SUMMARY

This paper proposes a novel sub-architecture to optimize the data flow of REMUS-II (REconfigurable Multimedia System 2), a dynamically coarse grain reconfigurable architecture. REMUS-II consists of a μPU (Micro-Processor Unit) and two RPUs (Reconfigurable Processor Unit), which are used to speed up control-intensive tasks and data-intensive tasks respectively. The parallel computing capability and flexibility of REMUS-II makes itself an excellent candidate to process multimedia applications, which require a large amount of memory accesses.

In this paper, we specifically optimize the data flow to deal with those performance-hazard and energy-hungry memory accessing in order to meet the bandwidth requirement of parallel computing. The RPU internal memory could work in multiple modes, like 2D-access mode and transformation mode, according to different multimedia access patterns. This novel design can improve the performance up to 26% compared to traditional on-chip memory. Meanwhile, the block buffer is implemented to optimize the off-chip data flow through reducing off-chip memory accesses, which reducing up to 43% compared to direct DDR access. Based on RTL simulation, REMUS-II can achieve 1080p@30 fps of H.264 High Profile@ Level 4 and High Level MPEG2 at 200 MHz clock frequency. The REMUS-II is implemented into 23.7 mm² silicon on TSMC 65 nm logic process with a 400 MHz maximum working frequency.

key words: REMUS-II, coarse grain reconfigurable architecture, multimedia application, data flow optimization, H.264 HiP

1. Introduction

With the growing demand for portable multimedia applications, such as video camera, wireless videophone, and portable wireless multimedia terminal, more and more handheld portable devices are required to support multimedia processing, and the efficiency of multimedia algorithm is being improved.

General purpose processors can offer sufficient flexibility to deal with various applications while may not provide sufficient performance to deal with complex computation. On the contrary, Application Specific designed Integrated Circuits (ASICs) can offer high performance and reduce power consumption of a range of specific applications at the expense of flexibility. However, due to new functionalities and features of modern mobile devices, the ASIC solutions have to integrate new IPs to satisfy the new requirements, which bring great burden to the die size. In addition, the streaming characteristic of multimedia process makes ASIC design hardly meet the time-to-market requirement. Therefore, how to provide a high performance System-on-a-Chip (SoC) within a relatively short time, which can meet varies computing demands, becomes a key issue. Coarse-grained reconfigurable architecture (CGRA) has emerged as a suitable solution. Extracting and exploiting parallelism of computing is a good way for computer architects to meet increasingly performance requirement and stringent energy constraints. CGRA is an excellent implementation of instruction level parallelism (ILP), which can be achieved by executing multiple instructions simultaneously with a multi issue processor called Processing Element (PE).

With the streaming of multimedia standards, the compression efficiency and quality of image has been improved. Generally speaking, processing high quality multimedia streaming data, such as H.264/AVS developed by ITU-T and ISO [1], requires a large amount of memory accesses, which is a serious problem for the total performance as well as power consumption. Thanks to the 2D calculation characteristics and increased calculation capability of reconfigurable array, CGRA can offer sufficient performance for multimedia processing, while propose higher challenge to the design of bandwidth and latency of memory hierarchy.

This paper introduces a CGRA named REMUS-II, short for REconfigurable MUltimedia System 2, to process multimedia data-streams, such as H.264 and MPEG2. The original version [2], REMUS-I, is based on transaction-level simulation and mainly focuses on reconfigurable array design. The REMUS-I calculation unit, Processing Element Array, has the ability to decode H.264 HiP in evaluation. However, the memory architecture design of REMUS-I cannot meet the requirement of high throughput as well as large bandwidth in multimedia algorithm processing, and become bottleneck of the system performance. Therefore, REMUS-II mainly focuses on the improvement of data flow and corresponding control flow to optimize the SoC performance. This paper mainly focuses on the data flow optimization.

