**Fast Parallel Reconfigurable Computing Architecture**

**for Multi-Standard Video Decoding**

**Chi-Chou Kao**

Department of Computer Science and Information Engineering, National University of Tainan, Tainan, Taiwan

E-mail: cckao@mail.nutn.edu.tw

**Abstract -** Video processing applications often require high computing capacity but have performance and power constraints, especially in portable devices. General purpose processors can no longer meet the requirements. In this paper, a parallel reconfigurable computing architecture consisting of reconfigurable processing units interconnected by an area-efficient routing is proposed. The hierarchical configuration contexts are proposed to reduce the implementation overhead and the energy dissipation spent on fast reconfiguration. The proposed architecture targets multiple-standard video processing. The design is able to provide high performance comparable to the fixed function ASICs through deep pipelining and large amount of computing parallelism. The experimental results demonstrate the proposed architecture has great performance and practicability.

**Keywords -** reconfigurable processing, performance, power, parallel, multiple-standard video processing.

# INTRODUCTION

Embedded devices including TV, TV games, digital cameras, tablets, smart phones are indispensable tools. Most of embedded devices need for fast calculations, lower power consumption, and functional requirements under limited resources. In the multimedia applications of embedded devices, codec is used for image storage and transmission. However, the codec must be adjusted to different requirements and conditions, including transmission speed, delay and bandwidth, image quality requirements, resolution, color. The bit rate, the picture update rate per second (FPS) and so on. Although the codec can be defined according to the usage requirements, it makes many difficulties in communication. Hence, various standards such as H.263, H.264, H.265, MPEG-2, MPEG-4, and AVS have been developed [1]. In the general architecture for various video decoding, image compression first converts color from RGB to YcbCr and divides into blocks for motion compensation and discrete cosine transform. Next, quantization (entropy) and entropy coding are performed to generate output. Finally, we can get the video decoding based on the reverse operation of compression. Obviously, the video codec requires a large of computation, fast calculation speed, and real time. With the rapid development of mobile devices, we must improve computing power and performance to meet energy consumption, cost, and flexibility.

Traditionally, deep pipelines and process shrinking are key technologies to improve system performance. The process shrinking is an efficient way to reduce crystal size for improved performance and integration. However, due to advances in process technology, the delay of the flip-flop has approached the delay of the combinational logic, making the benefits of multi-pipelined from the instruction level less and less important for frequency improvement. It makes high power consumption when a microprocessor is used in video processing. Therefore, using a microprocessor as a video processing architecture is not a good choice. The digital signal processors (DSPs) are often designed to provide high computational performance in digital signal applications. The DSPs provide high bandwidth and dedicated computing resources, such as multipliers and accumulators. The TI 8-way Very Long Instruction Word (VLIW) TMS320C64X DSP [2] can achieve up to 8k mega-instructions per second (MIPS). It shows that DSPs provide energy efficiency gain pipeline lengths, less communication overhead, and more parallel advantageous architectures compared to microprocessors. However, the VLIW DSP cannot be used in a large number of processing units because it shared control structure uses global register files for data sharing between different processing units. The systolic array architectures [3] in which data processing units (DPUs) are connected in a layout with local arrays can be used in video processing. The DPU operations are handled by conventional data flow scheduling in the pipeline. The input/output pipelines and concurrent data can support high throughput. The systolic array architectures provide high system scalability due to their regular structure. However, it has high cost and complex hardware and software implementation drawbacks.

The chip multi-processor (CMP) has been proposed as a general-purpose processor architecture design and can achieve higher performance and lower frequencies [4]. The CMP can provide high energy efficiency processor system and has low operating frequency and supply voltage. However, the CMP requires a large of control logic and memory resulting in poor scalability. In addition, the CMP architectures cannot be used for parallelism because of the limited parallel computational processing. The number of processor cores that integrate large multi-core systems on a single wafer has been developed. Compared to CMP, multi-core systems typically involve more processing of cells with a particular communication structure. For example, the Intel 80-core system [5] integrates 80 wafers in a 10×8 2D grid structure. Each chip contains a computing component and a 5-port router unit. Dynamic messaging protocols are implemented to provide high speed and network-on-a-chip data communications, which is more scalable than CMP interconnects. The multi-core architecture can achieve high computing power for data flow applications. But the multi-core systems have high power consumption such that it does not meet the power requirements of portable devices. The graphics processing units (GPUs) have a large number of integrated streaming processors to achieve high computing power [6] but power consumption is still a problem. Therefore, it is necessary to develop innovative architectures for performance efficiency gaps between fixed-function logic.

