ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

A Multiprocessor System by Using FPGA

Malladi Sunder Rao[1], G.Ramprabhu[2]

Research Scholar, Annamalai University, Annamalai Nagar, Taminadu, India. [1]

Professor, Annamalai University, Annamalai Nagar, Taminadu, India [2]

ABSTRACT: In the modern environment the System on Chips (SoCs) made a very huge quantity of applications in the field of computer networking, Computer Vision, Audio Signal Processing, Video Signal Processing, Image Processing, Wireless Communication Networks etc. In these types of applications it provides an property of imposing of Real Time Processing of an System with highly demanding throughput. Previously the researchers mentioned the problem only by the process networks which are used as the optimal architecture having synthesis of automatically idealized. The optimal architecture developed for process networks which is an automatic manner which is integrated for developing a framework is addressed in this thesis. In this the processors, memories, the I/O devises are taken as the resources which consist of the architectural resource allocation method and it is chosen from the library used for component. In added with the resource allocation in our thesis the various process are binding which is used to process the network beyond the processors present in the system as well as the Queues present in the memory. For the run time scheduling the assumption we take as the dynamic scheduler for processing of the Queue and the processors. The another one new invention in our thesis is the synthesis of Interconnection Network (IN) which gives the exploiting communication present in the patterns which runs the specific application of the process network. For estimating the delays present in the queuing we developed a new analytical model based on the IN synthesis which works in the multiprocessors with heterogeneous property and memory with non uniform accesses. For performing validation of this method there is no need of any benchmarks with all applications which has a modeled of the process network. For the Real time applications we further developed a benchmarks which is very much time consuming process and it is not an easy method which is portable across the different types of PN frameworks. For these kinds of processes we used the Random Process Network Generator (RPNG) which is used to create the sample network processes which has a characteristic of speedy and aided in speedy validation and it was compared with the Multiprocessor Systems on Chip (MpSoC) techniques used for synthesis.

KEYWORDS: Multiprocessor Systems on Chip, Random Process Network Generator, Interconnection Network, Directed Acyclic Graph, System on Chips.

1. INTRODUCTION

1.1. Main FPGA-Based Processors.

In FPGA-based multiprocessor systems, the most widely used FPGA soft processors are made by one of the two main FPGA companies: Xilinx or Altera. The other option is to use open-source soft processors.

Xilinx is the most widely used brand in MPoPC systems. It supplies three main processors: softcore MicroBlaze (MB), PicoBlaze (8 bits reduced soft processor), and hard-core PowerPC. The number of PowerPcs is limited by the FPGA model to a maximum of four. The number of MicroBlaze and PicoBlaze processors is limited only by logic resources.

1.2. Classification.

The traditional classification of MPSoCs is heterogeneous, when there are different processors or even accelerators in the same system and homogeneous, when all the processors in the system are identical. Normally, application-specific systems are heterogeneous. This is the common type of MPSoCs, as in FPGA- based multiprocessor systems. The reason is that MPSoC are usually implemented for embedded applications that behave intrinsically in a heterogeneous manner and require different kinds of processors.

Homogeneous MPSoCs are normally general-purpose systems, where all the processors are identical. In this

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

kind of system, it is possible to increase the number of processors without changing the architecture (scalability property). It is easier to develop software for homogeneous systems.

They use the dynamic reconfiguration FPGA feature to adapt hardware at run-time to a specific application. So, we have several dynamic modules which are loaded by an arbiter depending on the target application.

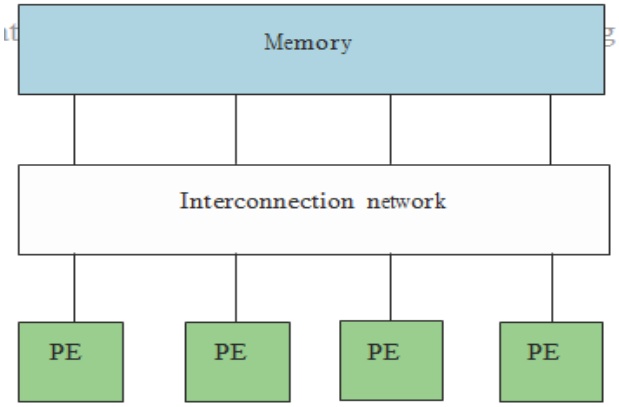

Figure 1. Shared memory multiprocessor systems.

