UDC 004:37:001:62

Vladimir V. Vychuzhanin[1], Doctor of Technical Sciences, Professor, Head of the Department of Information Technology, E-mail: 126.ist.onpu@gmail.com, Scopus ID: 57193025809, ORCID: http://orcid.org/0000-0002-6302-1832
[1]Odessa National Polytechnic University, Shevchenko Avenue, 1, Odessa, Ukraine, 65044

ON THE CONSTRUCTION OF A SOFTWARE ARCHITECTURE FOR NUCLEAR SYSTEMS ON A CRYSTAL

Annotation. *The article discusses how to build software architecture for multi-core systems on a chip (SoC), based on asymmetric and symmetric multiprocessing, the hypervisor. Asymmetric multiprocessing is a port for several operating systems on physically separate processor cores. In symmetric multiprocessing in systems with core isolation, one OS is launched on several cores. OS SMP- system is ported without user intervention with a growing number of cores. Since all cores are managed by a single OS, message transfer between cores can occur at the L1 data cache level, providing faster communication with less jitter. Kernel isolation allows you to reserve a kernel for a hard real-time application, protecting it from the influence of other high-performance kernels, which for the software architecture allows you to select your operating system without creating low-level software when managing multiple operating systems. The hypervisor refers to a low-level software system. It manages several independent operating systems that are at a higher level. Developing multi-core systems-on-chip offerings focused on the embedded market are well suited for asymmetric multiprocessing configurations. This architecture is useful for developers who use the performance of a real-time operating system in combination with a diverse set of Linux kernel functions. The article discusses the software and hardware solutions contained in the XAPP1079 environment, which are required to run Linux on a single Zynq-7000 All Programmable system on a chip, and open source applications on the second core. Designing systems based on systems on a chip for high-performance and a real-time applications requires an optimal solution taking into account the factors: data transfer time; separation of the operating system. A system solution for high-performance and real-time applications using a symmetric multiprocessor processing architecture with kernel isolation provides low latency, jitter and real-time system operation, while maintaining software SoC scalability. Programmable logic integrated circuits containing multi-core subsystems have an efficient architecture with symmetric multiprocessing of data to ensure a compromise between the actual data transfer time and the low latency of their processing. The advantages of using symmetric multiprocessing manifest themselves if the load is distributed among several resources. In this case, the time required to complete the task is reduced. However, the performance gain brought about by a simple multiplication of the number of performers will not necessarily be linear. Some tasks should be performed only sequentially. Multi-core systems are able to process packages much more efficiently than single-core ones - but only if they are managed by optimized software. It is expedient to develop multi-core computing software, including an OS with support for symmetric and asymmetric multiprocessor data processing architectures, an embedded hypervisor, high-speed packet processing modules, and an exhaustive set of tools for the entire cycle of multi-core computing systems. The results of such development will find application in multiprocessor supercomputers and server applications, in terminal devices, access aggregators and basic devices - where the highest throughput is required.*

Keywords: *multi-core system on chip; asymmetric multiprocessing; symmetric multiprocessing; programmable logic integrated circuit.*

Introduction

Multi-core processors provide an increasing level of performance and scalability required for network equipment, control systems, and many other embedded applications.

The use of multi-core processors as specific solutions for specific types is based on common criteria for evaluating the effectiveness of a given solution. Four such criteria can be distinguished: configurability; portability; scalability; performance.

Configurability allows you to determine how easy it is to: configure the system for a specific device; switch to a new device configuration without the need for a complete revision of the system settings – boot, initialization / launch; what operating system problems are associated with this. Portability refers primarily to portability of applica tions, and to some extent to the operating system used. Scalability includes consideration of the possibilities of switching to a larger number of cores, which is associated with higher performance multi-core processors.

Multi-core computing technology is firmly established in the network device architecture.

The main prerequisites for the use of multi-core processors in network equipment are closer integration of functions and performance gains due to the use of more advanced traffic processing techniques [1; 2].

Known methods of constructing software architecture for multi-core systems on a chip (SoC) are based on asymmetric and symmetric multiprocessing, the hypervisor.

Multi-core SoCs are used to run basic operating systems (OS), as well as for high-speed data processing [3-10]. SoC software often consists of mul-

© V. Vychuzhanin; 2019

This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/deed.uk)

tiple applications – from real-time systems to high-throughput systems.

Such hybrid system solutions are becoming more complex as modern SoCs are increasingly used in high-performance systems with a large number of processor cores and high-speed interconnects.

Providing a hard real-time mode (response of the order of a few μs and jitter no more than 1 μs) in high-performance systems requires careful selection of possible solutions to such a problem.

Formulation of the problem

Increasing the number of cores in the SoC requires solving problems related to the choice of hardware, as well as software that takes into account the optimal ratio between the actual data transfer time and the low latency of their processing.

Analysis of recent research and publications

Currently, the developer can use three approaches to implement a hybrid system based on SoC: asymmetric multiprocessing AMP (Asymmetric Multi-Processing,), hypervisor solution and symmetric multiprocessing SMP (Symmetric Multi-Processing) with core isolation [1-10].

AMP is a port for several operating systems on physically separate processor cores. For example, launching an OS of bare metal class specifically designed for solving real-time tasks on one core, and running a full OS, for example, Linux, on other cores.

The complexity level increases with the transfer of messages between the OS, when memory sharing and management are required along with other security measures.

Since the cache memory is not shared between different operating systems, it is necessary that the transmission of messages is not performed through the cache area (increases delay and jitter).

In addition, this is inefficient software architecture in terms of scalability, since reporting is required as the number of cores increases.

The hypervisor refers to a low-level software system. It manages several independent operating systems that are at a higher level.

Although the initial porting in this case is similar to the AMP system, the advantage of the hypervisor is that it eliminates the need to solve non-trivial resource management and messaging tasks.

On the other hand, the disadvantage of the hypervisor is that it increases the overhead due to the additional level of software, which reduces system capacity and performance in real time.

In SMP systems with kernel isolation, one OS is launched on several cores. OS SMP-system is ported without user intervention with a growing number of cores. Since all cores are managed by a single OS, message transfer between cores can occur at the L1 data cache level, providing faster communication with less jitter. Kernel isolation allows you to reserve a kernel for a hard real-time application, protecting it from the influence of other high-performance kernels, which for the software architecture allows you to select your operating system without creating low-level software when managing multiple operating systems.

Initial porting in SMP systems is difficult when using multiple operating systems.

However, this can be significantly reduced if porting begins with an SMP architecture.

If you need deterministic execution of programs in real time, for example, when visualizing multimedia data, the possibilities of highly symmetrical processing are very limited.

A situation may arise when applications running on different kernels access the same OS resource. In this case, access will receive only one of the cores.

The remaining cores will be idle until the release of the critical area. Naturally, the performance of real-time applications is sharply reduced. SMP architectures implemented on an OS with a monolithic core, such as Windows CE, are primarily vulnerable.