Among many solutions, DSPs are easy to program and have relatively low power consumption, but the computational complexity is limited by the Von Neumann bottleneck [7]. The general purpose processor (GPP) is easy to use and flexible, but the performance is not satisfactory. The application special integrated circuits (ASICs) provide the highest performance, but the structure of the circuit cannot be changed. The Field Programmable Gate Arrays (FPGAs) offer parallelism and a high flexibility, but have high power and can only be configured at boot time [8]. None of these architectures can balance performance and resiliency in hardware resources. The use of Reconfigurable computing has been suggested as a way to balance performance, cost, and flexibility [9]. However, because the access memory must pass through the public bus, performance can be delayed. The problem can be solved by using parallel processing techniques of multiple instructions (multiple instructions, multiple data, MIMD). The MIMD has multiple processors and I/O processors that can access one or more memories through a common bus. The parallel reconfigurable computing (PRC) architecture combine reconfigurable computing with parallel processing technologies such that each processor configured with a memory that reduces the number of accesses to the bus. Figure 1 shows an architectural diagram of a parallel reconfigurable computing. The architecture which consists of reconfigurable processing units (RPUs) can optimize further performance of system [10]. The PRC architecture is sufficient to handle high computation and data intensive applications such as MPEG-4 or H.264 video encoder [11-12].

The remainder of this paper is organized as follows. Section 2 gives design analysis and describe related works. Section 3 presents the details of our architecture. Section 4 presents the experimental results. Conclusions can be found in Section 5.

# Proposed Parallel Reconfigurable Computing Architecture

The proposed parallel reconfigurable computing (PRC) architecture consists of five main components: reconfigurable processing unit array (RPUA), input, output, inner memory, configuration stream storage and controlling logic. The first four parts consist of a data processing path, and the latter form a configuration path as shown in Figure 4. The operation of the RPU is driven by the inputs. When the streams is pushed into the stream storage, the control logic first translates and reorganizes the stream, and then configures the input or memory to prepare the data. At the same time, the RPU is reconfigured to the required functional structure. Once the stream is ready, the stream is distributed to the memory to retrieve, process, store, and output.

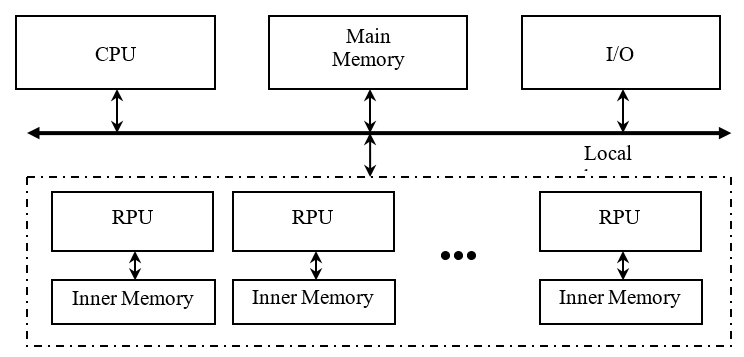


Fig. 1. Parallel reconfigurable computing architecture.

1. **Data Processing Path**

The data processing path is responsible for the fetching, processing, storing and exporting of the stream. The core part of the data processing path is the Reconfigurable Processing Unit Array (RPUA). The RPUA is organized in four RPUs, which contains reconfigurable Processing Elements (PEs). Each RPU can work independently thus providing a higher level of parallelism to improve the throughput. Moreover, certain RPUs can also be turned off to maintain very low power consumption.

1. **Reconfigurable Processing Unit**

Figure 2 depicts the reconfigurable processing unit architecture. Each cell block consists of four processing elements (PEs) along with the functional control and data switch. The PE can be configured to perform basic logic, shift and arithmetic functions. Multiple PEs can be chained together for more complex tasks. An interconnection network is designed for the processer communications, which includes nearest neighbor connection among adjacent cells and a hierarchical network for non-adjacent cells. Inner memory is designed for each PE to store the configuration contexts for both computing and communication. A serial peripheral interface (SPI) is designed that chains the instruction memories in a linear array fashion to efficiently load instruction contexts into active processing units.

