Applied Nanoscience (2023) 13:2103–2114
https://doi.org/10.1007/s13204-021-02098-7

ORIGINAL ARTICLE

Optimal energy efficient, load aware memory management system on SoC's for industrial automation

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

Received: 17 August 2021 / Accepted: 9 September 2021 / Published online: 3 February 2022
© King Abdulaziz City for Science and Technology 2021

Abstract

It has only recently become possible to build the system on a chip (SoC) platform that makes use of field-programmable gate arrays (FPGAs) to moderate the amount of computational load placed on the main processor's CPU core. On the reconfigurable fabric, data used by both the software and the hardware is mapped using optimised memory mapping algorithm that was developed specifically for this purpose. This memory mapping algorithm serves as the base for the entire method. In this work, proposed a novel technique 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. The experimental results reveal that the proposed ELMM system of FPGA memory resources can be obtained at a much lower latency with minimal resource overhead and lower power consumption. Implementation of this work can done by using the Xilinx ISE 14.4 simulator and also generated the waveforms.

Keywords Energy efficient load aware memory management (ELMM) · Latency · Memory management · Power consumption · System-on-chip

Introduction

System on chip (SoC) is a chip which has multiple different components on same silicon. These components can be processing units, memories or other functions. These SoCs are integrated circuits (ICs) but in larger scale and SoCs are usually considered to have more functionality where ICs usually are considered to have one specialised function (Chusov et al. 2021). The key is the combination of software and hardware. Hardware is very fast and has low power consumption but is not flexible, adaptable and is hard design and test. Software is very flexible, adaptable, easy to write and test but also slow and has high power consumption. This means that with programmability we lose performance and with performance we lose adaptivity. SoCs are a good middle ground when both programmability and performance are wanted (Ahmed et al. 2021).

SoCs communicate internally with interconnects and externally with communication protocols. For example, an SoC for smart phone would need a communication protocol to be able to communicate with peripheral devices. These devices can be cameras, screens or other chips. Communication protocols can be used internally as well between the different components of the SoC. Figure 1 shows an example of a high level representation of an SoC. The figure shows possible modules inside an SoC and how they are connected with an interconnect (Rumyantsev et al. 2020).

SoCs are becoming more complex and contain more functionality. Complexity of design means more transistors but short time to market forces designers to use transistors less efficiently (Frolova et al. 2020). The efficient usage of transistors comes from optimization for which the designers do not have enough time. One way to battle this is the design re-usability where same design would be used for multiple applications. These designs usually leave room for some customization in architectural parameters.

✉ K. Siva Sundari
sivasundari2029@gmail.com

R. Narmadha
narmadha1109@gmail.com

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

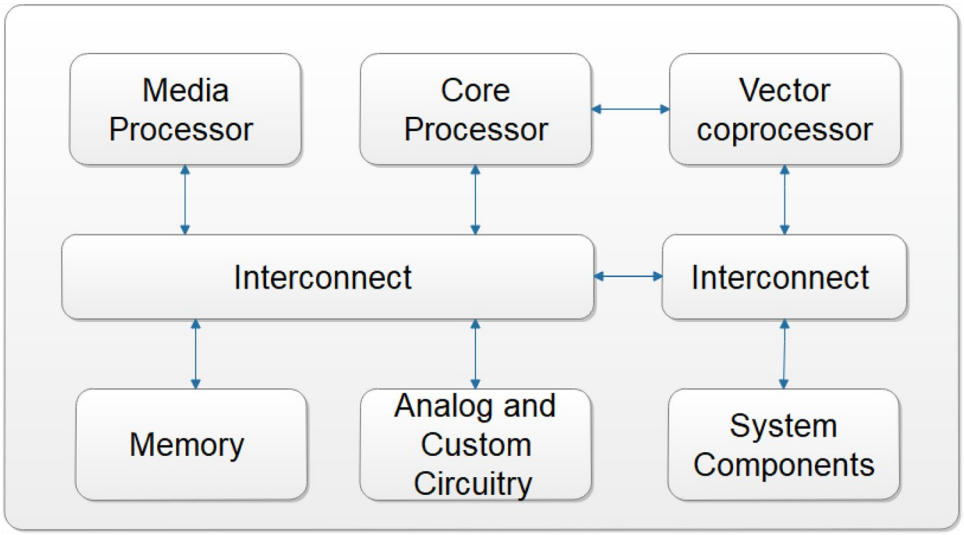


Fig. 1 Block diagram for high level representation of SoC

Energy optimization

The following section of this paper will provide a more in-depth discussion of an optimization concept for memory subsystems that are composed of STT-RAM blocks. Due in part to the optimization potential offered by the following two characteristics of this memory technology, which can be observed in conjunction with it and which can be observed in conjunction with it. Most importantly, the trade-off between the energy consumption of STT-RAM write operations and the latency of the write operation must be taken into account. As has been demonstrated for static random access memory memories, the impact of the required memory access logic results in an increase in dynamic energy consumption with increasing memory size (Rudolf et al. 2019). Also discussed is how to take advantage of this fact by optimising SRAM memory subsystems in the same way that DRAM memory subsystems are. However, while selecting an energy-efficient set of STT-RAM memories is straightforward, it does involve aspects that can be classified as allocation and binding problems, which will be discussed in greater detail further down this page on this page (Strobel et al. 2019). We have already discussed how allocating different-sized memory blocks can make a significant difference in terms of write and read energy consumption, particularly when combined with application segment binding and taking into account different memory access frequencies.

