PSYCHOLOGY AND EDUCATION (2022) 59(2): 256-266
ISSN: 1533-6939

FPGA IMPLEMENTATION OF MULTIBIT FLIP-FLOP USING MESOCHRNOUS TECHNIQUE

SAMBA ANUSHA[1], MR.M.SRI.VENKAT RAMI REDDY[2]
M.Tech Scholar[1], Assistant Professor[2]
Department of ECE[1,2]
Holy Mary Institute of Technology & Science, Keesara, Bogaram, Ghatkesar Rd, Kondapur, Telangana 501301[1,2]
sambaanurenu@gmail.com[1]

Abstract:

To make the system more modular and to make timing closure simpler, mesochronous clocking replaces strict synchronisation with more flexible clocking mechanisms. The clock signals that arrive at the two ends of the mesokronous interface have the same frequency, but there may be an unknown phase connection between the arrival clock signals on the margins. Sending data between modules requires clock synchronisation. First, in this brief, we offer a unique mesochron FIFO that can handle clock synchronisation and temporary data storage, synchronising data implicitly via explicit flow control synchronisation. The recommended method can work even if the transmitter and receiver are separated by a long link, such that the delay does not fit within the specified operating frequency. Multi-cycle connections may be accommodated using the recommended mesochronous FIFO, which is easily modifiable without affecting the basic concept. An implementation of the new design is proven to have a much lower cost compared to previous state of the art mesochronous FIFO architectures.

INTRODUCTION

TMultiprocessor System-on-Chips (SOCs) are the dominant development architecture in the area of fast computer interfaces (MPSoC). MPSoC has been necessitated by the development of new technologies. It's become necessary to optimise the computer due to its high overhead and energy consumption in this advanced architecture. There are two ways that designers are addressing this issue: by tailoring the design to the limitations of the application[1] and by limiting the operation to a limited voltage/frequency range[2,3]. However, although adaptation is the best method, it comes at a significant cost [4]. Monitoring the protocol and signal interface between various processor components [5] while optimising overhead power and processing is part of the design method. This design's MPSoC optimization is hampered by the wide range of design units and components that are used. Optimizing for specific applications also restricts operating frequency and system performance. So the design method is stated as having an internal clock allocation update procedure[8] and a FIFO-based synchronisation strategy for numerous units in sub-unit activities. Data interchange is synchronised across all core

units in this system[9]. Dual clock FIFOs are used in each IP core processor block.

All IP interfaces must be conservatively configured since the speed and throughput of each IP core are different[10], thus employing a dual-clock FIFO scheme for all IP blocks puts the resource at more risk than just providing it. [11] The worst-case situation might need a change in the buffering of this sync parameter, for example. Additional high-impact specialisations may result from the descriptive presence of frequency ratio information (such as the connectivity of a chip works at a faster rate than linked IP units) and performance restrictions[12]. Because of this, the FIFO dual clock architecture is able to cover a large area and save electricity. In [12-17], there are a variety of applications. The synchronisation method is difficult to accept between two clock variations[18] since the designs do not use clocks. In order to synchronise the core units, the clock system is disregarded. There is a restriction on synchronisation because of this limitation. MPSoC architecture's latency is being reduced, as seen by recent developments. It's not possible to get around the resource allocation latency until after the fact. Delays in the processing of the clock must be allocated to each instruction, which leads in system delays when those assignments are made. The improved clock library function provided in this article is a revolutionary latency monitoring approach. This method reduces the amount of data exchanged and instruction delay, and thus eliminates the overhead latency. The paper is organised into six sections, each of which is described in detail in the following paragraphs. MPSoC's approach to library coding is detailed in Section 2. Section 3 explains the preferred method for library code to assess delay in Mesochronous operations. Results of the simulation were shown in Section 4 and the conclusion was made in Section 5.

