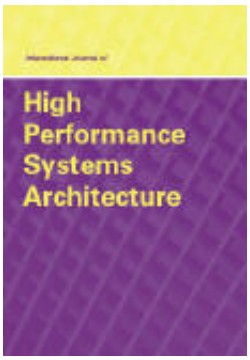


International Journal of High Performance Systems Architecture

ISSN online: 1751-6536 - ISSN print: 1751-6528
https://www.inderscience.com/ijhpsa

MPSoC design and implementation using microblaze soft core processor architecture for faster execution of arithmetic application

Prashant S. Titare, D.G. Khairnar

DOI: 10.1504/IJHPSA.2023.10053187

Article History:
Received: 19 April 2022
Accepted: 01 October 2022
Published online: 06 April 2023

MPSoC design and implementation using microblaze soft core processor architecture for faster execution of arithmetic application

Prashant S. Titare* and D.G. Khairnar

E&TC Department,
D Y Patil College of Engineering,
Akurdi, Pune, 411044, India
Email: pstitare@dypcoeakurdi.ac.in
Email: dgk@ee.iitb.ac.in
*Corresponding author

Abstract: The research paper presents the design methodology with novel task distribution technique on multi-processor system on chip (MPSoC) for speeding up the execution of arithmetic application. Utilisation of multiple soft core processors on field programmable gate array (FPGA) reduces the overload of adding external hardware to a system. Parallel processing of soft core processor with proposed task distribution technique makes any application to execute at faster rate. This task distribution based speed enhancement technique for arithmetic application is very feasible and appealing to the modern applications like neural networks, fuzzy logic, algorithms of machine learning etc. Experimentation on such architecture with arithmetic application shows significant increase in speed of operation with respect to conventional design. This is implemented using Microblaze soft core processor architecture on Xilinx Virtex 5 FPGA board.

Keywords: multiprocessors; soft core processor architecture; embedded systems; VLSI; versions like ultra large; MPSoC; multi-processor system on chip; parallel processing; FPGA; field programmable gate array; high performance; speed enhancement; arithmetic application.

Reference to this paper should be made as follows: Titare, P.S. and Khairnar, D.G. (2023) 'MPSoC design and implementation using microblaze soft core processor architecture for faster execution of arithmetic application', *Int. J. High Performance Systems Architecture*, Vol. 11, No. 3, pp.156–168.

Biographical notes: Prashant S. Titare is a Research Scholar, working under the Research Centre DYPCOE, Akurdi, Pune, affiliated to Savitribai Phule Pune University (SPPU), Pune-Maharashtra, India. He has completed his Mtech in VLSI design and pursuing PhD research work on MPSoC and Embedded Systems.

D.G. Khairnar received his BE from University-of-Pune, MTech and PhD from IIT-Powai-Bombay, India. From 1994-to-1996 he was R&D Engineer in Specialty-Metals Ltd, Pune. From 1999-to-2000, he was Research-Assistant at IIT, Bombay, India. He was session Chair for 4th International Conference (ITNG'07), Las-Vegas, USA. He was Technical Committee Member in 5thITNG-2008, USA. He completed Technical Program conducted by AICTE & British Council under UK India Education and Research Initiative (UKIERI-Phase-III) in 2018–2019. His research interests are in digital-signal-processing, neural-networks, internet-of-things, wireless-sensor-networking and VLSI. Presently he is Head-of-Department and as PhD Research-Guide at DYPCOE, Pune under Savitribai-Phule-Pune-University, India.

1 Introduction

Embedded systems are the systems performing specific task in stipulated time. The performance of embedded application in particular time deadline expects task to execute at faster rate. Also, the very large scale integration (VLSI) technology and its further versions like ultra large scale integration (ULSI) and nanotechnology shows that the amount of area available for particular logic will keep on reducing. Miniature through VLSI and embedded application demand for electronic system to work in faster manner without contributing additional area to the hardware motivates to design the multi-processor system. Most of the embedded devices right from cellular phones, satellites to household appliances like refrigerators, high definition television (HDTV), washing machine etc. are expecting faster execution of tasks. This can be definitely accomplished by using multi-processor system. The paper

contributes in designing of such multi-processor system which improves the execution speed of an application.

Multi-processor systems are the parallel computing processors designed to reduce workload on central or main processor. System can be made multi-processor either by adding hard core processors or by adding soft core processors. Hard core processor addition increases the silicon area on hardware wafer, but the soft core processor can be added or reconfigured without adding external hardware unit in existing system. Soft core configuration is just utilising the number of soft cores within the system. It is because of reconfigurable system like FPGA used for implementation of any application has soft cores available in it. Considering the prototyping level of FPGA, it will have the number of soft cores present in it. Utilising all the existing soft cores is preferred in this proposed technique. More the number of soft cores configured, more tasks can be mapped to it and accordingly more parallelism can be obtained. Such technique is used in proposed method presented in this paper. However, the limitation on utilising the number of soft core processor is the total amount of memory available on board (Lien et al., 2007). Thus for a particular FPGA there is a limitation of number of soft core processors to be used. Hence, an appropriate selection of board and number of soft core processors on it can solve the issue related to multi-processing. The paper presents such soft core based architecture with novel task mapping technique to improve performance in the form of speed, which is explained from Section 6 onwards.

Initially, embedded system architecture were preferred as simple controlling systems for explicit application. These system demands for faster speed of execution without consuming more amount of power in order to execute the modern requirement of application, like encoding or decoding of audio or video information, image processing, etc. and therefore, the multiprocessor system on a chip (MPSoC) is one of the solution to deal with such escalating computational needs (Huerta et al., 2005; Tero et al., 2006; Wolf et al., 2008; Wolf, 2004).

An MPSoC is a multi-processor on a single silicon-chip which may include about two or more number of Processor-Memory Modules (PMM). Considering the 10 PMM, if the memory modules are grouped together into a single first level cache (L1) that is shared between all the available cores then it is termed as a multi-core processor. The term multi-core architecture is used if two or more number of cores is developed on the chips that are interconnected together by appropriate resources (Lien et al., 2007; Cesario et al., 2002; Chandra et al., 2007; Titare et al., 2020).

Recent technologies are demanding such MPSoC architecture based approach for faster execution of application. Faster speed along with efficiency is the ultimate requirement of embedded consumers. All these necessities are possible in small integrated chips through MPSoC.

Thus MPSoC have emerged as an important class in the field of micro-electronics and VLSI. MPSoC shows an absolute method to incorporate multiple processing cores on a single silicon chip. It is promising to recognise following five major steps for proper designing of an MPSoC:

a to develop an application

b to configure platform accurately

c to program a code

d to map an application on to the platform

e to debug.

Most of the MPSoC development work is centred at one of the steps like platform configuration or application mapping. Also instruction set architecture (ISA) based system design approach is preferred for some MPSoC systems (Huerta et al., 2005; Titare et al., 2020).

According to the system architecture model, MPSoC are classified as Homogeneous and Heterogeneous. The Homogeneous MPSoC has processors of similar architecture while Heterogeneous type has different processor architecture on same platform (Lahiri et al., 2004). This research paper provides a design and implementation of homogeneous MPSoC system architecture using Microblaze soft core processor.

Designing of such MPSoC architecture with soft core processors is presented in this paper. Also the task mapping approach to respective soft core processors is illustrated in this research paper which shows significant improvement in speed of execution of arithmetic application. Different arithmetic algorithms are used in industrial applications like decision based self-automated Conveyor belt, temperature monitoring of industrial furnaces, etc. and embedded products like heart-rate monitoring in smart gadgets, etc. Computing such algorithms at faster rate is the demand of the current digital world. The proposed MPSoC approach shows how the desired results can be obtained at faster rate. Hence it reduces the total time consumption for any mathematical operation. Here, the arithmetic application based tasks are used for computations. If the mathematical operation speed is improved then automatically it improves the computational speed of the application. Therefore using the methodology elaborated in this research paper makes the embedded devices to work faster and to provide required output at the earliest. Thus the proposed method contributes in speed improvement of digital applications.