Because of this, the system's overall performance is influenced by several factors, including the memory subsystem's operation frequency and the overall system performance as a result, as well as memory blocks' operating voltage levels and the processor's operating frequency. In a single memory block, it is referred to as different operation voltage levels assignment when different memory blocks are assigned different voltage levels (Strobel and Radetzki 2019a). When integrated into a 45 nm node, a 4 MiB STT-RAM memory can operate at a maximum operation frequency ranging from 48 to 57 MHz, depending on the selected operation mode. Figure 1 depicts a trade-off diagram for the 45 nm node, which provides a more detailed illustration of the subject (Strobel and Radetzki 2019b). This paper investigates the write operation in relation to other memory sizes, and it is discovered that a much broader range of design possibilities is revealed; for example, it is possible to use write frequencies as low as 40 GHz in megabyte memory and write frequencies of more than 100 MHz in smaller memories with storage capacities as small as a few thousand bytes or even smaller than that. Because energy consumption changes at the same rate as the environment, it is not difficult to identify a good or even optimal solution within this design space (Fan, et al. 2019). In the following section, we will discuss a memory optimization method for STT-RAM memories that takes into account both the impact of memory size and the effects of different operation voltage levels at the same time. It is possible to account for the effects of memory size and different operation voltage levels in STT-RAM memories at the same time using this method, which saves both time and effort by eliminating the need for multiple calculations (Strobel and Radetzki 2019c).

Efficient load aware memory management (ELMM) algorithm

Variables and data structures in shared memory are declared in such a way that the compiler does not perform memory optimizations and makes the most efficient use of the processor registers that are currently available because there is insufficient local storage available. This ensures that the processor registers are used in the most efficient manner possible. As a result of this performance optimization, when accessing the shared memory pool, other forms of local storage, such as processor caches, are bypassed and used instead (Sergey et al. 2019). Data is read from shared memory whenever software requests it, and the results are immediately written back to shared memory to complete the transaction. Due to its simplicity, this scheme can be implemented with some difficulty. Any processor can be used to implement this algorithm.

Despite the fact that this scheme is feasible, it can only be implemented if the processor is capable of supporting explicit instructions for flushing all local storage into memory, which is not always the case due to limitations in the capabilities of some CPUs. As is readily apparent, the most significant advantage of both schemes is the fact that there are no costs associated with data transfer between software and hardware during the development process (Casini et al. 2018). The advantage of using shared memory rather than a local storage scheme, aside from the fact that data is not buffered within the processor's local memory, is the ability to maintain data consistency throughout the duration of the transaction. However, when the software portion of the application is in use, the amount of traffic that travels across the processor bus increases dramatically as a result of the increased activity on the processor bus. Because mapped memory access times are significantly longer than those of unmapped memory, using mapped memory in place of unmapped memory can result in a significant reduction in performance when compared to unmapped memory (Kuan and Adegbija 2018). To ensure that these schemes perform as efficiently as possible, it is critical to take into account factors such as the processor bus speed, the speed of mapped

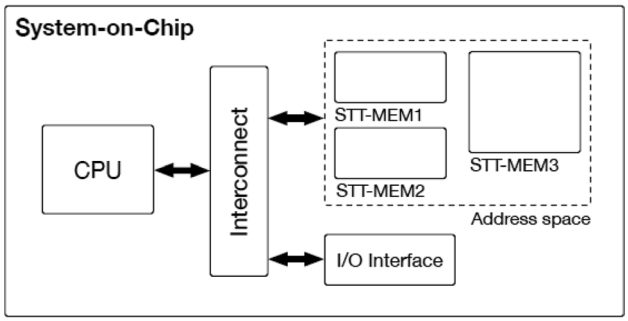


Fig. 2 System architecture

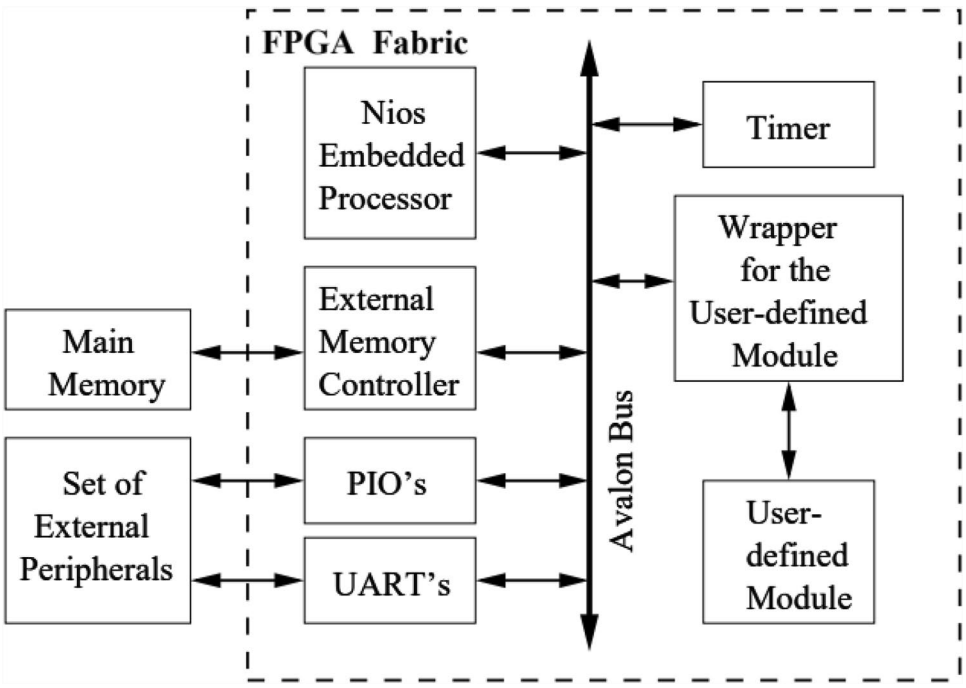


Fig. 3 Block diagram of the efficient load aware memory management

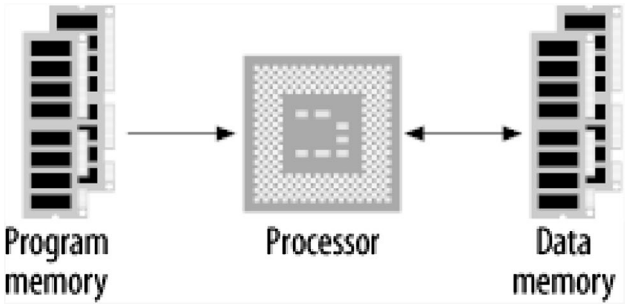


Fig. 4 Memory architecture

memory, and the speed of the access logic that is associated with mapped memory (Figs. 2, 3).

