Applied Nanoscience (2023) 13:1691–1701
https://doi.org/10.1007/s13204-021-02114-w

ORIGINAL ARTICLE

Design energy efficient shared distributed memory management system on SoC's to improve memory performance

K. Siva Sundari[1] · **R. Narmadha**[1]

Received: 28 August 2021 / Accepted: 24 September 2021 / Published online: 18 January 2022

Abstract
System-on-Chip (SoC) design is becoming increasingly complex as a result of trends such as miniaturisation and data-intensive applications, and it is becoming increasingly difficult to manage without the use of automated design methods. Prior work proposed a novel technique called optimal energy efficient load aware memory management, which concentrated on the amount of extra storage after mapping using a task monitoring algorithm and reduced the amount of energy consumed. Every shared distributed memory algorithm that has been proposed has been influenced by embedded system synthesis theory. The following are the contributions made by this work to the fields of SoC design in general and on-chip memory optimization in particular. This paper proposes a complete workflow to improve memory subsystems in an application-specific way when the system was designed. In general, the proposed memory optimization methods generate good results in Xilinx's 14.2 integrated simulation environment, when integrated into the complete simulation, optimization, and code generation flow.

Keywords System-on-Chip (SoC) · Efficient load aware memory management (ELMM) · Shared distributed memory (SDM) · Latency · Memory mapping mechanism

Introduction

A SoC can be specified as an integrated circuit (IC) which contains multiple independent very large-scale integration (VLSI) designs which forms an operational application to the chip. The predefined cores in SoC are integrated components, which usually include microprocessors, central processing unit (CPU), graphical processing unit (GPU), large memory arrays, audio and video controllers and so on (Yang et al. 2021). These cores are referred to as intellectual property (IP) blocks. Cores can be separated in two classes depending on their nature: soft cores are in form of synthesizable register-transfer level (RTL) description, and the hard cores have been optimized for performance and a certain process. The pay-off between these solutions are flexibility, performance, time-to-market and portability among others. Cores with more design specific attributes tend to have better performance and shorter time-to-market, but they have less reusability, flexibility and have higher cost.

Depending on the nature and scale of the SoC project, there might be third party IP blocks in use. In this case the customer usually offers few premade IP blocks for the deliverer company to build the SoC around, or the customer orders few IP blocks from the deliverer (Shan and Sun 2021). Since all the IP blocks communicate with each other via communication channels, it is critical for the SoC developers to have precise documentation of the IP blocks. SoC design is becoming faster due to the reusability of IP blocks. The fact that SoC designers can also use third party IP blocks speeds up the time-to-market process.

Embedded systems can be defined in a general form as "computer stored on other devices where the computer's existence is not immediately evident". Additionally, embedded devices classify as systems with little complexity that cannot operate external software from third parties in this classification (Failed 2021). This distinguishes them obviously from desktop and server devices. The embedded field is the fastest growing segment of the computer market, driven by the wide range of applications that it is intended to support. Examples include the advancement of mobile devices, the Internet of Things (IoT), natural language

✉ K. Siva Sundari
sivasundari2029@gmail.com

R. Narmadha
narmadha1109@gmail.com

[1] Department of ECE, Sathyabama Institute of Science and Technology, Chennai, India

processing, cloud technologies, multimedia content, robots, self-contained driving, health-care services, and renewable and smart power systems, among many other things. While embedded computer challenges can be extremely diverse, they are typically resolved using one of three approaches. Those are:

1. A hardware/software solution consisting of a custom hardware in conjunction with one or more embedded processors plus appropriate soft-to-use software (such as System-on-Chip (SoC) or System-on-Chip (MPSoC) multiprocessor software).
2. Hardware off-shelf in combination with custom software.
3. Customized software for a digital signal processor (DSP).

Regardless of whether concepts are adopted, certain fundamental problems are similar to embedded system design (Ahmed et al. 2021). In this respect, memory minimization and electricity consumption can be named above all. Robustness, safety and energy efficiency are further criteria.

High pressure on the market makes it difficult in short times and at a reasonable price to build new, creative embedded devices with increasing complexity. Consumers want more functionality and improved performance on mobile devices with stable or lower energy budgets. However, batteries do not improve in the same speed and the typical operation is hampered and already close to the specified stringent limit 0:6 V, which should be reduced the system supply voltage. Challenges that go beyond Moore law scaling are becoming more and more relevant (Rumyantsev et al. 2020). As a typical figure for future technological advances, the ITRS consortium addresses the Power Performance Area Cost (PPAC) value in its 2015 More Moore report. The necessity for instant generation of the data and related applications in Big Data and cloud, IoT, or RTC domains is dictated by $>30\%$ and $>50\%$ power improvements at $>50\%$ and by 35–40% lower costs in the next technology nodes, equivalent to the 2 or 3-year time frame.

As already mentioned, static energy consumption is a function of the current leakage and the current leakage stagnates when the system is transacted. However, there is a different dynamic power usage (Frolova et al. 2020). Define two measurements for the energy assessment, namely the architectural and the transferable power estimates, to emphasise this discrepancy. The estimate of architectural power is the estimate of power in view of the complete transactions of a network. In this example, for all NoC components estimate the entire static and dynamic power usage. When the NoC is run by a specified traffic pattern, the power estimate is the current consumption. For all components and dynamic power, only those that switch in the simulation time are estimated to be full statical power. For instance, when fluid switching is half the maximum NoC capacity, during the simulation time the dynamic power consumption will be half the maximum NoC capacity.

In this work, use two area metrics: router area and connection area. The chip area that all NoC routers occupy is the router area. The parameters of a NoC system are measurable. However, the area of the connection is different. Depending on where the routers are in a SoC system, the length of the link vary (Endo et al. 2020). During the floor plan of a chip which is not accessible during simulation time is allocated the location of routers. Here, assume an area for each core to solve this problem, and it can measure the NoC area without a connection area by measuring the area of routers. The overall link length of the NoC system can now be estimated with this area.

