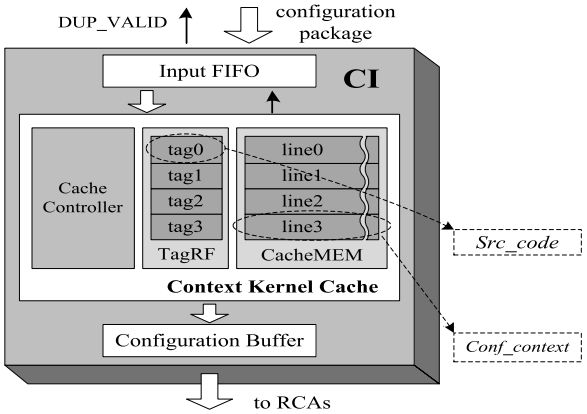


Fig. 5 Architecture of the CI.

cache memory has four cache lines (line0~line3), and each line can buffer the *Conf_context* of a configuration package. The cache register file has four cache tags (tag0 ~ tag3), and each cache tag buffers the *Src_code* of a configuration package. Furthermore, line*i* and tag*i* (*i* is 0~3) indicate a same configuration package. The cache controller supervises the working process of the Context Kernel Cache, which can be classified into two cases: the case of a cache hit and the case of a cache miss.

Since the *Id_head* is comprised in the first 64 bits of a configuration package, it will always be received by each RPU in the first clock-cycle of the transmission. By comparing the *Src_code* included in *Id_head*, the cache controller can detect whether the *Conf_context* of the incoming configuration package is already buffered in Context Kernel Cache. If it is already buffered, then this is the case of a hit, otherwise it is the case of a miss. In the case of a miss: ① the incoming configuration package is firstly received and buffered in Input FIFO; ② then the configuration package will be transferred to Configuration Buffer, and meanwhile a copy of the *Src_code* and *Conf_context* of this configuration package will be buffered in Context Kernel Cache (in TagRF and CacheMEM respectively); ③ finally the *Conf_context* of this configuration package will be sent to the corresponding RCAs. In addition, the content of Context Kernel Cache can be supervised by a simple Round-robin strategy. In the case of a hit: ① only the *Id_head* of the incoming configuration package will be received by Input FIFO; ② one buffered *Conf_context* will be selected from CacheMEM and it will combine with the *Id_head* buffered in Input FIFO to form an entire configuration package. This configuration package will be then stored in Configuration Buffer; ③ finally the corresponding RCAs will be mapped by the content of Configuration Buffer, just the same as the former case of a miss. Besides, in the case of a hit, the cache controller will inform CI to drive an output signal called DUP_VALID to μPU. Once getting the value 1 of DUP_VALID signal, μPU will terminate the transmission of current configuration package. In this case, the whole transmission will be accomplished in one clock-cycle. Therefore, the Context Kernel Cache, which can hold the recent configuration settings, will reduce the unnecessary transmissions between μPU and RPUs.

4.2 The Pipelined Reconfiguration Processing Structure

The reconfiguration process proposed is divided to three stages: the configuration word generating stage (Stage1), the context group pre-fetching and sending stage (Stage2, also called the context group preparing stage), and the context kernel remapping stage (Stage3).

The context kernel is a group of configuration data used to configure a RCA once. In REMUS-II, a configuration package is equal to a context kernel. The size of each context kernel is fixed as 2 K bits. There are totally about 315 different context kernels created in our proposed work for H.264 decoding, and totally about 85 different context kernels for MPEG-2 decoding.

The context group is a configuration set consists of several context kernels to perform a certain sub-algorithm of target application (e.g. IDCT4 $\times$ 4, IDCT8 $\times$ 8, MC, and so on). A context group can contain up to eight context kernels, and the context kernels involved in a context group are arranged in order of their precedence.

The information that determining which context group is chosen to be loaded and which RCA is to be mapped is defined as the configuration word. Although there are totally over three hundred context kernels for H.264, the frequently-used context kernels are less than one hundred. These frequently-used context kernels are normally used to deal with some sub-algorithms for MC and deblocking, such as the matrix-transposition and others. The data size of frequently-used context kernels takes up about 30 percent of the totals, but masks up about 80 percent of the access times. Therefore, the external memory access overhead can be greatly reduced with the technique of caching and reusing the frequently accessed data.