As a result, the vast majority of microprocessors currently available on the market are Von Neumann machines in their most basic configuration, which is true for the vast majority of microprocessors (Sayed et al. 2018). Keep in mind that the memory architecture (Fig. 4) is the most notable exception to this rule because it stores instructions and data in separate memory spaces and has separate data, address and control buses for each memory space, as opposed to the conventional architecture, which has one data, address and control bus for each memory space and one control bus for each memory space. In addition to the fact that instruction and data fetches can occur at the same time, this approach has a number of advantages, including the fact that the size of an instruction is not limited by the size of a standard data unit. The following are some of the additional benefits of employing this strategy, which are listed below (word).

The memory system architectures are made up of a cluster of high-speed processors, each of which has its own cache or local memory and access to a large, shared global memory pool, as well as a shared global memory pool, and a shared global memory pool (Fig. 5). Wikipedia states that data and programmes that will be executed by the computer are stored in the global memory of the computer before being executed by the computer itself. Also stored in this memory is a table containing the names of processes that are currently awaiting execution; this table is referred to as the "waiting list", and it contains information about each process' (or sub- program's) status while it is awaiting execution (Xu et al. 2018). To be able to run semi-independently of another processors in the system, it is necessary to load the processes and data associated with them into local memory or cache on each processor's behalf. This is accomplished by storing the processes and data associated with them in local memory or cache on each processor's behalf (Chen et al. 2018). It is also possible to communicate with other processes through the use of the global memory system, which is accessible through the global memory system, which is accessible through the global memory system.

Xtra storage mapping algorithm

Memories mapping is the process of associating data from one file with an aspect of a process's virtual address space that is currently being performed, and it is also known as memory mapping (also known as virtual address translation). In some circles, it is referred to as data association, data association and association, or data association and association, and it is also referred to as data association and association (Calinescu et al. 2018). When a file mapping object is created and stored in the database, it is possible for the system to keep track of the relationship between two files. When a process makes use of a portion of the virtual

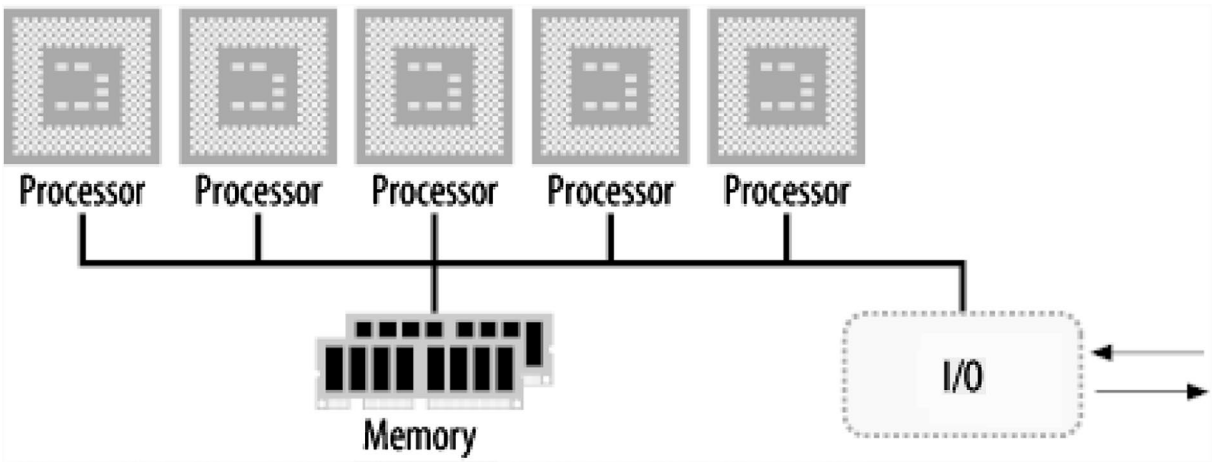


Fig. 5 ELMM shared-memory

address space that has been allocated to it to access the content of files that are currently being accessed, this is referred to as the file view in computing terminology (Long et al. 2018). Reading and writing from a file are both examples of file mapping. A process can benefit from both random and sequential input and output by using file mapping. Large data files can be processed without the need to map the entire data file into memory, which is particularly advantageous when dealing with large data files. Multiprocessing is a term that refers to the use of memory-mapped files to share data among multiple processes that are running at the same time.

It is possible to perform multiprocessing using memory-mapped files. The fact that processes can read from and write to the file view when using dynamically allocated memory means that when using dynamically allocated memory, pointers are used to read from and write to the file view rather than direct access to the file view when using dynamically allocated memory. In order to improve performance, file mapping stores the file on disc while keeping a copy of the file view on the computer's hard drive (Zhao et al. 2018). As a result, the system becomes more responsive as a result of the implementation of file mapping. The virtual protect function, which allows processes to manipulate the file view when the function is enabled, can be used to manipulate the file view by manipulating the file view. Figure 6 depicts the relationship between a, a file view, file mapping object and a file on disc (Chang et al. 2017). It also depicts the relationship between a file mapping object and a file on disc.