Properly used OS with SMP support allows you to use the benefits of SMP, without requiring the use of specialized APIs or programming languages.

Developers have used the POSIX standard (in particular, the API) for many years in high-performance SMP environments.

But in practice, these implementations have been proven only for a small number of processors or cores (4-8). A well-designed SMP OS allows you to run threads in an application at the same time on any kernel.

This concurrency makes all chips processing power available for applications at any time. In actual practice, SMP works best with “CPU/Execution bound” model processing. I/O delays tend to disrupt the linear increase in SMP performance on several cores.

This is because SMP is designed to exploit potential parallelism in load sharing software. Specific I/O operations for input data streams are not parallel in time.

They are parallel in “space”, i.e. Separate applications are required for each data stream that is not individually subject to “parallel operation”.

Since one OS controls each core in the SMP system, all inter processing (IPC) between the cores is considered “local”.

This improves performance because the system does not require a special IPC protocol to provide communication between applications running on different cores.

Thus, there is no specific IPC requirement for IPC to work effectively in an SMP environment.

Resource allocation in SMP is a concurrency problem. In the SMP system, all resources, including memory, are shared and managed by the OS.

However, if applications use a shared memory access model, then by definition, built-in security measures to ensure that different parts of the application cannot access these shared resources at the same time.

This is the main problem that underlies the so-called problem of "parallelism" in multi-core systems.

Typically, applications developed for a processor model should not be associated with concurrent access to shared resources, since the two threads are not executed simultaneously.

Despite this, there are well-understood scenarios in uniprocessor constructions in which there is a possibility of a "collision" between two threads, access to shared data caused by the presence of a multi-threaded or multitasking operating system.

In these cases, developer's use OS supported mutates or semaphores that control access to shared data or resources to prevent damage to shared resources.

But in a multi-core SMP scenario, even these defenses fail, since they only provide simultaneous access resulting from OS multithreading.

It follows from this that the SMP model is required so that all single-core applications are checked for the presence of the necessary protections for all threads that use common data or resources. Comparing AMP and SMP (Fig. 1)

When building hybrid systems based on SoC, the developer faces the accumulation of jitter and delays at the stages: data transfer to system memory from input/output ports; when the processor detects new data in the system memory; copying data into its own memory; performing calculations on the data; copying the result back to system memory; when transferring the results back to the I/O port. Since jitter and delay accumulate at all the listed stages, it is necessary to optimize each of them.

Purpose of the article

Providing a trade-off between real-time data transfer and low latency processing in multi-core systems on a chip based on the choice of the optimal software architecture.

Justification of the choice of the method of constructing software architecture for multi-core systems on a chip

Embedded systems can be high performance, but not working in real time [9]. There is a need to choose a real-time operating system or select the feature set of the Linux distribution, given its shortcomings. An alternative to this choice may be - asymmetric multiprocessing.

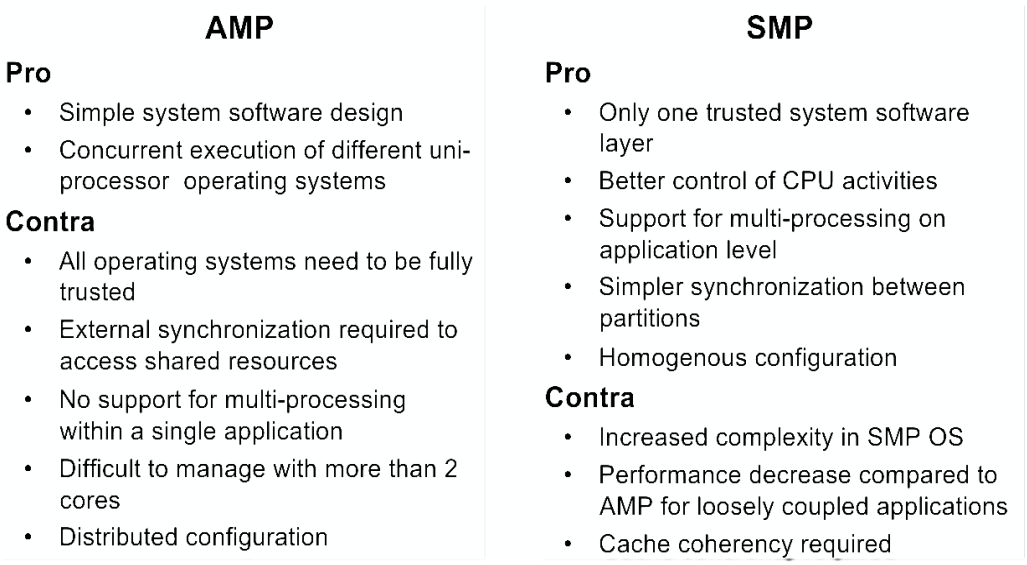


Fig. 1. Comparing AMP and SMP

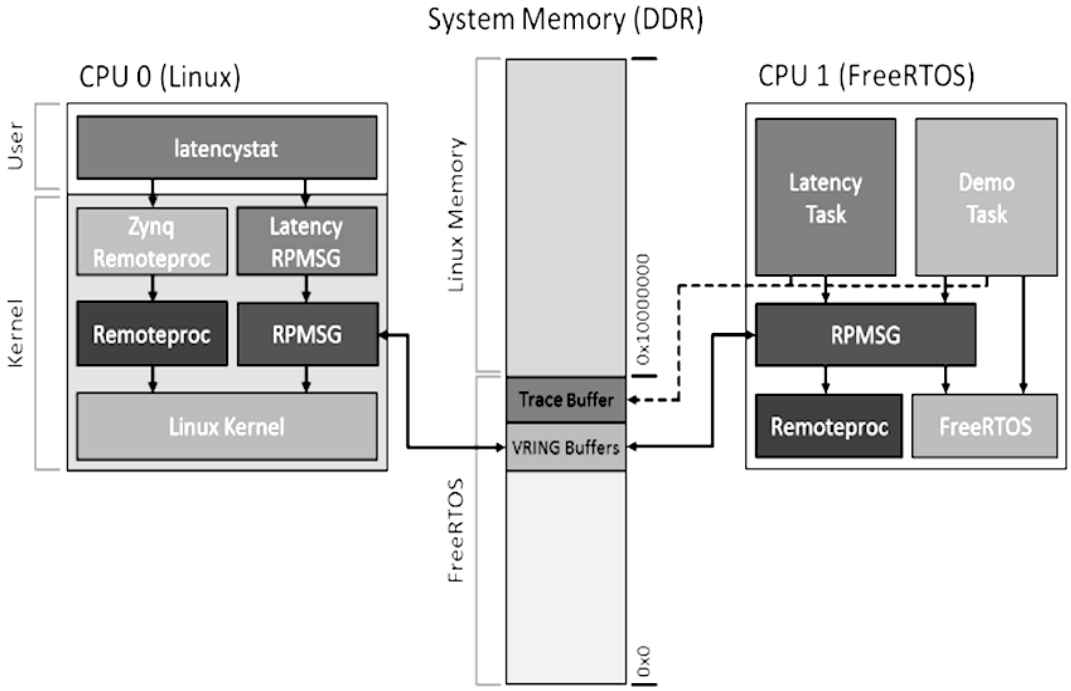


Fig. 2. Structure of Linux-FreeRTOS AMP

Modern SoCs contain various types of processors, a wide range of standard peripheral I/O devices, and programmable logic [11-23].