REMUS-II is composed of a data-intensive processing unit RPU (Reconfigurable Processor Unit), a control-intensive processing unit μPU (Micro-Processor Unit), a main processor ARM7TDMI and some assistant modules. The data–path of REMUS-II is specifically optimized to meet the access patterns of multimedia data. We implement multi-modes RPU internal memory (RIM), which supports 2D block and transposition access modes. The block buffers are designed to reduce off-chip memory accesses. Experiment results show that, 1080p@40 fps High Level MPEG2
and 1080p@32 fps of H.264 HiP@ Level 4 can be achieved on final architecture based on RTL simulation. REMUS-II is implemented into 23.7 mm² silicon on TSMC’s 65 nm logic process.

The remainder of this paper is structured as follows. Section 2 acknowledges related works in this field. Section 3 details the top architecture, from hardware design and software schedule respectively. The data flow optimization is shown in Sect. 4, including multi-modes RIM and block buffer mechanism for on-chip and off-chip memory respectively. Section 5 reports experimental results. Section 6 concludes the paper and presents future work.

2. Related Works

As CMOS technology continues to scale down, it is feasible to design SoCs with greater data processing capability, such as parallel computing. Many architectures, such as VLIW [3], Stream processor [4], vector processor [5], represent the evolution of the traditional processor architectures toward scalable solutions to achieve high data processing efficiency. The computational mode of these architectures is based on single-instruction-multiple-data (SIMD). As a highly parallel architecture, CGRAs are deeply influenced by these parallel computing architectures.

According to R. Hartenstein [6], CGRA refers to a class of reconfigurable architectures that provide operator level basic configurable block, word level data paths as well as of reconfigurable architectures that provide operator level basic configurable block, word level data paths as well as powerful and very area-efficient routing switches. CGRAs partly originated from fine-grained reconfigurable architectures (FGRA) represented by FPGAs. However, FGRA provides too much flexibility by allowing bit-level reconfiguration. In typical multimedia and telecommunication applications, the basic data types are usually 32-bit, 16-bit or 8-bit instead of 1-bit, so fine grain reconfigurable architectures are much less efficient than coarse grain ones [7].

Many CGRAs were proposed in last decades, such as MorphoSys [8], RaPiD [9], PipeRench [10], EGRA [11]. Efficiently mapping H.264/AVC decoder onto a flexible platform presents a big challenge to existing architectures and design methodology. PACT-XPP is a commercial reconfigurable architecture designed for multimedia and telecommunication applications [12]. The basic units of XPP include ALU-PAEs (Processing Array Elements) and RAM-PAEs. The data width is 24-bit. The XPP typically comprises dozens of these units connected by row-based routing channels. They map part of H.264 algorithm on XPP and the overall performance gain is up to 7.4x [13]. H.264 HD (1920×1080@24 fps) decoding is mapped on the XPP-III architecture [14]. The ADRES [3] consists of functional units (FU) and register files (RF), which are connected in a certain topology. The architecture has two functional views: a VLIW processor and a reconfigurable array, to accelerate the non-kernel code and kernel code separately. In [15], the ADRES achieve 4.2 time speed-up for H.264 algorithm kernels and 1.9 times speed-up for overall performance over an 8-issue VLIW.

Memory subsystem, which is always the performance bottleneck of SoC, is more important in CGRA since its powerful computing engineer requires higher throughput and wider bandwidth. The memory architecture usually includes on-chip memory and off-chip memory. Many methods have been introduced to optimize performance of on-chip memory. ADRES [3] uses rotated register as its Distribute Register File to reduce off-chip memory access. Each row of the computer elements of Zippy [16] can access a shared ROM memory through horizontal bus, the cell can read/write data from/to the FIFOs via bus. XPP[13] implements RAM-PAM as its distributed memory, located at the left and right side of the ALU-PAEs. In [17], they introduces data exchange buffer between main system and reconfigurable array. On-chip memory optimization could reduce off-chip memory accessing times effectively and improve the performance accordingly.

Off-chip memory design is also essential. An automatic bus matrix synthesis flow of a reconfigurable architecture is proposed in [18]. The external DRAM is implemented as off-chip memory to store encoded (or reconstructed) images, and the others are allocated in the internal SRAM. A DRAM controller is designed, containing an SRAM buffer which functions as a level 2 cache. However, the FPGA simulation result shows it is 2 times slower than expected since they underestimate the accessing time from external DRAM.