This paper is framed in following sequence, Section 2: explains the Soft core processor architecture used in MPSoC along with its feature; Section 3: elaborates the Communication architecture to be preferred while selecting MPSoC network; Section 4: describes the MPSoC Architecture and its features; Section 5: shares the information related to different types of Communication topologies; Section 6: explains the different mapping techniques available for MPSoC and the proposed mapping technique; Section 7: explains the complete system design using all the previous section parameters; Section 8: shows different Results and Analysis of an application using proposed techniques and the paper concludes in Section 9. Sections 2–5 is included here in order to explain the

different parts involved or selected in the design of proposed technique on MPSoC architecture. The importance of each part of architecture is elaborated here in these sections.

Designing MPSoC using soft core does not add any physical area to the system. Hence soft core approach is preferred. There are different soft core processors available as per (Titare et al., 2020) explored in Section 2.

2 Soft core processor architecture

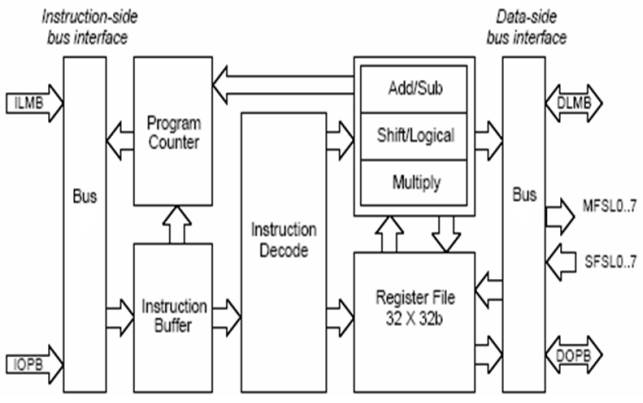
- *Microblaze soft core processor*

The embedded processor named Microblaze is a 32-bit reduced instruction set computer (RISC) soft core optimised for implementation on Xilinx Field Programmable Gate Array (FPGA), as shown in Figure 1 (Xilinx.com 2020; Tong and Anderson, 2006).

The Microblaze processor is exceedingly configurable, as it allows selecting a definite set of features required by design. Hence such soft core processor type is selected for the implementation of proposed application. Such features help MPSoC design to be reconfigurable. Microblaze being a Harvard architecture, it implicates that it has different interfacing units for data bus and instruction bus access. Picoblaze is 8 bit soft core processor, so instead of that 32 bit Microblaze is preferred as a 32 bit instruction set.

Each bus unit is further divided into a local memory bus (LMB) and on-chip peripheral bus (OPB). LMB provides an access to on-chip dual port block RAM. On-chip and off-chip memory and peripherals, both are interfaced using OPB. The Microblaze core also facilitates with 8 input and 8 output interfaces to the fast simplex link (FSL) buses. These FSL buses are unidirectional dedicated communication channels, detail explanation is in Section 3 (Huerta et al., 2005; Xilinx.com, 2020; Tong and Anderson, 2006; Titare et al., 2020). Microblaze is a soft core processor specifically designed for Xilinx FPGAs.

Figure 1 Microblaze soft core architecture



Source: Xilinx.com (2020)

There are different types of soft core processors available like 8 bit Picoblaze from Xilinx, NIOS II from Altera, Leon by Gaisler research and Xtensa series from Tensilica (Gaissler et al., 2005; Titare et al., 2020). Leon3 supports maximum operating frequency of around 400 MHz by using Application Specific Integrated Circuit (ASIC) implementation while Microblaze and NIOS-II has 200 MHz on their respective FPGA platform. However, Xtensa has the highest design flexibility since unlimited custom instructions and execution units can be implemented on the processor's core (Xilinx: XAPP529, 2004; Xilinx White Paper, 2007; Gaissler, 2005; Titare et al., 2020). Hence, based on application requirement, soft core processor needs to be selected. In order to interface such processors network, a communication medium is required between such soft core processors. So, communication architecture is presented in further Section 3.

3 Communication architecture

- *Fast simplex link (FSL) overview*

FSL buses are used as a communication link between Microblaze cores. Microblaze has eight input and eight output FSL interfaces. These FSL channels are dedicated, unidirectional, point-to-point data streaming interfaces.

The width of FSL interface is 32 bits. Data and control words can be exchanged through FSL channel. The FSL interface can have maximum speed up to 300 MB/sec depending on the target device.

The FSL bus system is an ideal choice for inter processor communication (like Microblaze processor to another Microblaze processor) or Input-Output streaming communications (Xilinx.com Logicore IP, 2018; Ryu et al., 2004).

The features of the FSL bus interface are:

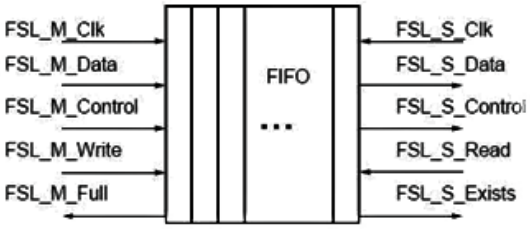
- unidirectional and Non-arbitrated communication
- dedicated, point-to-point communication
- support for data and instruction communication
- 600 MHz standalone operation
- configurable data size
- first-in, first-out (FIFO) based communication (Huerta et al., 2005; Xilinx.com Logicore IP, 2018).

One Master drives FSL bus and in turn FSL drives one Slave. Figure 2 signifies the working principle of the FSL bus system and its signals (referred from Xilinx LogicoreIP datasheet). FSL link allows data transfer through group of macro supported by Xilinx EDK (Embedded Development Kit). Reading and writing of FSL allows communication between processors (Lien et al., 2007; Huerta et al., 2005;

Zeferino et al., 2002; Titare et al., 2020; Brandstatter et al., 2014).

Communication architecture like FSL helps to interconnect soft core. The dedicated connections between cores ensure that data transition delay is less. Such communication link of FSL is used in the proposed method as it helps to interconnect soft core with dedicated connection. As the work is more concentrated on speed of execution of application, the propagation delay in communication link should be minimum. Hence, such dedicated connection of FSL is preferred in designing of MPSoC. Developing architecture with multiple processors and interfacing it using communication link is termed as MPSoC, as explained further in Section 4.

Figure 2 FSL bus signal

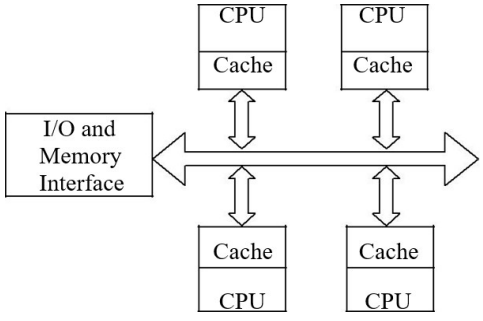


Source: Xilinx.com logicoreIP (2018)

4 MPSoC architecture

VLSI, the branch of microelectronics starts evolving since 1990. The single processor technique developed for an embedded system provides application from communication domain, networking domain, etc. However, multimedia application requires system having multiple processors to compute the given task in shorter stipulated time as compared to single processor time.

Figure 3 Block diagram of MPSoC architecture



Source: Wolf et al. (2008)

Processors involved in multiple processing systems should have the capacity of parallel programming to improve its performance. Very long instruction word (VLIW) based processors are the example for the same, having parallelism approach for the execution of a task (Ryu et al., 2002). Digital signal processing (DSP) processors used in smart phones for signal processing purpose is also executing parallel processing of instructions. On the other hand, ASIC architecture has specific blocks and is not preferred for the design of general purpose application. Following block diagram shown in Figure 3 corresponds to multiple processor architecture.

