Acta Technica 62 No. 2B/2017, 551–560 © 2017 Institute of Thermomechanics CAS, v.v.i.

Analysis of the key technologies of practical application based on embedded reconfigurable computing system

Jingshu Cao[1,2], Wenxin Li[1]

Abstract. With the continuous development of science and technology, reconfigurable computing has become one of the current research hotspots as a new computing model. However, with the increasing complexity of the embedded reconfigurable computing system, it is increasingly necessary to design from the system level to improve the design effect. Therefore, the practical application technology of the reconfigurable computing system has become a very important research topic. In this paper, the development of the subject was understood, the relevant theory of the subject was analyzed, the key technology was researched and analyzed, the example experiment was carried out, and the results were compared and analyzed, which proves the feasibility and effectiveness of the study, and is of great significance to the development of key technologies.

Key words. Reconfigurable computing, system level design, operating system.

1. Introduction

Traditional computing systems usually use two different methods to implement the algorithm. Firstly, the use of special integrated circuits, this method has a very high execution speed and accuracy of the operation. Once designed, its function cannot be changed. In order to implement different algorithms, it is necessary to redesign the circuit, the development cycle and the cost is high, and only the mass production can show its superiority. Secondly, the use of general-purpose microprocessors, the flexibility of the method is high, and once the requirements change, the function of the system can be changed by software instruction, but the serial execution of the instruction and the bottleneck of memory access bandwidth make the

[1]Lanzhou Institute of Physics, GanSu, LanZhou, 730000, China
[2]Corresponding author

performance of general-purpose microprocessors often difficult to meet the actual requirements. A single processor-based software implementation does not provide good support for complex algorithms with potential parallelism [1]. Semiconductor industry has been in accordance with the generalization and specialization of the alternating development, once every twenty years. The first common and dedicated loop is characterized by pure hardware design, that is, through the hard wired way to achieve the algorithm. The second loop is characterized by sequential programming of software, that is, through the order to achieve the algorithm. The third cycle is characterized by structural programming, that is, through the reconfigurable logic device to implement the algorithm [2]. As a new way to calculate, reconfigurable computing is in line with the development trend of semiconductor technology. It combines the advantages of generalization and specialization, uses the FPGA to reconfigure the logic unit functions and interconnection characteristics, which is a form based on the application requirements of the dynamic configuration of the circuit and has the high performance of hardware implementation and software implementation flexibility [3]. Reconfigurable systems based on reconfigurable computing techniques can be simply defined as computing systems that contain at least one reconfigurable hardware module. The function of the hardware module can be modified by the end user. The modification process is realized mainly by refactoring or partially reconstructing the programmable logic device in the system. The reconfigurable system can combine the advantages of software implementation and hardware implementation with only a small amount of hardware resources. Its appearance makes the boundaries of hardware and software in the traditional sense become blurred, the hardware system to be software [4]. Reconfigurable systems have been presented with powerful computational performance and data processing capabilities in a number of data-intensive applications, particularly in the presence of large parallelism and waterborne applications within the algorithm. One of the most representatives is the Splash 2 developed by the US Supercomputer Research Center in 1992 [5].

2. State of the art

2.1. Reconfigurable computing overview

Estrin of the University of California, USA, first proposed a reconfigurable concept and developed a prototype system. The system consists of non-flexible but programmable processors and flexible digital logic components reconstructed by program control. Although the abstraction level of the system software and hardware is not high, it can be programmed and reconstructed [6]. As the implementation of the technology is not yet perfect, the system developed by Estrin is only a rough approximation of the design concept, but it is the core of the reconfigurable computing system.

In 1986, Xilinx companies developed the world's first FPGA chip, and obtained a very good application effect in practice. To the early nineties of last century, there has been some FPGA-oriented development for a certain type of application computing equipment. The common practice is to combine one or more FPGAs, CPUs, and memories together. As part of the coprocessor accelerator program that can be executed in parallel (typically the loop body), the FPGA is managed by the refactoring [7]. The computing device of this structure was known as a reconfigurable computer. This way of working is also called refactoring.

From a different research point of view, the understanding of reconfigurable computing is not the same. We see it as a class of computer organizational structure, and has the following characteristics which are different from other organizational structure, that is, the chip customization capabilities after manufacture (different from the ASIC), can achieve the mapping of the algorithm to the computing engine to a large extent (different from the general microprocessor) [8].