Literature survey

These tasks generating overheads are regarded as vital, even after prefetching (Enemali et al. 2017). They must be reused to remove their overhead reconfiguration (In future iterations of the same task graph execution). Reuse the configurations already loaded (Enemali et al. 2017) is therefore one of the best ways to reduce the overall reconfiguration costs.

The traditional configuration methods have already been extended by Clemente et al. (2016) by dividing the set-up into blocks. This reduces and efficiently manages the granularity of configurations to reduce overhead reconfiguration.

Although the combination of pre-fetch and reuse methods leads to better results, the application's performance isn't optimal (Clemente et al. 2014). Consequently Clemente et al. (2014) supplied a two-storey memory hierarchy. The two levels are on and off the chip. The high speed (HS) and low energy memory on the chip is broken down into (LE). The authors have also offered two configuration mapping algorithms specifically for static and dynamic systems, which reduce energy and time overheads.

Background

FIFO designs and architectures of two types exist: serial and parallel. As shown in Fig. 1, the first FIFO generation that operates with a fall-through principle, such as the shift register, was the serial FIFO. The traditional FIFO architecture, however, is continually improving. Most FIFOs are currently parallel, a suitable way to enhance the number of words stored with faster speed. For two key reasons, this trend is suited for a chip network. The first reason is the fall via concept in which the newly arrived data unit is stored at the FIFO tail, and a step is shifted to the head of the FIFO queue

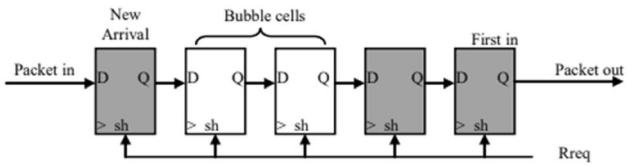


Fig. 1 Conventional shift register (serial) FIFO

at each application shift. The data units are thus shifted at every request throughout the storage space (Tasoulas et al. 2019). This concept has three drawbacks of long downtime, high dynamic energy consumption and high bubble cells. The first disadvantage is that the FIFO increase its capacity and will lead to an increased FIFI latency as its downtime increases. The minimum latency of FIFO depends not on the number of stored items but on the depth of the physical FIFO. Figure 1 illustrates the second disadvantage of bubble cells in FIFO. When data entry/output rates are different, bubble cells can form. Third disadvantage is that data shifts from tail to head of FIFO generates dynamic power consumption. Serial FIFO is simpler, but is unfit to implement on-chip. The architecture of FIFO should not move data elements across all memory sites (Strobel and Radetzki 2019). In other words, the arrival packet should be stored at the front of empty cell rather than at the tail of a queue.

With 0.1 µm technology with relatively smaller buffer sizes and less buffer utilisation the register-based implementation is still possible, but it is not a good choice for 35 nm technology. This is mainly because of increased static power of 35 nm or less, which is completely reduced by the advantage of less activity due to the additional transistors. As the buffer capability increases to dozens, register-based implementation is inadequate because of the larger buffer chip area.

Schematic diagrams used for large-capacity FIFOs are shown in Figs. 2 and 3 respectively for a dual-port SRAM cell and a D-type flip flop (Fan et al. 2019). Provided that two and four transistors in the structure of the NOT gate and the NAND gate are needed. Only one third of the flip-flop

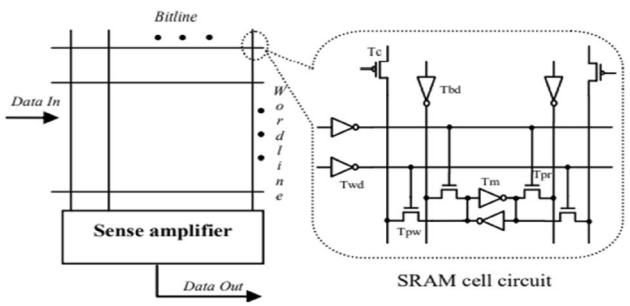


Fig. 2 SRAM-based FIFO

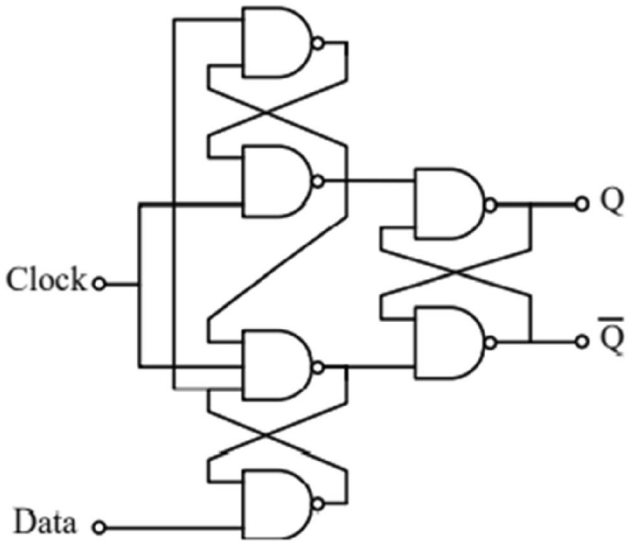


Fig. 3 A positive-edge-triggered D-FF

area of type D occupies an SRAM cell. Nowadays, SRAM implements noc buffers mainly because of the area, power cost and availability of the respective IP cores. The above facts have encouraging us in all proposals in this dissertation to use an SRAM-based buffer and mechanism for parallel style.