In the RPU design, we integrate multiple processors to partition tasks onto different computing resources and process in parallel for high system performance. The temporary results from one processor can be forwarded directly to the next one through interconnections, which reduces memory utilization and requirements. The design can avoid high execution overhead such that it can achieve high performance and energy efficiency. Each PE has the full visibility of the temporary results from all PEs in the same cell. The cells are grouped into clusters with shared components. This hierarchical interconnection efficiently alleviates the long wire delay impacts meanwhile maintaining good scalability for different system dimensions. The design is well suited to the stream processing applications, characterized with continuous data flow from one processor to another without much data feedback and reutilization.

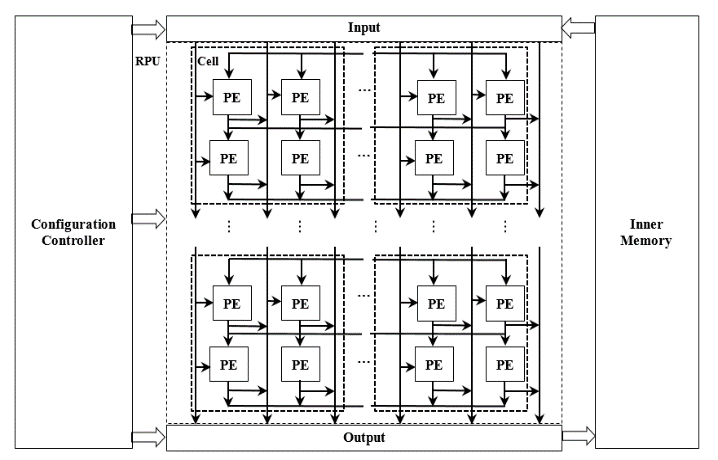


Fig. 2. Reconfigurable processing unit architecture.

Due to the rich computing and communication resources, numerous computing architectures can be mapped onto the architecture, including SIMD, MIMD, and systolic array structures. Moreover, by loading new instruction codes into the configuration memory through the SPI structure, new operations can be executed on the desired PEs without interrupting other PEs. The number of PEs involved in the application is also adjustable for different system requirements. In the design, defective cells, caused by manufacturing fault or malfunctioned circuits, can be easily turned off and isolated from the functional ones to achieve good fault tolerance. A program counter (PC) is designed to schedule instruction execution time for each PE. Variant delays are also available for input/output signals inside each PE. Therefore, the design can provide explicit synchronization that eases the exploration of computing parallelisms.

1. **Processing Element**

The PE is composed of an arithmetic logic unit (ALU), a group of input/output (Regs), temporary result registers (T\_Regs), I/O MUX, instruction controller, and instruction memories, as shown in Figure 3. It can be configured to perform basic logic, shift and arithmetic functions and operators. Multiple PEs can be chained together through the programmable on-chip connections to implement more complex algorithms.

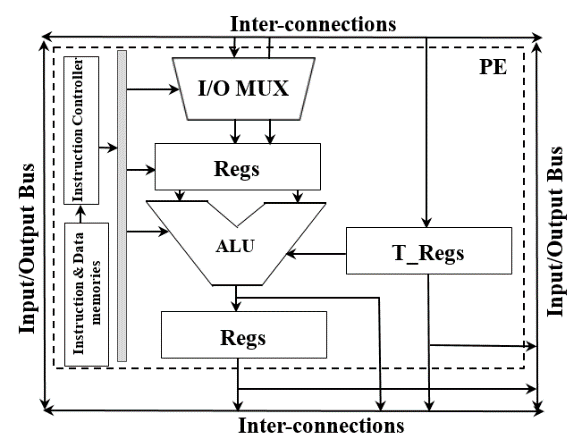


Fig. 3. Processing element structure.

In the PE, an instruction code, pre-stored into the instruction memory, is loaded into the instruction controller on a cycle by cycle to provide both functionality and datapath control for a specific algorithm. Additionally, the instruction code can be dynamically reconfigured in various modes to different application requirements. Therefore, the design can provide comparable energy efficiency while maintaining dynamic programmability. The datapath and operation control signals specify the configuration of data flow and computing units, while the I/O delays are used for synchronization scheduling among multiple computing units. The instructions are accessed in a cyclic manner that supports periodical execution of a set of operations. We can find that the multiplication related operations do not appear often in video decoding applications. Therefore, only the ALUs contain fixed-point multipliers such that the total area of the PEs is reduced largely. To reduce the power usage on continuous toggling of the configuration registers, each bit of the fixed-length operation code is designed to disable the toggling of the configuration registers of the corresponding arithmetic or logic circuits in the ALU when a certain bit of the fixed-length operation code is zero. The transition probabilities of each of bits are kept in minimum so the power consumption of ALUs can be further reduced.