Storage systems have not improved in terms of performance when measured in terms of the amount of storage available, even though processor performance has increased dramatically. While data can be processed

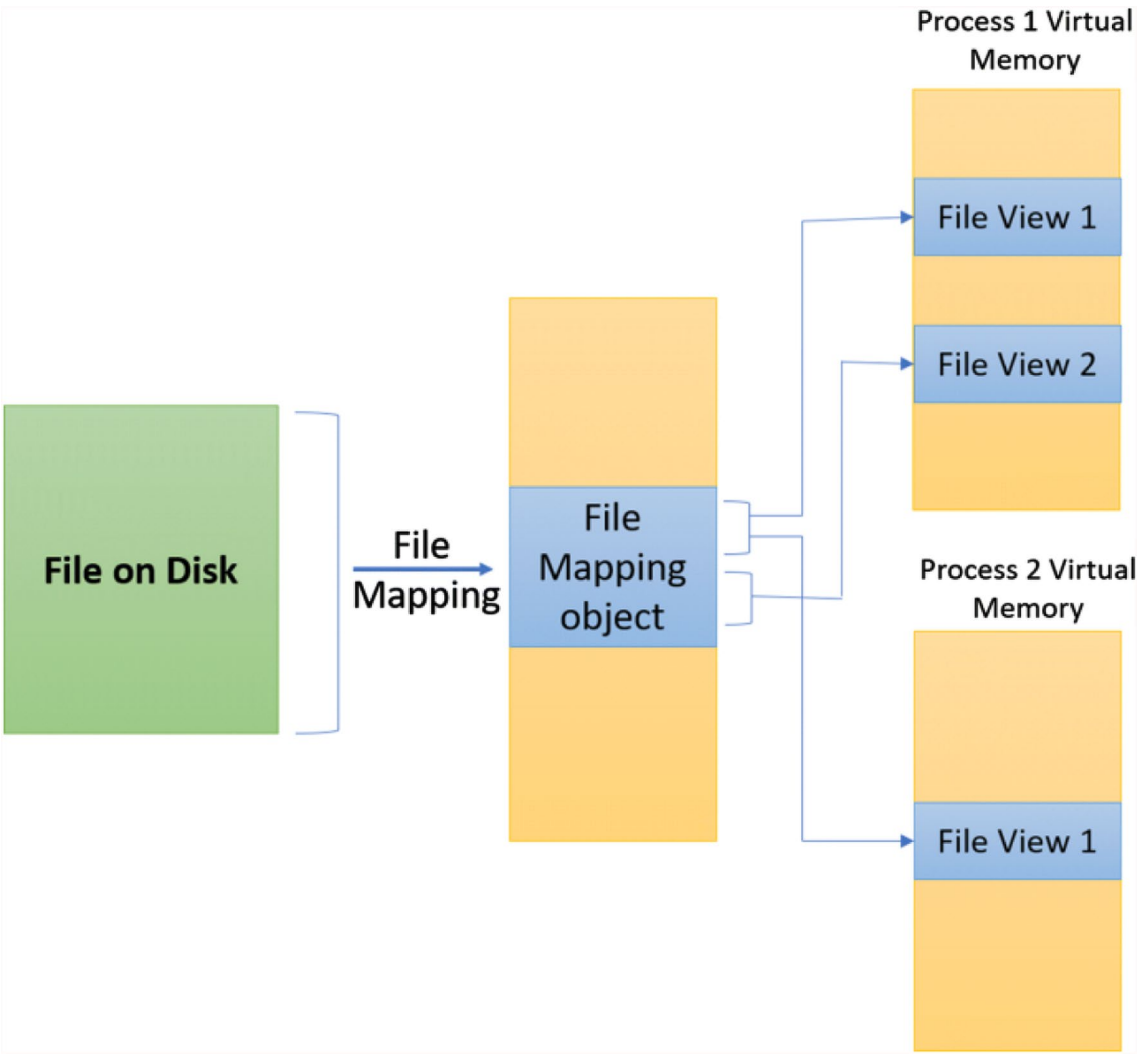


Fig. 6 Xtra storage mapping algorithm

quickly by the CPUs, system problems can still occur as a result of the latency introduced by a data storage device (Park et al. 2017). Data-intensive workloads are associated with a significant amount of overhead, which accounts for a significant portion of the time lost to inefficiency. With the recent increase in the amount of data that is being processed by applications, there is an urgent need for a significant improvement in the overall performance of the system. In recent years, in-memory data processing has gotten a lot of attention because of its ability to process workloads more quickly by eliminating the need for I/O. In-memory data processing is becoming increasingly popular because of its ability to process workloads more quickly. Such issues, on the other hand, can be avoided in the long run by including memory-mapped file I/O functionality in your application.

Memory access latency

The actual data transmission latency calculation starts by dividing the "chunks" parameter with the mem_t data structure's bus_width variable to obtain the total amount of data columns to be transmitted. Once the total amount of data columns is determined, the latency function will divide the amount by two and store the result to a temporary variable known as access_transfer (DOR3 memory can transmit twice per memory clock and transmit one data column per transmission).

At this point, the latency function is still expressed in terms of memory clock cycles, as it was previously stated (Lapshev and Hasan 2016). The latency function will convert the result to processor clock cycles by multiplying the values ("latency" and "Access _transfer") by the mem t variable, mem to CPU clk, in the latency function. When a value is converted to a processor clock cycle, the latency function stores the value of access transfer in the channel's pre v burst variable for use in subsequent latency calculations.

Because the data bus will be busy until the most recent memory access (the current access) is completed, the latency function will also store the value of start time + "latency" + access transfer to the bus timer variable of the channel (Li et al. 2016). Finally, the latency function performs one final action, which is to calculate the memory access latency that is relative to the "now" parameter in the input parameter. This is completed by returning the value of bus timer, which is the word "now".

Latency optimisation

It is necessary to employ the latency pragma because it ensures that the implemented design executes all of the operations in the functions within a specified range of clock cycles. However, while it is possible to announce the latency pragma while still inside the loop, this will result in the loop's total latency being specified for each repetition (Kulkarni et al. 2016), however, if it is desired to operate on the total latency of the loop, it is necessary to announce the latency pragma while outside the loop as described above (Fig. 7).