Since the data path is always the bottleneck of the whole system, the data flow of REMUS-II is optimized to meet the challenge of the high data throughput requirement of multimedia applications, which is introduced in this paper.

3. Target Architecture

REMUS-II, our scalable coarse-grained reconfigurable architecture will be introduced in this section with hardware design specifications as well as software scheduling.

3.1 Top Architecture

Compared with REMUS-I, REMUS-II primarily focuses on the performance improvement of the data flow and control flow processing. This paper mainly introduces the data flow part. REMUS-II contains a μPU to deal with control intensive processes and two RPUs to process the streaming data. The reconfigurable SoC also including an ARM7TDMI, which is employed for application control and reconfiguration scheduling, and some assistant modules, such as a Direct Memory Access Controller (DMAC), an Interrupt Controller (IntCtl), a SRAM memory, an External Memory Interface (EMI). These modules are connected to the AMBA2.0 AHB bus. The top architecture is shown in Fig. 1.

The RPU0 and RPU1 can work in sequence synchronous mode or parallel synchronous mode. Sequence mode means the two RPUs have different tasks, and one RPU starts processing after the results of another has trans-
ferred to the Exchange Memory (EM), such as the schedule of H.264, while parallel mode means the RPU0 and RPU1 have the same task, such as the schedule of MPEG2.

Each RPU contains a 16x16 PEA (Processing Element Array) which is composed of four 8x8 RCA (Reconfigurable Cell Array). The PE adopts the common ALU architecture with arithmetic operations and logical operations. Crossbars interconnection architecture is implemented as the data-flow route between different PEs in each RCA.

The μPU is implemented to handle control flow and generate configuration words dynamically. The μPU contains μPEA (microprocessor element array) and SPA (stream processor array). The SPA contains 2 stream processors to keep entropy decoding efficiently. The μPEA, on the other hand, will generate configuration information according to the information of MB (Macro Block) generated by SPA, such as block type, size, motion estimation type, and et al. Most of multimedia algorithms contain a large number of if-then-else control flows, for example, there are 7 different types of MBs and each kind of MBs has 16 Motion Compensation (MC) cases and 8 De-blocking cases. It is difficult to deal with such complex situations with single master processor such as ARM11 or other higher performance processor, so we design μPU to handle this problem. Similarly, ADRES employs tightly coupled VLIW processor [15] and XPP uses Function PAEs array [14] to handle complex control flow processing of H.264.

3.2 Memory Hierarchy

To satisfy the performance requirement and control the energy budget of REMUS-II, we implement hierarchical memory architecture, shown in Fig. 1.

The RPU memory: The input and output FIFO is the interface of data flow and RCA. Each PE can get data from input FIFO and store data to output FIFO within a RCA. The input/output FIFOs can transfer data from/to RIM or EM. RIMs and EM process intra-data and inter-data exchange of RPU separately.

The on-chip memory: A SRAM is connected to the AHB bus as on-chip memory. The main purpose of SRAM is to store the results of SPA, such as IDCT information. The RIMs and ARM7TDMI can access SRAM though AHB bus.

The off-chip memory: Because of their large volumes at commodity, DDR SDRAM chip [19] is selected as off-chip memory in our reconfigurable SoC. The reference frames of H.264/MPEG2 are store in off-chip DDR. In the reconfigurable SoC of REMUS-II, we built a local bus between EM and RPU to avoid AHB competition.

The RIMs play an important role in the data flow. It can exchange data with FIFOs, SRAM and off-chip DDR. There are two critical paths of the data flow:

1. From RIM to input FIFOs: Many sub algorithms always access memory in blocks. In some cases, the data of block need transpose during computing. Therefore, the RIMs are designed to support multi-modes in order to meet the access patterns of multimedia streaming data and optimize the data flow. The RIM modes are detailed in Sect. 4.1.

