Multi-MicroBlaze-based SoC Design and Its Evaluation on Multi-threaded Applications

Zeynep Bilge, Berkehan Altin, Asst. Prof. Ismail San
University Name: Eskisehir Technical University
Email of students and Supervisor: berkehanaltin@eskisehir.edu.tr
zeynepbilge@eskisehir.edu.tr
isan@eskisehir.edu.tr
Which board used: ZedBoard Zynq Evaluation and Development Kit
GitHub Link: https://github.com/DHLSan/FlashCore
Video Link: https://youtu.be/WHOWx3jIIoo

Abstract— Performance increasing in embedded systems are becoming primary occupation since newer technologies cannot yield expected performance. A novel approach to overcome such problem is using multi-processor systems on chip (MPSoC) since MPSoCs provide parallelism. Parallelism not only increases performance but also complexity of programming since it involves shared memory usage and synchronization. Thus, it is a challenge to develop such system. In this study, it is aimed to develop a multi-processor system on chip consists of more than one MicroBlaze soft-core processor possibly with a hard-core processor and measuring the performance output. In order to do that mutex and mailbox IP cores have been used to make it synchronized the process between the soft-core processors. By doing so, a methodology to implement multi core processors or an architecture will be realized.

Index Terms—MicroBlaze, soft-core processor, hard-core processor, MPSoC, multi-threaded benchmark, FPGA, Zynq-7000

1 Introduction

Multiprocessor systems on chips (MPSoCs) have been utilizing in many different areas since they supply through-put efficiency and reconfigurability options.[1][2] Such systems mainly forming by processors and different peripherals and aims for parallelize task performing. A desired output of multi-core approach is acceleration of a certain task; but on the other hand, it also increases the complexity of programming since it incorporates shared memory and synchronization concepts. Acceleration is required since as the technology developed there are no expected performance increment. When it is evaluated in this context, application specific software and hardware design on multi-core systems could be a solution to overcome expected performance problem. MPSoCs achieves high performance by using thread-level parallelism without strong programming aspects.[3] MPSoCs encounter challenges following:[4]

(1) real-time operation,
(2) low power consumption,
(3) developing environment tool,
(4) operating systems,
(5) software security.

To address such challenges, it is aimed to develop a multiprocessor system on chip by using MicroBlaze soft-core processor. MicroBlaze soft-core processor is based on Reduced Instruction Set Computer (RISC) and developed by Xilinx. Mainly, focuses on a solution for inter-processor synchronization in hardware level by considering shared memory. During the progress phase, multi-threaded benchmarks, and fundamental concepts of synchronization are being researched.

In this study, efficient shared memory usage and synchronization tasks are being tried to solve. The main motivation for this project is demonstrating the parallelism within soft-core processors and acceleration of a given specific task and it is believed that the architecture developed can be re-used for any other implementation. By developing such a architecture, multi-soft-core processors can be available and easily run on any FPGA system.

1.1 Implementing Single MicroBlaze

MicroBlaze is based on Hardvard architecture means that there are separate memories for reserved memory and instructions. MicroBlaze soft-core processor supports four memory interfaces such as: Local Memory Bus, AXI4 Interface, IBM Processor Local Bus (PLB) and Xilinx Cache Link (XCL). One can use Local Memory Bus to access on-chip block RAM, AXI4 Interface to access both on-chip and off-chip peripherals and memory. AXI4-Stream also available and lastly XCL can be used for specialized external memory controllers.

To implement and demonstrate a simple task, a single MicroBlaze has been put in block design. This system consists of one AXI GPIO with two channel and AXI UartLite.

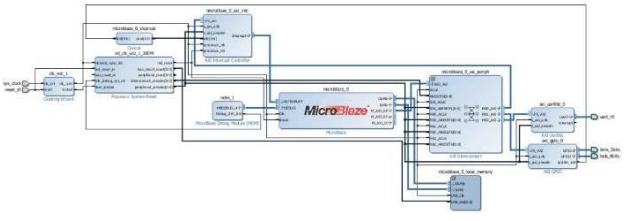

Figure 1: Single MicroBlaze hardware design

2 Inter Processor Communication

2.1 Need for Communication

Technology is improving every day to adhere the increasing need and desire from consumers. In microelectronic technology, the need for faster operations and computing power requires rapid development. In the light of Moore's Law, as transistors per chip is continuing to increase, clock frequencies have hit a wall because of power dissipation. In order to discard this obstacle, using multiple processor cores and turning to task level parallelism makes sense.[5] However this substitution is progressing slowly since it comes with the burden of design challenges. Therefore, managing shared resources and designing inter processor communication plays a big part in these improvements.

When implementing a multi core solution, usually the current tasking is distributed among available cores in order to get the maximum performance out of each core. As a requirement to this operation, processors must communicate and share system resources safely and reliably. Inter processor communication, helps to conduct this process without causing conflict. Using mailboxes and mutexes is an efficient way of using system resources reliably.