The following is the declared separating latency for all iterations:

```
Loop A: for (i=0; i<n; i++)

#pragma HLS latency max=10

Declaration of latency for all iterations:

#pragma HLS latency max=10

Loop A: for (i=0; i<n; i++)
```

In addition to using sequential loops for further optimization, another method of reducing latency is to combine them all. By flattening the nested loop, it is possible to reduce additional latency, which is a positive development (Balasa et al. 2017). This directive should be applied to the loop that is located at the innermost point of its parent loop body in order to flatten it out. It can be defined as: Set_directive_loop_flatten top/inner.

Implementation and results

The memory controller implementation were evaluation using trace based simulation. A VHDL testbench was used to read request from a text file and them to the controller via the host input interface. The testbench records requests as complete when they are signalled on the host output

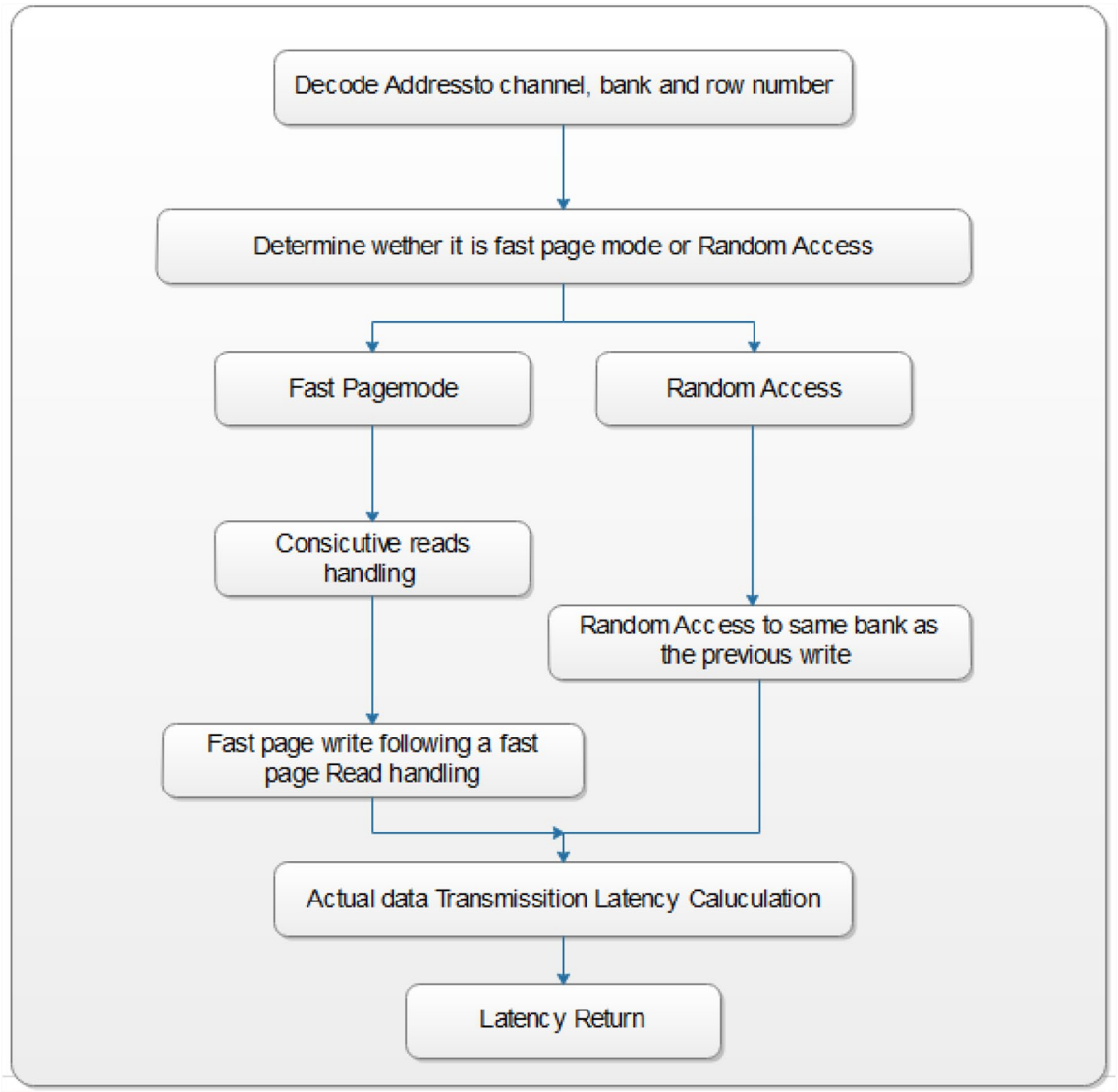


Fig. 7 Memory_access_latency function work flow

interface, which simulates the overall latency of memory request to real host as shown in Fig. 8.

The testbench load request as fast as possible without regard for the timing information in the trace is itself dependent on the average memory access time and using it would not give an accurate picture of how the design implemented in this work actually affect the average memory latency. It would have been preferable to perform the trace-based simulation using realistic timing, but the inter dependence of the rate at which memory requests are used with the average response time for the memory requests makes the synthesis of such timing difficult. Therefore, the traces were read into the simulation as quickly as posible. Accelerated traces allow the controller to achieve its maximum possible performance because the maximum amount of concurrency is available. Although using accelerated traces can hide some of the downfalls of certain controller implementations.

However, because of intellectual property protection, obtaining memory figures that are suitable for use in industrial applications is difficult. The use of nonlinear regression modelling of memory properties in conjunction with a black box approach is one approach that can be used to deal with a lack of available numbers in this scenario.