MPSoC architecture includes multiple processors which collectively has central processing unit (CPU) and memory together, to control all the peripherals (I/O or Input/Output devices) over a network. Memory sharing, data transfer and interconnection are main concern of multi-processing system. Almasi and Gottlieb et al. defined multiprocessors as parallel processing elements that collectively cooperate and communicate to compute complex problem at faster rate (Almasi and Gottlieb et al., 1989; Sabry et al., 2014; Selvameena et al., 2017).

Designing an environment for MPSoC architecture comprise of CPU, cache memory, I/O units and memory interface, as shown in Figure 3. CPU interconnection supports higher level of component integration which will reduce the design area and design time that too without sufficient loss in efficiency. MPSoC environment can be designed with hardware-software co-design, including synthesisable hardware interfaces, hardware accelerators, operating systems (OS) and device drivers. All these units are controlled by OS and application programming interface (API) (Cesario et al., 2002; Ali et al., 2018; Iturbe et al., 2013; Gohringer et al., 2011). This MPSoC will be used as a platform to develop the proposed design methodology with novel technique in this paper.

Communication networks and soft core study inference to choose FSL and Microblaze as more compatible soft core architecture. Thus, all the above mentioned units along with communication link (like FSL for Microblaze) complete the MPSoC architecture. Subsequent Section 5 will elaborate on methods to interconnect multiple processors in a network.

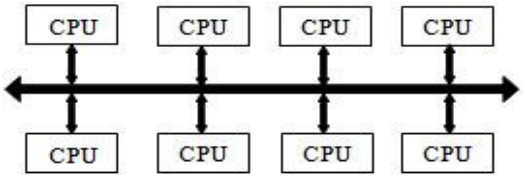
5 Communication technologies

Multiple processors can be inter-connected to each other using different topologies. Selection of topology depends on the type of soft core to be used. Some of the network topology used for connecting various processors (CPU) in a cluster through the FSL link for point-to-point data transfers is discussed as follows: (Huerta et al., 2005; Titare et al., 2020)

Bus topology: In this topology, all the computers are interconnected using a single line through trans-receivers, as shown in Figure 4. All lines should be closed with matched termination. Speed of operation and network performance depends on the number of CPU available on a network. If one CPU transmits a message, then rest all the CPU will be waiting to transmit their data. At a time only one can send data.

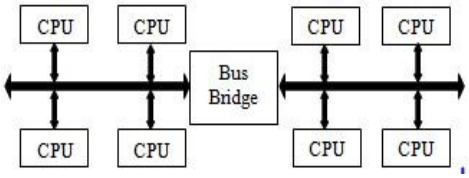
The complete communication network will fail if there is even a single break in the main line. Such bus networks are also called passive topology as the computers or CPU on the bus only responds (or listens) to data transmitted. They do not transfer data from CPU to CPU.

Figure 4 Bus network



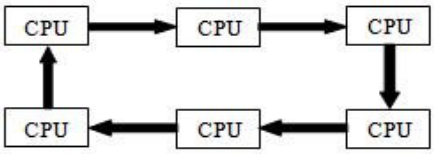
Hierarchical bus network: It is also termed as split-bus architecture. It is a network that connects two or more buses using Bus Bridge, as displayed in Figure 5. This bus bridge is a controllable connection point, which when enabled makes the connection otherwise disconnect it.

Figure 5 Hierarchical bus network



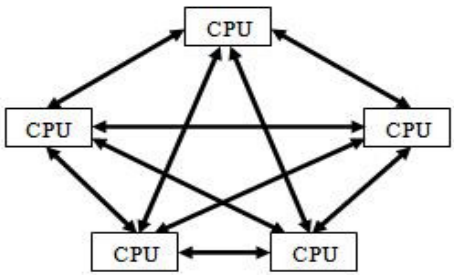
Ring topology: A network in which one CPU is connected to next CPU, in continuous manner and in turn last CPU is connected to first CPU, forming a ring pattern. Its symbolic representation is shown in Figure 6. The data is transferred from one node (CPU) to another until it reaches the last destination node present in a network. The main drawback of this topology is that the data propagation delay will be longer if transmitting CPU and receiving CPU are present at longer distance in a ring. More the number of nodes, longer will be the propagation delay. However, it has the advantage lesser circuit complexity as the number of interconnections is less.

Figure 6 Ring network



Mesh network: It is a network where each and every node (CPU) is interconnected to each other, demonstrated in Figure 7. This type of topology has the least travelling time for data transfer between any nodes over a network. This is because of direct connection between transmitting node and receiving node.

Figure 7 Mesh network

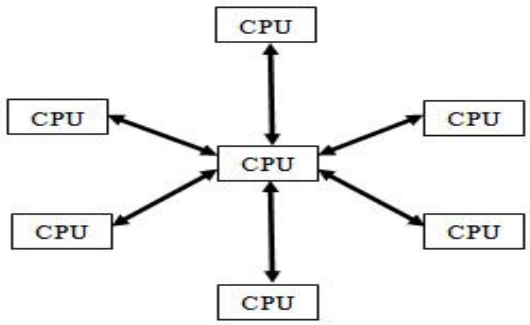


The major drawback of such topology is that the circuit complexity increases as the number of interconnection increases due to increase in the number of nodes in a network.

Star topology or network: This topology has a central node connected to each and every node present in a network. Its structure seems to be like a star (as shown in Figure 8) and hence the name. One centralised controlled called Master CPU and multiple slave type of configuration can be achieved in such topology.

Due to master slave architecture, central CPU decides which task has to be allocated to which CPU, and can also collect results from all these CPU. The main drawback is if the central node fails, then entire control is lost and the complete system fails. As all the operations are regulated through central node, then communication bottleneck can occur for the large bulk of data operations.

Figure 8 Star network



Multi-Star network can be created by grouping individual star network. The central star network is responsible for control of multiple star networks attached to it (Othman et al., 2012; Piscitelli et al., 2011).

All these topologies have some advantages and disadvantages. Depending on the application requirements, a particular type of topology can be selected and mapped on MPSoC. Novelty of this research paper lies with the mapping of tasks. This technique along with the development of MPSoC is explained in forthcoming Sections 6 and 7.

6 Mapping for MPSoC

Mapping of tasks for MPSoC means tasks can be mapped to different CPU as per the topology selected. Previous literature shows the availability of multiple task mapping algorithms (Chen et al., 2011; Wang et al., 2013). However, the proper approach has to be followed while mapping the tasks on MPSoC. Different types of approach for task mapping are mentioned below which are classified on the basis of timing instants when these tasks are mapped:

- *During design time*: If the static or offline mapping approach is preferred then MPSoC resources can be explored at better level using complex process. The only major issue with static mapping is to handle dynamic tasks

- *During run time*: the dynamic or also called online mapping approach needs simple but faster processes as it is handling application when it is in execution mode.

There are two different dynamic mapping approaches for designing an MPSoC, as mentioned below:

- *Mapping with resources reservation*: This method helps to verify whether enough MPSoC resources are available or not before mapping their respective application tasks on system.
- *Mapping without resources reservation*: This approach helps to map the initial task of the application keeping the remaining tasks in waiting state. Task belonging to wait-state are mapped whenever required. Hence, in this case, application execution will start faster, but may wait for other resources to get into ready state.

Run time mapping of tasks in MPSoC is a need for the dynamic mapping approach. Such mapping can be controlled using:

- *Centralised*: One processing element will be centralised, which will control, regulate and combine the mapping process. This central (or single) master will manage the mapping of resources. Due to single centralised control, bottleneck is observed with respect to scalability and performance.
- *Distributed*: Multi-master or multi-cluster approach is preferred. One processing element in each network or in each cluster is responsible for mapping of tasks. It is complicated but better approach with respect to performance and speed (Mandelli et al., 2011).