1. **Reconfigurable Interconnections**

As the CMOS technology scaling down, interconnect has become an increasingly important issue for integrated circuit design. In many signal processing applications, the system throughput is significantly affected by communication costs. The shared bus connections with high bandwidth are usually adopted in modern multicore CPU designs. The bus topology is simple, but the lack of scalability and high power consumption and timing penalty make it not attractive. In our structure, the PEs within each RPU are organized in a line-to-line manner, i.e., each PE could be connected to any PEs in the adjacent upper and lower lines through the inter-layer mesh interconnections as shown in Figure 2. For I/O data, the RPUA is connected to a pair of I/O through an I/O bus. Unlike the traditional full mesh connection scheme, where every PEs within different lines are equally connected to the I/O data ports. In the proposed PRC structure, only one particular line of PEs and a corresponding line of data ports are selected and connected to the I/O bus at one configuration. In our design, the homogeneous cell units are tiled in a 2D mesh structure. Thus the adjacent cells can be connected directly through short wires. Since four PEs in a cell are placed at one cell, each PE can be directly linked to the nearest PE located in the adjacent cell.

1. **Configuration Path**

The instruction code determines what operation each PE performs and how data is routed among multiple PEs. The initial instructions are loaded to the memory at compile time and can be updated to accommodate new applications or performance requirements at run time. A serial peripheral interface (SPI) is designed to configure the instruction memories as shown in Figure 2. In this structure, the instruction memories are linked in a ripple array fashion with the inputs and outputs chained one to another. At cell level, each cell is able to receive one instruction code from the previous cell and forward it to the next cell after local propagation. Because of only one unified configuration port, it eliminates a large amount of global configuring wires to provide better performance and scalability.

In the initial configuration procedure, the instruction contexts are loaded into the first PEs instruction memory and is then shifted down to the second one and so on. This procedure stops after the last active PE is configured. The new instruction code and the ID of the PE to be configured are sent into the instruction memory. The PE bypasses the information to the next one if the transmitted ID doesn’t match the ID. This procedure continues until it reaches the desired PE. By this means, only the execution of the reconfigured PE is temporarily suspended. The other PEs remain unaffected and keep operating during the reconfiguration procedure. Figure 4 depicts the dynamic control flow for a processing unit. At run time, a configuration context is loaded into the instruction register and is then partitioned into interconnection and functionality controls. In general, each instruction selects two operands from local registers or inputs. The basic arithmetic, logic and shift operations are then executed concurrently on these operands. Finally, one result is selected and forwarded to the output.

We organize the context for the proposed architecture in three levels. The first level of context contains the ALU OP codes, the inter-interconnection configuration bits and the data loading/storing commands. By using a group of OP codes, the RPU can be configured to execute a specific operation. The configuration bits for the inter-interconnections control the flow of the processed data stream. The data loading/storing commands for the input/output interfaces select the data sources for each RPU. The indexes of the core contexts are grouped into the second level. It represents a serial of data-dependent RPU level calculations. The second level is used for certain applications, such as an iterative algorithm, the computation requires similar operations and the processed data may be repeatedly used, which means that the RPU is likely to be reconfigured using the same structures. The highest level is an extension of the group context with synchronization command to control the flow of the implemented algorithm. The complete contexts are dynamically generated by the host processor and will not be stored within the RPU.

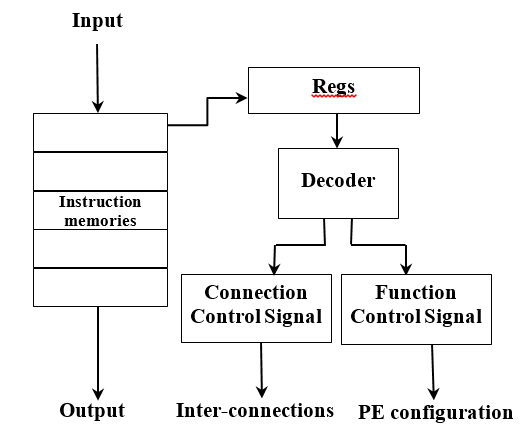


Fig. 4. Configuration controller architecture.