Existing work

In this work, which is optimal energy efficient load aware memory management (ELMM) technique and it concentrated on the amount of extra storage after mapping using task monitoring algorithm and it reduces the energy consumption. In shared memory, variables and data structures are specified in such a way that the memory optimization compiler does not perform and uses the most efficiently available Processor Registers as local storage is not available at all. This provides the most efficient usage of the processor registers. With this improved performance, other forms of local storage, such as caches, are circumvented and utilised when accessing the shared storage pool (Failed 2018). Data from common memory will be read when requested by the software, and the results will be returned immediately to the shared memory to complete the transaction. This scheme can be implemented with moderate effort because of its simplicity. To implement this algorithm, any CPU can be utilised.

This approach can be implemented only if the processor can handle explicit instructions for flushing any local storage in memory, which is not always the case due to limits on specific CPU's capabilities. The most important advantage of both methods is, as is clear, the lack of expenditures incurred during the development process in connection with

data transfer between software and hardware (Fig. 4) (Ding 2018).

The benefit of shared memory rather than of local storage systems is the ability to preserve data consistency throughout the course of a transaction, apart from the fact that data are not buffered within local memory of the processor. But the volume of traffic that passes via the processor bus grows significantly as a result of increased activity in the processor bus while the software part of the programme is in use. Due to the much higher mapped memory access times than those of unmapped memory, mapped memory can lead to a significant drop in performance in comparison to unmapped memory (Ahn et al. 2018). It is important to take into consideration elements such as processing autobus speed, mapped memory speed, and the access logic speed connected with mapped memory to ensure these systems operate as effectively as feasible.

Latency and memory management mechanism of proposed method

The method structure for shared distributed memory (SDM) is shown in this section, which is positioned somewhere between the memory controller, which serves as the main character in the share memory illusion, and the scheme for distributed shared memory management (Xu et al. 2017). When the SDM algorithm is used in conjunction with the memory controller, the memory controller's utilities are delivered. With all of the executions, the memory controller for the proposed architecture is ecstatic about handling them all.

Maintaining the index and other statistics of distributed shared memory becomes increasingly important when shared data is transferred at a quicker rate than data transferred by physically dispersed devices (Lapshev and Hasan 2016). A lookup, for example, is one of its key responsibilities, as is mapping statistics and activities into the distributed shared memory system. Consequently, the memory controller is in charge of the index mechanism and action required to maintain a coherent view of the data. Figure 5 demonstrates the construction of a memory controller in conjunction with SDM to create a logical shared space for the purposes of storing data. SDM method and the programme text or client application can be discovered on any distributed network, as can the SDM method and method. Because of the employment of virtual address space, which is depicted in Fig. 5, the memory controller will assist in creating the illusion of having a global address space. The SDM approach will establish links to all of the local memory in the computer system under consideration (Li et al. 2016). Different processes can read and write shared data contents in accordance with the SDM technique, which is based on full replication and allows for the sharing of data contents between processes. As shown in Fig. 6, it will, among other things, manage memory read and write operations to the memory controller in the way depicted in the figure. Accordingly, whenever an application requires access to data from a remote node that has been shared with it, a copy of the data will be made available to it through the memory

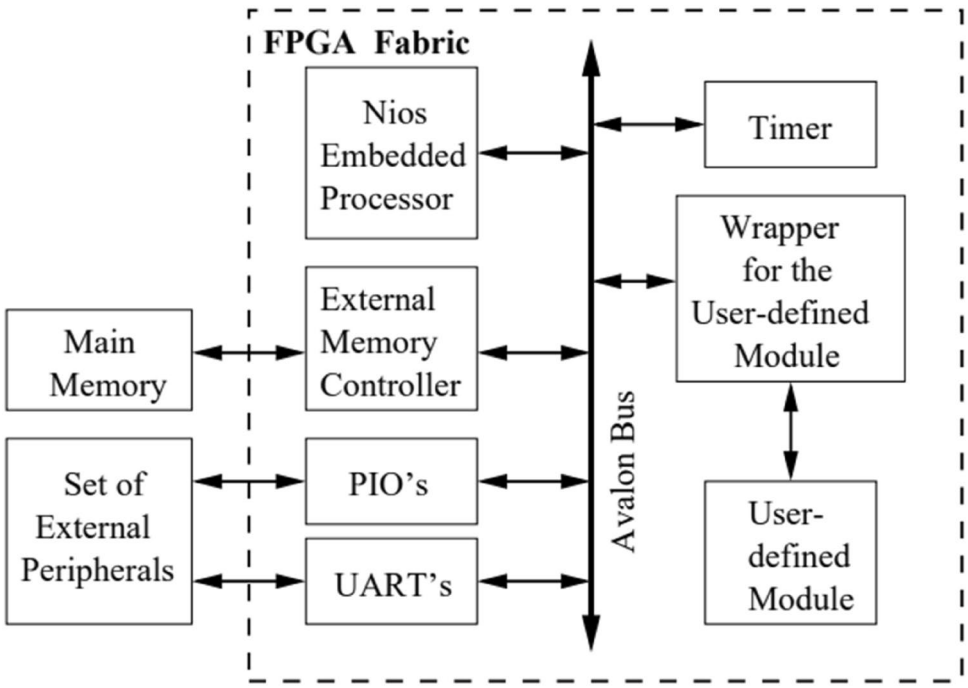


Fig. 4 Block diagram of the efficient load aware memory management (existing method)