The timing chart of 3-stage reconfiguration processing structure for MPEG-2 and H.264 is shown in Fig. 6 (a) and Fig. 6 (b), respectively. Since the context kernel remapping stage costs quite little cycles, and is always accompanied with the MB decoding operations, the corresponding MB decoding process is depicted instead. These reconfiguration stages are performed in a pipelined manner as follows.

Stage1. In this stage, the system allotting and scheduling tasks are processed by eight μP elements in advance. Therefore, information about the control flow of next MB can be obtained by parsing the configuration word generated and buffered. Therefore the context groups required in future can be completely predicted and pre-fetched in advance.

Stage2. In this stage, the required context groups are pre-fetched and the context kernels contained are then sent to relevant RCAs in appropriate order.

Stage3. This stage is performed on each RCA and controlled by the former context group pre-fetching and sending stage. When one RCA is under the context kernel remapping stage, the other RCAs in the same RPU can execute their tasks at the same time, and this is also called the partial

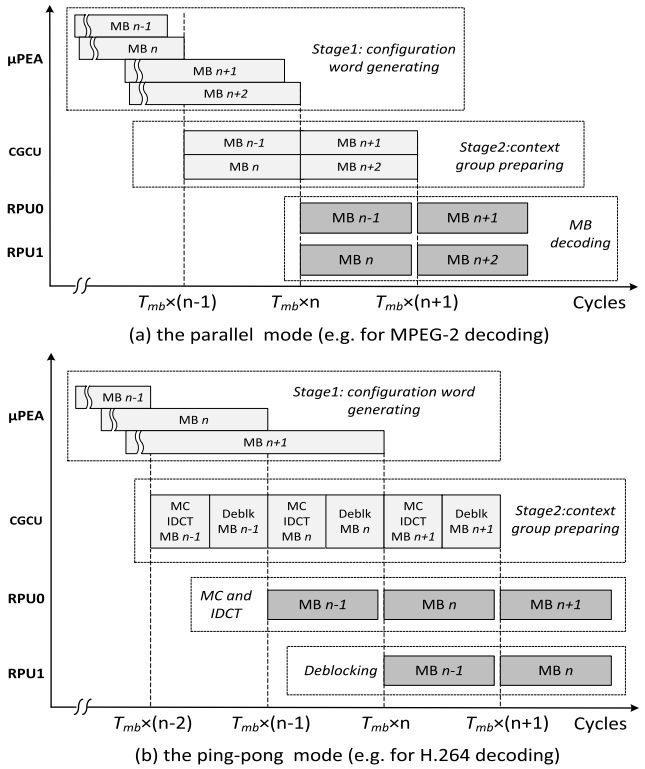


Fig. 6 Timing chart of the reconfiguration processing structure.

reconfiguration feature. Besides, there is also a so-called none-wait-reconfiguration feature which can further reduce the mapping overhead in Stage2. In each RCA, there are two register files for configuration mapping. When one register file is active, the next incoming context kernel will be loaded to a shadow register file. Once the current mapping task is finished, the shadow register file will switch to be active, and the former register file will then switch to be the shadow register file.

As illuminated above, the operations in Stage3 can be easily carried out within T_{mb}. Besides, since eight sequential tasks can be processed by eight μP elements in parallel, each task in Stage1 can be processed within $8\times T_{mb}$, which is also easy to be satisfied. Among these stages, the Stage2 where numerous operations for context group loading and sending are involved is the critical path. In order to satisfy the performance requirement of this stage, two optimization methods are introduced for reducing the transmission required as following:

Firstly, a Context Group Cache is implemented in μPU to pre-fetch and buffer the context groups for reusing purpose. Considering the decrease in transmission of configuration data by the Context Group Cache, the average clock-cycles required for transferring a configuration cluster is given in Eq. (6) (defined as T''). The parameter $P_{hit\text{-}CGC}$ represents the Context Group Cache hit ratio, and T' given in Eq. (3) represents the clock-cycles required for transferring a configuration cluster without Context Group Cache. Besides, T_{CGC} represents the clock-cycles for loading a context kernel from Context Group Cache to RPUs, and T_{DDR} represents the average clock-cycles for loading a context kernel from DDR memory to Context Group Cache. The value of T_{CGC}/T_{DDR} is equal to the inverse proportion of the bandwidth of Write FIFO Channel to that of the EMI: $64/512 = 0.125$.