I. DISTRIBUTION OPERATION IN VA-MPSOC

CoreVAMPSoC is a nice example of a hierarchical interconnect architecture that is both integrated and energy efficient. Because the core CPUs in this cluster all share the same address space in the core, it's simple to link them. Originally, each CPU could read and write local data from other CPUs through a bus-based interface. This is still the case. By combining a FIFO buffer with cluster connection, CPU penalty cycles may be avoided. During the design of the processing unit, the standard data bus width topology is established. An AXI4 Interconnect Standard 32-bit or 64-bit data bus width, for example, is used by the Advanced Bus Architecture Microcontroller (AMBA). For a given address, AXI4 defines how data and addresses should be sent. R/W bus requests may be issued at the same time thanks to separate reading and writing channels. As a result of the inclusion of extra registration processes, the clock rate may be increased to a maximum of MPSoC clocks. Because the architecture does not allow the best read requests for all cores in the execution sequence, it cannot support the

best read requests in this situation. With a four-clock operating cycle delay of less than one clock cycle, this processor is the least latency-intensive. Share Bus installation requires a total of five intermediates, one for each channel and two for crossbar linkages. The NI (Network Interface) interface is formed by a network-on-a-chip (NoC) that links two CPUs. Each CPU cluster is arranged in a 2D structure using its X and Y coordinate indices. The cluster uses a single address space for all its memory and units. NI bridge-based communication is employed when CPU clusters and packets are transferred over router links. Consequently, it allows for a reduction in CPU working time for this contact in the CPU core. As a consequence, packet data is stored and retrieved directly from the CPU's local memory. Because of this, CPUs make use of the delays associated with reading from and writing to local memory. As a DMA controller for the CPU, NI is also a useful tool. There are two ports on NI that connect to the cluster in the original architecture. Packets may be routed to several R/W separation channels. Where data is written when the NI AXI master port is being broadcast.

An efficient method of communication is required for exchanging schedules across multiple CPU interfaces. Using a single communication channel, CoreVA-MPSOC[19] implements a new communications paradigm. Memory accesses may be interrupted while using this strategy, but it is more scalable and efficient. Most of the time, a single operation will read from and write to many output channels. The data storage of one or more R/Ws is controlled by each channel. Buffer granularity affects synchronisation. An insufficient amount of data may be acquired or no free basic buffer can be written by the CPU when a channel's request for buffering is made.

A random programme may access any memory address in the buffer since data is received through channel. Furthermore, there is no need for extra synchronisation. When the job is done, the CPU connects with the other units to exchange the status and reuse the register. As a result, the administrative burden is reduced thanks to the coordination of efforts. A major delay in allocating funds is still present. Due to resource allocation and sync delays, the time delay calculation is limited to the data interface between CPU units using the NoC interface. As a data exchange route bus, the NoC serves as an arbitrator for buses. This transaction demonstrates a significant lag, resulting in a slowdown in data processing. There must be a reduction in the time restriction, which is the emphasis of this essay, The time stamp library is used in conjunction with a new delay mapping approach to speed up the synchronisation process.