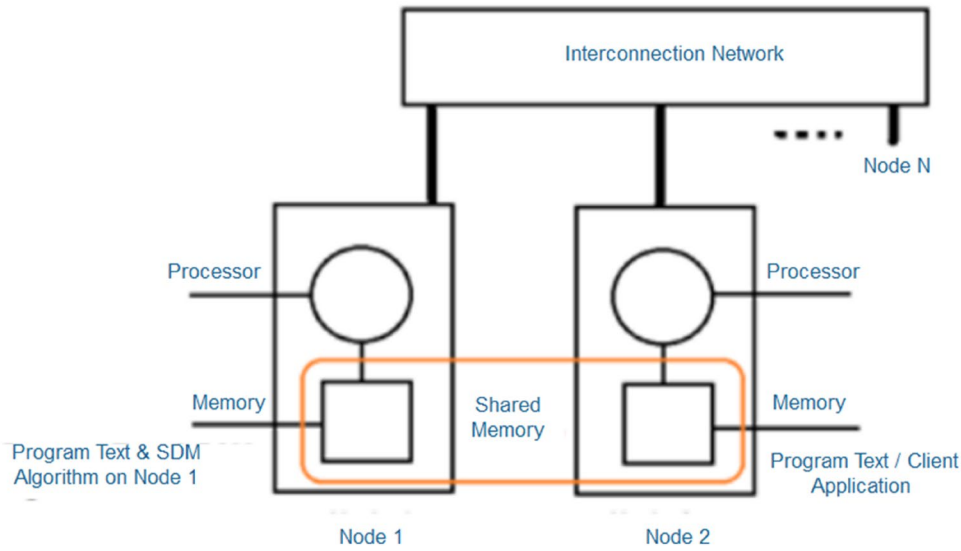


Fig. 5 Shared virtual space with SDM

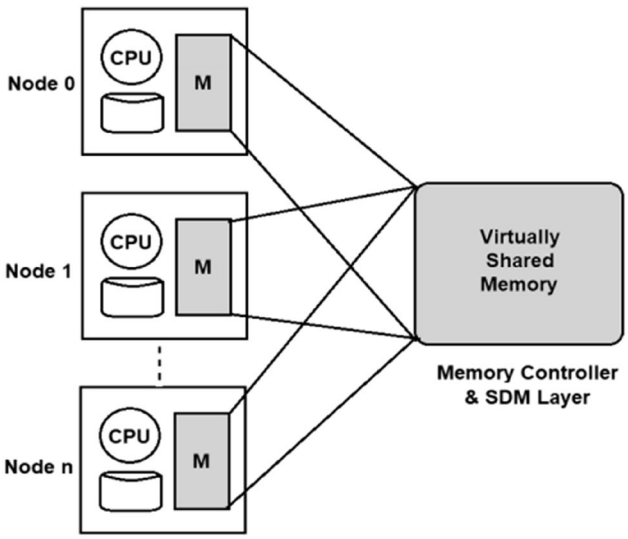


Fig. 6 Memory controller with SDM algorithm

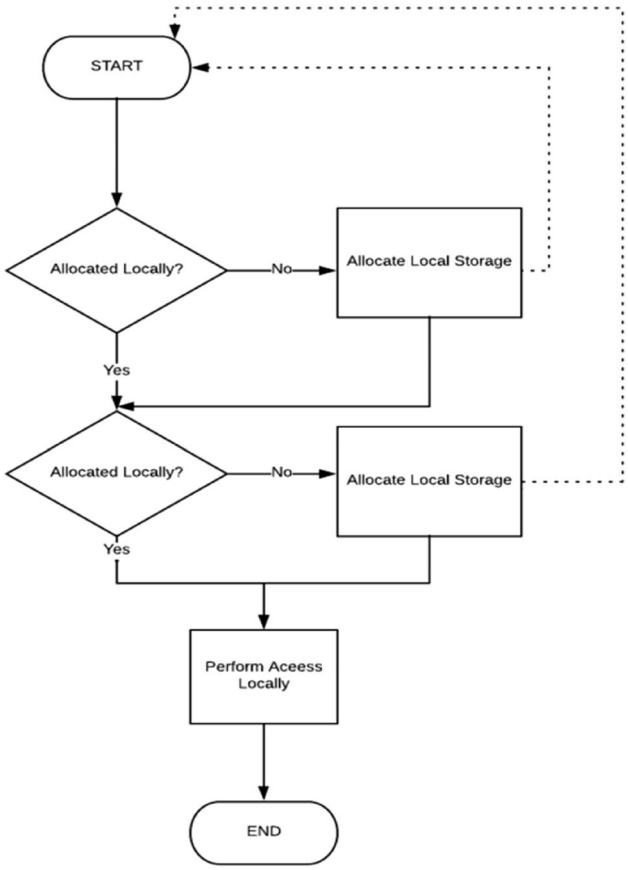


Fig. 7 Flow of memory access semantics

controller application. Write operations can be accessible in the same way that they can be accessed for reading operations (Kulkarni et al. 2016). When many nodes are running the same programme at the same time, it will adhere to the consistency mechanism that is currently in use. As well as the memory controller's design, Fig. 6 shows an example of mapping conducted by the memory controller in the local memory of specific sites.

Access control policies include the global allocation policy (which determines which node will receive a particular request) and the coherence protocol, which are both defined by an access control system (which determines how requests are handled). Figure 7 depicts the memory access policy for shared data items that are both locally and remotely located, as well as how they are accessed. In this policy, both messaging and access controls are used to accomplish

its objectives. Using this approach, the policy could send messages to other sites instructing them to invalidate their copies of the data before allowing the local node to write its own copy of the data.

A single transition and mapping between physical and logical address space is performed by the memory controller, and it is carried out by the memory controller and executed by the memory controller. Figure 8 shows a diagram of how the shared memory region is allocated in terms of memory (right). It also aids in the management of shared memory resources in a more efficient manner. Each site requires both local and global representations of shared data content mapping, and each site is required to have both of these representations. This means that in order for a process to access shared memory every time it grants a logical address to a request process, the mapping policy must convert the logical address into a physical memory location that is appropriate for the request process (50). Given the fact that each individual memory reference points to a different memory content, this conversion is less complicated, resulting in lower execution costs and higher memory performance.

Latency

Chip multiprocessor (CMP) systems should have their memory request serving latencies kept to a bare minimum to improve their overall performance and energy consumption. If you schedule the appropriate memory commands at the appropriate times, you can keep this to a bare minimum. Our scheduler reduces the latency associated with serving read memory requests when the written tail is not full and if memory traffic is not heavy, by delaying switching to write drain mode (Fig. 9).