$$\begin{aligned} T'' &= \frac{P_{hit\text{-}CGC}\times T_{CGC} + (1-P_{hit\text{-}CGC})\times (T_{DDR}+T_{CGC})}{T_{DDR}}\times T' \\ &= \left(\frac{T_{CGC}}{T_{DDR}} + (1-P_{hit\text{-}CGC})\right)\times T' \end{aligned} \tag{6}$$

Furthermore, the Context Kernel Cache in each RPU will also significantly reduce the total transmission of context kernels from μPU to RPUs. In the case of a hit, the whole transmission of a context kernel can be accomplished with only one clock-cycle. Therefore, 75 percent system cycles will be saved when the Context Kernel Cache access hits. Considering the decrease in transmission of configuration data by Context Kernel Cache, the average clock-cycles required for transferring a configuration cluster is then calculated in Eq. (7) (defined as T'''). The parameter $P_{hit\text{-}CKC}$ represents the context kernel cache hit ratio, and T'' given in Eq. (6) represents the clock-cycles required for transferring a configuration cluster without the Context Kernel Cache. The parameter $T_{hit\text{-}CKC}$ represents the clock-cycles required by the transmission of a context kernel when the Context Kernel Cache access hits, and $T_{miss\text{-}CKC}$ represents the clock-cycles required in the case of a miss. As mentioned above, the $T_{hit\text{-}CKC}$ is 1 and the $T_{miss\text{-}CKC}$ is 4 respectively in our implementation.

$$T''' = \frac{P_{hit\text{-}CKC}\times T_{hit\text{-}CKC} + (1-P_{hit\text{-}CKC})\times T_{miss\text{-}CKC}}{T_{miss\text{-}CKC}}\times T'' \tag{7}$$

As can be seen in Fig. 4 and Fig. 6, the proposed optimization methods enhance the processing performance of H.264 and MPEG-2 by improving the reconfiguration efficiency of their kernel sub-algorithms such as MC, IDCT and deblocking, which are also widely used in other media applications, including AVS [23], MPEG-4, H.263, and so on. Therefore, the proposed reconfiguration architecture is not specified for H.264 or MPEG-2, and can be also suitable for other multimedia applications such as H.263, MPEG-4, AVS, and so on.

4.3 The Context Group Control Unit

The Context Group Control Unit (CGCU) is responsible for the configuration data pre-loading from external DDR memory and distributing to RPUs. As shown in Fig. 3, CGCU consists of four sub-modules: the Context Group Cache, the Configuration Word FIFO, the Look-up Table, and the Parser.

The Context Group Cache is responsible for caching the context groups required by RPUs. When a new

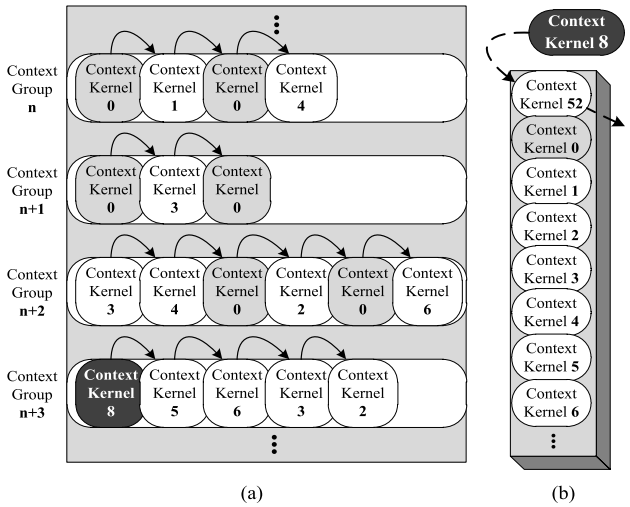


Fig. 7 Content of the Context Group Cache: (a) the logical view, (b) the physical view.