II. LITERATURE SURVEY

Most MPoPCs are application specific. In this case, the system is application dependent, so the architecture is designed to achieve the best performance for the specific application. This architecture can be obtained by using design flow tools provided by FPGA vendors (known as hand-tune design) or using custom tools that obtain the architecture automatically from the specifications (known as automatic design). There are systems that target different areas: multimedia, networking, control, and bioinformatics. Network application is implemented in [7]. The solution proposed for IPV4 packet forwarding is a master- slave/pipeline approach. Communications among processors is point to point using MicroBlaze Fast Simplex Link (FSL) ports. There are different pipeline branches replicated in space to increase throughput. To demonstrate the viability of this solution, they compare the FPGA-based system proposed to an ASIC MPSoC with the same functionality. The FPGA-based system only loses a factor of 2.6X in performance normalized to area.

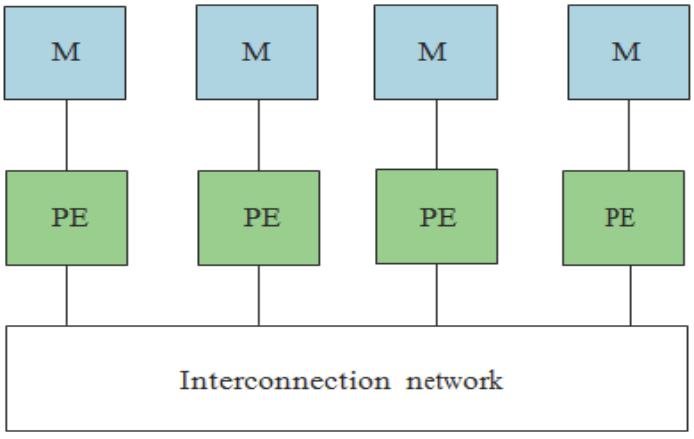

Figure 2. Distributed memory multiprocessor systems.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

MPLEM [20] is a homogeneous, general-purpose 80 processor FPGA-based multiprocessor system. Tseng and Chen [2] present a shared-bus/shared- memory multiprocessor system implemented on an Altera Cyclone. They use an instruction cache and Mutex core provided by Altera for synchronization. A NoC-based homogeneous multiprocessor system is presented in [3]. This has 24 processors, is implemented in a Xilinx Virtex-4, and is external DDR2 constrained. To increase its compatibility and to facilitate the reutilization of the architecture, the IP Open Core Protocol (OCP-IP) standard is used to connect processor elements and NoC. Network interfaces (NIs) translate OCP to the NoC protocol. In this case, every MicroBlaze has an OCP adapter.

The new evolution in reconfigurable multiprocessor systems is the run-time reconfigurable system. This type of systems adds the dynamic reconfigurability feature of FPGAs to the power of having multiple processors. It adds a new degree of freedom in the design of multiprocessor systems. This freedom allows designers to adjust system performance at run-time obtaining better efficiency in accordance with the application. Here, they propose a "meet-in-the-middle" design flow, which is to combine traditional top-down design with bottom-up approach. The bottom-up design is possible due to run-time reconfigurability. It allows hardware to be re designed at run-time in order to obtain better efficiency in terms of speed, area, and power for a specific application. Depending on the context, a different algorithm will be re-configured. They utilize the Dynamic Partial reconfigurable FPGA feature.