The handling of storage reads is more important than the handling of storage writes because the memory reads are more critical for the performance of the system than that recorded. If a read queue becomes entirely empty while serving read requests, the postponed writing drop policy assumes that it is more beneficial to wait for forthcoming requests than to enter written drain mode immediately to save time. Because reading performance is more important than writing performance in the system, this has happened.

Write drain mode should be used if read requests are not received within a certain timeframe or if the writing queue is overburdened to the point where the high watermark is exceeded. The delaying written drain mode is used in an adaptive manner to accommodate the amount of memory request traffic. Due to the high volume of traffic in memory requests, written requests are recommended after reading requests as soon as possible instead of waiting for more read requests. However, the delayed write drain will automatically be activated if there's not much read/write traffic. It is determined how often memory requests were made in the past whether there are large or low volumes of memory requests.

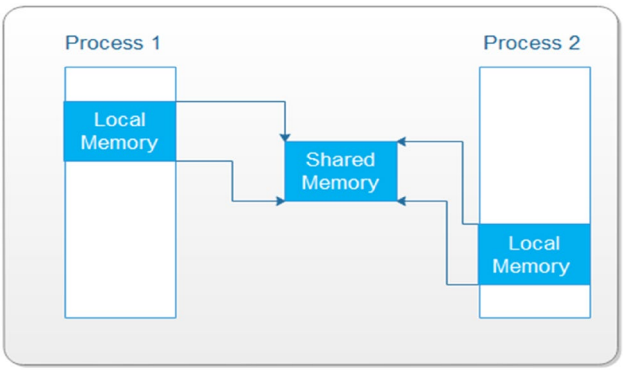


Fig. 8 Memory mapping mechanism

Results and comparison

The RTL schematic of the cache memory compression block is depicted in Fig. 10. It is a design abstraction that models the circuit in terms of digital signals flowing between hardware registers, and it is used in the design of integrated circuits.

The RTL schematic of the compression algorithm is depicted in Fig. 11 (right). Figure 12 depicts a summary of the device utilisation for the design that was implemented. This is demonstrated following the synthesis. This is used to determine the number of devices that were used in the design implementation process (Table 1).

Figure 13 shows the power analysis for the proposed SDM algorithm. This analysis shows total power, supply power etc., Figure 14 shows timing summary for the algorithm developed (Fig. 15).

Conclusion

According to the results of this work, we can see that the proposed scheduler outperforms the single channel memory system in terms of execution time when used in a multi-channel configuration. When comparing the proposed scheduler to other simulated policies, the proposed scheduler outperforms the other simulated policies in terms of power consumption. Despite the fact that the amount of hardware has increased, the proposed scheduler consumes less power than the previous one. The proposed