II. MESOCHRONOUS CLOCK VIRTUALIZATION IN MPSOC (VR-MPSOC)

The proposed solution for virtualizing MPSoC activity includes a virtual delay

calculation unit. An MPSoC core unit now has a library unit, and each operation instruction of the processor unit specifies a pre-calculated delay parameter for the library unit. Multi-core processing unit instructions are referred to as 1, 2, or 3 byte instructions in this method of design. Every action is carried out using one of three methods: direct addressing, direct addressing, or two methods of indirect addressing. Each of these classes has a corresponding clock delay. Physical delays due to manufacturing and overall set-up delays are combined into a single time estimate, and the time for data is also included. For each kind of instruction, a delay is computed and stored in a library specific to that core unit. Design processors have continuous instructions and constant delays, hence this delay input technique is carried out at all times throughout a design process. For each instruction, a corresponding delay is set in the library function, which completes delay computation. The core unit keeps a synchronisation table for each library. Arbiter unit delays are mapped to this library unit at the operational stage of processing instructions. Arbitration takes place through the MPSoCNoC interface, which is referred to as "arbiter." The core unit's request for a core bus determines which bus lines will be assigned to it. Each allocation is delayed by the amount of time the mapped instructions take to complete. MPSoC performance is improved as a result of this reduction in latency. Due to clock allocation and computation delays, cumulative delay is the latency parameter for this proposed method. To keep the process in sync between each execution of an instruction, data or instructions are buffered. A delay value is assigned to each unit based on the instruction type. A large portion of the additional latency is due to clock synchronisation and the overall delay from allocation to computation. In addition to the processing delay, a route delay is seen in the allocation of buses and the interchange of data. The functioning of a building with an acceptable architectural floor plan results in a significant amount of traffic delay. In order to circumvent this latency, the library unit removes the primary switching and calculating component. It's a virtual implementation of a time stamp unit that gives back the correct delay value while it's in use. This leads to the development of 'Vr-MPSoC' units, which are essentially virtual MPSoCs. The algorithm for the suggested method is shown below.

Algorithm (Clock switch virtualization)

Process Initialize:

Step 1: *define the cluster of CPU*

Step 2: *allocate the arbiter for data and instruction*

Step 3: *allocate the operation instruction for each CPU*

Process read:

Step 1: *generate a read offset signal to library latency*

Step 2: *recover the time stamp for each instruction*

Step 3: *record the delay to offset library*

Processes execute:

Step 1: *Read instruction*

Step 2: *Decode instruction type*

Step 3: *Read offset value*
Step 4: *Allocate to data and instruction register*
Step 5: *Read data*
Step 6: *Execute instruction*
Step 7: *Write back*
End

The operational block diagram for the proposed approach is presented in Fig. 1 below.

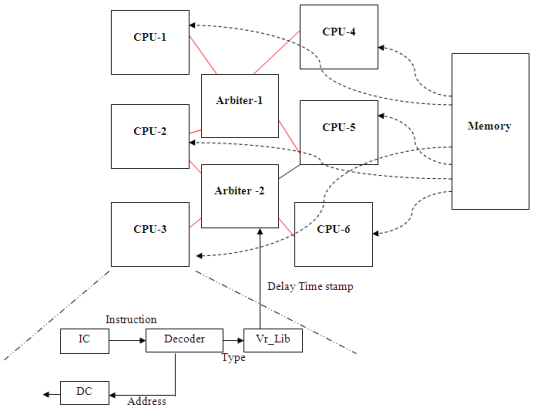


Fig. 1: System Architecture for the proposed MPSoC interface

When a decoded command is mapped to a library unit, the processor unit is updated in Phase-1 of the two operating phases. Every kind of instruction decoded by the arbiter has a delay stamp given to it, which internally sets a delay value based on the current clock cycle. Arbitration is carried out on the basis of the scalar time delay value instead of calculation and allocation. This results in a low latency in the system.

III. SIMULATION RESULTS

The proposed method's operational functionality is evaluated for the scheduling of the given job in three phases of simulation. The delay in allocating and operating resources is computed in advance. The Xilinx ISE synthesiser for the FPGA device is used to test the proposed approach at the second level of implementation. Performance metrics including throughput, area, latency, and overall system responsiveness are shown here. The simulation's ultimate outcome is analysed to determine the different levels of instructional density.

1) Functional verification of operational processes

The Aldec tool generates the HDL description of a simulated specific task for time monitoring. The suggested solution is implemented using Xilinx FPGA devices in the design. For example, measurements are taken for power, latency, throughput, and area, among others. Below are the findings. It is shown in Fig. 2 how the planned work was tested. Rather of requiring a dedicated instruction cache, this method makes use of the core CPU's instruction buffer set. The data acquired from the main memory is buffered by the processing instructions for each core unit.