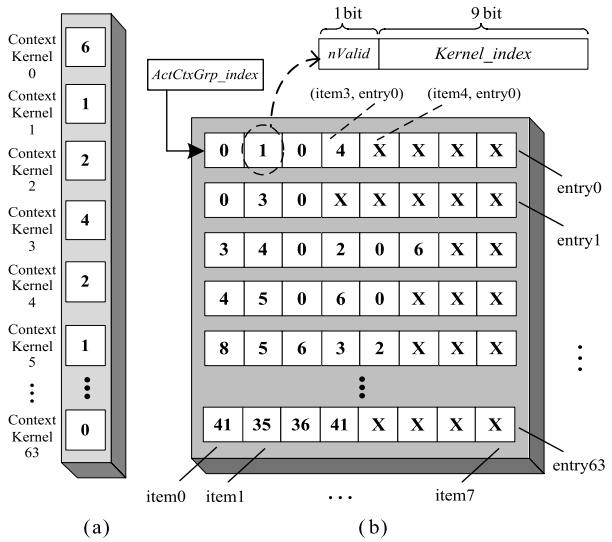


Fig. 8 Content of the Look-up Table: (a) the *Kernel_weight*, (b) the *nValid* flag and the *Kernel_index*.

configuration word is generated by μPEA and written to CGCU, CGCU will parse the incoming configuration word and then pre-load the context group required from external DDR memory. Besides, there is also a register defined as the Active Context Group Index (*ActCtxGrp_index*). The *ActCtxGrp_index* is designed to indicate which context group is currently transferred from CGCU to RPUs. The updating strategy for the value of *ActCtxGrp_index* is a simple Round-robin algorithm.

Since the context groups in Context Group Cache are always likely to contain same context kernels, the Context Group Cache is designed in a so-called two-side-view way: the logical view and the physical view. From the logical view, context group is regarded as the basic unit processed in Context Group Cache, as shown in Fig. 7 (a). From the physical view, configuration data stored in Context Group Cache is in the unit of context kernel, as shown in Fig. 7 (b). Furthermore, the same context kernel involved in multiple context groups will be only stored once, and the information about context groups is recorded in the Look-up Table, as shown in Fig. 8. Depending on the consumption rate of RPUs and the generating rate of μPEA, the average context groups cached in Context Group Cache is about sixty. Besides, the average number of context kernels contained in each context group is about four (4.2 for H.264, and 3.7 for MPEG-2). Therefore, from the logical view, the number of context kernels required by these context groups is: $60 \times 4 = 240$. However, from the physical view, the number of context kernels actually stored in Context Group Cache is in maximum 64. Therefore, with the two-side-view design, totally 44 K bytes of the Context Group Cache will be saved (with a 73% reduction on memory space occupancy).

The Look-up Table is responsible for tracing two kinds of information: the content record of a context group, and the usage record of each context kernel. Once a new configuration word is received, or a context group is moved out from the Context Group Cache and sent to the RPUs, the content of Look-up Table will be updated accordingly. As shown in Fig. 8 (b), a 64-entry table (defined as entry0 ~ entry63) is used to record the content of each context group. Each entry of the table consists of eight 10-bit items (defined as item0 ~ item7). The lowest 9-bit (defined as the *Kernel_index*) registers the serial number of the context kernel (the lowest 9-bit of *Src_code*). The most significant bit is used to indicate whether the context kernel is valid or not (defined as the *nValid* bit: 0 is valid, and 1 is not valid). If the *nValid* bit of item4 in entry0 (marked as (item3, entry0) in Fig. 8 (b)) is 1, and the *nValid* bit of item3 in entry0 (marked as (item4, entry0) in Fig. 8 (b)) is 0, it indicates that the entry0 context group contains four context kernels (the invalid items are marked with X in Fig. 8 (b)). Besides, a 16-bit register is used to indicate the total number of context kernel involved in all context groups, and this number is also defined as the *Kernel_weight*, as shown in Fig. 8 (a).

The Parser is responsible for receiving configuration words sent from μPEA. The Configuration Word FIFO is responsible for buffering the incoming configuration words. When there is no more space in Context Group Cache for the incoming context groups, the Parser will inform μPEA to hold the current write operation.

By parsing configuration words buffered in Configuration Word FIFO, context groups required in near future can be completely predicted in advance. If some context kernels involved in these context groups are currently not cached in Context Group Cache, then victim context kernels should be selected for the context kernels going to be pre-loaded. To select the most appropriate victims, a dynamic strategy is implemented to organize Context Group Cache as follows: Firstly, the context kernels with a kernel weight of 0 will be selected as candidates for the victims. Afterwards, a simple Round-robin algorithm is performed to choose the victims from these candidates. Finally, the memory space used by the victims will be rewritten with the context kernels going

to be pre-loaded. When the memory space provided by the victims is not enough for a certain context group going to be pre-loaded, CGCU will not load this context group at once, until more victims and enough memory space are available.