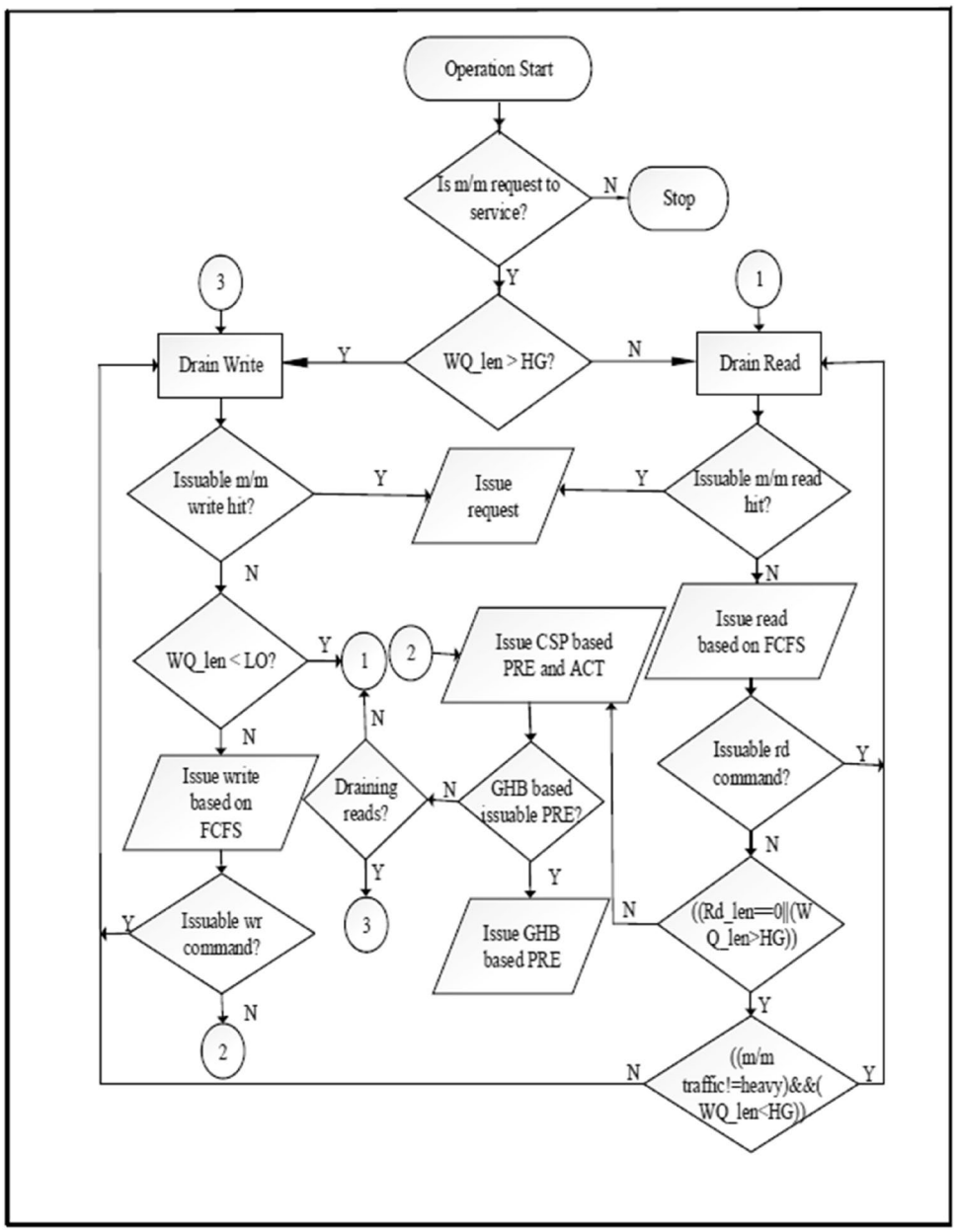


Fig. 9 Flow chart of proposed scheduling approach

scheduler generates speculative precharge and activation commands, which results in an increase in performance as a result of the increased performance. In addition, row hit read/write commands are given a higher priority than memory requests in the queue. With respect to all simulated policies, the proposed approach consumed significantly less energy, resulting in overall performance that was 47.54% better than the existing ELMM technique. Eventually, the proposed approach may be combined with other scheduling policies to improve the overall efficiency of the system. Additional scheduling mechanisms can be implemented to reduce the amount of energy consumed by the system as a result of the refresh operations themselves, as well as the amount of energy consumed during the refresh operations themselves. In future work, With increasing chip density, FPGAs become increasingly resourceful. SoCs are often employed in FPGA application design instead of memory mapped bus to use the resource