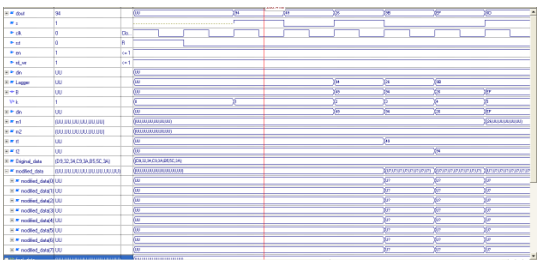


Fig. 2: Operational instruction used for testing

Arbiter units are used to map each instruction to the delay restriction in the test procedure, as seen in Figure 3.

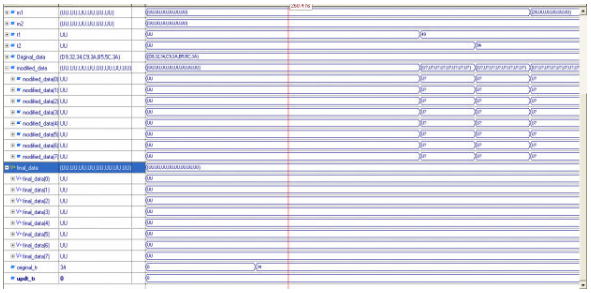


Fig. 3: Mapped instruction of delay metric at arbiter unit

New data is created by mapping a clock pulse, which is decoded by the arbiter as a delay instruction. During execution, the processing unit assigns a clock delay value to each register. When the mapped clock pulse generates fresh data for each instruction, the arbiter interprets this as a delay instruction. Each CPU unit is instantly assigned during the execution of a reading. Figure 4 depicts the mapping and allocation of delay.

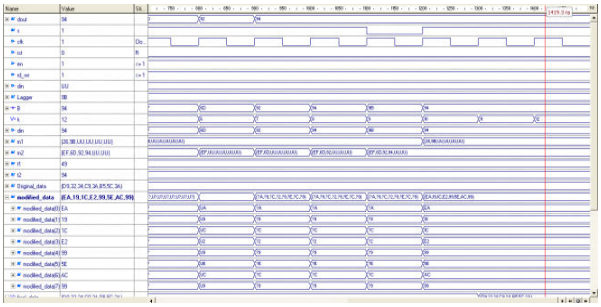


Fig. 4: Delay instruction allocation at arbiter unit

The suggested technique has a latency of 49 compared to the VA-MPSoC design's 54 in the execution of a four-instruction set. Figure 5 depicts what we've discovered.

Fig. 5: Latency measurement for the developed system

B) Implementation result

The Xilinx FPGA device for a Spartan family is used to implement the described technique. Figure 6 depicts the final results of the implementation.

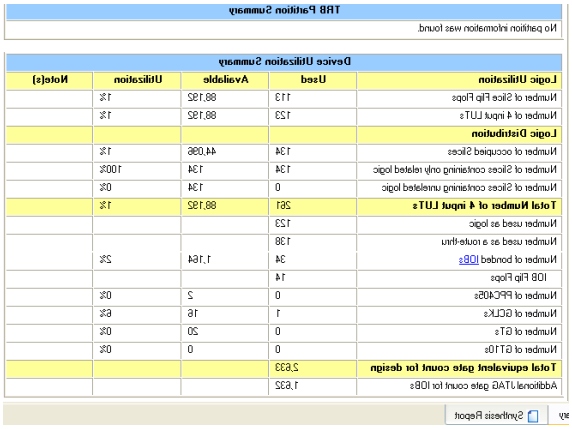


Fig.6: Report of Xilinx FPGA implementation for the developed system

Xilinx's X-power analyzer is used to do a power analysis on the implemented design. It is possible to get a power rating of 181mW.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		181

Vccint 1.50V:	85	128
Vccaux 2.50V:	20	50
Vcco25 2.50V:	2	4

Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco25	0	0
Signals:	0	0

Quiescent Vccint 1.50V:	85	128
Quiescent Vccaux 2.50V:	20	50
Quiescent Vcco25 2.50V:	2	4

Fig. 7: X-power report for developed system

129.98MHz and a time duration of 7.6ns were recorded in the system's timing report during testing. The device has a 2.9 ns setup delay and a 3.6 ns hold delay. Routing and area coverage may be seen with the help of Xilinx-route-and-place. Is Everything Connected?