Besides, the multi-bank interleaving mode of DDR memory is also chosen to improve external memory access efficiency. Taking the advantage of bank interleaving, the extra latencies caused by the DDR reading operations from a different row can be greatly reduced [24].

5. Experimental Results

As illuminated in Sect. 4, in order to support the 1920 × 1080@30 fps stream decoding at the clock frequency of 200 MHz, the average reconfiguration overhead for driving the RPUs to process each MB should be limited to T_{mb} (816 clock-cycles). Since the average reconfiguration overhead is indicated by the parameter T''', it can be easily calculated in Eq. (8). $T_{total\text{-}reconf}$ is the total clock-cycles spent by the two RPUs for getting configuration data, $N_{total\text{-}CG}$ is the number of the context groups totally loaded to the two RPUs, and $N_{mb\text{-}CG}$ is the average number of the context groups loaded to the two RPUs for one MB.

$$T''' = \frac{T_{total\text{-}reconf}}{N_{total\text{-}CG}} \times N_{mb\text{-}CG} \tag{8}$$

The actual performance gain of the reconfiguration process can be calculated in Eq. (9) (defined as G'). The formula for T'/T, T''/T', and T'''/T'' are given in Eq. (3) ∼ Eq. (7) respectively.

$$G' = \frac{1}{(T'''/T'') \times (T''/T') \times (T'/T)} \tag{9}$$

The profit of the reconfiguration structure on several benchmark streams for H.264 and for MPEG-2 is shown in Table 2 and Table 3, respectively.

The results show that, with the proposed reconfiguration architecture, REMUS-II can support the 1080p@32 fps of H.264 HiP@Level4 and 1080p@40 fps High-level MPEG-2 stream decoding at the clock frequency of 200 MHz. Based on all benchmark streams for both H.264 and MPEG-2 decoding, the reconfiguration overhead for processing one MB (the parameter T''') is much less than the restricted T_{mb} (816 clock-cycles). The results also depict that, with the proposed reconfiguration architecture, the performance of reconfiguration process can achieve a great increase in profits of about 4 times (the parameter $G' \approx 4$). The simulation results show that, for both H.264 and MPEG-2 benchmark streams, the actual performance gain of reconfiguration process (denoted by G') is much greater than the required performance gain for H.264 and MPEG-2 real-time 1080p@30 fps stream decoding (denoted by G). In contrast to the CGRAs mentioned in Sect. 2, REMUS-II can comprise and support a much larger array scale, with the proposed reconfiguration architecture design. As the array scale of REMUS-II is much greater, it can obtain much higher processing parallelisms. Therefore, REMUS-II can achieve a much higher processing performance.

Table 2 Profit of reconfiguration on benchmark streams for H.264.

Stream	Foreman	Airshow	Van.Helsing
Size	Qcif	1080p	1080p
Profile	High	High	High
T'/T	0.86	0.84	0.89
$P_{hit\text{-}CGC}$	0.79	0.79	0.80
T''/T'	0.34	0.34	0.33
$P_{hit\text{-}CKC}$	0.33	0.37	0.31
T'''/T''	0.75	0.72	0.77
G'	4.56	4.86	4.22
T'''	701cycles	659cycles	725cycles
Fps	2674	32.7	32.4

Table 3 Profit of reconfiguration on benchmark streams for MPEG-2.

Stream	Foreman	Airshow	CatsAndDogs
Size	Qcif	1080p	1080p
Profile	High	High	High
T'/T	0.64	0.61	0.67
$P_{hit\text{-}CGC}$	0.66	0.67	0.67
T''/T'	0.47	0.46	0.46
$P_{hit\text{-}CKC}$	0.28	0.31	0.24
T'''/T''	0.79	0.77	0.82
G'	4.21	4.62	3.96
T'''	457cycles	415cycles	486cycles
fps	3340	40.3	40.1

Besides, Table 2 and Table 3 also show that, the reconfiguration overhead for MPEG-2 is much less than that for H.264. As illustrated in Sect. 4, there are mainly two reasons: Firstly, since the MB decoding task for H.264 contains much more computations than that for MPEG-2, the amount of configuration data to be transmitted for H.264 is much greater than that for MPEG-2, and therefore the reconfiguration overhead for H.264 is also greater. Secondly, since the parallel working mode is chosen for MPEG-2, the two RPUs are mapped with same configuration settings, and therefore the reconfiguration overhead can be significantly reduced with multi-target reconfiguration technology. However, in ping-pong working mode, the two RPUs are mapped with different configuration settings, and therefore the reconfiguration overhead for H.264 cannot be reduced by as much as that for MPEG-2 with the multi-target reconfiguration technology.