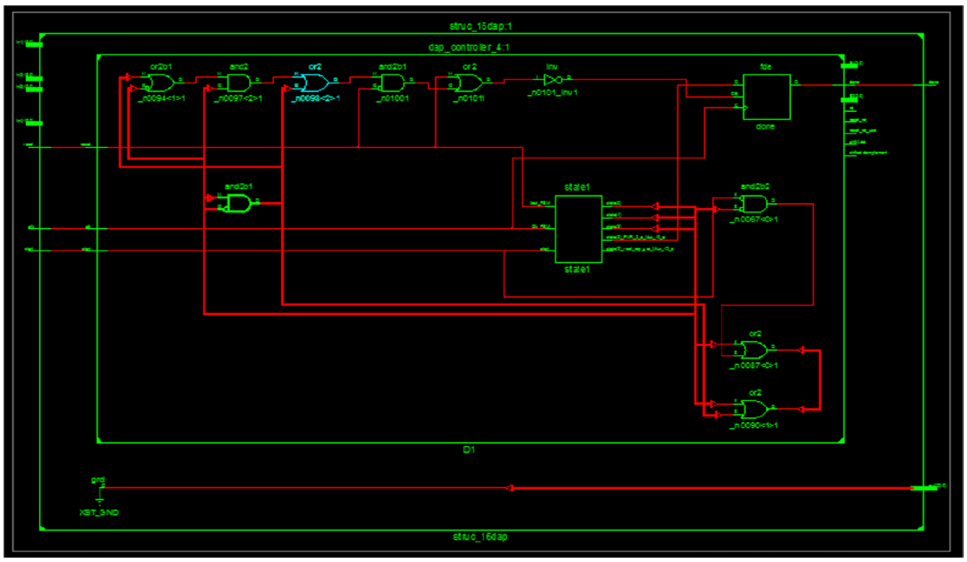


Fig. 10 RTL schematic of the memory block in proposed system

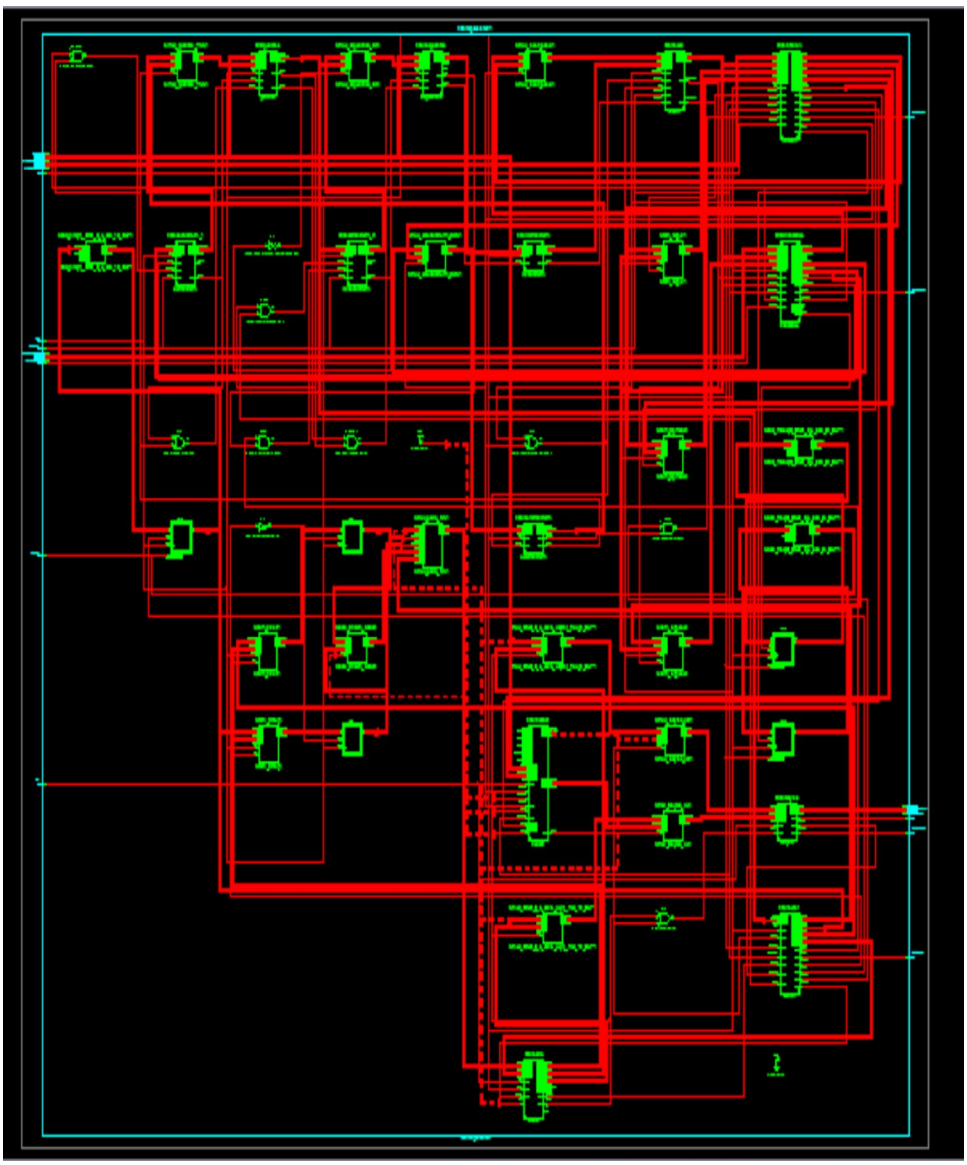


Fig. 11 Technology view of cache memory block

Fig. 12 Device utilization summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	18,176	126,800	14%	
Number used as Flip Flops	18,176			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	23,487	63,400	37%	
Number used as logic	23,487	63,400	37%	
Number using O6 output only	23,350			
Number using O5 output only	0			
Number using O5 and O6	137			
Number used as ROM	0			
Number used as Memory	0	19,000	0%	
Number used exclusively as route-thrus	0			
Number of occupied Slices	6,210	15,850	39%	
Number of LUT Flip Flop pairs used	23,487			
Number with an unused Flip Flop	5,311	23,487	22%	
Number with an unused LUT	0	23,487	0%	
Number of fully used LUT-FF pairs	18,176	23,487	77%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	0	126,800	0%	

Table 1 Comparision table between the existing and proposed methods

	Supply power (W)	Maximum frequency (MHz)	Input arrival clock time (ns)	Output required time (ns)
ELMM	0.082	736.594	3.648	2.437
SDM (proposed)	0.049	194.74	2.921	1.618

completely and to achieve maximum parallelism. This thesis is based on the suggested memory allocator that serves customers connected to the same bus, but it can adapt the communication protocol to systems that have various communication systems, such as SoC. More research is needed in this adaptability.

Fig. 13 power consumption based on power analysis

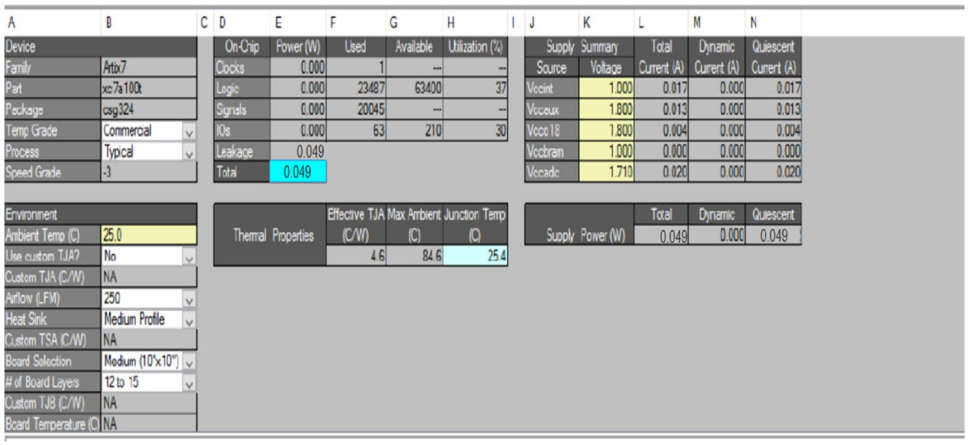
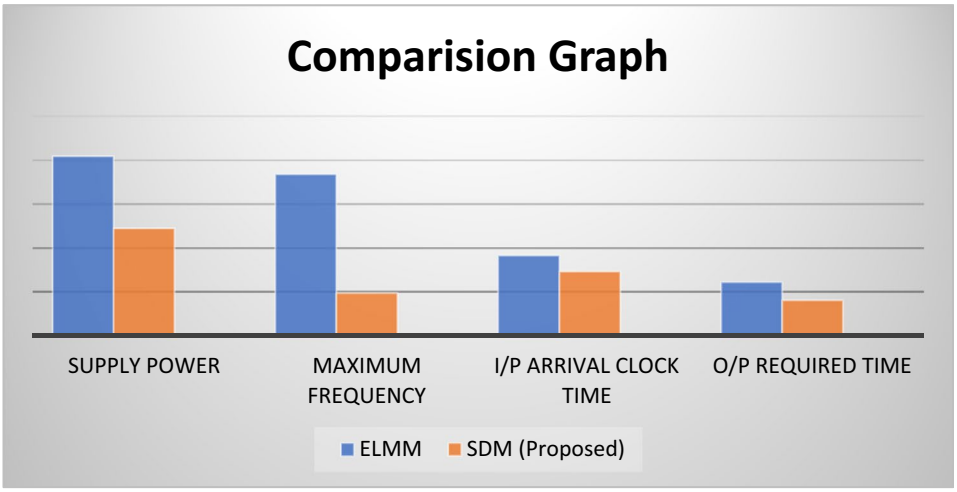


Fig. 14 Timing summary

```
Asynchronous Control Signals Information:
----------------------------------------
-------------------------------------------------------------------------+-----------------------+
Control Signal                                                           | Buffer(FF name)       |
-------------------------------------------------------------------------+-----------------------+
xbuffer/NLOAD(xbuffer/NLOAD:Q)                                           | NONE(Mram_nintop)     |
xbuffer/ichf_GND_53_o_OR_460_o_0(xbuffer/ichf_GND_53_o_OR_460_o11:O)| NONE(xbuffer/Mram_buf) |
-------------------------------------------------------------------------+-----------------------+

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.135ns (Maximum Frequency: 194.744MHz)
   Minimum input arrival time before clock: 2.921ns
   Maximum output required time after clock: 1.618ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'CLK2'
  Clock period: 4.237ns (frequency: 236.032MHz)
  Total number of paths / destination ports: 161483 / 6134
-------------------------------------------------------------------------
```

Fig. 15 comparision graph between existing and proposed methods



Declarations

Conflict of interest The authors declare no conflict of interest.