For example, Xilinx Zynq-7000 SoCs include dual-core ARM Cortex-A9 processors, standard peripherals (Gigabit Ethernet MAC, USB, DMA, SD / MMC, SPI and CAN) and a large programmable logic array. Such SoC products can be the basis of the Linux / RTOS AMP system, providing substantial flexibility for the real-time OS [24-32].

A typical AMP configuration is similar to a PCI-based system, where the Linux domain functions as a host, the RTOS domain, functions as an adapter, and one or more shared memory areas are used for inter-domain communication. In addition, the Linux / RTOS AMP system can dynamically reconfigure programmable logic based on time requirements for operation, taking into account the presence or absence of various external devices.

The Linux drivers used are designed to control the loading and unloading of AMP from the secondary processor. Fig. 2 shows the distribution of VRING buffers containing messages arranged in a specific structure.

The SMP core can function both on one core and simultaneously on several cores (Fig 2).

The ability to dynamically control the number of cores is the main reason AMP developers prefer the SMP core to the UP core. Remote Processor (Remoteproc) is a Linux component that provides for starting and stopping individual cores (remote processors), as well as loading kernel software in the AMP system.

It is possible for the Linux OS infrastructure used on the main processor to manage the life cycle and communication with the software context on the remote processor.

The Linux infrastructure has limitations, namely, that Linux metal must be compatible at the API level and have functional symmetry with its Linux counterpart. The scheme of the software stack Multicore Framework and its use in RTOS is shown in Fig. 3.

The Framework contains a well-abstracted transfer level, consisting of the hardware interface level and the OS abstraction level.

This allows users to transfer the Framework to other processors and operating systems.

Fig. 3,b shows the remoteproc and rpmsg infrastructure for the Linux kernel, which are kernel space drivers.

Rpmsg implicitly assumes that Linux will always be the main OS and does not support Linux as a remote OS in the AMP configuration. In addition, the remoteproc and rpmsg APIs are only accessible from the Linux kernel space. There is no equivalent API or library that can be used with other operating systems.

For example, it is possible to dynamically reconfigure the SMP system shown in Fig. 4, in the

AMP system shown in Fig. 5, and then returns to SMP again, using the capabilities of remoteproc.

Full reconfiguration control is possible using a user application or a system initialization script.

Reconfiguration management allows user applications to stop, restart, and start RTOS applications based on the dynamic needs of the system.

Kernel software (RTOS and user application) is loaded from a standard Executable and Linkable Format (ELF) file containing a section known as a resource table. The resource table is similar to the PCI configuration space.

These include the memory required for code and RTOS data. Trace buffers are areas of memory that automatically appear as files in the Linux file system.

They provide basic traceability for a remote processor that writes tracing, debugging, and status messages to buffers, in which messages are available for verification through the Linux command line or custom applications.

The resource table can also be used to define virtual input/output devices (VDEV), representing pairs of shared memory queues that support message passing between the Linux kernel and a remote processor, as well as interrupt signaling between the processors.

The Linux kernel initializes virtual I/O queues. Software running on a remote processor includes only the VDEV description in its resource table.

The remote processor messaging structure (rpmsg) is a software messaging bus based on the Linux kernel's Virtual I/O system.

The exchange bus is similar to a local subnet in which individual processors can create addressable endpoints and exchange messages through shared memory. The rpmsg framework acts as a switch, routing messages to the appropriate endpoint based on the recipient address contained in the message.

Since the message header contains the source address; special connections between different processors can be established.

For a Linux SMP kernel, this is a common thing, and this means another reason that the SMP kernel is preferred in AMP configurations.

Remote processor infrastructure manages interrupts with minimal device driver support.

Multiprocessor kernel support is not limited to homogeneous multiprocessor systems (systems using only the same type of processor).

All the functions described above can also be used in heterogeneous systems (systems with different types of processors).

For example, you can implement the PCB hardware architecture (Fig. 6) on the Xilinx SoC Zynq-7000, using one of the ARM processors as a control processor and Xilinx MicroBlaze processors in programmable logic.

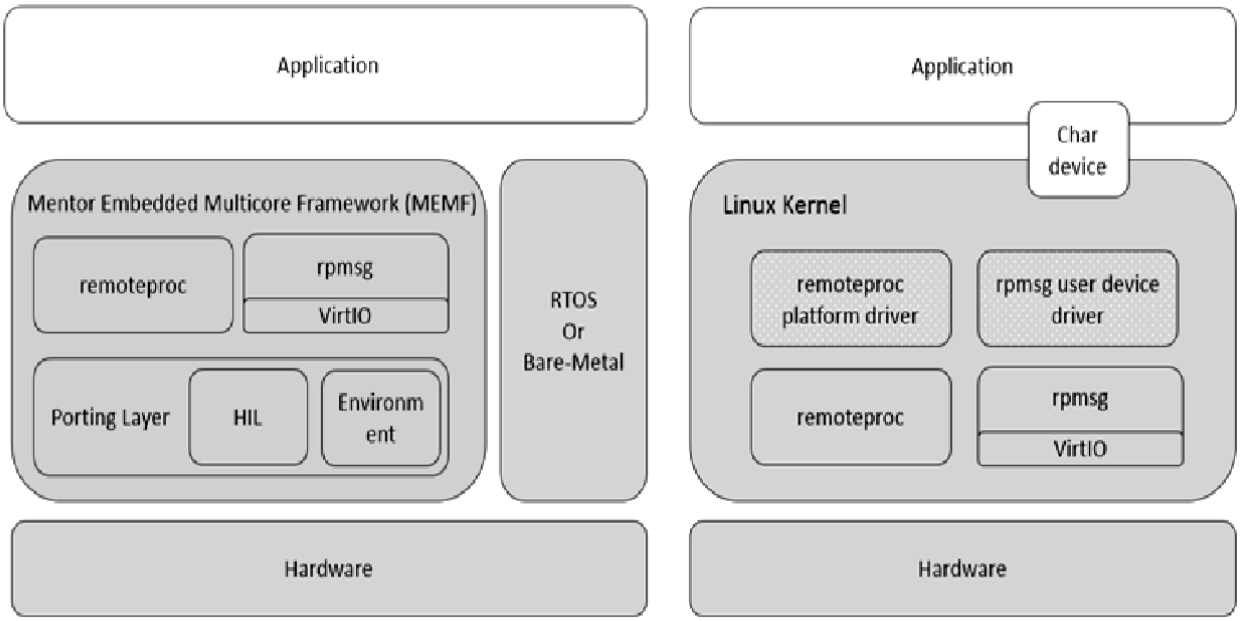


Fig. 3. Multicore Framework in RTOS and Bare Metal Environments (a) and remoteproc and rpmsg in the Linux kernel (b)