2. From DDR to RIM: The row organized character of DDR makes block data transfers from DDR to RIM inefficiently. In order to solve this problem, we design block buffers to cache data in blocks. The block buffers composed of a memory and associated control logic, which can be configured according to several parameters. When accessing DDR, the control logic will judge whether the data required is buffered. If so, the data will be transferred immediately through buffers. The block buffers, detailed in Sect. 4.2, can optimize the data flow through reducing DDR access times.

3.3 Software Schedule

We take H.264 as an example to illustrate the software scheduling procedure and the data flow between memories. The scheduling is shown in Fig. 2 while the black dotted line in Fig. 1 shows the data flow of H.264.

Step1: the SPA of μPU entropy decodes the original multimedia stream. Notice that, the RPU process the application as MBs while the entropy process the stream at slice [20]. Some of the outputs are written to the SRAM memory as data flow ①, other information including control flow is sent to μPEA.

Step2: the RPU0 processes MC and IDCT. Since the RCA is 8 × 8, the 16 × 16 MB is divided into four 8 × 8 MBs. The line ② shows the data flow of RPU0: the MC will access reference frame in DDR through block buffer, while the SRAM data is prepared for IDCT.

Step3: the RPU1 processes de-blocking. Part data of de-blocking is the result of RPU0, which transfer to RPU1 through EM; other data are in the frame buffer, shown in lines ③. After processing, the data will be written back to DDR, shown in line ④.

From the software schedule view, the time of data flow is hidden. That is because we separate PE operations (including the configuration of PE, routing, input and output data of FIFO) and memory operations (identifying the data
flow between memories). The memory configuration is also divided into two levels: on-chip memory access and off-chip memory access. The on-chip memory access includes data transfer between RIM/EM and RCA while the off-chip one includes data transfer between SRAM/DDR and RIM. Figure 3 shows the time chart of data transfer and execution of RPU1 in the de-blocking of H.264. After sent to RPU through FIFO Write Channel, the memory operation configuration would be decode immediately to pre-fetch data to RIM and then write data to input FIFOs (if not full). On the contrary, the PE operation configuration will be decoded after previous operation is completed. The separated configuration benefits not only the information compression but also the data pre-fetching.

This design makes LOAD, EXECUTE, STORE three steps pipeline possible and hides memory operation time correspondingly.

However, the data flow also need to be optimized to fit the pipeline, otherwise the RCA may be stalled to wait for the required data when the memory access time is longer than execution time.

4. Data Flow Optimization

The total performance of REMUS-II is limited by memory bandwidth rather than processing resources especially when data-flow is parallel and pipelined. Some software method, such as data re-organization, can be used to reduce memory access. Furthermore, hardware mechanism is also introduced in REMUS-II to alleviate memory bandwidth limitations.

We are about to optimize the data flow from the following aspect. First of all, the target applications of our reconfigurable architecture are multimedia applications, such as H.264 and MPEG2. Therefore, analysis of data access pattern of these algorithms is essential for optimization. Secondly, the features of the off-chip memory should be taken into account. Finally, hierarchical memory is designed to support load/store pipeline effectively.

4.1 On-Chip Data Flow Optimization: Multi-Modes RIM

RIM is composed of not only memory but also corresponding control logic and dedicated DMA, as shown in Fig. 4. RIM includes five separate memory blocks. RIM 0–4 can exchange data within four RCAs separately and simultaneously while RIM 5 is implemented to swap data among RIM 0–4. The internal data width of RIM is 256 bits, which can offer sufficient throughput for computing.