In addition to use such mapping of tasks, the proposed technique of mapping based on percentage of RAM available with respect to look up table (LUT) available, is to be used.

Comparing with the previous work done by researchers with respect to task mapping, as explained above, proposed method is classified under Distributed dynamic mapping technique. As presented in Sections 7 and 8 the mapping of task based on Block RAM (BRAM) shows the improvement in speed of operation. In order to demonstrate this experimentation, multiple mathematical algorithms were implemented. Some of the arithmetic application is explained with this novel technique of task mapping in further sections.

Consequently, such mapping of tasks on MPSoC with communication link forms the complete system as discussed in further section. This Section 7 presents the design and development of MPSoC as per arithmetic application requirement.

7 The complete system architecture

MPSoC system is collectively a combination of multiple processing cores, having tasks mapped onto it with the respective algorithms. These cores are interconnected to each other using communication link like FSL, depending on the type of FPGA. A single processor system on a chip (SPSoC) consists of one CPU, program and data memory, timer and an application based peripheral. The major difference between SPSoC and MPSoC is that in MPSoC, the number of CPU is not limited to one (Anjam et al., 2010; Nikolov et al., 2007; Beltrame et al., 2008).

Depending on an application requirement, program or data memory can be configured. Additional program memory can be used to operate them in parallel. So, MPSoC provides an advantage of speed through parallelism, thus improves the overall performance (Lien et al., 2007; Huerta et al., 2005; Titare et al., 2020).

MPSoC is defined as group of 2 or more number of processors developed on system on chip (SoC). Hence, the system designed here is of dual processor system. System with more number of processors connected using topology like ring and star is implemented. Star topology based five soft core MPSoC system is developed and demonstrated in Section 8. Dual processor system is developed using Xilinx XPS EDK tool (Xilinx Platform Studio and Embedded Development Kit) (Kangas et al., 2006; Xilinx White Paper, 2007; Xilinx.com EDK, 2018).

Memory units, clocking circuitry and peripherals are interfaced with Microblaze soft core processor using FSL communication link, as shown in Figure 9. The diagram represented in Figures 3 and 9 are almost similar. Figure 3 is symbolic representation of multiple soft core processors connected using FSL bus. While Figure 9 indicates the experimental representation of Figure 3, which is generated after the simulation and synthesis process is completed in Xilinx tool. From Figure 9, black coloured block represents soft core processor connecting to RAM memory using blue coloured LMB bus. Purple coloured bus connects the processors with Microblaze debug module and brown coloured bus connects processor with peripherals. Additionally the blocks like mailbox and mutex is used for inter task communication and synchronous purpose. The red coloured horizontal block indicates Microprocessor Microcontroller Module interface interconnecting the system with external system.

As shown in Figure 9, MPSoC is implemented using mathematical algorithms by distributing entire logic in the form of tasks to different soft core processors. Task mapping to each processor is done using the proposed technique of Computational compatibility ratio, as explained further. Such tasks are distributed to both the processor for parallel execution. Proposed work is having distribution of tasks to different processors based on percentage of RAM available with respect to LUT available with that soft core processor. The technique used to distribute tasks is shown in Table 1.

Table 1 Percentage of LUT and RAM available for dual processor system

Processor no.	*% RAM available*	*% LUT available*
P1	14%	21%
P2	13%	18%

Figure 9 Dual processor system (representing multi-processor system) architecture (see online version for colours)

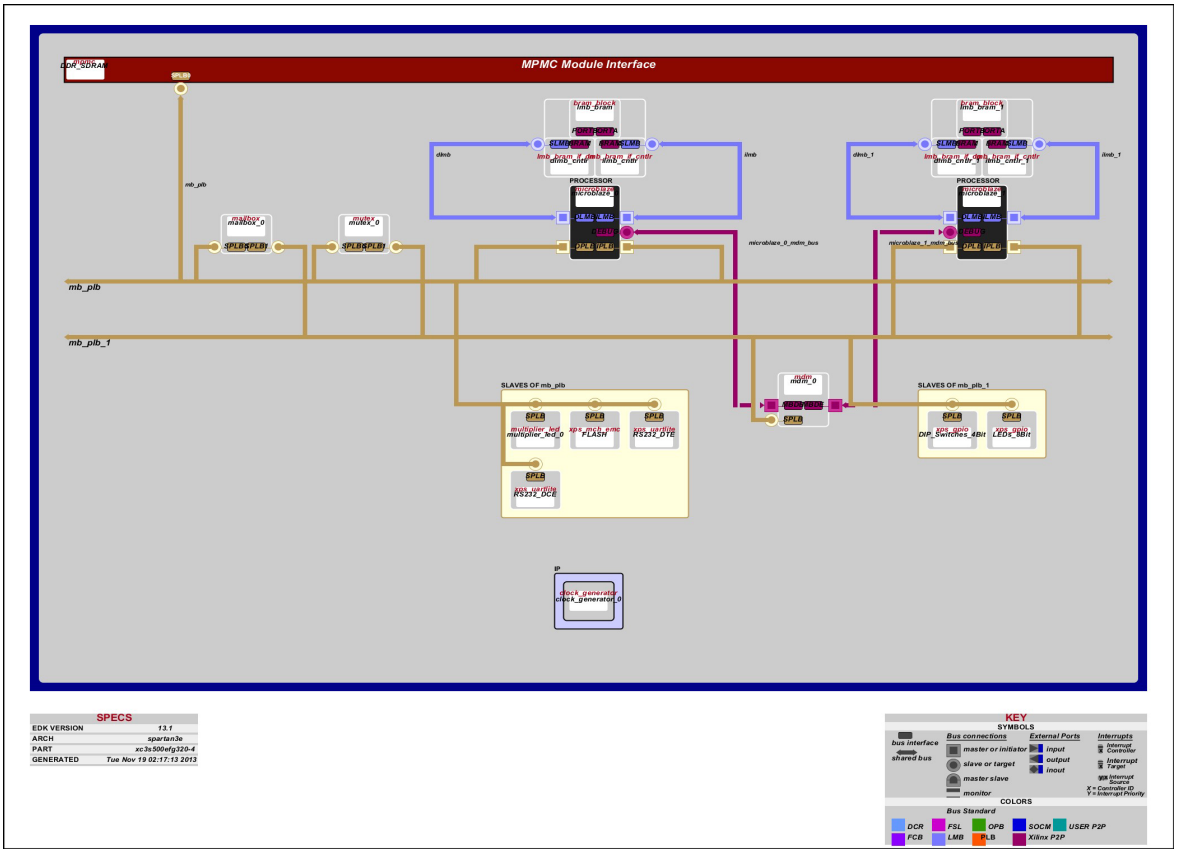


Table 1 is showing the details about the amount of LUT and RAM used by processor for the arithmetic application. P1 and P2 are Microblaze soft core processors with distinct memory size for Block RAM. Based on this table, as the percentage of RAM and LUT available is more, processor computing any task in faster manner can be decided. Ratio for processor P1 is 14/21 = 0.6667 and for processor P2 is 13/18 = 0.7222. Henceforth, this RAM/LUT ratio is called as computational capability. It means if computational capability is more then, that processor will work at faster rate than other processor.

Consider the example shown in Figure 9 and Table 1, where MPSoC is designed with dual soft core processors and with specified RAM memory distribution. This memory distribution is performed by using different address pointer in Addressable bus memory under Xilinx Build Support package tool. After compiling the instance of a code, it generates design summary which clearly specifies the LUT available with soft cores. Now comparing this ratio of computational capability the tasks can be mapped to different processors. From these two processors named P1 and P2, as the P2 processor has higher computational capability ratio, it works faster than P1 processor. This was demonstrated and similar results were observed in multiple examples as shown in Section 8. Hence the second instance of code which executes immediately after the conclusion of first, the processor having higher ratio is allotted with the major task. Mapping this task is in similar manner of calling functions automatically using labels. Higher is the ratio associated with processor then accordingly the major task is labelled to this processor. The entire execution shows that the tasks are mapped adaptively to the processors based on this computational capability with that processor. Therefore the major tasks can be allotted to the processor having higher computational capability ratio. Hence the architecture with more number of processors, can be classified further with their computational capability ratio and all together improves the speed of application. Thus the MPSoC with such technique shows significant improvement in speed.