Reconfigurable computing is a numerical model of the spatiotemporal domain which uses FPGA as the technical basis, changes the function of the logical unit in FPGA and the interconnection mode of the connection according to the programming information in the reconfigurable device configuration file, thus changing the function of the computing system. It can use custom reconfigurable devices to customize the computing components while designing the implementation, and can reuse the computational resources to achieve a number of different computational tasks [9]. Reconfigurable computing makes up for the performance gap between microprocessor implementation and ASIC implementation, the calculation speed is comparable to that of ASIC at the same time, which has the flexibility similar to the microprocessor, and provides a flexible and effective computing solution for a wide range of applications.

2.2. Research at home and abroad

As the reconfigurable system has broad application prospects and potential market value, programmable device manufacturers Xilinx and Altera are represented. Industry has carried out in-depth research on reconfigurable devices since the mid-1990s. The goal is to improve the device reconstruction speed, there have been some partially reconfigurable devices, as well as reconfigurable devices using compressed configuration files [10]. In addition, field-specific reconfigurable chips have also been successful, such as reconfigurable multiprocessor systems developed by IMEC and Freescale semiconductors, reconfigurable communication chip series which are capable of handling 50 channels at a chip rate of CDMA 2000 developed by Chameleon Systems, and the dynamic reconfigurable processor "DAPDNA-2" developed by Japan IPFlex for image processing and so on.

At the same time, the academic research on reconfigurable system has made great progress, and many new architectures, algorithms and tools have emerged. According to the degree of coupling, the reconfigurable computing platform is divided into system-level loose coupling system, chip-level loose coupling system and on-chip tight coupling system. In order to improve the speed of reconstruction, the researchers put forward a number of strategies and methods, some of which have been implemented in commercial products, such as partial reconfiguration. In addition, scholars have proposed to use multi-context and configuration data Cache to reduce the configuration time [11]. In terms of compilation tools, the Single-Assignment C project already has the ability to compile some C fragments into assembly language and map it to a reconfigurable system.

With the progress of the level of programmable devices and integrated circuits, reconfigurable computing has been widely used in many areas from embedded systems to high performance computing, including digital image processing, network security, bioinformatics and supercomputing, However, for the further application of reconfigurable computing, there are still many key problems that need to be solved and improved. In the aspect of system design support, with the increasing complexity of embedded reconfigurable computing system design, it is necessary to design from the system level at the same time. In the aspect of running environment support, because the traditional operating system can't adapt to the new reconfigurable system application requirements, it is one of the most important problems to be solved that how to manage the reconfigurable computing resources through the operating system, how to shield the details of the implementation, and how to provide the hardware and software unified programming model to the application developers [12].

In the domestic, Zhejiang University, China University of Science and Technology, Xi'an University of Electronic Science and Technology, National University of Defense Technology and the Chinese Academy of Sciences Institute of Computing and other research institutes have carried out fruitful research on the related problems in reconfigurable operating system. Our labs use the improved unified multitasking model to design the SHUM-UCOS operating system for reconfigurable computing, which has been able to support the dynamic creation of hardware tasks and provide a variety of topology and communication support.

3. Methodology

3.1. Reconfigurable device foundation

FPGA is the most commonly used reconfigurable device. The most widely used is based on SRAM technology. It uses the configuration SRAM to store configuration information for logical and routing resources, and the configuration information is written to the internal configuration SRAM of the FPGA to change the logic functions and interconnections between the logical units [13]. As shown in Fig. 1, Q is determined by the circuit design or hardware description language, and the corresponding configuration is controlled by programming "read" or "write".

The island structure is an early programmable device structure, and it is also a commonly used device model for academic discussion. The island structure consists of a logical cell array, routing resources, and input and output pins. Fine-grained logical units are small lookup tables (LUTs).

Existing reconfigurable devices mainly support the following reconfiguration models:

Single context model: In this model, each rewrite of configuration information must override the entire configuration memory.

Multi-context model: The main idea of this model is to prepare a plurality of

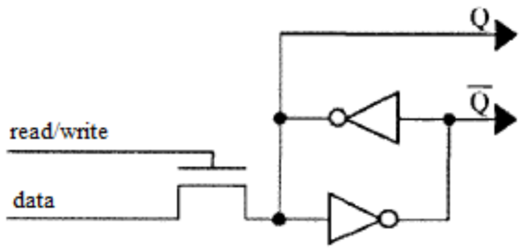

Fig. 1. SRAM-based FPGA programming principles

configuration files for the logical array of devices and store them in different addresses.