Taking the benchmark stream *foreman_qcif* as example, the value of T'/T is about 0.86 for H.264 and 0.64 for MPEG-2. As mentioned above, the two RPUs are performed in a parallel way for MPEG-2, and therefore the value of P for MPEG-2 is much less than that of H.264. This result shows that with multi-target reconfiguration, the transmission overhead for configuration data will be reduced to less than 86% for H.264 and 64% for MPEG-2. Besides, as $P_{hit\text{-}CGC}$ is about 0.79 for H.264 and 0.66 for MPEG-2, the value of T''/T' is then about 0.34 for H.264 and 0.47 for MPEG-2 respectively (by Eq. (6)). This results show that with the context group cache placed in μPU, the

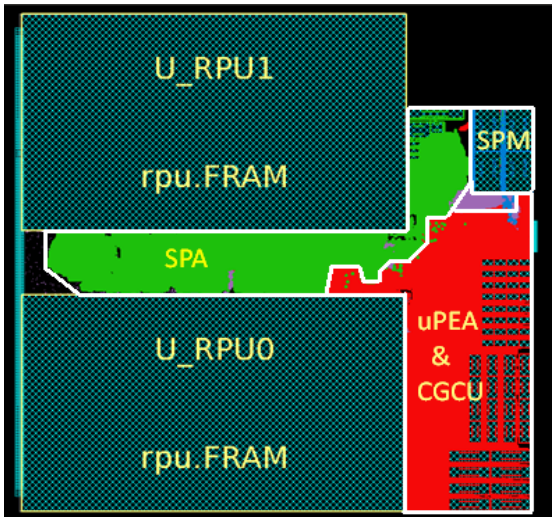


Fig. 9 Layout of REMUS-II.

Table 4 Comparison among REMUS-II and other reconfigurable platforms in H.264 decoding.

Platform	XPP-III[10][11][12]	ADRES[8]	REMUS-II
Process	90nm	90nm	65nm
Frequency	450MHz	300MHz	200MHz
Area	42.5mm^2	NA	23.7mm^2
Profile	BP	BP	HiP
Max Spec	1920×1080@24fps	1280×720@30fps	1920×1080@32fps

transmission overhead for configuration data will be further reduced to less than 50 percent. Furthermore, as the $P_{hit\text{-}CKC}$ is about 0.33 for H.264 and 0.28 for MPEG-2, the value of T'''/T'' is then about 0.75 for H.264 and 0.79 for MPEG-2 respectively (by Eq. (7)). This results show that with a tiny cache (the Context Kernel Cache) implemented in RPU, the configuration data sent from μPU will be further reduced to less than 80 percent.

To validate the proposed optimization methods, the hardware architecture of REMUS-II was described with Verilog HDL language and simulated with Synopsys Verilog Compiler Simulator (VCS) to show the system performance, as well as the performance of reconfiguration process. This design was implemented under TSMC 65 nm low power process. The area and timing results were generated by Synopsys Design Compiler (DC) using the worst case conditions, and the dynamic power was estimated by Synopsys PrimeTime-PX (PTPX).

The layout of REMUS-II is shown in Fig. 9, and it is mainly composed of several components: the two RPUs, the μPU (including the SPA, the μPEA and the CGCU), and the SPM. In addition, because the inter circuits of μPEA and CGCU are tightly coupled, they are placed in one block of the layout. The area of REMUS-II is 23.7 mm^2 and the dynamic power is estimated to 620 mW by PTPX (the power consumption of off-chip memory is not included).

Table 4 gives the comparisons for H.264 decoding on different reconfigurable platforms. The results show that the processing performance of REMUS-II is better than that of XPP-III and ADRES. Besides, since the computations of HiP decoding are 54% more than that of BP, REMUS-II achieves 105.3% better performance with the comparison to XPP. Moreover, since the dynamic power of XPP-III can be as high as 3,420 mW (450 MHz × 7.6 mW/MHz) [12], REMUS-II consumes much less power due to the much more advanced semiconductor process and the lower clock frequency.

Table 5 Implementation details of REMUS-II.