Multimedia applications always access memory in blocks and transposition blocks [20]. The Table 1 shows statistics of transposition times needed of one MB in H.264 sub-algorithms.
Table 1  Transposition times of different cases in H.264.

| Transposition times | Case Name               |
|---------------------|-------------------------|
| 0                   | MC_00,01,02,03, IDCT4, MC_1(Intra) |
| 2                   | MC_10,11,12,13,20,22,30,31,32,33 Deblocking(all), IDCT8 |
| 4                   | MC_21,23,               |

Table 2  Profit of transposition mode of RIM.

| Sub-algorithm | Case   | Basic | Transpose | Ratio |
|---------------|--------|-------|-----------|-------|
| MC            | P_8x8_02 | 97cycle | 97cycle  | 0%    |
| MC            | P_8x8_20  | 496cycle | 382cycle | 23%   |
| MC            | B 4x4     | 873cycle | 646cycle | 26%   |
| Deblocking    | luma     | 774cycle | 656cycle | 15%   |
| Deblocking    | chroma_intra | 272cycle | 254cycle | 10%   |
| IDCT          | IDCT8    | 199cycle | 165cycle | 17%   |

Fig. 5  Four access modes of RIM. (a) basic mode, (b) single 2D mode, (c) multi-2D mode, (d) transpose mode.

The control logics of RIMs are designed to support multiple modes, shown in Fig. 5 (a)–(d). The figure shows how the data is transformed from RIM to input FIFO. While the data transform from output FIFO to RIM after processing, there are also corresponding modes.

Figure 5 (a) shows the basic mode, which transfers continuous data to input FIFO. The parameter start address and the length of data are needed in this mode.

RIM supports two kinds of 2D data access modes shown in Fig. 5 (b) and (c). Single 2D mode and multi-2D mode separately. In single 2D mode, the parameters of start address and offset are used to identify the block location, length and height define the block shape, and the joint parameter defines how the block lines are sent to FIFO line. In multi 2D mode, the block number is a parameter additionally defining how many different blocks should combine together. These two types of 2D modes transform the data from irregular format to regular one, which can improve the utilization of the 256-bit width FIFOs. The FIFO utilization is defined as the ratio of valid data and FIFO line size. For example, in basic mode, a 16 × 16 block has 16 × 16 × 8 bit data, which needs 16 FIFO lines of 50% utilization (128 bit/256 bit) and the RCA need 16 cycles to read data from FIFO. By contrast, in single 2D mode, two 128-bit block lines data are combined to one 256-bit line and then sent to the FIFO, so only 8 cycles are needed for RCA to read data. Similarly, RCA needs 4 cycles to read a 4 × 4 block in basic mode while needs only one cycle in single 2D mode.

Figure 5 (d) shows the transposition mode. This mode is similar with single 2D mode, except the data are transferred to the FIFO in columns. Matrix transposition is very common in multimedia applications, many method can deal with this problem, such as using software algorithm [21], occupying FPGA area [22] or PE array [14] to implement transpose. Compared with those methods, transposition mode can complete transposition during transferring data, which is very efficient. We select several cases to exam the performance gain of transposition mode, shown in Table 2. This novel design can improve the performance up to 26% compared to traditional on-chip memory.

The various modes of RIM offer sufficient and efficient data access, which could meet the access pattern of multimedia algorithm and improve the data flow of on-chip memory.

4.2 Off-Chip Data Flow Optimization: Block Buffers

Due to its low price, DDR is nowadays widely used in state-of-art SoCs. DDR is organized in banks and each bank has several linear rows. Accessing data in different rows needs pre-charge and active commands which require much more time than accessing active row [19]. This characteristic of
DDR makes block access far from efficient. It is common to place frame buffer in raster scan order, which may make each line of a MB in different rows of DDR [23].

For example, a frame of H.264 1080p contains 1920 × 1080 pixels and each luminance pixel is 8 bit. Since the DDR bank is 1024-Bytes width [19], each line of the MB is located in different rows of DDR while pixels are placed in raster scan order, shown in Fig. 6. Accessing each line of a 16 × 16 MB need switch active row once (when a line locate in one row) or twice (when a line falls across the edge between two rows).

Multimedia algorithms process the producer-consumer locality with little global data reuse [24], so the traditional caches are largely inefficient in these applications. We design a block buffer to improve the performance.

The motivation of block buffer is: Since accessing the data of active DDR row is much faster than inactive one, the block buffer will fetch not only the target MB but also the adjacent data. If all data of next MB are buffered, the hardware will access the block buffer rather than DDR.