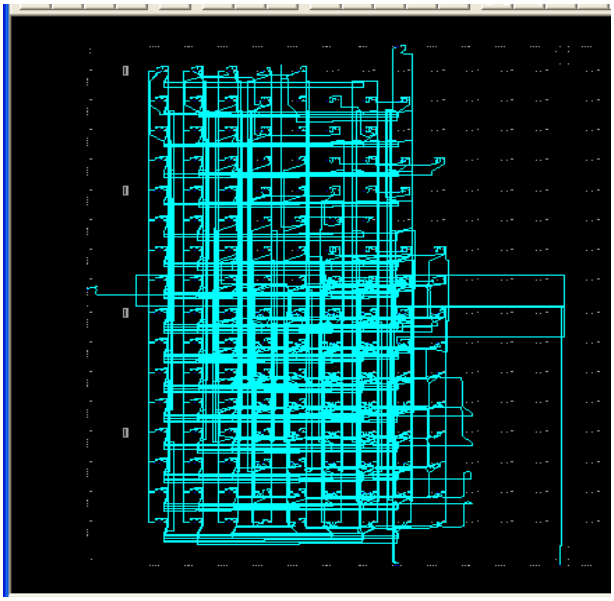


Fig. 8: Logical interconnect of CLB in targeted FPGA device

The logical placement of CLB unit is shown in Fig. 9.

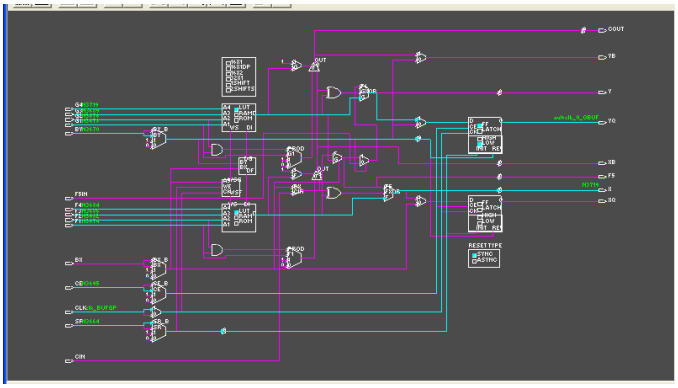


Fig. 9: Logical Placement of CLB unit

Fig. 10 shows the pin configuration for the intended design. There are 12 dedicated IO lines with Vcc and ground pins in this configuration, as shown in Figure 10.

The blue encircled are the allocated line here,

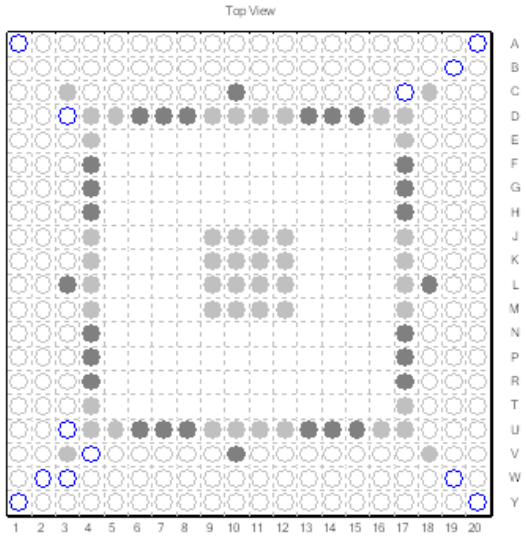


Fig. 10: Pin layout of the implemented design for the targeted FPGA

C) Analysis of developed approach

The ability to accurately measure and record power is critical for evaluating system performance in real time. What is the capacitance, the voltage, and the operating frequency of a set of instructions that have been successfully executed? There are two factors that determine how much power a gadget has: its parameters and its processing frequency. A gadget that is often used will require more power. The power consumption of smaller computations may be reduced by reducing the number of operational iterations and hence the number of operational cycles. Results of the power analysis are shown in Table 1.