Dynamic partial reconstruction model: In this model, only some of the resources on the reconfigurable device can be reconfigured selectively without affecting the other resources on the device [14].

3.2. Dynamic reconfigurable system architecture

In this paper, the dynamic reconfigurable system based on bus connection is studied. The system is mainly composed of CPU, reconfigurable device, system memory, configuration controller and communication controller, as shown in Fig. 2.

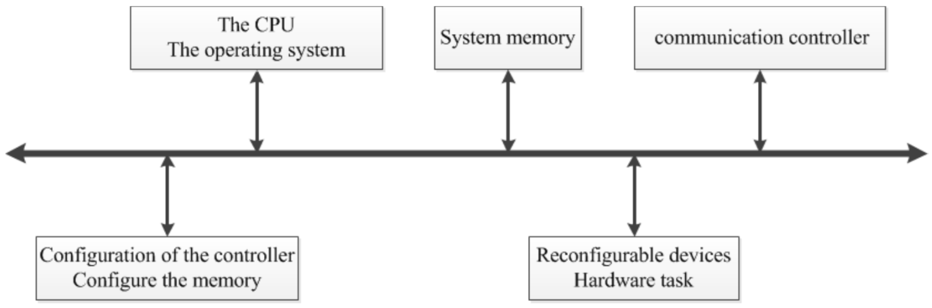

Fig. 2. Dynamic reconfigurable system based on bus connection

In Fig. 2, the operating system runs on CPU, which is responsible for managing the entire reconfigurable computing system and handling software tasks. As an additional computing unit other than the CPU, the reconfigurable device contains a number of CPUs for performing coarse-grained computationally intensive tasks. The communication controller is responsible for handling the underlying communication details and provides support for inter-task communication. The system has a dynamic partial refactoring capability that can be reconstructed on a reconfigurable device to load new hardware tasks without affecting the other running parts. Refactoring refers to the process in which the configuration controller reprograms the RPU in units of frames according to the corresponding hardware task configuration information in the configuration memory.

3.3. Experimental platform and operating environment

The experimental platform consists of XC2VP30 FPGAs and related peripherals that support DPR. The PPC 405 hardcore CPU and reconfigurable computing resources are included in the FPGA, both sharing the OPB bus, CPU clock frequency and bus frequency are 100 MHz [15]. According to the Xilinx EAPR reconfigurable system design flow, the platform is designed to be divided into AES encryption/decryption PRM and the basic design except PRM. They are located in FPGA dynamic PRR and static area. Due to the limited resources within the FPGA, the experimental platform contains only one PRR, the PRR and the static region are communicated via the bus macro, and the routing results are shown in Fig. 3.

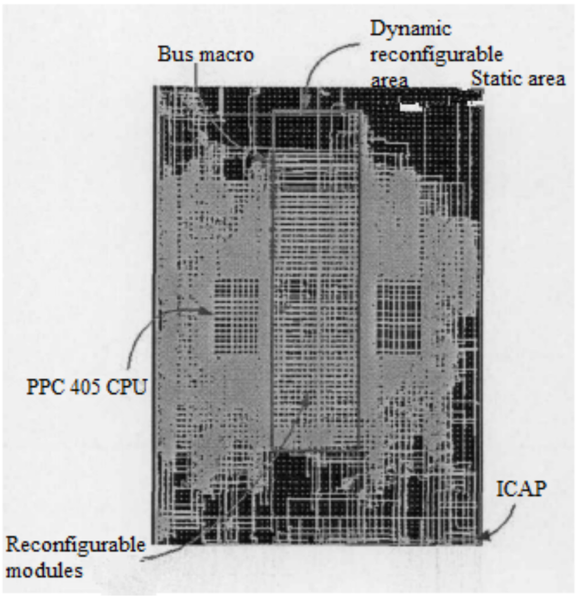

Fig. 3. XC2VP30 FPGA wiring diagram

The extended Xilkernel 3.0 operating system kernel is loaded on the above experimental platform, in addition to support priority preemptive scheduling, dynamic/static buffer allocation and semaphore synchronization, the kernel also added PRR resource management and hardware task management module. In the process of extending the thread concept from the software, the computing resource management is extended from CPU to PRR. As Xilkernel itself supports Pthread multi-threaded programming model, on this basis, it is further extended to support the creation of hardware threads, control and data transmission, to provide designers with dynamic reconfigurable hardware and software unified multi-threaded programming model SHUMDR.