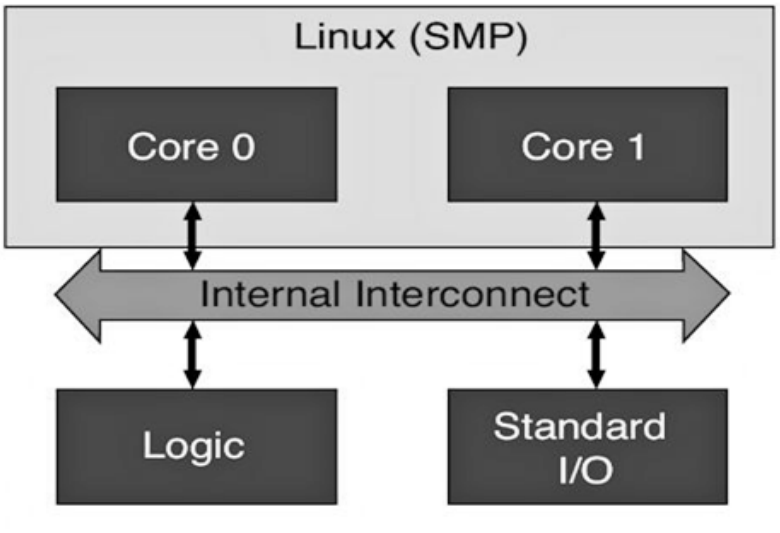


Fig.4. Structural diagram of SMP data processing

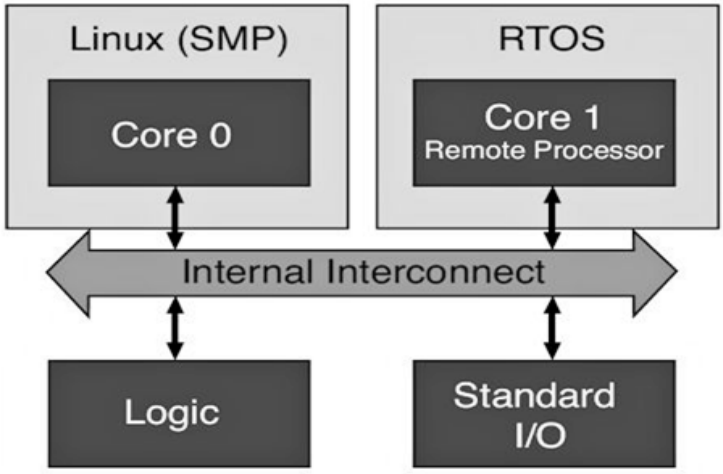


Fig. 5. Migrating the SMP system to the AMP data processing system

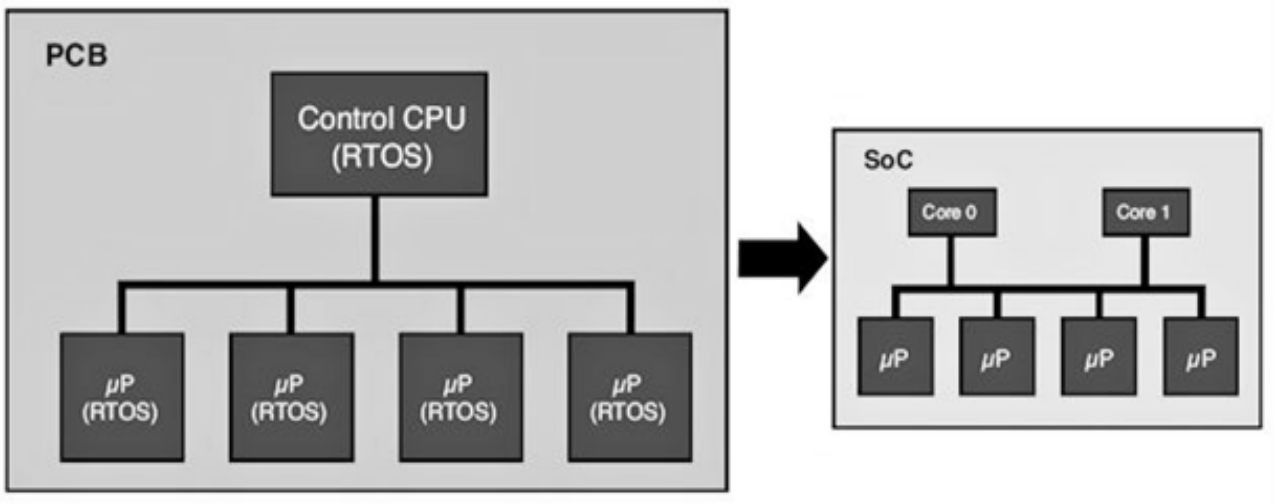


Fig. 6. Implementation of PCB hardware architecture on SoC

Xilinx provides XAPP1079, which includes both software and hardware solutions needed to run Linux on one Zynq-7000 AP SoC processor core, and open source applications on the second core (Fig. 7).

The ARM processor can be used to run the Linux SMP kernel (Fig. 8).

Adding Linux to the original design provides all the standard multiprocessing functions described above for both the ARM cores and the soft core processors.

It also has a wide range of Linux functions that support various network interfaces (Ethernet, Wi-Fi, Bluetooth), network services (web servers, FTP, SSH, SNMP), file systems (DOS, NFS, cramfs, flash memory) and others. Interfaces (PCIe, SPI, USB, MMC, video).

The Xilinx SoC Zynq-7000 provides two Cortex-A9 processor cores [32-34] sharing common memory and peripheral devices. AMP allows both processors to run their own operating systems.

The reference circuit includes the hardware and software necessary to create a reference design in which both Cortex-A9 processor cores operate in AMP configuration.

Measures were taken to prevent processor conflicts on shared hardware resources. To load and debug the processor cores in the available templates, select Zynq FSBL for AMP (Fig. 9).

The optimal ratio between real-time determinism and low latency data processing

Analysis of alternatives showed that SMP architecture with core isolation provides the best solution for optimizing high-performance real-time systems based on SoC [35].

For a real-time OS, the system response time to polling / interruption may lie in the nanosecond range, and the time it takes to perform calculations on the data depends on the application and is a fairly predictable value.

There are two main ways to transfer data that have different effects on DMA (Direct Memory Access) time – transmission.

An example is the use of a source project based on a development kit, for example, for a SoC Cyclone V [36-38] containing two nuclear subsystems ARM Cortex-A9.