Table 1: Observation for power utilization

Instruction density	Power (*mW*)	
	VA-MPSoC [19]	Vr-MPSoC
4	131.5	114.2
5	274.5	182.8
7	321.4	281.3
9	388.7	311.4
12	451.6	411.3

The comparison of power utilization for different instruction density is shown in Fig. 11

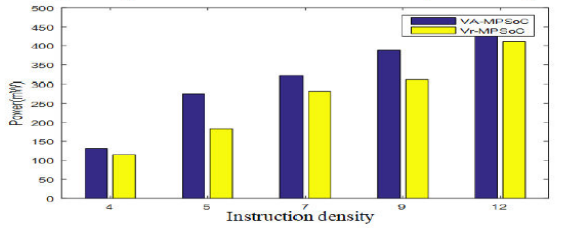


Fig. 11: Comparison of power utilization for different instruction density

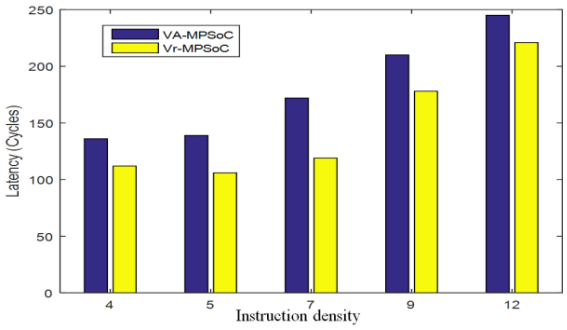
Because of the Vr-MPSoC architecture's short computation cycles, power consumption is lower than it would be for a broad range of instructional systems.

Latency is the amount of time it takes for a computation to complete. Observed delays for the suggested method are summarised in Table 2 below.

Table 2: Latency observation for the developed approach

Instruction density	Latency (Cycles)	
	VA-MPSoC [19]	Vr-MPSoC
4	136	112
5	139	106
7	172	119
9	210	178
12	245	221

The comparison of latency for different instruction density is shown in Fig. 12



System performance may be measured by the number of processing blocks completed each cycle, or throughput, which is the efficiency of the number of processing blocks completed per cycle. In a digital system, the throughput is defined as

$$THR = \frac{Fmax \times Bsize}{LAT} \qquad (2)$$

Where , and LAT are the maximum operating frequency, block size and latency measured.

Table 3: Throughput observation

Instruction density	Throughput	
	VA-MPSoC [19]	Vr-MPSoC
4	196.4	212.3
5	226.1	272.7
7	270.2	321.1
9	321.4	342.4
12	387.1	396.2

The comparison of the processing throughput for different instruction density is shown in Fig. 13

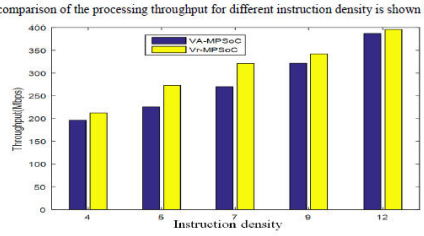


Fig. 13: Comparison of the processing throughput for different instruction density

IV. CONCLUSION

This research mapped the architecture of a chip multiprocessor system using a new approach (MPSoC). Because of the delays in allocating resources, the functional performance of distributed computing is limited. The MPSoC architecture introduces a revolutionary clock time allocation virtualization with library mapping in order to achieve optimal operational performance in spreading processor units. With the foregoing method, latency is greatly reduced, resulting in a reduction in electrical consumption. This signifies that the system's processing power has increased.