The reconfigurable computing tasks will result in most of the energy consumption and a large of configuration time, so a good reconfigurable strategy to reduce energy consumption and improve performance is necessary.

1. **Video Decoding Mapping**

In our design, the data processing and configuration path have high level of flexibility such that multimedia computation intensive tasks can also be efficiently executed on RPU. The PEs design can be considered as the ALUs in general-purpose CPUs that support a wide range of instructions. The interconnection of the PE array is a non-blocking network, which enables the data communication between PEs within one RPU. Since the inner loop only contains one operation, i.e., multiply-add, and there are no data dependences between successive operations, the inner loop can be unfolded to fully exploit parallelism. The proposed interconnection scheme and the multi-level buffering structure can efficiently reduce the interval time between successive pipelines, which maximize the data throughput and the utilization of the PEs. Moreover, the PE only needs one-time configuration by rearranging the input data stream to further boost the execution of the kernel. The input and output data are updated in the same cycle without any dependences. Therefore, the RPU can be efficiently utilized to execute the computation tasks. The proposed configuration procedure significantly reduced the configuration time of the RPU. Generally speaking, the key of improving the computation and energy efficiency is to maintain a very high utilization of the hardware resources of the PEs.

The stream processing tasks, including intra-prediction, IDCT, and deblocking filtering, are processed in parallel within each Macro Block (MB). The primary performance constraint used is the average decoding speed per MB. In high video stream, one frame consists of 8160 MBs. Assuming the operating frequency is 200 MHz, the equivalent average decoding performance requirement for processing one MB is around 816 cycles. It can be seen that the deblocking filtering task requires the longest processing time. For most cases, it is even larger than the sum of the prediction time and the IDCT time. Hence, we decide to allocate independent hardware resources, i.e., two identical RPUs to implement deblocking filtering processing in parallel to improve the decoding efficiency.

# Experimental Results

The stream processing tasks, including intra-prediction, IDCT, and deblocking filtering, are processed in parallel within each Macro Block (MB). The primary performance constraint used is the average decoding speed per MB. In high video stream, one frame consists of 8160 MBs. Assuming the operating frequency is 200 MHz, the equivalent average decoding performance requirement for processing one MB is around 816 cycles. It can be seen that the deblocking filtering task requires the longest processing time. For most cases, it is even larger than the sum of the prediction time and the IDCT time. Hence, we decide to allocate independent hardware resources, i.e., two identical RPUs to implement deblocking filtering processing in parallel to improve the decoding efficiency.

The benchmark applications listed in Table 1 have been manually mapped onto the PRC design. These benchmarks represent a wide range of real-time applications from signal/image processing to scientific computing. For power consumption and system throughput evaluations, all benchmarks are simulated at the same operating frequency of 100 MHz. The same benchmarks are also directly implemented on the FPGA platform to provide performance comparison. The Altera Stratix II FPGA is selected as the benchmark platform. The benchmarks are designed in Altera Quartus II CAD tool. The PowerPlay Analyzer is used to evaluate the power consumption based on the switching annotation generated from the gate level simulations. The ASIC implement is also generated for every test bench using the same HDL code in the FPGA designs.

Table 2 lists the power consumption performance for benchmark applications. All figures are generated at 100 MHz. Figure 5 shows the power dissipation for different components of the PRC system. The processing units consume about 42% of total power. The memories and interconnections consume about 51% of total power. On average, the design consumes about 147 *m*W at 100 MHz. The energy efficiency, evaluated by the total number of operations per second per watt, is also calculated, as shown in Table 4. The architecture provides an average 31.8 GOPS/W energy efficiency from all benchmarks. The matrix multiplication shows only 10.1 GOPS/W energy efficiency. This is because the systolic array mapping scheme only uses 1/4 of the computing resources, while the other PEs still consumes power to bypass data.

Table 1. Application and benchmarks mapped onto the PRC system.