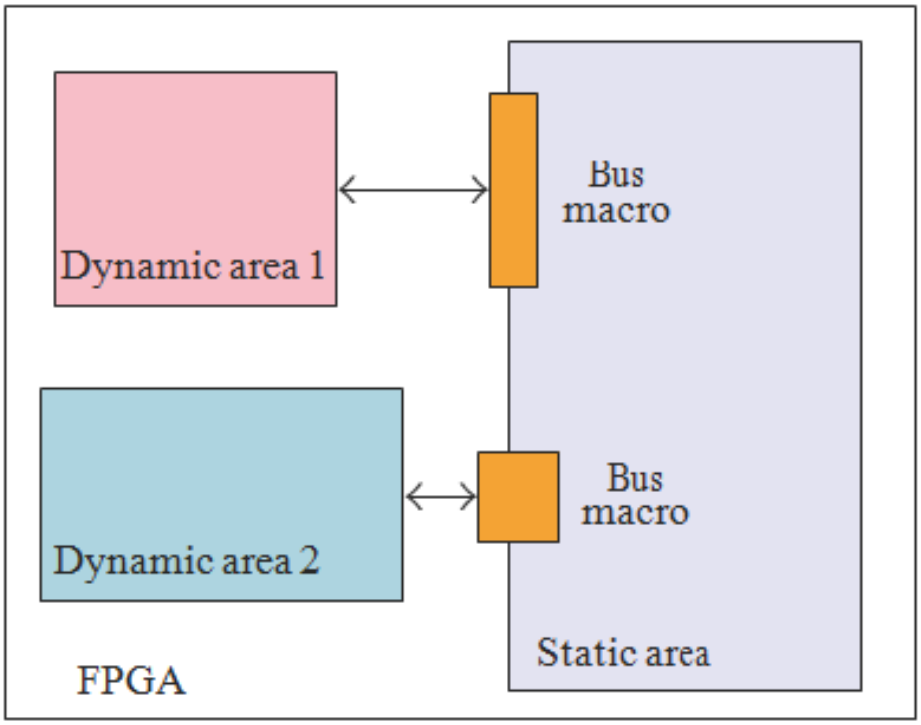

Figure 3. Architecture of a basic run-time reconfigurable system.

III. PROPOSED SYSTEM

The principal limitation of building a multiprocessor system on a FPGA is the amount of logic resources, specifically the amount of on-chip memory. Therefore, Shared-Memory architectures are widely used. It is possible to share data memory and/or instruction memory. Another solution is to use external memory to increase the amount of memory. In this case, the number of external memory blocks is limited by the package and pins of the FPGA [10, 12]. If the multiprocessor system does not fit in a single FPGA, it is possible to implement the system using a multi-FPGA approach [13]. Another important research area is parallel software efficiency. To maximize the potential of

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

multiprocessors, it is necessary to develop parallel programming..

3.1. Design Methodology

The main two ways of building multiprocessor systems in an FPGA are: (a) manually, using the design flow provided by FPGA companies or (b) automatically, mapping applications to a specific architecture.

3.1.1 Hand Tuned Design.

The first and most common method is manual design using the design flow available from FPGA companies. The two most commonly-used tools are EDK from Xilinx and SOPC Builder from Altera. Their advantage is the familiarity designers have with these tools and their ease of use [10]. The problem is that it is a hand tuned method so it is difficult to explore all the design space manually to get the best possible architecture for a specific application [8]. This method may be a good choice to build small systems where the architecture to be implemented is known, or where there are not so many possibilities for Design Space Exploration (DSE), so in these cases it is not difficult to perform experiments and then choose the best configuration.

Another important issue is that, with FPGA company design flows, it is not possible to explore all the possibilities for designing multiprocessor systems. Normally, these tools are made to design uniprocessor systems so they have a number of shortcomings when dealing with multiprocessor systems. The most important shortcomings are as follows.

(i) Limitations in architectures for interprocessor communication. The main design flows only allow shared-bus and point-to-point communications. Complex MPSoC may require a higher bandwidth than a bus can offer or may need to be more area- efficient than point-to-point connections.

(ii) Lack of effective mechanisms to share resources. Mutex Core provided by FPGA vendors seems to be inefficient as a synchronization method for multiprocessor systems [12].

(iii) Limitation in IP cores. The cores that can be included in one's design are restricted by the vendor library.

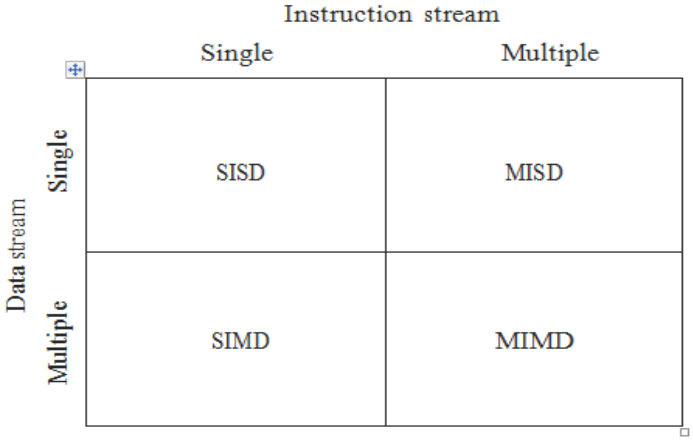

Figure 4. Effective Mechanisms with Data Stream

Researchers try to solve these limitations by creating *ad-hoc* IP blocks. These IP blocks act like add-ons to resolve this specific limitation. Some examples are

(i) Efficient Synchronization. Tumeo et al. [13] introduced two hardware synchronization modules for Xilinx MicroBlaze Systems. The modules can be completely integrated in the system using the EDK tool chain.