Hardware Costs	Memory Size (Kbytes)	Standard Cell Count (Kgates)	Power for H.264 (mW)
RPUs	404	12,043	439
SPA	64	1,982	61
µPEA & CGCU	194	2,236	83
Others	177	495	37
Total	839	16,756	620

The implementation details of REMUS-II are outlined in Table 5. As shown in Fig. 9, the hardware costs for the proposed reconfiguration architecture design are composed primarily of the μPEA and CGCU. The hardware costs and the energy consumption of μPEA and CGCU are as follows: the memory space occupied accounts for 23.1% of the embedded memory total, the equivalent logic gates used accounts for 13.3% of the total, the estimated dynamic power consumed accounts for 13.4% of the total on-chip power consumption, which are all not too high.

6. Conclusions

This paper presents a novel architecture design to optimize the reconfiguration process of a coarse-grained reconfigurable architecture called REMUS-II. To optimize the reconfiguration architecture, a hierarchical configuration storage structure and a 3-stage reconfiguration processing structure are proposed. Besides, the multi-target reconfiguration method and the configuration caching strategies are also introduced to dynamically detect and exploit the temporal locality of configuration data for reusing purpose. Since the proposed reconfiguration architecture is not specified for H.264 or MPEG-2, it can be also suitable for other multimedia applications such as H.263, MPEG-4, AVS, and so on. With the reconfiguration architecture proposed, the performance of reconfiguration process can be improved by 4 times. Based on RTL simulation, the proposed reconfiguration process can meet the requirement for H.264 and MPEG-2 HiP decoding. Experimental results show that REMUS-II can support the 1080p@32 fps of H.264 HiP@Level4 and 1080p@40 fps High-level MPEG-2 stream decoding at the clock frequency of 200 MHz. The proposed REMUS-II system has been implemented on a TSMC 65 nm low power process with a die size of 23.7 mm^2. Comparison results also show the processing performance of REMUS-II is better than that of XPP and ADRES.

Acknowledgments

This work was supported by the National High Technology Research and Development Program of China (863 Program) (grant no.2009AA011701), and the Natural Science Foundation of Jiangsu Province of China (grant no.BK2010167 and no.BK2010166).

The authors would like to thank to C. MEI, J. XIAO, J.J. YANG, Y.C. LU, Y.Q. FAN, H. LEI and C.X. ZHANG for their helpful discussions and technical support.

References

[1] T. Cervero, S. López, G.M. Callicó, F. Tobajas, V. de Armas, J. López, and R. Sarmiento, "Survey of reconfigurable architectures for multimedia applications," VLSI Circuits and Systems IV (Proceedings Volume), SPIE Microtechnologies for the New Millennium, Dresden, Germany, 2009, doi:10.1117/12.821713.

[2] J. V. T. of ITU-T, "Draft itu-t recommendation and final draft international standard of joint video specification (itu-t rec. h.264 iso/iec 14496-10 avc)," Document JVT-GO50, Dec. 2003.

[3] T. Wiegand, G.J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol.13, no.7, pp.560–576, July 2003, doi:10.1109/TCSVT.2003.815165.

[4] A. Joch, F. Kossentini, H. Schwarz, T. Wiegand, and G.J. Sullivan, "Performance comparison of video coding standards using Lagrangian coder control," Image Processing. 2002. Proceedings. 2002 International Conference on, pp.II-501–II-504, vol.2, 2002.

[5] M. Zhu, L. Liu, S. Yin, C. Yin, and S. Wei, "A cycle-accurate simulator for a reconfigurable multi-media system," IEICE Trans. Inf. & Syst., vol.E93-D, no.12, pp.3202–3210, Dec. 2010.

[6] T. Geng, L. Liu, S. Yin, M. Zhu, and S. Wei, "Parallelization of computing-intensive tasks of the H.264 high profile decoding algorithm on a reconfigurable multimedia system," IEICE Trans. Inf. & Syst., vol.E93-D, no.12, pp.3223–3231, Dec. 2010.

[7] B. Mei, B. De Sutter, T. Vander AA, M. Wouters, S. DuPont, and A. Kanstein, "Implementation of a coarse-grained reconfigurable media processor for AVC decoder," J. Signal Processing Systems, vol.51, pp.225–243, 2008.

[8] B. Mei, F.J. Veredas, and B. Masschelein, "Mapping an H. 264/AVC decoder onto the ADRES reconfigurable architecture," International Conference on Field Programmable Logic and Applications, pp.622–625, 2005.