The results obtained using this proposed technique of task distribution and its analysis is explained in further section named “Results and Analysis”. Huerta et al. (2005) have defined performance parameters named ‘Speedup’ and ‘Efficiency’ to check MPSoC performance as elaborated in the next section. So, all together make the MPSoC to work at faster rate.

8 Results and analysis

Result obtained for MPSoC can be evaluated using metrics like speedup and efficiency as discussed in next sub-section. This task distribution based speed enhancement technique (presented in Section 7 using dual processor system) for arithmetic application is very feasible and appealing to the modern applications like neural networks, fuzzy logic, algorithms of machine learning etc. Each of these applications desires the fundamental usage of arithmetic application. Therefore, further sub-section demonstrates

execution of such arithmetic application on MPSoC. All these applications are selected as these arithmetic applications are most frequently used in any computations or in arithmetic logical unit (ALU) operations. The speed improvement observed in these applications can be achieved through parallelism also. But in addition to parallelism if tasks are mapped as presented earlier then the further speed improvement is observed. To understand this, mathematical parameter like speedup and efficiency can be used. The further sub-sections are organised as first explaining the parameters, then the application with proposed technique and later the overall graphical analysis of application. The expression of speedup and efficiency which determines the system architecture performance is explained first. Based on which the further applications are mapped with these parameters. Arithmetic applications like matrix multiplication, 32-bit multiplier and GCD computation is performed using the proposed technique of MPSoC. Later, the dual processor system explained in previous section is extended with more number of processors and its analysis is presented. Further all this application execution is compared with conventional technique and the comparison is presented in graphical manner.

- *Speedup and efficiency*

Speedup is the ratio of time taken by the single processor (ts) to the time taken by the 'p' number of processor (tp) executing in parallel for MPSoC. Ideally, this ratio should have the number equal to the number of processor (p). It indicates as the number of processor increases, speedup increases.

However, speedup never remains equal to 'p', as there is always a communication overhead. It means certain time is consumed during task mapping and inter- communication, because of which it is lower than p (Huerta et al., 2005).

$$\text{Speedup} = ts/tp \tag{1}$$

For parallel operating processors in MPSoC, performance can also be measured through efficiency. Efficiency is defined as the ratio of speedup to the number of processor (p). An ideal value of efficiency should be one (Huerta et al., 2005).

$$\text{Efficiency} = \text{speedup}/p \tag{2}$$

- *Matrix multiplication application*

The Matrix multiplication of integers and floating-point numbers were performed by Huerta et al. (2005). An application of matrix multiplication was tested to check the parallel execution of the task. The demonstration involves a technique or an algorithm for computing matrix multiplication application.

This technique was implemented by transmitting rows of first matrix (M1) and entire second matrix (M2) to each processor present in a network. For example consider both these matrices are of 2×2 in size. Same is represented in Figure 10(a) as the data transmitted by nodes (2 slave processors) to central node (master processor). Then each processor computes the matrix multiplication for that particular row and corresponding second matrix received. Later after computation, this processor returns the result back to main central processor acting as master processor. Master or central processor is responsible to collect results from each processor and produce output in desired form. The empirical result on console window for experimentation of matrix multiplication is demonstrated in Figure 10(b). It clearly indicates that even if the tasks are distributed between the processors using proposed technique still the same result is obtained. The proposed method of task mapping shows the same result as observed in conventional technique.

Figure 10(a) Matrix multiplication on dual processor system demonstrated using nodal network diagram (see online version for colours)

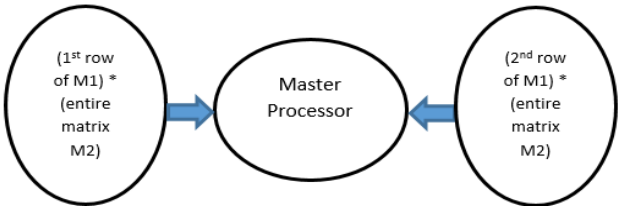


Figure 10(b) Result observed for Matrix multiplication on dual processor system (see online version for colours)

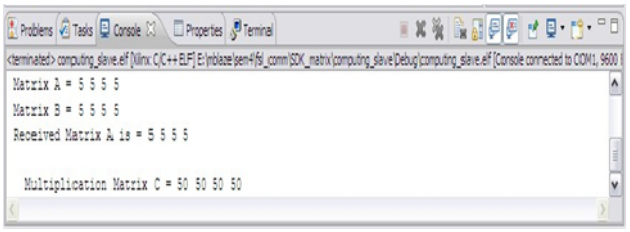


Figure 10(b) shows output obtained for 2×2 matrices. Matrix A and matrix B are transmitted to the slave. And the statement showing 'Received Matrix A' is for the experimentation purpose to indicate that console receives the text information from the program. Slave multiplies each row and column, and gives the result in matrix C. Thus value of each element will be obtained as 5×5 + 5×5 = 50. This can also be performed if only one row of matrix A and entire matrix B is sent to each of the slave, they will compute the matrix multiplication and will return their results to master. And master will display the complete result.

Huerta et al. (2005) performed the matrix multiplication on floating-point values as well. As compared to integer value, the floating-point matrix multiplication consumes more time as the information contents present in the data are larger than the integer values. However, the time required to transfer data will remain same. This extra time consumed in the floating- point operation as compared to integer matrix multiplication can be reduced using floating-point processors.

As the processing time for floating-point multiplication is longer, improvement in efficiency and speedup for MPSoC is observed. However, in floating-point as well,

there will be communication overhead for the transfer of matrix on multiple processors and collecting results back. To overcome this problem of overhead, the common message can be broadcasted to all the processors, was suggested (Lien et al., 2007; Titare et al., 2020). This will reduce the time required to transfer the same message to each processor individually.

With respect to the previous application of matrix multiplication, second matrix is common to all processors and hence can be broadcasted to all the processors, as shown in Figure 10(a). Here, broadcasting communication time is also needs to be considered. Broadcasting common tasks and mapping as presented earlier helps to reduce the sufficient amount of time.

By using this application, different test can be performed by varying different parameters like matrix size, data types, number of processors, etc. Such tests were performed to check the application execution time. The total time consumption includes time taken for the collection of data (matrix values) from memory, transferring data to each processor, processing (multiplication) of the collected data, the collection of results from each processor and storing final results back to memory (Lien et al., 2007; Huerta et al., 2005; Titare et al., 2020). Also, there is research available based on improvement of clocking circuitry (Johnson et al., 2017; Almeida et al., 2011; Logue et al., 2013). In addition, the work referred in Darwish et al. (2006), Arnold et al. (2014), Sultan et al. (2018), Aust and Richter (2012), Stavrou et al. (2007) and Chawki et al. (2022) is based on developing MPSoC or network on chip (NoC) using different modelling style, different threading and task mapping approach for different application. Comparing this with the proposed method presented in the paper, the task mapping is distinct and it shows approximately 6–7% improvement in speed for those applications also.