Figure 7 (a) shows the hardware view of the block buffer module. It includes two buffers, which store forward prediction and backward prediction separately. Each buffer includes a memory block, which can be configured to different logical view, such as one luminance memory and two corresponding chroma memories in H.264 and MPEG2 applications.

When access a MB, such as MB1, shown in Fig. 7 (b), block buffer will fetch its surrounded data, as Block1. When access next MB, the module will judge whether all of the data is already in that buffer. For example, MB2 meets the requirement and do not need to access DDR any more. Part data of MB3 is not in the block, so it needs to update block buffer to Block2 according MB3. Similarly, while accessing MB4, the block buffer needs to be updated.

Hit ratio and the profit ratio are responses to evaluate the benefits of block buffer in Eq. (1) and Eq. (2):

\[ \text{Hit ratio} = \frac{\text{hit num}}{\text{total access num}} \]
\[ \text{profit ratio} = 1 - \frac{\text{optimized access time}}{\text{previous access time}} \]

Hit ratio stands for the chance that the data which will be processed has already in the block buffer. Low hit ratio will hazard the total performance because of the penalty. Profit ratio is more intuitive, that higher value means better profit, zero value means no profit at all, and negative value indicates the buffer decreases system performance.

In order to obtain higher hit ratio as well as profit ratio, there are several important parameters to design block buffer: the offset, the length and the height, shown in Fig. 7 (b). The length and height determine the shape of the block. The offset stands for the position relationship between the MB and the block. For example, suppose the left-top point of MB1 is A(x, y), we use Eq. (3) to calculate the position of block buffer Block2 B(x’, y’). The point B should locate at internal of the frame.

\[ x' = [x - \text{length} \times \text{offset}] \]
\[ y' = [y + \text{height} \times \text{offset}] \]

In order to obtain higher profit, we do design exploration of offset, length and height respectively, shown in Fig. 8 (a)–(c), based on stream forman_gif_main.264. The left pictures show the hit ratio while the right ones show the profit ratio. The line 0 and line 1 stand for forward and backward prediction.

The Fig. 8 (a) shows that, 15%−20% offset is preferable. The design exploration of length indicates that large size obtains better performance, which is shown in the right picture of Fig. 8 (b). However, when the length is larger than 64 Bytes, the performance is improved very slowly. Therefore, the 64-Byte is preferable to obtain trade-off between die size and performance. The right figure in Fig. 8 (c) shows that too large height will compromise performance. When the height large than 32 Byte, the hit ratio hardly increases. However, the profit ratio decreases since fetching data of different lines of MB needs to access new row of DDR.

Finally, we point out the best parameters for forman_gif_main.264: Offset 20%, Height 32-Byte, Length 64-Byte. Since the buffer block is flat-shaped, it is more suitable for horizontal motion predictions rather than vertical ones. Those three parameters are configurable to fit different streams. We tested some streams and the results are shown in Table 3. The block buffer reduces up to 43% accessing time compared to direct DDR access.

The RIM and Block Buffer are designed to optimize the block-based accessing pattern, which is very common.
in multimedia algorithm, especially in video and image processing. Therefore, not only H.264 and MPEG4 but also algorithms like H.263, MPEG2, AVS could take advantages of RIM and Block Buffer.

5. Experimental Results

To evaluate the performance of REMUS-II, a functional RTL model is firstly designed in Verilog and then synthesized in Synopsys DesignCompiler to map onto TSMC 65nm low power technology. The area and timing results are generated by DesignCompiler using worst case conditions.

We will map two different multimedia streams, H.264 and MPEG2, onto REMUS-II to evaluate the performance of proposed architecture. Some typical cases of core algorithms are taken as examples and the results are shown in Table 4.

In order to process real-time 1080p stream, the frames per second (fps) should larger than 30.

\[
\text{fps} = \frac{\text{frequency}}{(\text{Total MB} \times T_{MB})} \tag{4}
\]

According to Eq. (1), while the working frequency is 200 MHz and total MBs of one frame (Total MB) number is \((1920 \times 1080) / (16 \times 16) = 8100\), the time to process an MB \((T_{MB})\) should be limited to 816 cycles.