The basis of the array of programmable logic of the PL Cyclone V family, unlike other families of the Cyclone series, are adaptive logic modules (as well as in the Arria and Stratix series).

In addition, the Cyclone V family of chips contain digital variable-precision signal processing units, embedded RAM blocks, high-speed transceivers, hardware IP blocks (PCI Express controllers and external synchronous memory controllers), and project protection against unauthorized copying and modification.

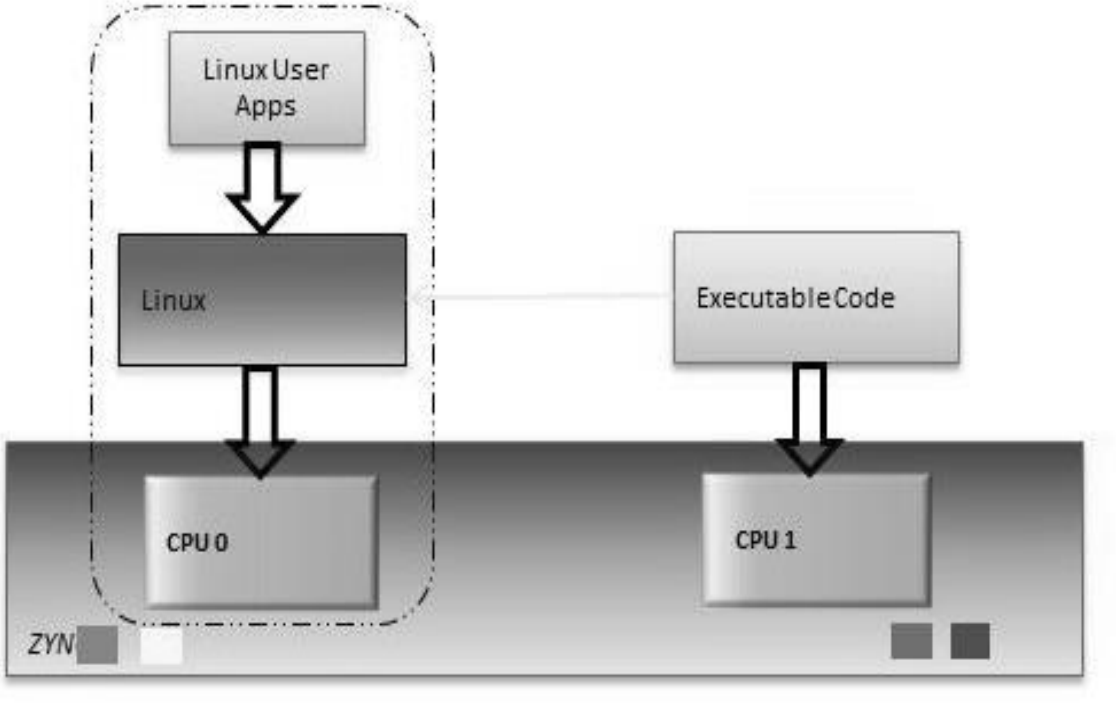


Fig.7. Xilinx XAPP1079 structure

Software architecture includes:

• VxWorks real-time OS, running in SMP mode on a dual-core ARM processor.

• The real-time application continuously performs calculations on the data and sends the results back to the I/O port.

VxWorks real-time OS includes a multitasking kernel, interprocess communication and synchronization tools, tools for cross-compiling, performance monitoring.

The VxWorks operating system has, a client-server architecture and is built in accordance with the microkernel technology, i.e. at the lowest uninterrupted kernel level (WIND Microkernel), only task scheduling and management of their interaction / synchronization are processed.

The rest of the functionality of the operating core - memory management, I / O, etc. – is provided at a higher level and implemented through processes.

This ensures the speed and determinism of the kernel, as well as the scalability of the system.

VxWorks can be configured for small embedded systems with strict memory constraints, as well as for complex systems with advanced functionality.

Moreover, the individual modules themselves are scalable.

Specific functions can be removed during assembly, and specific nuclear synchronization objects can be omitted if the application does not need them.

Although the VxWorks system is configurable, i.e. individual modules can be loaded statically or dynamically; it cannot be said that it uses a component-based approach.

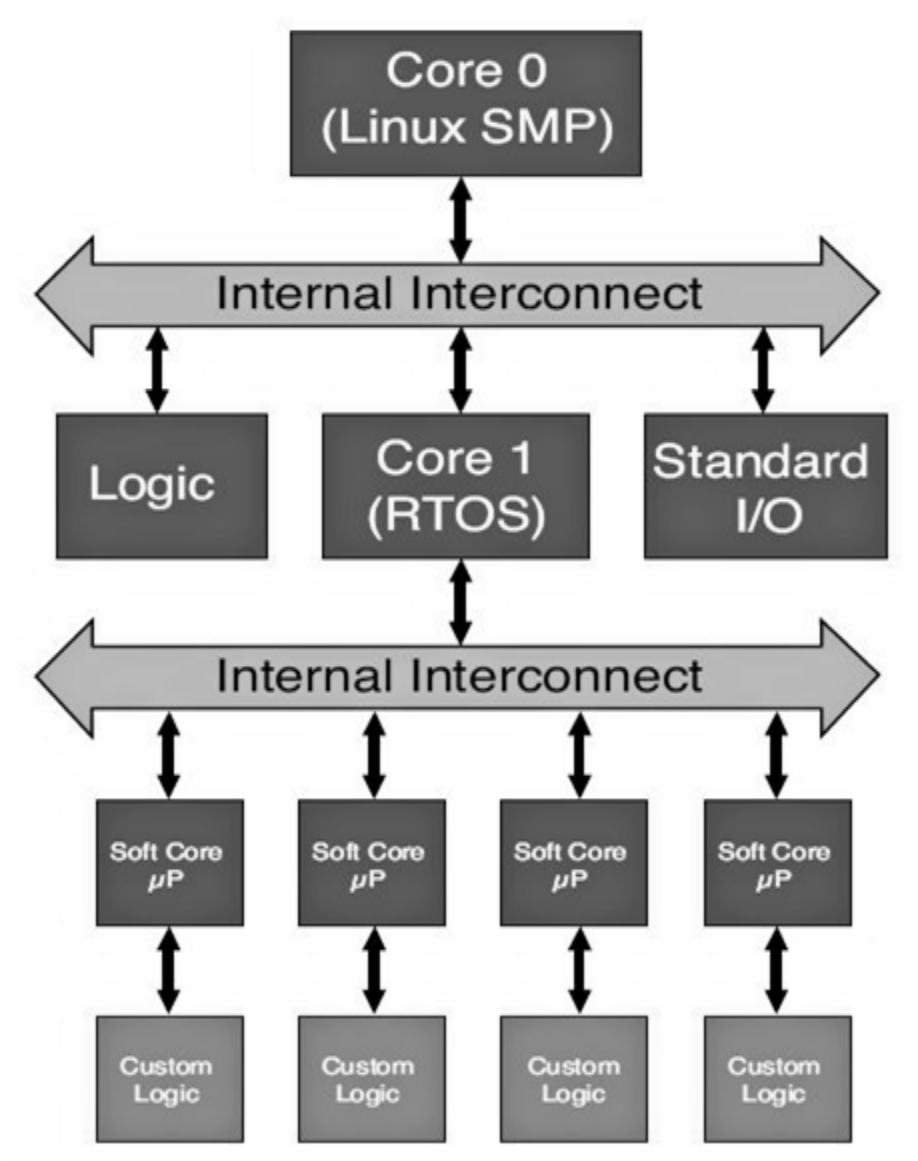


Fig. 8. Running the Linux SMP kernel



Fig. 9. Selection of a custom template FSBL

All modules are built above the base core and are designed in such a way that they cannot be used in other environments.