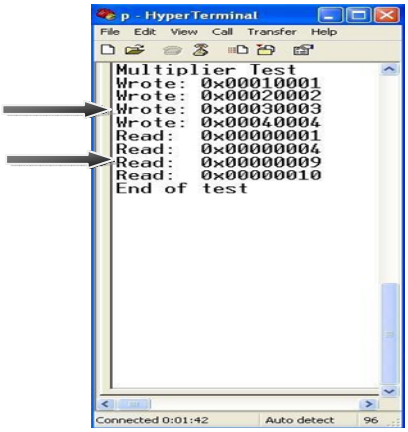
From this it can be concluded that the results were not as good as it could be expected since the efficiency gets affected as the number of processors increases. However, this effect is less than compared to speed improvement obtained from parallelism. For example, if the matrix multiplication by single processor takes 10 micro second (µs) to conclude with results, then ideally dual processor should compute it in 5 µs. But this time is more than 5 µs. This is due to communication overhead. Time taken for distributing the data in multiple processors and collecting it back gets added in total time. However, due to parallelism, the overall time consumed shows significant reduction as compared to the single processor system.

- *32-bit-Multiplier design application*

Similar arithmetic application of 32 bit multiplier can also be executed on HyperTerminal using UART (Universal Asynchronous Receiver Transmitter). From this Figure 11 it can be clearly observed that multiplication of two 32 bit numbers written in memory can be multiplied to get 64 bit output. And this result can be read from memory using Read address variable for that BRAM memory.

For instance, memory is written with value '3' in 32 bit word then its 32 bit multiplication result can be observed in Read register as '9' value as output. Figure 11 shows similar results as observed in conventional system. Computational time required for this system is reduced and same can be observed in graph shown in Figure 14.

Figure 11 32-bit multiplier application and its execution on UART HyperTerminal (see online version for colours)

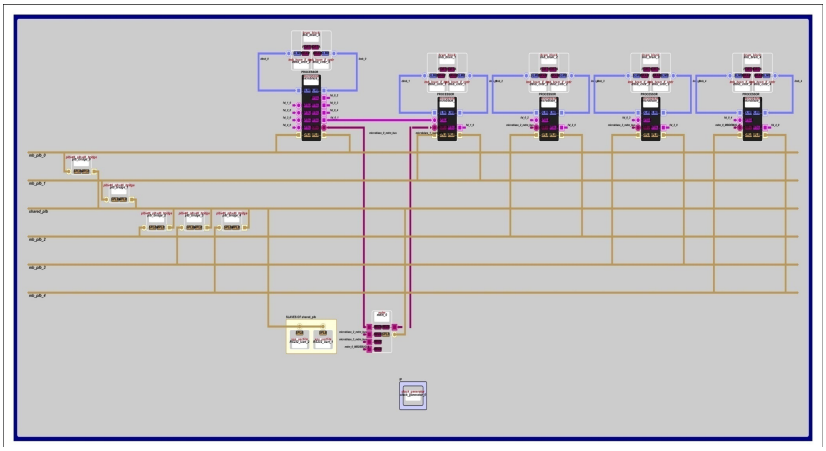


- *Greatest common divisor (GCD) computation*

While using MPSoC with proposed task distribution technique for GCD arithmetic computation, sufficient amount of speed improvement is observed. GCD is calculated by comparing two numbers and obtaining their common divisor. In Dual processor system, Master takes 2 input values from user and forwards it to slave. This is possible in programming part of the system, using Base System Builder. While writing a program for an application, it can be decided that which soft core processor should be made as master and others as slave. Later, while executing the application the module two instance of program decides the task mapping to soft core based on RAM/LUT ratio. These examples are divided into small tasks using program functions. Further these functions are mapped to soft core using ratio and accordingly they are called by function labels. For this application of GCD, program is written to compute GCD by taking test case values from user. Master program stores it in variable and then transfer it to function where the computation is calculated. The functions are interconnected using FSL bus macro like PUTFSL and GETFSL (which transmits and receive the preprocessing values within the soft core processors) It transfer the data between one soft core to other, for example, value from one processor is transferred using PUTFSL and the same value is received in another processor using GETFSL. Thus the value from master can be transferred to slave and is returned using these macros. Master sends it to slave and slave then calculates GCD and returns result back to master in order to display the answer for user. This application also demonstrates that the proposed task distribution technique is helpful for faster execution of arithmetic application.

As discussed in previous Section 7 (related to dual processor system), similar thing is demonstrated further for more number of processors. Results observed for such system and its analysis is elaborated in this section. The same dual processor system presented in previous section is extended to five processor system to verify the proposed mapping technique on MPSoC architecture. If five Microblaze soft core processors are connected in star topology manner, as shown in Figure 12, then results obtained for the same is displayed graphically in Figure 13. The amount of RAM to LUT ratio available with each processor helps to decide the processor that can work at faster rate.

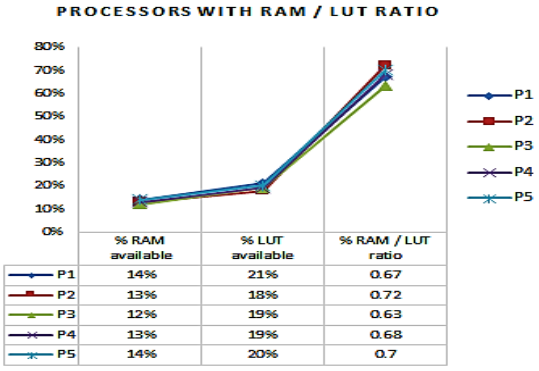
Figure 12 Environment for multi-processor system on chip architecture (star topology) (see online version for colours)



The reason to present graph in Figure 13(a) is to show the similarity in result observed for different processors. As the columns shown in Figure 13(a) is partially co-related, the same graph is demonstrated in bar chart format in Figure 13(b), as RAM and LUT are distinct entity. Also for some of the arithmetic applications presented earlier the amount of RAM utilised is approximately 14%. Hence the ratio observed is 0.7 approximately. The resultant values observed for ratio is in closed range (0.6–0.72) indicates that mapping of task in such similar valued range processor also shows significant improvement in speed. Like the processor with computational compatibility ratio 0.72 shows faster execution than processor with 0.68. Hence the processor P2 having ratio value 0.72 is mapped with major task. Therefore the speed improvement is observed in this MPSoC system. Similarly, more the computations involved in processor, more RAM memory will be utilised and accordingly the ratio value will change. In such case also, the processor having higher ratio will give faster result. Hence such processor will be automatically mapped with major task of computation using function call, as presented earlier.

From this experimentation and analysis, as shown in Figure 13(a) and (b), it is observed that processor P2 is having higher ratio of computational capability than others. Hence, from this it can be concluded that the task expecting more computation is to be allotted to the processor P2, as it is having more computational capability ratio. More the number of processors, more will be the speed of execution considering the tasks are correctly allocated. However if many processors are added in the network then the task distribution time comes into consideration which increases the communication overhead. In addition to more number of processors, if tasks are mapped to processor using this computational capability ratio, then significant increase in speed is observed as presented in Figure 14.