Some H.264 cases seem to need more than 816 cycles, especially for the \(4 \times 4\) MBs. However, the proportion of \(4 \times 4\) cases is relatively small. For example, \(4 \times 4\) MBs only account to 2.7% of total MBs in stream QCIF. Therefore, most MBs can be processed within 800 cycles and the average execution time of one MB is less than 800 cycles finally. The performances of H.264 and MPEG2 streams are shown in Table 5.

The results show that REMUS-II can process 1080p@32.fps of H.264 High Profile@Level 4, and 1080p@40.fps High Level MPEG2 when exploiting a 200 MHz working frequency.

The die size of REMUS-II is 23.7 mm\(^2\) on TSMC 65 nm logic process. The layout of REMUS-II is given in Fig. 9. Two RPU, one \(\mu\)PU (including SPA and \(\mu\)PEA) and

| Stream | foreman | Van.Helsing | city | mobile |
|--------|---------|-------------|------|--------|
| Algorithm | H.264 | H.264 | MPEG2 | MPEG2 |
| Size | QCIF | 1080p | D1 | CIF |
| Profile | Main | High | Main | Base |
| Hit ratio | 0.94 | 0.90 | 0.93 | 0.92 |
| Profit ratio | 0.43 | 0.32 | 0.37 | 0.35 |

Table 4 The execution time of different cases on REMUS.

| Stream | foreman | Van.Helsing | city | aesshow |
|--------|---------|-------------|------|---------|
| Algorithm | H.264 | H.264 | MPEG2 | MPEG2 |
| size | QCIF | 1080p | D1 | 1080p |
| Profile | High | High | High | High |
| fps | 2674 | 32.4 | 202 | 40.3 |

Table 5 Performances of H.264 and MPEG2 decoding on REMUS-II.
the on-chip SRAM can be recognized clearly.

In order to illustrate the merits of the proposed architecture, we would like to list the performance comparison as well as some relevant parameters of XPP, ADRES and REMUS-II when decoding H.264 streams, shown in Table 6.

The results show that, in general, REMUS-II obtains higher performance and less die size occupation than XPP and ADRES.

6. Conclusions and Future Work

This paper introduces the data flow optimization of a coarse grain reconfigurable architecture named REMUS-II, which is designed for data-parallel and computation-intensive multimedia applications. REMUS-II has passed the verification under veloce-based simulation acceleration. It includes two RPU s to do computation-intensive tasks, a $\mu$PU to deal with control flow and a main processor ARM7TDMI to do scheduling. Thanks to multi-mode internal memory architecture, we could improve the performance of execution up to 26%. Meanwhile, the block buffer is implemented to reduce up to 43% off-chip memory accesses. After optimization, the data flow of REMUS-II can meet the data requirement of computation-intensive array. The experiment results show that, 1080p@30 fps High Level MPEG2 and H.264 HiP@Level 4 can be achieved on final architecture. It has been implemented into 23.7 mm² silicon on TSMC 65 nm logic process.

In the future, we will continue to optimize the energy consumption as well as die size occupation of REMUS-II by reducing the off-chip memory accessing and increasing the on-chip memory usage. Hopefully, new compiling techniques will support our future architecture, and make auto-mapping process and memory re-layout more efficient.

Acknowledgements

This work was supported by the National High Technology Research and Development Program of China (863 Program) (grant no.2009AA011700). The authors would like to thank to J Yang, B Liu, J Xiao, H LEI, JY Yang, YC Lu, YQ Fan, D Wang, CX Zhang, X Wang, YJ Chen, for their helpful discussions and technical support.