The VxWorks core has the following options:
– the number of tasks is not limited;
– the number of task priority levels is – 256;
– task scheduling is possible in two ways - priority preemption and cyclic;
– the means of task interaction are message queues, semaphores, events and channels (for task interaction within the CPU), sockets and remote procedure calls (for network interaction), signals (for managing exceptions) and shared memory (for data separation);
– several types of semaphores are provided for managing critical system resources: binary, computational (counting) and mutually exclusive with priority inheritance;
– deterministic context switching is supported.

On the experimental system [35] on the basis of the SoC Cyclone V kit, the cycle completion time and jitter were measured using various amounts of transmitted data.

With intensive FTP traffic processing the second core, after performing test runs, the delay was reached at the μs level with less than 300-ps jitter.

An FTP application using both kernels was also launched on VxWorks SMP [36].

At the same time the speed doubled.

Thus, when using the SMP method with kernel isolation, the performance of the system during data transfer is reduced.

In order to minimize the decrease in system performance, it is necessary to find a compromise between bandwidth and the solution of the real-time support problem.

Conclusions

Designing a system based on SoC for high-performance and real-time applications require an optimal solution taking into account the factors: data transfer time; separation of the operating system.

A system solution for high-performance and real-time applications using an SMP architecture with kernel isolation provides low latency, jitter, and real-time system performance while maintaining software scalability of SoC.

Programmable logic integrated circuits containing multi-core subsystems have an efficient architecture with symmetric multiprocessing of data to ensure a compromise between the actual data transfer time and the low latency of their processing.

The advantages of using SMP are manifested if you distribute the load among several resources.

In this case, the time required to complete the task is reduced.

However, the performance gain brought about by a simple multiplication of the number of performers will not necessarily be linear.

Some tasks should be performed only sequentially.

Multi-core systems are able to process packages much more efficiently than single-core ones - but only if they are managed by optimized software.

It is advisable to develop multi-core computing software, including an operating system with SMP and AMP architectures, an embedded hypervisor, high-speed packet processing modules, and a comprehensive set of tools for the entire development cycle of multi-core computing systems.

The results of such development will find application in multiprocessor supercomputers and server applications, in terminal devices, access aggregators and basic devices - where the highest throughput is required.

References

1. Nepomniachtchi, O. V. (2008). “Verification problems when designing systems on a crystal”. [Electronic resource]. – Available at : http: //www.mrwolf.ru. Nauka_i_obrazovanie . Tochnye_nauki / 9644. – Active link : 07.12. 2017.

2. Mark Gunter. (2012). “Optimized Software for Multi-Core Processors Provides Breakthrough in network capacity”. In MKA: VC, No. 4, pp. 38-47 (in English).

3. Putrya, F. M. (2009). Arkhitekturnyye osobennosti protsessorov s bol'shim chislom vychislitel'nykh yader, [Architectural features of processors with a large number of cores], *In Information Technology*, No. 4, pp. 2-7 (in Russian).

4. Grles, M. (2005). “Building ASIPs. The Mescal Methodology”, University of California at Berkeley electronics research laboratory, 25 p.

5. Evans, D. J. & Margaritis, K. G. (1992). “Algorithms for VLSI processor arrays”, *In Elektroteh. Vestn.* V. 59, No 2, pp. 61-67. DOI.10.1016/0141-9331(93)90102-D.

6. Kung, S. Y. (1988). “VLSI Array Processors”, Prentice-Hall, Inc., 600 p. DOI.10.1109/iscas.1988.14929.

7. Borkar, S., Cohn, R., & Cox, G. (1988). “Parallel Computing”, Proc. Supercomputing ’88. Kissimmee, pp. 330 – 339, doi. 10.1109/superc.1988.44670.

8. Eds. Ole-Johan Dahl, Edsger, W. Dijkstra, & C. A. R. Hoare Dijkstra. (1972). “Notes on structured programming”, Structured Programming, *Academic Press, Publ.*, 88 p. DOI. 10.1007/978-3-642-59412-0_19.

9. Ni, Nick. “Best practices for designing high-throughput, real-time SoC systems”. [Electronic resource]. – Available at : http:// www.embedded-computing.com. – Active link : 07.12. 2017.

10. Shcherbakov, K. S., & Shcherbakov, S. A. (2009). Tekhnologiya razrabotki vstroyennogo programmnogo obespecheniya dlya PLK. [Technology of development of embedded software for PLC.]. In News of Tomsk Polytechnic University, V. 315 (5), pp. 28-32 (in Russian).

11. Palagin, A. V., & Yakovlev, YU. S. Osobennosti proyektirovaniya komp'yuternykh sistem na kristalle PLIS. [Features of designing computer systems on a FPGA chip], *In Mathematical Machines and Systems*, No. 2, pp. 1-14 (in Russian).

12. Koch, D. (2013). “Partial reconfiguration on FPGAs. Architectures, tools and applications”, *Springer-Verlag*, 296 p.

13. Palagin, A. V. (2007). “Reconfigurable computing technology”, In Cybernetics and Systems Analysis, Springer New York, V. 43(5), pp. 675-686. DOI.10.1007/s10559-007-0093-z.

14. Mentens, N. (2015). “Dynamic Hardware Reconfiguration in Industrial Applications”, *In Lecture Notes in Computer Science*, pp. 513-518. Doi.10.1007/978-3-319-16214-0_47.

15. Evtushenko, N. D. “Methodology of designing systems on a chip. Basic principles, methods”. [Electronic resource]. – Available at : http: //www.mriprogress. msk.ru/news.php@id=7. – Active link : 07.12. 2017.

16. Bukhtev, A. (2004). Sistemy na kristalle. Novyye tendentsii. [Systems on a crystal], New Trends in Electronics NTB, No. 3, pp. 52-56 (in Russian).

17. Bukhteyev, A. (2003). Metody i sredstva proyektirovaniya sistem na kristalle, [Methods and means of designing systems on a chip], In Chip News, No. 4 (77), pp. 4-14 (in Ukraine).