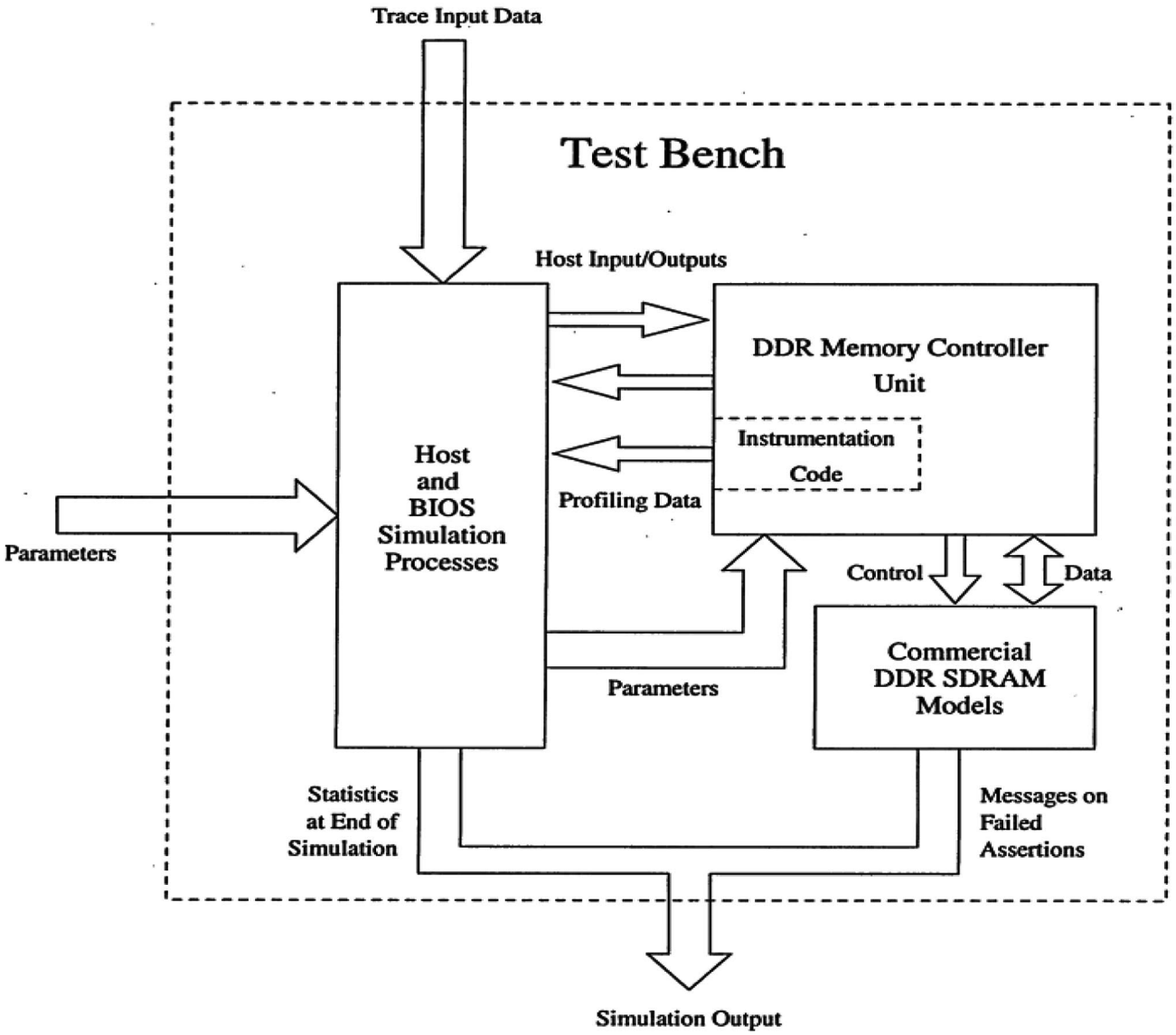


Fig. 8 Simulation method

Figures 9 and 10 show the simulation results of the proposed method. RTL and Technology diagram of the cache memory block of the proposed method. Everything about memory characteristics that were used in the experimental evaluation was presented in the same way it had been before.

As shown in Fig. 11, the results of dynamic energy minimization experiments conducted on 16 benchmark applications in terms of normalised average power consumption, along with a power analysis performed on the proposed algorithm, were compared to the theoretical results. This analysis displays total power, supply power, and other information (Fig. 12).

Conclusion

When developing and designing for high performance microprocessors, it is essential to develop and design an effective cache memory algorithm that generates an effective system. Memory-intensive applications can benefit from the first method, which combines optimization of memory allocation with application binding to provide a more efficient solution. It is possible to achieve application-specific dynamic energy minimization in memory subsystems when running memory-intensive applications using this method, which is particularly advantageous when running memory-intensive applications. Based on