Taking the AES application driven by data flow as an example, it is assumed that the data source to be processed is stored in Compact Flash in binary file format.

The application is divided into multiple threads for processing. Among them, the data producer software thread acquires the data block to be processed, the data encryption hardware thread, the thread of the data decryption hardware sharing the PRR resources, and the data block is processed. Data consumer software thread will handle the results of the data block output, through the serial port on the host. The process first allocates two shared data buffers, and then creates three threads. Among them, when hardware encryption thread was created, the dynamic binding of the pile thread, the thread between the pile thread and the producer software thread, the consumer software thread can achieve an orderly access to the shared buffer through two semaphores to complete the communication between software and hardware. According to SHUMDR, when the system is running, the operating system can dynamically reconstruct the PRR according to the resource usage and hardware task description. Once the hardware encryption thread is exited, the PRR resource is released and the data decryption thread can be created to decrypt the data. The data decryption process is similar to the data encryption process.

4. Result analysis and discussion

4.1. Hardware thread resource usage

In the previous experimental platform, the hardware thread is the hardware logic module which is run on the PRR through the HTI package, and is communicated with the CPU through the OPB bus, and is accepted by the operating system. The results are shown in Table 1.

Table 1. Usage of HTI resources

	HTI resource usage	Number of XC2VP30 FPGA resources	HTI resource occupancy
Slice	433	13696	3.20 %
FF	773	27382	2.80 %
LUT	574	27392	2.10 %
BRAM	2	136	1.40 %

As shown in the table above, the number of Slice used is only 3.2 % of the review.

4.2. Hardware thread creation

Table 2 shows the time required to create encryption and decryption hardware thread. The average creation time of the software thread set by the default attribute is 19 µs; and the hardware thread creation time includes task configuration time, task management time, and stub thread creation time.

As can be seen from Table 2, if the hardware task configuration hits, there is no need to reconfigure, the hardware thread creation time only includes task management time as well as the thread creation time for 50 µs. If the hardware task configuration is invalid, task reconfiguration is needed. In order to reduce the creation time of the hardware task configuration failure, SHUMDR uses CBB in the system memory to improve the data transmission rate. With AES encryption hardware thread, for example, task configuration time reduced from 821.9 ms to 32.29 ms. As can be seen from Table 2, the use of CBB significantly shortens the hardware thread creation time.

Table 2. Hardware thread creation time

	AAES encrypts hardware threads (bit stream file size: 307525Bytes)		AES decrypts the hardware thread (bit stream file size: 293821Bytes)	
	Use CBB	Not used CBB	Use CBB	Not used CBB
Task configuration time	32.29	821.90	31.48	776.63
Task management time+ Pile thread creation time	005	0.05	0.05	0.05
Create time for hardware task configuration failure	32.34	821.95	31.53	776.68
Create time for hardware task configuration hit	0.05		0.05	

4.3. Communication and synchronization between threads

Table 3 describes that when data cache (D-Cache) is turned on, shared buffer size is 2 K, 4 K, 8 KByte, CPU direct data I / O read and write mode is used to extend the communication time between threads corresponding to the communication API. Among them, the extended communication API combined with different shared buffer allocation, which is corresponding to different data transmission delay.

As can be seen from Table 3, compared with the way that CPU is directly responsible for data transmission, SHUMDR uses the DMA in the communication controller, which effectively improves the communication efficiency between threads. Among them, the dynamically allocated buffer is located in the D-Cache buffer memory area, while the static allocated buffer is located in the non-buffer address area. Compared with the static buffer allocation method, because of the need to maintain the consistency of Cache data, the communication time of dynamic buffer allocation is relatively long, which is characterized by better communication flexibility and programming transparency.

Table 2. Hardware thread creation time

Shared buffer size (bytes)	Communication time from software thread to hardware thread			
	Dynamic buffer allocation		Static buffer allocation	Direct I/O read and write
	D-Cache update	Data transmission	Data transmission	Data transmission
2 K	85.64	18.46	24.52	184.29
4 K	167.56	33.50	43.12	369.21
8 K	331.41	63.45	81.22	739.02
Shared buffer size (bytes)	Communication time from software thread to hardware thread			
	Dynamic buffer allocation		Static buffer allocation	Direct I/O read and write
	D-Cache void	Data transmission	Data transmission	Data transmission
2 K	83.47	18.46	24.52	184.29
4 K	165.59	33.50	43.12	369.21
8 K	329.10	63.45	81.22	739.02

5. Conclusion

As a new computing model, reconfigurable computing has high performance and flexibility, which is one of the hot research areas of the current architecture. It can configure the realization of the circuit according to the application requirements and can be used in the field of embedded system design. In this paper, through the analysis and research of the previous technology, the key technologies of practical application were deeply studied, the experimental platform was built, and the case analysis was carried out.