(ii) Cache Coherency Problem. A cache coherency module is presented in [4] as an IP Block. This paper assesses the solution and reports good results in performance.

(iii) An EDK-based tool is presented in [4]. It allows the designer to abstract low-level details of EDK and to create NoC-based systems rapidly.

(iv) Tumeo et al. also present a number of interesting modules to improve the performance of multiprocessor systems on FPGA. An Interrupt controller is presented in [5], and a DMA mechanism in [6].

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

3.1.2. Automatic Synthesis Design.

Another trend consists of using automatic synthesis tools to map an application to architecture. The input parameters are: the application and, sometimes, the platform for the architecture and the cores to be used in the design. The result is a synthesized system that fits in the FPGA target. The problem with this method is that there are no standards. Some automatic tools have appeared in the research community, but they have not been standardized or commercialized. So, they are not widely used. Nevertheless, this is a very important research area, but is still on-going. Several papers present frameworks to perform this automation process; some of which we discuss here. The processor has its own special memory, which is relatively small. These portions of memory are called registers, and there are some special registers that are used to keep track of things as a program executes. One of the most notable is the extended instruction pointer (EIP). The EIP is a pointer that holds the address of the currently executing instruction. Other 32-bit registers (depends upon the architecture) that are used as pointers are the extended base pointer (EBP) and the extended stack pointer (ESP). All three of these registers are important to the execution of a program.

When programming in a high-level language, like C, variables are declared using a data type. These data types can range from integers to characters to custom user-defined structures. One reason this is necessary is to properly allocate space for each variable. An integer needs to have 4 bytes of space, while a character only needs a single byte. This means that an integer has 32 bits of space (4,294,967,296 possible values), while a character has only 8 bits of space (256 possible values).

3.2. Memory Segments

Program memory is divided into five segments: text,data,bss,heap,stack. Each segment represents a special portion of memory that is set aside for a certain purpose. The text segment is also sometimes called the code segment. This is where the assembled machine language instructions of the program are located. As a program executes, the EIP is set to the first instruction in the text segment.

- Read the instruction that EIP is pointing to.
- Add the byte-length of the instruction to EIP.
- Execute the instruction that was read in step 1.
- Go to step 1.

Sometimes the instruction will be a jump or a call instruction, which changes the EIP to a different address of memory. The processor doesn't care about the change, because it's expecting the execution to be non-linear anyway. So if the EIP is changed in step 3, the processor will just go back to step 1 and read the instruction found at the address of whatever the EIP was changed to.

Write permission is disabled in the text segment, as it is not used to store variables, only code. This prevents from actually modifying the program code, and any attempt to write to this segment of memory will cause the program to alert the user that something bad happened and kill the program. Another advantage of this segment being read-only is that it can be shared between different copies of the program, allowing multiple executions of the program at the same time without any problems. It should also be noted that this memory segment has a fixed size, because nothing ever changes in it. The data and bss segments are used to store global and static program variables. The data segment is filled with the initialized global variables, strings, and other constants that are used through the program. The bss segment is filled with the uninitialized counterparts. Although these segments are writable, they also have a fixed size.

The heap segment is used for the rest of the program variables. One notable point about the heap segment is that it isn't of fixed size, meaning it can grow larger or smaller as needed. All of the memory within the heap is managed by al- locator and deallocator algorithms, in our srijan allocator and deallocater manager are not supported so all the memory allocated statically at the time of compilation. But in actually program scenario the heap will grow and shrink depending on how much memory is reserved for use ,the growth of the heap moves downward toward higher memory addresses whereas it remain static in srijan frame work. The stack segment also has variable size and is used to store context during func- tion calls. When a program calls a function, that function will have its own set of passed variables, and the function's code will be at a different memory location in the text (or code) segment. Because the context and the EIP must change when a function is called, the stack is used to remember all of the past variables and where the EIP should return to after the function is finished.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

The ESP register is used to keep track of the address of the end of the stack, which is constantly changing as items are pushed into and popped from it. Because this is very dynamic behavior, it makes sense that the stack is also not of a fixed size. Opposite to the growth of the heap, as the stack changes in size, it grows upward toward lower memory addresses. When a function is called, several things are pushed to the stack together in a structure called a stack frame. The EBP register (sometimes called the frame pointer (FP) or local base pointer (LB)) is used to reference variables in the current stack frame. Each stack frame contains the parameters to the function, its local variables, and two pointers that are necessary to put things back the way they were: the saved frame pointer (SFP) and the return address. The stack frame pointer is used to restore EBP to its previous value, and the return address is used to restore EIP to the next instruction found after the function call.