References

[1] Recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC) and J. in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2003.
[2] 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.
[3] B. Mei, S. Vernalde, D. Verkest, H.D. Man, and R. Lauwereins, “ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix,” Lect. Notes Comput. Sci., vol.2778/2003, pp.61–70, 2003.
[4] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krueger, A.E. Lefohn, and T.J. Purcell, “A survey of general purpose computation on graphics hardware,” Computer Graphics Forum, vol.26, pp.80–113, 2007.
[5] C. Kozyrakis and D. Patterson, “Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks,” Proc. International Symposium on Microarchitecture (MICRO), pp.283–293, 2002.
[6] P.R.B. De Sutter and A. Lambrechts, “Coarse grain reconfigurable architecture (embedded tutorial),” Proc. 2001 Asia and South Pacific Design Automation Conference, pp.564–570, 2001.
[7] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for deep-submicron FPGAs, Kluwer Academic Publishers, Norwell, MA, USA, 1999.
[8] H. Singh, M.H. Lee, G. Lu, et al., “MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive multimedia applications,” Field Programmable Logic and Applications, New Paradigms and Compilers, pp.126–135, 1996.
[9] 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.
[10] G. Ansaloni, P. Bonzini, and L. Pozzi, “EGRA: A coarse grained reconfigurable architectural template,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.19, no.6, pp.1–13, 2010.
[11] V. Baumgarte, G. Ehlers, F. May, A. Nickel, M. Vorbach, and M. Weinhardt, “PACT XPP: A self-reconfigurable data processing architecture,” J. Supercomputing, vol.26, pp.167–184, 2003.
[13] W.H. Lee and J.H. Kim, “H.264 implementation with embedded reconfigurable architecture,” presented at the Conference on Computer and Information Technology, 2006.

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

[15] B. Mei, FJ. 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.

[16] C. Plessl and M. Platzner, “Zippy-A coarse-grained reconfigurable array with support for hardware virtualization,” IEEE International Conference on Application-Specific Systems, Architecture Processors, pp.213–218, 2005.

[17] F. Campi, A. Deledda, M. Pizzotti, L. Ciccarelli, P. Rolandi, C. Mucci, A. Lodi, A. Vitkovski, and L. Vanzolini, “A dynamically adaptive DSP for heterogeneous reconfigurable platforms,” pp.9–14, 2007.

[18] G. Lee, Y. Ahn, S. Lee, J. Son, K. Yoon, and K. Cho, “Communication architecture design for reconfigurable multimedia SoC platform,” Design Automation for Embedded Systems, vol.14, pp.1–20, 2010.

[19] Micron Technology Inc. Double Data Rate (DDR) SDRAM 256Mb: x4, x16 Preliminary, 2001.

[20] T. Wiegand, G.J. Sullivan, G. Bjontegaard, 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, 2003.

[21] C. Qiao, “Efficient matrix operations in a reconfigurable array with spanning optical buses,” in Frontiers of Massively Parallel Computation, pp.273–280, 1995.

[22] T. Miyamori and K. Olukotun, “A quantitative analysis of reconfigurable coprocessors for multimedia applications,” IEEE Symposium on FPGAs for Custom Computing Machines, pp.2–11, 1998.

[23] J. Probell, “Architecture considerations for multi-format programmable video processors,” J. Signal Process. Syst., vol.50, pp.33–39, 2008.

[24] J.D. Owens, S. Rixner, U.J. Kapasi, P. Mattson, B. Towles, B. Serebrin, and W.J. Dally, “Media processing applications on the Imagine stream processor,” IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp.295–302, 2002.

[25] F. Campi, R. Konig, M. Dreschmann, M. Neukirchner, D. Picard, M. Juttner, E. Schuler, A. Deledda, D. Rossi, and A. Pasini, “RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip,” International Symposium on System-on-Chip, pp.110–113, 2009.

Xinning Liu received B.S. and M.S. degree of electronics engineering from Southeast University, China, in 2000 and 2003 respectively. Now, he is a lecture of the School of Electronic Science and Engineering, Southeast University. His research mainly focuses on the SoC design technology and low power design technology.

Chen Mei received B.S. degree of electronics engineering from Anhui University in 2007 and M.S. degree of Circuit and System from Southeast University in 2010, China. Now, she is working toward the Ph.D. degree in Southeast University. Her research mainly focuses on reconfigurable architecture design, memory optimization of SoC.

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 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.

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.