References

Ahmed MR, Zheng H, Mukherjee P, Ketkar MC, Yang J (2021) Mining message flows from system-on-chip execution traces. In: 2021 22nd international symposium on quality electronic design (ISQED), pp 374–380. https://doi.org/10.1109/ISQED51717.2021.9424306

Ahn S, Kim J, Kang S (2018) Poster: a novel shared memory framework for distributed deep learning in high-performance computing architecture. In: 2018 IEEE/ACM 40th international conference on software engineering: companion (ICSE-companion), pp 191–192

Clemente JA, Ramo EP, Resano J, Mozos D, Catthoor F (2014) Configuration mapping algorithms to reduce energy and time reconfiguration overheads in reconfigurable systems. IEEE Trans Very Large Scale Integr Syst 22(6):1248–1261

Clemente JA, Gran R, Chocano A, del Prado C, Resano J (2016) Hardware architectural support for caching partitioned reconfigurations in reconfigurable systems. IEEE Trans Very Large Scale Integr Syst 24(2):530–543

Ding Z (2018) vDSM: distributed shared memory in virtualized environments. In: 2018 IEEE 9th international conference on software engineering and service science (ICSESS), pp 1112–1115. https://doi.org/10.1109/ICSESS.2018.8663720

Diep T-D, Furlinger K (2021) Nonblocking data structures for distributed-memory machines: stacks as an example. In: 29th Euromicro international conference on parallel, distributed and network-based processing (PDP), pp 9–17, https://doi.org/10.1109/PDP52278.2021.00012

Endo W, Sato S, Taura K (2020) MENPS: a decentralized distributed shared memory exploiting RDMA. IPDRM 2020:9–16. https://doi.org/10.1109/IPDRM51949.2020.00006

Enemali G, Adetomi A, Arslan T (2017) FAReP: fragmentationaware replacement policy for task reuse on reconfigurable FPGAs. In: IEEE international parallel and distributed processing symposium workshops (IPDPSW), pp 202–206

Fan H et al (2019) High-precision adaptive slope compensation circuit for system-on-chip power management. In: 2019 IEEE 38th international performance computing and communications conference (IPCCC), pp 1–2

Frolova PI, Chochaev RZh, Ivanova GA, Gavrilov SV (2020) Delay matrix based timing-driven placement for reconfigurable systems-on-chip. EIConRus 2020:1799–1803. https://doi.org/10.1109/EIConRus49466.2020.9039108

Kulkarni N, Yang J, Seo J, Vrudhula S (2016) Reducing power, leakage, and area of standard-cell ASICs using threshold logic flip-flops. IEEE Trans Very Large Scale Integr Syst 24(9):2873–2886

Lapshev S, Hasan S (2016) New low glitch and low power DET flip-flops using multiple C-elements. IEEE Trans Circ Syst I Regul Pap 63(10):1673–1681

Li Y, Wang H, Liu R, Chen L, Nofal I, Chen Q, He A, Guo G, Baeg S, Wen S, Wong R, Wu Q, Chen M (2016) A 65 nm temporally hardened flip-flop circuit. IEEE Trans Nucl Sci 63(6):2934–2940

Rumyantsev A, Krupkina T, Losev V, Maksimov A (2020) Development of a measurement system-on-chip and simulation on FPGA. EIConRus 2020:1851–1854. https://doi.org/10.1109/EIConRus49466.2020.9039249

Shan L, Sun H (2021) Distributed collaborative simulation middleware based on reflective memory network. In: 2021 IEEE 24th international conference on computer supported cooperative work in design (CSCWD), pp 274–279, https://doi.org/10.1109/CSCWD49262.2021.9437793.

Strobel M, Radetzki M (2019) Design-time memory subsystem optimization for low power multi-core embedded systems. In: 2019 IEEE 13th international symposium on embedded multicore/many-core systems-on-chip (MCSoC), pp 347–353. https://doi.org/10.1109/MCSoC.2019.00056.

Tasoulas Z, Anagnostopoulos I, Papadopoulos L, Soudris D (2019) A message-passing microcoded synchronization for distributed shared memory architectures. IEEE Trans Comput Aided Des Integr Circ Syst 38(5):975–979. https://doi.org/10.1109/TCAD.2018.2834423

Xu P et al. (2017) The research of distributed shared memory technology in power system. In: 2017 IEEE 2nd information technology, networking, electronic and automation control conference (ITNEC), pp 1309–1313. https://doi.org/10.1109/ITNEC.2017.8285008

Yang P, Wang Q, Huang X, Mi X (2018) Work in progress: an confidentiality and integrity scheme for the distributed shared memory of embedded multi-core system. In: 2018 international conference on compilers, architectures and synthesis for embedded systems (CASES), pp 1–2. https://doi.org/10.1109/CASES.2018.8516886

Yang Z, Zhang A, Mo Z (2021) Psm Arena: partitioned shared memory for NUMA-awareness in multithreaded scientific applications. Tsinghua Sci Technol 26(3):287–295. https://doi.org/10.26599/TST.2019.9010036