2.2 Mailbox and Mutex

In brief, mailbox core provides bi-directional communication among multiple cores by adopting FIFO type messaging. It has configurable depth of mailbox configurable interrupt thresholds and maskable interrupts. It supports bi-directional communication and can operate synchronously or asynchronously.

Mutex core enables processors to lock shared resources in order to block multiple access at the same time. It has configurable number of AXI4-Lite interfaces which reaches up to 8 and configurable number of mutexes.

Both mailbox core and mutex core can be implemented directly from Xilinx IP library and they can be in- dependently configured to use AXI4-Lite or AXI4-Stream interfaces.

3 Hardware Design

3.1 Designing Hardware for Mutex and Mailbox

In order to understand these concepts better, an exper iment to drive the LEDs using ARM Cortex™-A9 and MicroBlaze was planned. In the experiment, MicroBlaze core was driving the LEDs and ARM Cortex™-A9 sent messages to turn the LEDs on and off. The hardware in figure 2 was designed and software to apply designed experiment was conducted.

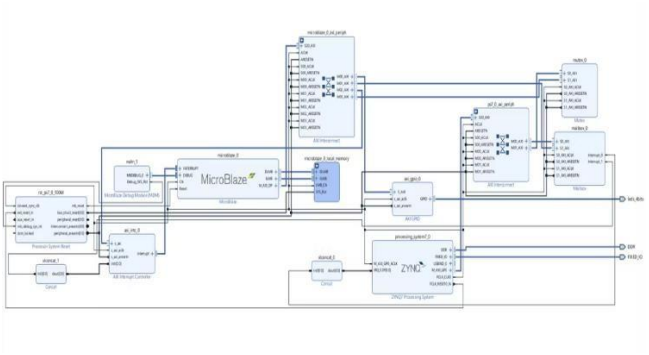

Figure 2: Hardware design for testing mailbox and mutex IP cores.

Since the project includes two different applications running on each core, debug and run configurations had to be made. Each core is selected for each application and debug operation was conducted. The software designed for each core is ran step by step in order to understand the working principle of the IP cores.

In the experiment process, usage of the mailbox and mutex IP cores with multiple master MicroBlaze cores was decided to be researched. After the research and experiments conducted, it was discovered that the mutex IP core can be used with multiple master MicroBlaze cores, however mailbox IP core could not. As a result, it was understood that usage of the mutex IP is more crucial for the project. Mailbox IP will be used when performing consecutive data sharing between only two cores.

3.2 Multi-MicroBlaze Cores

When designing hardware for the project, general plan was starting with two MicroBlaze cores and using Mailbox and Mutex IP cores in order to provide synchronization. According to the plan which have been made, hardware design in figure 4 was implemented.

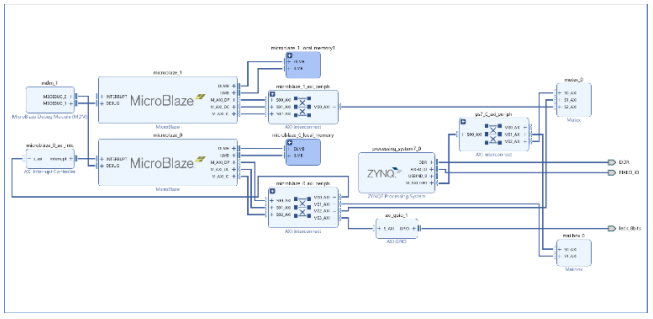

Figure 3: Multi MicroBlaze core hardware design

In this hardware design, two MicroBlaze cores were used for parallel computation and mutex IP core was used for synchronization.

As a test, one MicroBlaze is programmed to do addition operation of dummy values and write the calculated value to the mutex. Second MicroBlaze is programmed to read the calculated value from the mutex and write this value to the LED's.

After this step, designed software is examined by debugging. As a result, it was understood that one MicroBlaze core is writing values to the mutex however mutex is resetting the written values. After the trials that have been made, it was discovered that usage of the mutex as a shared memory element is not possible. Therefore, in order to provide a shared memory resource for different MicroBlaze cores, adding a Bram to the design was decided.

3.3 Adding Bram to Design

One of the possible solutions to overcome shared resource usage problem is adding a BRAM into design. One can access the same resources by storing data in a single memory. Mutex can be used to synchronize and control memory access from processors.

After deciding to add BRAM for accessing the same resources, the implementation in figure 3 was applied into design. In the beginning, for the sake of simplicity only 2 MicroBlazes have been used. In this experiment demonstration of usage of shared memory is aimed. This experiment have been realized by writing into BRAM and reading from BRAM. In research phase, it was discovered that in order to be able to add and connect a BRAM an additional AXI interconnect is needed to be added. This is necessary since each MicroBlaze core has one data port and this port is needed to be connected to mutex and BRAM at the same time to accommodate synchronization. In the experiment process, it was also decided to disable the data and instruction cache of the MicroBlaze cores in order to avoid requirement of providing cache coherency.