Figure 13(a) Graphical analysis indicating processor having higher RAM/LUT ratio (see online version for colours)



	% RAM available	% LUT available	% RAM / LUT ratio
P1	14%	21%	0.67
P2	13%	18%	0.72
P3	12%	19%	0.63
P4	13%	19%	0.68
P5	14%	20%	0.7

Figure 13(b) Bar chart indicating processor having higher RAM/LUT Ratio (similar to Figure 13(a)) (see online version for colours)

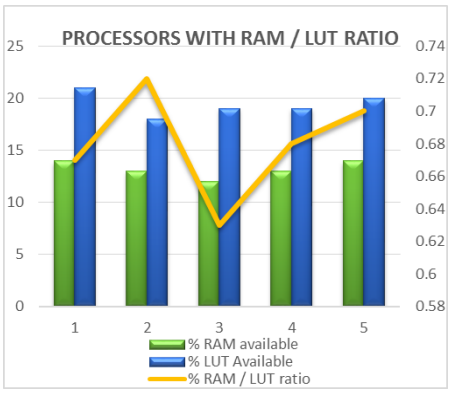
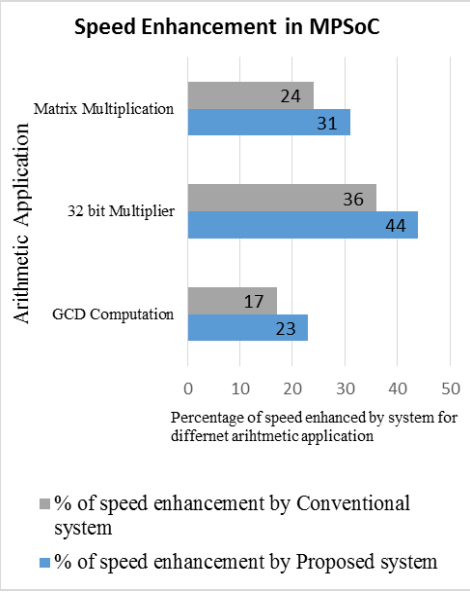


Table 2 Percentage of speed enhancement observed in arithmetic application

Arithmetic applications	*% of Speed enhancement*		*Percentage improvement observed*
	Conventional	*Proposed*	
Matrix Multiplication	23.96	31.05	7.09
32-bit Multiplier	36.02	43.72	7.7
GCD Computation	16.89	23.14	6.25

Table 2 displays speed improvement for the arithmetic application and corresponding graph is demonstrated in Figure 14.

Figure 14 Graph of speed enhancement by different applications for Conventional method and proposed method (see online version for colours)



All these applications are implemented on Xilinx Virtex 5 FPGA (XUPV5LXT) (Xilinx.com ML505, 2021) and the clock rate measured through System builder function shows the improvement in speed of execution as displayed in Figure 14. From Table 2 and graph as shown in Figure 14, it is observed that proposed system gives higher speed of execution than conventional system. Results and analysis of each and every application shows that the amount of time consumed by conventional system is more than the proposed system. Hence the speed enhancement is more for proposed system. This means the proposed system will execute an arithmetic application at faster rate than conventional system. By measuring the difference between the speed enhancements between these two types of system, it implicates that 7% of speed is improved by proposed system. For matrix multiplication, proposed system provides 31% of speed and conventional as 24%. This 24% of speed improvement is of Dual processor system as compared to single processor system.

And for same dual processor system, if proposed technique (RAM/LUT) of task mapping is used then 7% more speed improvement is observed. Similarly, by performing the other two applications like 32 bit multiplier and GCD computations implicates significant improvement of speed. Hence, the enhancement in speed of around 7% is observed in all the three applications. This clearly indicates that if the tasks are distributed using this proposed technique of RAM to LUT ratio then significant improvement in speed is observed. Similar results were obtained for LCM (Least Common Multiple), where 54 microsecond time was consumed by proposed technique compared to 76 microsecond of time consumed using conventional technique. Thus the execution performance of arithmetic application for Microblaze based MPSoC architecture is improved.

9 Conclusion

This research work presents the design and implementation of MPSoC architecture using soft core processors named Microblaze for faster execution of arithmetic application. The research paper concludes that use of soft core processor reduces the overhead of adding extra hardware to the system. Hence, with- out adding extra hardware to an embedded system, tasks can be computed at faster rate. And the utilisation of FSL link inferences that it can be preferred for inter-communication between multiple parallel processing processor. The selection of topology also contributes to make system execute the task at faster rate.

By using the proposed technique, it shows that the task having major complexity of computation should be allotted to the processor having higher value of RAM to LUT ratio. From results obtained for arithmetic application and its analysis with respect to speed enhancement clearly shows that proposed technique will save the CPU execution time. By experimenting some of the arithmetic application, this research paper concludes that proposed system shows 7% of speed improvement as compared to conventional system.

Usage of these technique in other application like neural networks, fuzzy logic, and machine learning algorithm also shows the significant improvement in the speed of execution.

References

Almasi, G.S. and Gottlieb, A. (1989) *Highly Parallel Computing*, 2nd ed., Benjamin/Cummings Publishing Co., Inc. Redwood City, CA, USA, pp.670–678, ISBN: 0-8053-0443-6.

Almeida, G., Carara, E., Busseuil, R. and Hebert, N. (2011) 'Predictive dynamic frequency scaling for multi-processor system on chip', *IEEE Conference on Programmable Logic,* pp.1500–1503, Id No. 978-1-4244-9472-9.

Anjam, F., Wong, S. and Nadeem, F. (2010) *A Shared Reconfigurable VLIW Multi-Processor System*, Delft, Netherlands, IEEE, pp.1–8, ID No. 978-1-4244-6534-7/10, ISBN: 978-1-4244-6533-0.

Arnold, O. and Fettweis, G. (2014) 'Adaptive runtime management of heterogeneous MPSoCs: analysis, acceleration and silicon prototype', *2014 International Symposium on System-on-Chip (SoC*), October 2014, Tampere, Finland, pp.1–4, doi: 10.1109/Issoc.2014.6972444.

Arpinen, T., Kukkala, P., Salminen, E., Hännikäinen, M. and Hämäläinen, T.D. (2006) *Configurable Multiprocessor Platform with RTOS for Distributed Execution of UML 2*.0 *Designed Applications*, Tampere University of Technology, Institute of Digital and Computer Systems, Korkeakoulunkatu 1, FI-33720 Tampere, Finland., ISSN: 1558-1101, IEEE, July.

Aust, S. and Richter, H. (2012) 'Energy-aware MPSoC for real-time applications with space-sharing, adaptive and selective clocking and software-first design', *International Journal on Advances in Software*, Vol. 5, pp.368–377.

Beltrame, G., Fossati, L. and Sciuto, D. (2008) 'High level modeling and exploration of reconfigurable MPSoCs', *NASA Conference on Adaptive Hardware Systems,* IEEE Conference, October, pp.324–337, 978-0-7695-3166-3/08.

Ben Othman, S., Ben Salem, A.K., Abdelkrim, H. and Ben Saouo, S. (2012) *MPSoC Design Approach of FPGA-Based Controller for Induction Motor Drive*, L.E.C.A.P-E.P.T./I.N.S.A.T., Tunisia, IEEE, pp.134–139.

Benchehida, C., Benhaoua, M.K., Zahaf, H.E. and Lipari, G. (2022) 'Memory-processor co-scheduling for real-time tasks on network-on-chip many core architectures', *International Journal of High Performance Systems Architecture*, Vol. 11, No. 1, March, pp.1–11, doi: 10.1504/IJHPSA.2022.121877.

Brandstätter, S. and Huemer, M. (2014) 'A novel MPSoC interface and control architecture for multistandard RF transceivers', *IEEE Access Journal*, Vol. 2, August, pp.771–787, doi: 10.1109/ACCESS.2014.2345194.

Cesário, W.O., Lyonnard, D., Nicolescu, G., Paviot, Y., Yoo, S. and Jerraya, A.A. (2002) 'Multiprocessor SoC platforms: a component-based design approach', *TIMA Laboratory, IEEE Design and Test of Computers*, *IEEE Design and Test of Computers*, Vol. 19, No. 6, December, pp.52–63, doi:: 10.1109/MDT.2002.1047744(2002 IEEE).

Chandra, V.B., Sharma, V. and Chaudhari, M. (2007) *Issues with Designing a Dual Core Processor with a Shared L2 Cache on a Xilinx FPGA Board*, Project report of researcher at Indian Institute of Technology, Kanpur, IIT Kanpur, Y3383, Y3393, May.

Chen, H., Yin, L. and Peng, G. (2011) *Implementation of Multi-Core Embedded System on Compound Guiding System*, Beijing China, IEEE, pp.4348–4352, ID No. 978-1-4577-0321-8/11.

Darwish, T., Kabbani, S. and Sleiman, A. (2005–2006) *Multi-Processor System Design on FPGA*, Final Year Project Report at The American University of Beirut, Faculty of Engineering and Architecture, Spring.