18. Shagurin, I. “Systems on a crystal. Features of implementation and prospects of application”. [Electronic resource]. – Available at : http: http://www.russianelectronics.ru/leader-review/2189/doc/40316/. – Active link : 07.12. 2017.

19. Shagurin, I. I. (2006*)*, Sozdaniye “sistem na kristalle” na osnove PLIS s ispol'zovaniyem sinteziruyemykh protsessornykh yader. [Creation of “systems on a chip” on the basis of FPGA using synthesized processor cores], In Problems of developing promising microelectronic systems: collection of scientific articles. Scientific Tr, Moscow, Russian Federation, *IPPM RAS*, pp. 382-385 (in Russian).

20. Gorbunov, V. C. “Application of programmable reconfigurable schemes for solving problems of processing large graphs”. [Electronic resource]. – Available at : http://www.Rosta.ru./GraphHPC-2014_04 – Gorbunov.pdf. – Active link : 07.12. 2017.

21. Adamov, Yu.F. “Designing systems on a chip”. [Electronic resource] – Available at : http://www.bmstu-sm5.narod.ru/puchkov/puchkov_lec.pdf. . – Active link : 07.12. 2017.

22. (2017) “Route and method of designing a system-on-chip controller chip for SD-cards of standard SDHC”. [Electronic resource] – Available at : http://www.kit-e.ru/articles/circuit/2012_11_154.php. – Active link : 07.12. 2017.

23. Tarasov, I. “Systems on a chip based on FPGA Xilinx FPGA with built-in PowerPC processors”. Part 2. [Electronic resource]. – Available at : http: //kit-e.ru/articles/plis/2005_8_82.php. – Active link : 07.12. 2017.

24. (2017). “Advantages of Xilinx 7 Series All Programmable FPGA and SoC Devices”. [Electronic resource]. – Available at : http://www.ni.com/white paper / 14583 / en /. – Active link : 07.12. 2017.

25. John A. “Carbone. RTOS devices with downloadable app modules”. [Electronic resource] – Available at : http: // www.eetimes.com/. – Active link : 07.12. 2017.

26. Vychuzhanin, V. V. (2018). “Distributed software package based on the APACHE SPARK framework for processing streaming BIG DATA from complex technical systems”, *In Informatics and Mathematical Methods in Simulation*, V 8(2), pp 146-155, (in Ukraine). Doi.org/10.15276/imms.v8.no2.146.

27. Vychuzhanin, V. V. (2011). Povysheniye kachestva peredachi vysokoskorostnykh signalov s ispol'zovaniyem ustroystva na PLIS, [Improving the quality of transmission of high-speed signals using the device on the FPGA], *In Modern electronics*, No 5, pp 46-51 (in Russian).

28. Vychuzhanin, V. V. (2012). Primeneniye PLIS dlya uvelicheniya propusknoy sposobnosti ustroystv, [The use of FPGAs to increase the bandwidth of devices], *In Modern electronics*, No 3, pp. 60-62 (in Russian).

29. (2018). “Zynq-7000 SoC. Technical Reference Manual”. [Electronic resource] – Available at : http: // www.xilinx.com, 2018. – Active link : 01.12. 2018.

30. (2018). “Zynq-7000 SoC. Data Sheet: Overview”. 2018. - DS190 (V1.11.1). [Electronic resource]. – Available at : http: // www.xilinx.com. – Active link: 01.12. 2018.

31. (2018). “Platform Zynq-7000. Another round of innovation”. [Electronic resource]. – Available at : http://www.russianelectronics.ru/developer-r/review/2189/doc/57818/. – Active link : 01.12. 2018.

32. Mario, Vestias, & Horacio, Neto. (2014). “A many-core overlay for high-performance embedded computing on FPGas”. *In 1st International Workshop on FPGAs for Software Programmers*, Munich, Germany, pp. 71 – 76. Doi.org/10.1109/fpl.2014.6927483.

33. McDougall, John. (2014). “Simple AMP”. In Bare-Metal System Running on Both Cortex-A9 Processors APP1079, (V1.0.1), 32 p.

34. Vychuzhanin, V. V. (2010). PLIS serii Cyclone s vstroyennymi apparatnymi transiverami. [FPGA of Cyclone series with built-in hardware transceivers], In Modern Electronics, No. 5, pp. 28-33 (in Russian).

35. Ni, Nick. “Best practices for designing high-throughput, real-time SoC systems”. [Electronic resource]. – Available at : www.embedded-computing.com. – Active link : 07.12. 2017.

36. Smith, M. C., & Peterson, G. D. (2012). “Optimization of Shared High-Performance Reconfigurable Computing Resources”, *In ACM Transactions on Embedded Computing Systems*, V. 11, pp. 1-22. Doi.10.1145/2220336.2220348.

37. Chakma, K. A. (2011). “Hierarchical Scheduling Approach for Symmetric Multiprocessing Systems”. *In Software Engineering*, V. 1(1), pp. 61-65. Doi.10.7763/lnse.2013.v1.14.

38. Cyclone, V. (2018) “Hard Processor System Technical Reference Manual”. [Electronic resource]. – Available at : https://www.intel.com/content/dam/www/ programmable/us/en/pdfs/literature/hb/cyclone-v/cv_54006.pdf. – Active link : 01.12. 2018.

Received 18.12.2018

[1]**Вичужанін, Володимир Викторович**, доктор техніч. наук, професор, завідувач каф. інформаційні технології, E-mail: 126.ist.onpu@gmail.com, Scopus ID: 57193025809,
ORCID: http://orcid.org/0000-0002-6302-1832
[1]Одеський національний політехнічний університет, пр. Шевченка, 1, Одеса, Україна, 65044