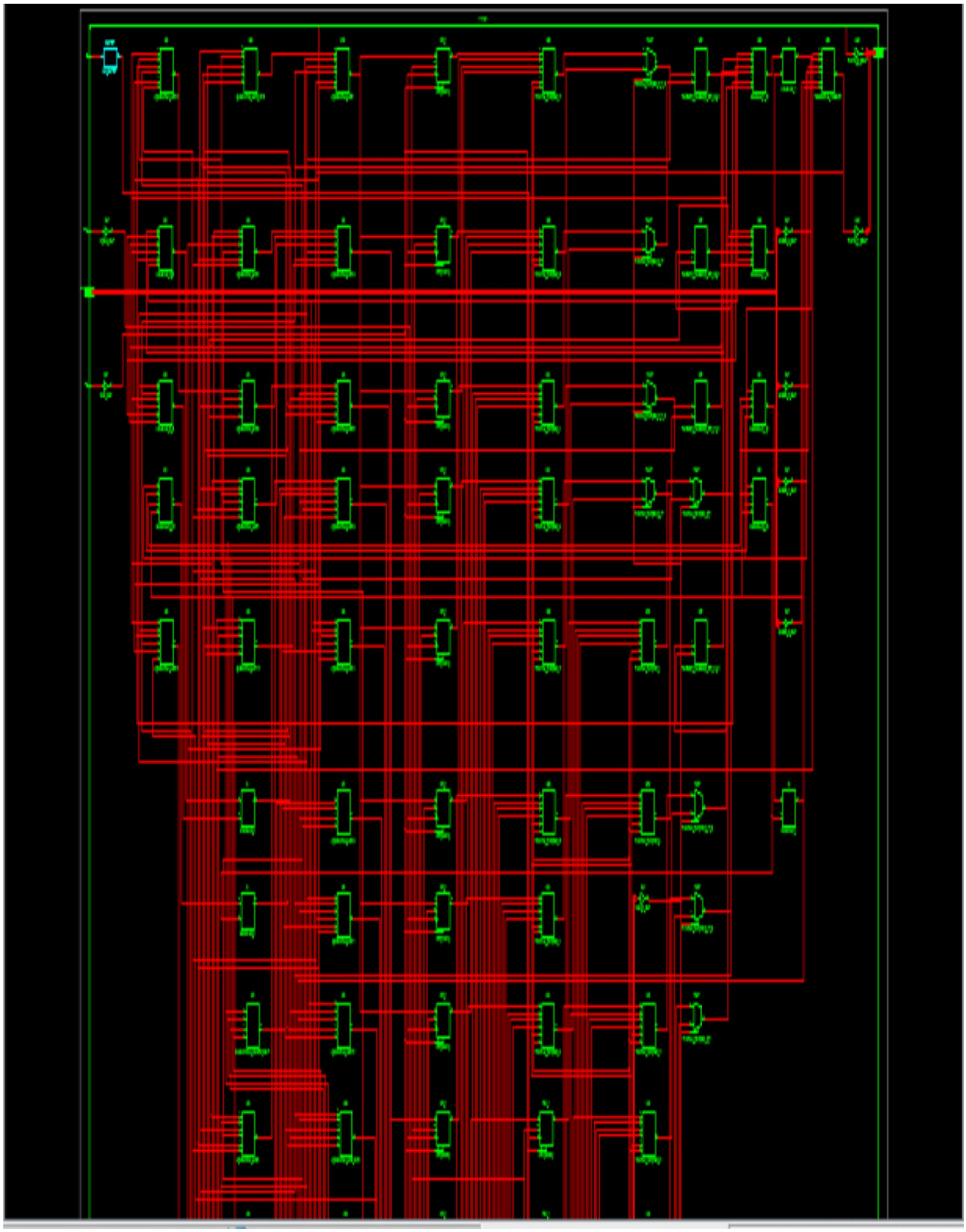


Fig. 9 Technology view of cache memory block

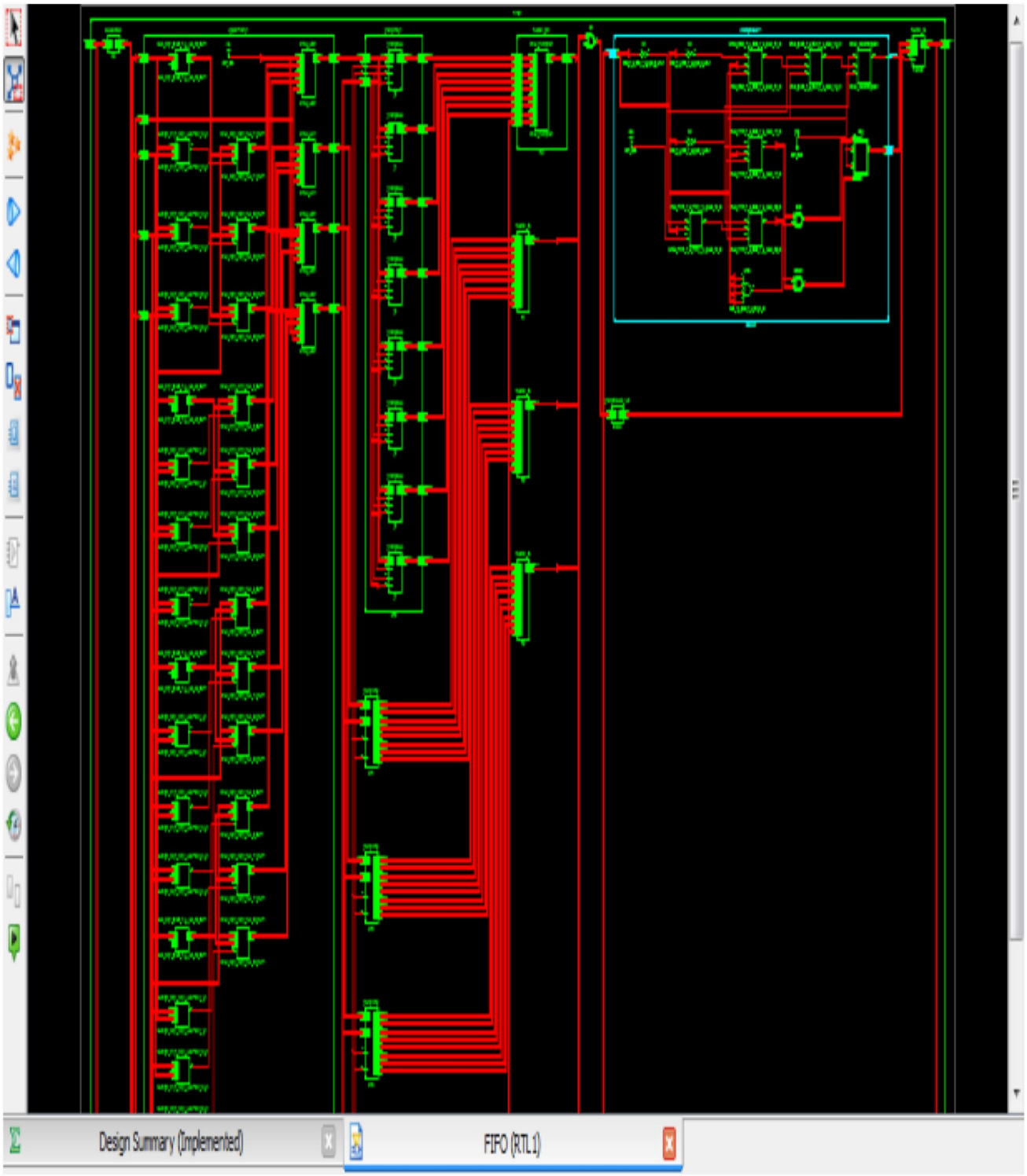
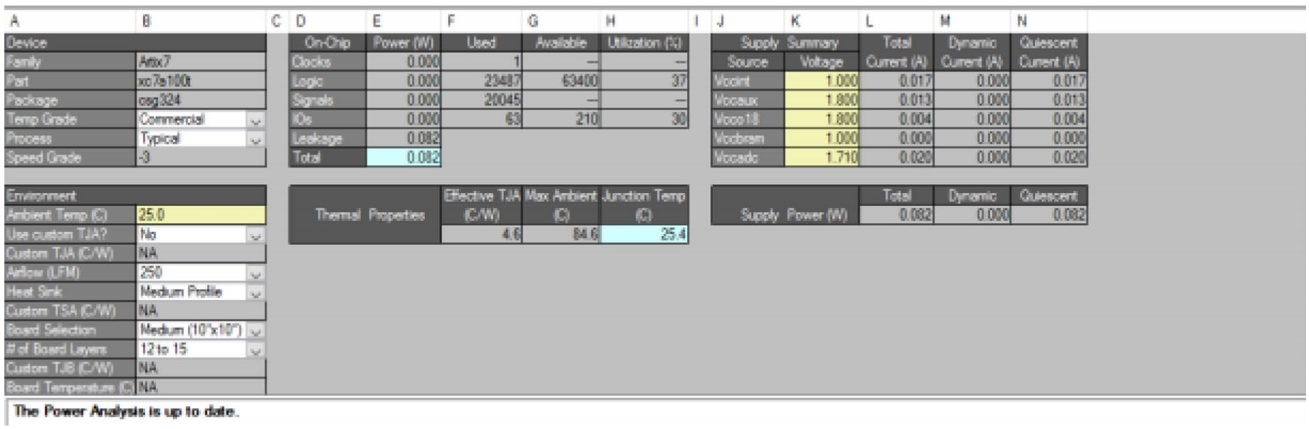