When a function is called, the EIP is changed to the address of the beginning of the function in the text (or code) segment of memory to execute it. Memory in the stack is used for the function's local variables and the function arguments. After the execution finishes, the entire stack frame is popped off the stack, and the EIP is set to the return address so the program can continue execution. If another function were called within the function, another stack frame would be pushed onto the stack, and so on. As each function ends, its stack frame is popped off the stack so execution can be returned to the previous function. This behavior is why this segment of memory is organized in a FILO data structure.

Srijan Simulator has complete control on data variables. In order to provide fine grain control on data variables, srijan provide the interface the system designer. Data variable can be specified to any memory module. MEMORY command which consists the memory region initialization with start address and size of free space of memory modules.

IV. CONCLUSION

This paper surveys recent FPGA-based multiprocessor systems appearing in the literature. There has been a significant increase in publications in this area over the last three years. After revising the literature, we can draw the conclusion that one of the most important issues with regard to designing multiprocessor systems in FPGA is the use of block RAM. The amount of on-chip RAM is fixed, and it limits the number of processors that can be included in one's design more than logic resources. Therefore, it is important to design memory efficient systems. When building shared memory systems, cache coherency and synchronization are critical aspects because the simple softcore processors offered by FPGA vendors have certain shortcomings in these areas. NoC is a weak point in design flows provided by vendors, since they do not include this approach in their tools. In our opinion, it is only a matter of time before they do so. Other important challenges in multiprocessor design are parallel software and automation of the design process. There are a number of important advances in this area but no standards have been established as yet. Run-time reconfigurability uses the dynamic reconfiguration feature of FPGAs to obtain a new degree of freedom in the design of multiprocessor systems, making these systems more flexible to target different applications using the same.

REFERENCES

[1] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, 2008.
[2] W. Wolf, "The future of multiprocessor systems-on-chips," in*Proceedings of the 41st Design Automation Conference*, pp. 681– 685, ACM, New York, NY, USA, June 2004.
[3] G. Martin, "Overview of the MPSoC design challenge," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pp. 274–279, 2006.
[4] R. Joost and R. Salomon, "Advantages of FPGA-based mul- tiprocessor systems in industrial applications," in *Proceedings of the 31st Annual Conference of the IEEE Industrial Electronics Society (IECON '05)*, pp. 4451–4506, November 2005.
[5] C. Wright and M. Arens, "Fpga-based system-on-module approach cuts time to market, avoids obsolescence," *FPGA and Programmable Logic Journal*, vol. 6, no. 6, 2005.
[6] S. Raje, "Catching the FPGA productivity wave," *Electronic Design*, vol. 52, no. 24, p. 20, 2004.
[7] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer, "An FPGA- based soft multiprocessor system for IPV4 packet forwarding," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 487–492, August 2005.
[8] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for FPGA-based soft multiprocessor systems," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '05)*, pp.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 3, March 2017

273–278, ACM, Jersey City, NJ, USA, September 2005.
[9] P. Huerta, J. Castillo, J. I. Ma´ rtinez, and V. Lo´ pez, "A microblaze based multiprocessor SoC," *WSEAS Transactions on Circuits and Systems*, vol. 4, no. 5, pp. 423–430, 2005.
[10] J. Dykes, P. Chan, G. Chapman, and L. Shannon, "A multiprocessor system-on-chip implementation of a laser- based transparency meter on an FPGA," in *Proceedings of the International Conference on Field Programmable Technology (ICFPT '07)*, pp. 373–376, December 2007.
[11] O. Lehtoranta, E. Salminen, A. Kulmala, M. Ha¨nnika¨inen, and T. D. Ha¨ma¨la¨inen, "A parallel MPEG-4 encoder for FPGA based multiprocessor SOC," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 380–385, August 2005.
[12] R. K. Karanam, A. Ravindran, and A. Mukherjee, "A stream chipmultiprocessor for bioinformatics," *SIGARCH Computer Architecture News*, vol. 36, no. 2, pp. 2–9, 2008.
[13] A. Tumeo, M. Branca, L. Camerini et al., "A dual-priority real-time multiprocessor system on FPGA for automotive applications," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 1039–1044, ACM, New York, NY, USA, March 2008.