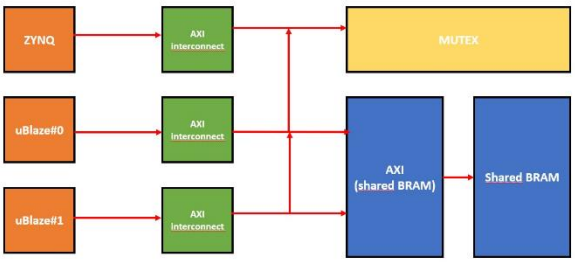

Figure 4: Bram diagram

During the process, it has been observed that reading from the value written BRAM is not successful. In order to understand the reason of this issue research has been performed. It is concluded that a reason was the reset connections of AXI interconnects. Reset connections has been fixed and designed software was tested in new hardware. As a result, the hardware for two MicroBlaze core and one Mutex IP was successfully designed. The design of the resulting hardware is given in figure 5.

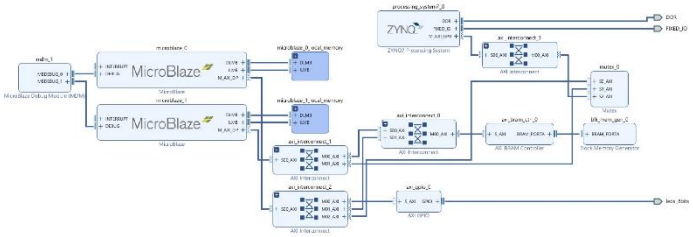

Figure 5: Hardware design

4. APPLYING BENCHMARKING

In order to see the effects of using multiple cores in project, a benchmarking algorithm was decided to be applied.

4.1 Monte Carlo: Pi Number

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. One of the examples of Monte Carlo algorithm is the estimation of Pi. The idea of Pi estimation depends on a hypothetical circle with radius r inside of a square with width of 2r. The next step for the algorithm is to generate a large number of random points with in the square and count how many of these points fall into area of circle. Since the area of the square is $4r^2$ and the area of the circle is πr^2, the estimation formula for pi would be below equation.

$$\pi = \frac{number\ of\ points\ generated\ inside\ circle}{number\ of\ points\ generated\ inside\ the\ square}$$

To apply the explained algorithm, number of steps for designing the software is given below.

- Generation of random point x
- Generation of random point y
- Calculation of d = square root (x*x + y*y)
- If d is smaller then 1, increment circle points
- Increment square points
- Repeat for number of iterations
- Calculate pi with equation 1
-

By following the given steps, a software was designed. In the design process of the software, 32-bit random number generation was achieved with implementation of linear feedback shift register. Inside the algorithm, usage of square root function from math header file was needed however including this header file to the program resulted in overflowing the instruction local memory of MicroBlaze cores.

Therefore, implementation of the square root function was applied using binary search inside the program. The applied software was run in single core of MicroBlaze firstly. As a next step, number of total iterations were dropped half and applied for two microblaze cores. Since these estimations ran concurrently for each core, the software was updated to write the result of each calculation to Bram. The writing process was synchronized using mutex IP. As the final step of this multi-core application, one of the microblazes was attended with responsibility of reading two calculated value from Bram, adding them and diving the result to 2 in order to achieve final estimation of pi.

6. Conclusion

In the final results of the benchmarking application, timing and accuracy of the estimations for single core and multi core applications were examined. As a consequence, when total iterations of 500000 was applied, single core achieved the estimation in 9 minutes. When total iterations were applied to two cores being 250000 iterations for each core, the estimation was completed in 4 minutes. The accuracy of the two estimations were on same level.

In conclusion, it was understood and proved that usage of multiple cores in an algorithm which depends on heavy calculations and high number of iterations can help to reduce the runtime in an application.

REFERENCES

[1] Janac, K. C. (2021). Network-on-Chip (NoC): The Technology that Enabled Multi-processor Systems-on-Chip (MPSoCs). Multi-Processor System-on-Chip 1: Architectures, 195-225.W

[2] Lemonnier, F., Millet, P., Almeida, G. M., Hübner, M., Becker, J.,Pillement, S., ... & Lemaire, R. (2012, July). Towards future adap- tive multiprocessor systems-on-chip: An innovative approach for flexible architectures. In 2012 International Conference on Embedded Computer Systems (SAMOS) (pp. 228-235). IEEE.K. Elissa, “An Overview of Decision Theory," unpublished. (Un- published manuscript)

[3] Acharya, G. P., & Rani, M. A. (2018). FPGA Prototyping of Micro-Blaze soft processor based Multi-core System on Chip. Interna- tional Journal of Engineering & Technology, 7(2.16), 57-60.

[4] Wolf, W. (2004, July). The future of multiprocessor systems-on- chips. In Proceedings. 41st Design Automation Conference, 2004.(pp. 681-685). IEEE.

[5] A. Munir, A. Gordon-Ross and S. Ranka, “Multi-Core Embedded Wireless Sensor Networks: Architecture and Applications”, IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 6, (2014), pp. 1553-1562.