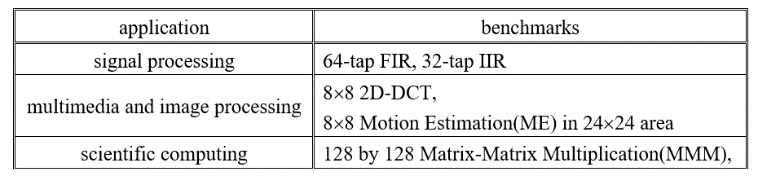
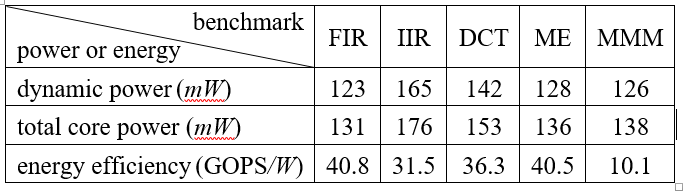


Table 2. Power consumption of different benchmarks.



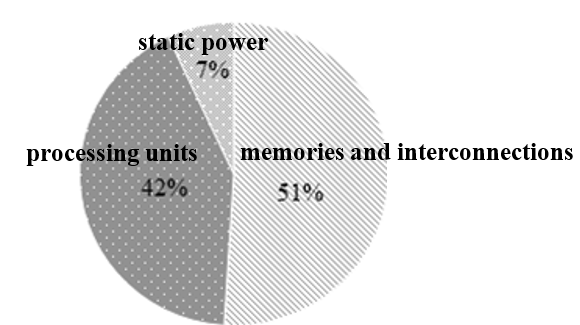


Fig. 5. Average power consumption

We have compared the PRC architecture power consumption with the computing platforms, FPGA, and reconfigurable architecture, RaPiD [13]. The PRC is about 360% less power consumption than FPGA. On the other hand, the PRC is about 210% less than RaPiD in power consumption. The results demonstrate that in most applications, the PRC architecture requires less clock cycles to finish the same amount of task comparing with FPGA and RaPiD implementations. This is benefitted from more processing parallelism provided by the PRC architecture with reduced computational and configuration complexity. The RaPiD integrated heterogeneous computing and data storage components, in which case the power consumption may not be well balanced among different modules. Its segmented interconnections and single control decoder may also involve high power consumption.

In our architecture, we use the reconfigurable processing unit (RPU) as the basic unit. In order to understand the parallelization effect, the decoding performance of the difference number of RPUs is compared with different resolutions as shown in Table 3. It can be seen that the more RPUs, the better decoding performance. The performance of four RPUs increases 2~3 times than the performance of one RPU, while the performance of eight RPUs is only increased about 4% than the performance of four RPUs. It is the reason that we use four RPUs to form one cell. We have compared the video decoding performance with ADRES [14] in Table 4. From the measured data, one can clearly see that the PRC design achieves a 41.8% performance boost than the ADRES design.

Table 3. Comparison of parallelization effect.

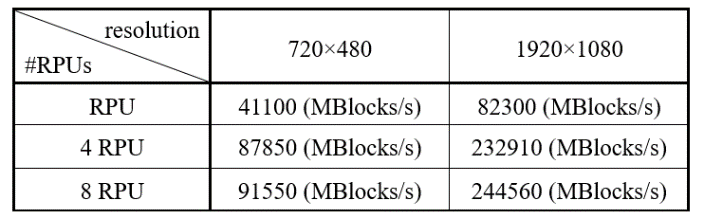
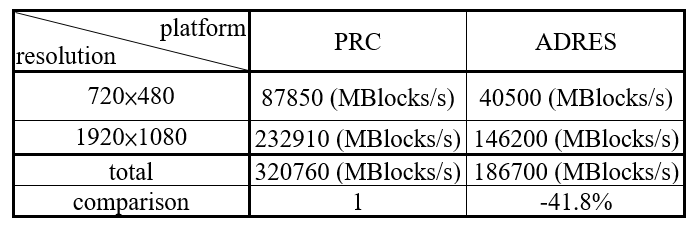


Table 4. Comparison of decoding performance.



# Conclusion

This paper has presented a performance-driven parallel reconfigurable computing architecture. The design integrates a large number of homogenous cell units in a 2D structure. In each cell, four processing units are attached with their own instruction and data memories. A hierarchical configurable network has been developed in the architecture, which includes three level of interconnections. The cell broadcasting and ID-based configurations were also developed for dynamic reconfiguration to address various control requirements. The architecture is flexible to exploit deep pipeline and large amount of parallelism with various operation modes. In combination of these features, the design can achieve high energy efficiency while maintaining high computational performance for the computing intensive data streaming applications. To achieve best performance, the architecture was designed to operate under various modes in these benchmarks, including pipelined structure, SIMD mode, systolic array structures. The notable improvements on both performance and energy efficiency have proved that the PRC design is one of the ideal choices for implementing multiple standard video decoding applications. The performance and energy efficiency are verified and compared favorably with the general-purpose processors and FPGAs.