Gohringer, D., Hubner, M., Zeutebouo, E.N. and Becker, J. (2011) 'Operating systems for runtime reconfigurable MPSoCs', *Research Article at International Journal of Reconfigurable Computing*, KIT, Germany, February, Hindawi Publication, Vol. 2011, ID. 121353, p.16.

Huerta, P., Castillo, J., Mártinez, J.I. and López, V (2005) *A Microblaze Based Multiprocessor SoC*, HW/SW Co-Design Group, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid Spain, Research-Gate – 29226 7954, November, pp.423–430.

Iturbe, X., Benkrid, K., Hong, C., Ebrahim, A., Torrego, R., Martinez, I., Arslan, T. and Perez, J. (2013) 'R3TOS: a novel reliable reconfigurable real time operating system for highly adaptive, efficient, and depandable computing on FPGAs', *IEEE Trans on Computers*, Vol. 62, No. 8, pp.1542–1556, August.

Johnson, A.P., Chakraborty, R.S. and Mukhopadyay, D. (2017) 'An improved DCM based tunable random generator for xilinx FPGA. IEEE Trans. on Circuit and Systems, Vol. 64, No. 4, April, pp.452–456, ISSN: 1549-7747.

Kangas, T., Kukkala, P., Orsila, H. and Salminen, E. (2006) 'UML based multi-processor SoC design framework', *ACM Transactions on Embedded Computing Systems* (*Nokia Research Centre*), Vol. 5, No. 2, May, pp.281–320.

Lahiri, K., Raghunathan, A. and Dey, S. (2004) 'Design space exploration for optimizing on-chip communication architectures', *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 23, No. 6, pp.952–961.

Lien, C-M., Chen, Y-S. and Shih, C-S. (2007) 'On-chip bus architecture optimization for multi-core SoC systems', *Published in Software Technologies for Embedded and Ubiquitous Systems. SEUS. (2007) Lecture Notes in Computer Science*, Vol (4761) *Springer*, Berlin, Heidelberg, ISBN 978-3-540-75663-7, pp.301–310, '2007.

Logue, J.D., Percey, A.K. and Erich, G.F. (2013) *Synchronized Multi-Output Digital Clock Manager*, European Patent International, Publication no WO 2002/029974, Application No., PCT/US2001/031251, Vol. E81-C, No. 2, February, pp.277–283, ISSN: 0916-8524.

Mandelli, M., Amory, A., Ost, L. and Moraes, F.G. (2011) *Multi-Task Dynamic Mapping Onto NoC-Based MPSoCs*, 1PUCRS – FACIN – Av. Ipiranga 6681 France, August, doi: 10.1145/ 2020876.2020920.

Mohammed, M.S., Tang, J.W., Ab Rahman, A.A-h., Paraman, N. and Marsono, M.N. (2018) 'Rapid prototyping of NoC based MPSoC based on dataflow modeling of real world applications', *9th IEEE Control and System Graduate Research Colloquium (ICSGRC Malaysia),* August, pp.217–222, doi: 978-1-5386-6321-9/18.

Nikolov, H., Stefanov, T. and Deprettere, E. (2007) *Efficient External Memory Interface for Multi-Processor Platforms Realized on FPGA Chips*, LIACS, Leiden University, The Netherlands, IEEE, ISSN: 1946-147, elec: 1946-1488.

Piscitelli, R. and Andy, D. (2011) 'A high-level power model for MPSoC on FPGA', *International Parallel and Distributed Processing Symposium*, Pimentel Computer Systems Architecture Group Informatics Institute, University of Amsterdam, The Netherlands, IEEE, pp.259–272.

Ryu, K.K. and Mooney III, V.J. (2004) 'Automated bus generation for multiprocessor SoC design', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 23, No. 11, November, pp.1531–1549.

Ryu, K.K., Shin, E.S. and Mooney, V.J. (2002) 'A comparison of five different multiprocessor SoC bus architectures', *Published in Proceedings, IEEE Euromicro Symposium on Digital Systems Design,* Augus, ISBN: 0-7695-1239-9t, doi: 10.1109/DSD 0.2001.952283.

Sabry, M.M. and Atienza, D. (20145) 'Temperature-aware design and management for 3D multi-core architectures', *IEEE Conference on Now Foundations and Trends*, March, pp.96*,* Print ISBN: 9781601987747, doi:: 10.1561/100000 0032.

Sarbazi-Azad, A.H.H. (2018) 'Dark silicon and future on-chip systems', *Elsevier and Science Direct as Dark Silicon and Future On-Chip Systems*, 1st ed., Vol. 110, July, p.304, eBook ISBN: 9780128153598.

Selvameena, T. and Arun Prasath, R. (2017) 'Out-of-order execution on reconfigurable heterogeneous MPSOC using particle swarm optimization', *2017 International Conference on Innovations in Information, Embedded and Communication Systems* (*ICIIECS*), IEEE, pp.1–6, doi: 978-1-5090-3294-5/17@2017.

Stavrou, K., Kyriacou, C., Evripidou, P. and Trancoso, P. (2007) 'Chip multiprocessor based on data-driven multithreading model', *International Journal of High Performance Systems Architecture,* pp.34–43, https://doi.org/10.1504/IJHPSA. 2007.013289, April

Titare, P.S. and Khairnar, D.G. (2020) 'Development of multi-processor system on chip using soft core: a review', *Presented in 2nd IEEE International Conference on Emerging Smart Computing and Informatics (IEEE-ESCI)*, IEEE Pune Section, March.

Tong, J.G. and Anderson, I.D.L. (2006) 'Soft core processors for embedded systems', *18th International Conference on Micro-Electronics-2006,* Research Centre for integrated micro-systems, University of Windsor, ISSN: 1-4244-0765-6/06/$20.00 @2006 IEEE.

Wang, C., Li, X., Zhang, J., Chen, P., Chen, Y., Zhou, X., Ray, C. and Cheung, C. (2013) 'Architecture support for task out of order execution in MPSoCs', *IEEE Trans on Computers,* August, pp.1296–1310, doi:.10.1109/TC.2014.2315628.

Wolf, W. (2004) 'The future of multiprocessor systems-on-chip', *Proceedings of the 41st Annual Design Automation Conference (DAC'04)*, San Diego, New York, USA, ISBN: 1-58113-828-8, doi: 10.1145/996566.996753, 7 June, pp.681–685.

Wolf, W., Jerraya, A.A. and Martin, G (2008) 'MPSoC technology', *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, Vol. 27, No. 10, October, pp.1101–1113, doi: 10.1109/Tcad.2008.923415.

Zeferino, C.A., Kreutz, M.E., Carro, L. and Susin, A.A. (2002) 'A study on communication issues for system-on-chip', *Proceedings of the 15th Symposium on Integrated Circuits and System Design (SBCCI),* SBCCI, December, Porto Alegre, Brazil, pp.121–126.

Websites

EDK Concepts, Tools, and Techniques, Xilinx, http://www.xilinx.com, July 2018.

Gaissler, J: The LEON processor. www.gaissler.com, 2005.cpu

LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11f), Xilinx, http://www.xilinx.com, November 2018.

Microblaze Processor Reference Guide, Embedded Development kit 13.1, Xilinx, http://www.xilinx.com, September 2020.

ML505/6/7 Virtex-5 Evaluation Platform, ML505 Schematic, Xilinx 1280415, November 2021.

Xilinx White Paper on "Designing multiprocessor systems in Xilinx Platform Studio", WP262 (v2.0), November 2007.

Xilinx White Paper on "Getting started with the Microblaze Development kit – Spartan 3E 1600E", November 2007.

Xilinx: XAPP529: Connecting Customized IP to the Microblaze Soft Processor core using the Fast Simplex Link (FSL) Channel. (v1.3). 2004.