[9] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP: A self-reconfigurable data processing architecture," J. Supercomputing, vol.26, pp.167–184, 2003.

[10] M.K.A. Ganesan, S. Singh, F. May, and J. Becker, "H. 264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture," Field Programmable Logic and Applications (FPL07), International Conference on, pp.467–471, 2007.

[11] F. Campi, R. Konig, M. Dreschmann, M. Neukirchner, D. Picard, M. Iuttner, E. Schuler, A. Deledda, D. Rossi, A. Pasini, M. Hubner, J. Becker, and R. Guerrieri, "RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip," Proc. 11th International Conference on System-on-chip (SOC09), pp.110–113, 2009.

[12] E. Schuler, "NoC concepts with XPP-III," International Symposium on Reliability of Optoelectronics for Space (ISROS 2009), Cagliari, Italy, May 2009.

[13] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," Proc. Design, Automation and Test in Europe (DATE 01), IEEE CS Press, pp.642–649, 2001.

[14] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiD-reconfigurable pipelined datapath," Field-Programmable Logic Smart Applications, New Paradigms and Compilers, pp.126–135, 1996.

[15] S.C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R.R. Taylor, "PipeRench: A reconfigurable architecture and compiler," Computer, vol.33, pp.70–77, 2000.

[16] P. Dai, X. Wang, and X. Zhang, "Implementation of H.264 algorithm on reconfigurable processor ReMAP," Microelectronics & Electronics, Asia Pacific Conference on Postgraduate Research, pp.237–240, 2009.

[17] M. Lanuzza, S. Perri, and P. Corsonello, "MORA: A new coarse-grain reconfigurable array for high throughput multimedia processing," Proc. International Symposium on Systems, Architecture, Modeling and Simulation (SAMOS), pp.159–168, 2007.

[18] H. Singh, M.H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, and E.M.C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput., vol.49, no.5, pp.465–481, 2000.

[19] B.D. Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in Handbook of Signal Processing Systems, Springer, 2010, ISBN: 978-1-4419-6344-4.

[20] PACT XPP Technologies, "XPP-III Processor Overview: White Paper," 2006.

[21] ARM Limited, "ARM7TDMI: Technical Reference Manual Rev3," 2001.

[22] Micron Technology, "Double Data Rate (DDR) SDRAM: MT46V2M32," 2001.

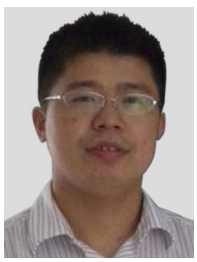
[23] AVS Video Expert Group, "Information Technology — Advanced coding of audio and video — Part 2: Video (AVS1-P2 JQP FCD 1.0)," Audio Video Coding Standard Group of China (AVS), Doc. AVS-N1538, 2008.

[24] S. Whitty and R. Ernst, "A bandwidth optimized SDRAM controller for the MORPHEUS reconfigurable architecture," 2008 IEEE International Symposium on Parallel and Distributed Processing, pp.1–8, 2008.

Bo Liu was born in 1984. He received the B.S., M.S. degrees in Electrical Engineering from Southeast University in 2006, 2008 respectively, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests mainly include multimedia processing, reconfigurable computing and related SoC designs.

Peng Cao received the B.S., M.S. and Ph. D degrees in Information Engineering and Electrical Engineering from Southeast University in 2002, 2005 and 2010 respectively. His research interests mainly include digital signal and image processing, image/video compression, reconfigurable computing and related VLSI designs.

Min Zhu was born in 1984. He received the B.S. degree from the Department of Micro & Nano Electronic, Tsinghua University, Beijing, China, in 2006, where he is currently working toward the Ph.D. degree in the Institute of Microelectronics. His research interests include reconfigurable computing and multimedia processing.



Jun Yang received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1999, 2001, and 2004, respectively, all in electronic engineering. He is currently a research fellow and the chairman of SoC department of National ASIC system Engineering Research Center (CNASIC), Southeast University. His research interests include chip architecture design and VLSI design.

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an Associate Professor in Institute of Microelectronics, Tsinghua University. His research interests include Reconfigurable Computing, Mobile Computing and VLSI DSP.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.

Longxing Shi received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1984, 1987, and 1992, respectively, all in electronic engineering. He is currently a Professor and the Dean of Integrated Circuit (IC) College, Southeast University. His research interests include system-on-a-chip design, VLSI design, and power IC design.