# ACKNOWLEDGMENT

I would like to thank the Ministry of Science and Technology, Taiwan, MOST 106-2221-E-024-005 to support this work.

# REFERENCES

1. Dhiman Chattopadhyay, Aniruddha Sinha, and T. Chattopadhyay, “A Low Cost Multiparty H.264 Based Video Conference Solution for Corporate Environment,” International Conference on Computational Intelligence and Communication Networks, pp. 68-72, 2010.
2. Mounir Bahtat, Said Belkouch, Philippe Elleaume, and Philippe Le Gall, “Efficient implementation scheme of a real-time radar beamformer on a VLIW DSP processor, TMS320C66x TI DSP implementation,” IEEE International Conference on Complex Systems (ICCS), pp. 1-6, 2012.
3. Li Liu and Yan Zhang, “Design and Implementation of Reconfigurable Discrete Hilbert Transform Based on Systolic-Arrays,” International Conference on Communications and Mobile Computing, vol. 1, pp. 245-249, 2010.
4. Zhe Lin, Sharad Sinha, Hao Liang, Liang Feng, and Wei Zhang, “Scalable Light-Weight Integration of FPGA Based Accelerators with Chip Multi-Processors,” IEEE Transactions on Multi-Scale Computing Systems, vol. 4, issue 2, pp. 152-162, 2018.
5. Saurabh Dighe, Sriram R. Vangal, Paolo Aseron, Shasi Kumar, Tiju Jacob, Keith A. Bowman, Jason Howard, James Tschanz, Vasantha Erraguntla, Nitin Borkar, Vivek K. De, and Shekhar Borkar, “Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor,” IEEE Journal of Solid-State Circuits, vol. 46 , issue 1 pp. 184-193, 2011.
6. Joong-Ho Won, Yongkweon Jeon, Jarrett K. Rosenberg, Sungroh Yoon, Geoffrey D. Rubin, and Sandy Napel, “Uncluttered Single-Image Visualization of Vascular Structures Using GPU and Integer Programming,” IEEE Transactions on Visualization and Computer Graphics, vol. 19, issue 1, pp. 81-93, 2013.
7. B.I. Pawate, G.R. Doddington, S.S. Mahantshetti, M.G. Harward, and D. Smith, “Memory based digital signal processing,” International Conference on Acoustics, Speech, and Signal Processing, vol. 2, pp. 941-944, 1990.
8. Xiangqiu Yang, Leibo Liu, Shouyi Yin, Min Zhu, Wen Jia, and Shaojun Wei, “Mapping deblocking algorithm of H.264 decoder onto a reconfigurable array architecture,” International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 4166-4169, 2011.
9. R. Maestre, F.J. Kurdahi, M. Fernandez, R. Hermida, N. Bagherzadeh, and H. Singh, “A framework for reconfigurable computing: task scheduling and context management-a summary,” IEEE Circuits and Systems Magazine, vol. 2, issue 4, pp. 48-51, 2002.
10. Chen Yang, HaiBo Zhang, XiaoLi Wang, and Li Geng, “An Energy-Efficient and Flexible Accelerator based on Reconfigurable Computing for Multiple Deep Convolutional Neural Networks,” 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp. 1-3, 2018.
11. David Andrews, “Operating Systems Research for Reconfigurable Computing,” IEEE Micro, vol. 34, issue 1, pp. 54-58, 2014.
12. C. Fisher, K. Rennie, G. Xing, S. Berg, K. Bolding, J. Naegle, D. Parshall, D. Portnov, A. Sulejmanpasic, and C. Ebeling, “Emulator for exploring RaPiD configurable computing architectures,” Proceedings of the 11th International Conference on Field-Programmable Logic and Applications, pp. 17–26, 2001.
13. D. Cronquist, C. fisher, M. Figueroa, P. Franklin, and C. Ebeling, “Architecture design of reconfigurable pipelined datapaths,” Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI, pp. 23–40, 1999.
14. B. Mei, B. D. Sutter, T. V. Aa, M. Wouters, A. Kanstein, and S. Dupont, “Implementation of a coarse-grained reconfigurable media processor for AVC decoder,” Journal of Signal Processing Systems, vol. 51, no. 3, pp. 225–243, 2008.