REFERENCES

1. L.Benini and G.DeMicheli, "Networks on chip: a new SoC paradigm". IEEE Computer, 35(1):70-78, January 2002.

2. T.Ono, M.Greenstreet, "A Modular Synchronizing FIFO for NOCs", Proceedings of International Symposium on Networks-on-Chip (NOCS), 2009

3. T.Chelcea, S.M.Nowick, "Robust Interfaces for Mixed-Timing Systems", IEEE Transactions on Very Large Scale Integration

Systems, 12(8): 857-873, 2004.

4. D.Ludovici, A.Strano, G.N.Gaydadjiev, L.Benini, D.Bertozzi, "Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs", Proceedings of Design, Automation and Test in Europe (DATE'10), pp. 679–684, Dresden, Germany, 2010.

5. C.Cummings, P.Alfke, "Simulation and Synthesis Techniques for Asynchronous

FIFO Design with Asynchronous Pointer Comparison", SNUG-2002, San Jos`e, CA, 2002.

6. A.Edmanand, C.Svensson, "Timing Closure through Globally Synchronous, Timing Portioned Design Methodology", Proceedings of Design and Automation Conference (DAC), pp.71–74, 2004.

7. P.Caput, C.Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", IEEE Solid-State Circuits Conference (ISSCC), pp.1765–1774, 2006.

8. I.M.Panades, A.Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp.83–94, 2007.

9. D.Ludovici, A.Strano, D.Bertozzi "Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches", Proceedings of 2nd. International Workshop on Network on Chip Architecture (NoCArc), pp.31–36, New York City, NY, 2009.

10. D.Ludovici, D.Bertozzi, L.Benini and G.N.Gaydadjiev, "Capturing Topology-Level Implications of Link Synthesis Techniques for Nanoscale Networks-on-Chip", Proceedings of the 19th ACM/IEEE Great Lakes Symposium on VLSI (GLSVLSI), pp.125-128, 2009.

11. I.Loi, F.Angiolini, L.Benini, "Developing Mesochronous Synchronizers to Enable 3D NoCs", Proceedings of International Conference on VLSI Design, 2007.

12. D.Ludovici, A.Strano, D.Bertozzi, L.Benini, G.N.Gaydadjiev, "Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture", Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip, pp.244-249, 2009.

13. S.Stergiou, F.Angiolini, S.Carta, L.Raffo, D.Bertozzi, G.DeMicheli, "XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips", Proceedings of Design, Automation and Test in Europe (DATE'05), pp.1188–1193, 2005.

14. F.Angiolini, L.Benini, P.Meloni, L.Raffo, S.Carta, "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", Proceedings of Design, Automation and Test in Europe (DATE'06), March 2006.

15. "Specification of optimized GALS interfaces and application scenarios", GALAXY Project deliverable D3, online at http://www.galaxyproject. org/publ deliv.html

16. J.Ebergen, "Squaring the FIFO in GasP", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.194–205, 2001.

17. C.E.Molnar, I.W.Jones, W.S.Coates, J.K.Lexau, "A FIFO ring performance experiment", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.279–289, 1997.

18. R.Apperson, Z.Yu, M.Meeuwsen, T.Mohsenin, B.Baas, "A scalable dual-clock FIFO for data transfers between

arbitrary and haltable clock domains", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15(10), pp.1125–1134, 2007

19. Johannes Ax, Gregor Sievers, Julian Daberkow, Martin Flasskamp, Marten Vohrmann, Thorsten Jungeblut, Wayne Kelly, Mario Porrmann and Ulrich Ruckert, "CoreVA-MPSoC: A Many-core Architecture with Tightly Coupled Shared and Local Data Memories", IEEE Transactions on Parallel and Distributed Systems, Post-Print , December 2017.

Author Details:

A.SAMBA ANUSHA is pursuing Mtech in Holy Mary Institute of Technology & Science, Keesara, Bogaram, Ghatkesar Rd, Kondapur, Telangana 501301. Her interest areas Low Power VLSI,Artifical Intelligence and Machine Learning (AI & ML), Internet of Things(IOT).