In this paper, a model-driven design method for reconfigurable embedded systems was proposed to design and implement a dynamic reconfigurable operating system prototype guided by a unified multitasking model, which provides designers with a unified multi-threaded programming model of hardware and software combined with the programming model. Performance of load applications was analyzed and predicted. From the different semantics and implementation of hardware and software tasks, an operating system framework based on unified multitasking model was designed.

In this paper, the key issues in system-level design and operational environment support were discussed, some progress was achieved, but there are still many areas to be improved. For example, in the prototype of reconfigurable operating system, the communication between hardware tasks still needs to be realized by the task of the pile, and the communication efficiency between hardware tasks has not been fully exploited. It is necessary to further improve the existing hardware task interface and the inter task communication mechanism, so as to reduce the communication and synchronization overhead between the hardware tasks, to improve the performance of the operating system.

References

[1] K. Compton, S. Hauck: *Reconfigurable computing: A survey of systems and software*. ACM Computing Surveys *34* (2002), No. 2, 171–210.

[2] K. Bondalapati, V. K. Prasanna: *Reconfigurable computing systems*. Proceedings of the IEEE *90* (2002), No. 7, 1201–1217.

[3] G. Estrin: *Reconfigurable computer origins: The UCLA fixed-plus-variable (F+V) structure computer*. IEEE Annals of the History of Computing *24* (2002), No. 4, 3–9.

[4] W. A. Najjar, W. Bohm, B. A. Draper, J. Hammes, R. Rinker, J. R. Beveridge, M. Chawathe, C. Ross: *High-level language abstraction for reconfigurable computing*. Computer *36* (2003), No. 8, 63–69.

[5] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz. E. Komp, P. Ashenden: *Programming models for hybrid FPGA-cpu computational components: A missing link*. IEEE Micro *24* (2004), No. 4, 42–53.

[6] M. Vuletic, L. Pozzi, P. Ienne: *Seamless hardware-software integration in reconfigurable computing systems*. IEEE Design & Test of Computers *22* (2005) No. 2, 102–113.

[7] J. M. P. Cardoso, P. C. Diniz, M. Weinhardt: *Compiling for reconfigurable computing: A survey*. Journal ACM Computing Surveys (CSUR) *42* (2010), No. 4, Article No. 13.

[8] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, D. Buell: *The promise of high-performance reconfigurable computing*. Computer *41* (2008), No. 2, 69–76.

[9] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J. W. Lee, W. Lee, A. Ma, A. Saraf, S. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal: *The Raw microprocessor: A computational fabric for software circuits and general-purpose programs*. IEEE Micro *22* (2002), No. 2, 25–35.

[10] C. A. Moritz, Y. Donald, A. Agarwal: *Simple fit: A framework for analyzing design trade-offs in Raw architectures*. IEEE Transactions on Parallel and Distributed Systems *12*, (2001), No. 7, 730–742.

[11] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, R. R. Taylor: *PipeRench: A reconfigurable architecture and compiler*. Computer *33* (2000), No. 4, 70–77.

[12] T. J. Callahan, J. R. Hauser, J. Wawrzynek: *The garp architecture and C compiler*. Computer *33* (2000), No. 4, 62–69.

[13] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, J. Fabula: *Radiation testing update, SEU mitigation, and availability analysis of the virtex FPGA for space reconfigurable computing*. Third Military and Aerospace Programmable Logic Devices International Conference (MAPLD'2000), 26–28 Sept., Maryland, USA, 2000.

[14] M. Sanchez-Elez, H. Du, N. Tabrizi, Y. Long, N. Bagherzadeh, M. Fernandez: *Algorithm optimizations and mapping scheme for interactive ray tracing on a reconfigurable architecture*. Computers & Graphics *27* (2003), No. 5, 701–713.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker: *NOX: Towards an operating system for networks*. ACM SIGCOMM Computer Communication Review *38* (2008), No. 3, 105–110.

Received September 12, 2017