ПРО ПОБУДОВУ ПРОГРАМНОЇ АРХІТЕКТУРИ ДЛЯ БАГАТОЯДЕРНИХ СИСТЕМ НА КРИСТАЛІ

Анотація. *У статті розглянуті способи побудови програмної архітектури для багатоядерних систем на кристалі (SoC), осно-ванні на асиметричною і симетричною багатопроцесорної обробки, гіпервізора. Асиметрична многопроцессорная обробка являє собою порт для декількох операційних систем на фізично окремих процесорних ядрах. У симетричній багатопроцесорної обробки в системах з ізоляцією ядер запускається одна ОС на декількох ядрах. ОС SMP-системи портується без участі користувача при зростаючій кількості ядер. Оскільки всі ядра підкоряються одній ОС, передача сообще-ний між ядрами може відбуватися на рівні L1 кеша даних, забезпечуючи більш швидкий зв'язок з меншим джиттером. Ізоляція ядра дозволяє зарезервувати ядро для застосування жорсткого реального часу, захищаючи його від впливу інших високопроизводит-них ядер, що для програмної архітектури дозволяє вибрати використовувану ОС, не створюючи програмне забезпечення низького рівня при управлінні декількома ОС. Гипервизор відноситься до системи низького програмного рівня Вон управляє декількома незалежними ОС, що знаходяться на більш високому рівні. Багатоядерні пропозиції систем на кристалі, орієнтовані на вбудований ринок, добре підходять для конфігурацій з асиметричною багатопроцесорної обробкою. Подібна архітектура корисна розробникам, що використовують продуктивність операційної системи реального часу в поєднанні з різноманітним набором функцій ядра Linux. У статті розглянуті програмні і апаратні рішення, що містяться в середовищі XAPP1079, необхідні для запуску Linux на одному процесорному ядрі Zynq-7000 All Programmable системи на кристалі, і додатки з відкритим вихідним кодом на другому ядрі. Проектування систе-ми на базі систем на кристалі для високопродуктивних додатків і додатків реального часу вимагає оптимального рішення з урахуванням факторів: часу передачі даних; поділу операційної системи. Системне рішення для високопродуктивних додатків і додатків реального часу з використанням симетричною багатопроцесорної архітектури обробки даних з ізоляцією ядра забезпечує малі затримки, джиттер і роботу системи в ре-жимі реального часу, зберігаючи при цьому програмну масштабованість SoC. Програмовані логічні інтегральні схеми, які містять багатоядерні підсистеми, мають ефективної архітектурою із симетричною багатопроцесорної обробкою даних для забезпечення компромісу між реальним часом передачі даних і малою затримкою їх обробки. Переваги використання симетричною багатопроцесорної обробки проявляються, якщо розподілити навантаження між декількома ресурсами. У цьому слу-чаї час, необхідний для виконання завдання, зменшується. Однак приріст продуктивності, що привноситься простим множенням числа виконавців, не обов'язково буде лінійним. Деякі завдання повинні виконуватися тільки послідовно. Багатоядерні системи здатні обробляти пакети значно ефективніше одноядерних - але тільки за умови, що ними керує оптимізатора-зірованним програмне забезпечення. Доцільною є розробка програмного забезпечення багатоядерних обчислень, вклю-чающего ОС з підтримкою симетричною і асиметричною багатопроцесорної архітектури обробки даних, вбудований гипервизор, модулі швидкісної обробки пакетів, а також вичерпний набір інструментарію для всього циклу розробки багатоядерних обчислювальних систем. Результати такої розробки знайдуть застосування в багатопроцесорних суперкомп'ютерах і серверних при-положеннях, в кінцевих пристроях, агрегаторах доступу і базових пристроях - там, де потрібно найбільша пропускна здатність.*

Ключові слова: *многоядерная система на кристалі; асиметрична багатопроцесорна обробка даних; симетрична багатопроцесорна обробка даних; програмована логічна інтегральна схема*

[1]**Вычужанин, Владимир Викторович**, доктор технических наук, профессор, зав. каф. информационных технологий, E-mail: 126.ist.onpu@gmail.com, Scopus ID: 57193025809,
ORCID: http://orcid.org/0000-0002-6302-1832
[1]Одесский национальный политехнический университет, пр. Шевченко, 1, Одесса, Украина, 65044

О ПОСТРОЕНИИ ПРОГРАММНОЙ АРХИТЕКТУРЫ ДЛЯ МНОГОЯДЕРНЫХ СИСТЕМ НА КРИСТАЛЛЕ

Аннотация. *В статье рассмотрены способы построения программной архитектуры для многоядерных систем на кристалле (SoC), основанные на асимметричной и симметричной многопроцессорной обработке, гипервизоре. Асимметричная многопроцессорная обработка представляет собой порт для нескольких операционных систем на физически отдельных процессорных ядрах. В симметричной многопроцессорной обработки в системах с изоляцией ядер запускается одна ОС на нескольких ядрах. ОС SMP- системы сортируются без участия пользователя при растущем числе ядер. Поскольку все ядра управляются одной ОС, передача сообщений между ядрами может происходить на уровне L1 кэша данных, обеспечивая более быструю связь с меньшим джиттером. Изоляция ядра позволяет зарезервировать ядро для приложения жесткого реального времени, ограждая его от влияния других высокопроизводительных ядер, что для программной архитектуры позволяет выбрать используемую ОС, не создавая программное обеспечение низкого уровня при*

управлении несколькими ОС. Гипервизор относится, к системе низкого программного уровня. Она управляет несколькими независимыми ОС, находящимися на более высоком уровне. Развивающиеся многоядерные предложения систем на кристалле, ориентированные на встроенный рынок, хорошо подходят для конфигураций с асимметричной многопроцессорной обработкой. Подобная архитектура полезна разработчикам, использующим производительность операционной системы реального времени в сочетании с разнообразным набором функций ядра Linux. В статье рассмотрены программные и аппаратные решения, содержащиеся в среде XAPP1079, необходимые для запуска Linux на одном процессорном ядре Zynq-7000 All Programmable системы на кристалле, и приложения с открытым исходным кодом на втором ядре. Проектирование системы на базе систем на кристалле для высокопроизводительных приложений и приложений реального времени требует оптимального решения с учетом факторов: времени передачи данных; разделения операционной системы. Системное решение для высокопроизводительных приложений и приложений реального времени с использованием симметричной многопроцессорной архитектуры обработки данных с изоляцией ядра обеспечивает малые задержки, джиттер и работу системы в режиме реального времени, сохраняя при этом программную масштабируемость SoC. Программируемые логические интегральные схемы, содержащие многоядерные подсистемы, обладают эффективной архитектурой с симметричной многопроцессорной обработкой данных для обеспечения компромисса между реальным временем передачи данных и малой задержкой их обработки. Преимущества использования симметричной многопроцессорной обработки проявляются, если распределить нагрузку между несколькими ресурсами. В этом случае время, требуемое для выполнения задачи, уменьшается. Однако прирост производительности, привносимый простым умножением числа исполнителей, не обязательно будет линейным. Некоторые задачи должны выполняться только последовательно. Многоядерные системы способны обрабатывать пакеты значительно эффективнее одноядерных - но только при условии, что ими управляет оптимизированное программное обеспечение. Целесообразной является разработка программного обеспечения многоядерных вычислений, включающего ОС с поддержкой симметричной и асимметричной многопроцессорной архитектур обработки данных, встраиваемый гипервизор, модули скоростной обработки пакетов, а также исчерпывающий набор инструментария для всего цикла разработки многоядерных вычислительных систем. Результаты такой разработки найдут применение в многопроцессорных суперкомпьютерах и серверных приложениях, в оконечных устройствах, агрегаторах доступа и базовых устройствах - там, где требуется наибольшая пропускная способность.

Ключевые слова: *многоядерная система на кристалле; асимметричная многопроцессорная обработка данных; симметричная многопроцессорная обработка данных; программируемая логическая интегральная схема*