Fig. 10 RTL schematic view of memory block



Device	
Family	Artix7
Part	xc7a100t
Package	csg324
Temp Grade	Commercial
Process	Typical
Speed Grade	-3

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	—	—
Logic	0.000	23487	63400	37
Signals	0.000	20045	—	—
IOs	0.000	63	210	30
Leakage	0.082			
Total	0.082			

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.000	0.017	0.000	0.017
Vccaux	1.800	0.013	0.000	0.013
Vcco18	1.800	0.004	0.000	0.004
Vccbram	1.000	0.000	0.000	0.000
Vccadc	1.710	0.020	0.000	0.020

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x10")
# of Board Layers	12 to 15
Custom TJB (C/W)	NA
Board Temperature (C)	NA

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	4.6	84.6	25.4

Supply Power (W)	Total	Dynamic	Quiescent
	0.082	0.000	0.082

The Power Analysis is up to date.

Fig. 11 Power consumption based on power analysis

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2664	408000	0%
Number of Slice LUTs	5034	204000	2%
Number of fully used LUT-FF pairs	2000	5698	35%
Number of bonded IOBs	89	600	14%
Number of Block RAM/FIFO	2	750	0%
Number of BUFG/BUFGCTRL/BUFHCEs	2	200	1%
Number of DSP48E1s	2	1120	0%

Fig. 12 Device utilisation

a comparison with using a single memory block only, the determined results are optimal, with average reductions of more than 80% on an ongoing basis on an ongoing basis. ELMM memory subsystem configurations with split ELMM memory subsystems have proven to be a highly efficient switch when it comes to designing low-power System-on-Chip devices. Two memory blocks can be used in this area to achieve the greatest possible improvement instead of just one memory block, which is the case with the former. It turns out, on the other hand, that the allocation of additional memories results in only a marginal increase in the amount of energy saved.

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, 2021 22nd International symposium on quality electronic design (ISQED), pp. 374–380. Doi: https://doi.org/10.1109/ISQED51717.2021.9424306.

Balasa F, Abuaesh N, Gingu C, Luican I, Zhu H (2017) Energy-aware memory management for embedded multidimensional signal processing applications. EURASIP J Embed Syst 1:2016

Calinescu G, Fu C, Li M, Wang K, Xue C (2018) Energy optimal task scheduling with normally-off local memory and sleep-aware shared memory with access conflict. IEEE Trans Comput 67:1–1

Casini D, Biondi A, Nelissen G, Buttazzo G (2018) Memory feasibility analysis of parallel tasks running on scratchpad-based architectures. In: 2018 IEEE real-time systems symposium (RTSS). Doi: https://doi.org/10.1109/RTSS.2018.00047

Chang D, Lin I, Yong L (2017) Rohom: requirement-aware online hybrid on-chip memory management for multicore systems. IEEE Trans Comput Aided Des Integr Circuits Syst 36(3):357–369

Chen D, Edstrom J, Gong Y, Gao P, Yang L, McCourt M, Wang J, Gong N (2018) Viewer-aware intelligent efficient mobile video

embedded memory. IEEE Trans Very Large Scale Integr (VLSI) Syst 26(4):684–696

Chusov SA, Primakov EV, Savchenko YV, Pereverzev AL, Barkov ES (2021) Configurable test environment for RTL simulation and performance evaluation of network on chip as part of SoC. IEEE Conf Russ Young Res Electr Electron Eng (ElConRus) 2021:1969–1974. https://doi.org/10.1109/ElConRus51938.2021.9396634

Fan H et al. (2019) High-precision adaptive slope compensation circuit for system-on-chip power management. 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. IEEE Conf Russ Young Res Electr Electron Eng (EIConRus). https://doi.org/10.1109/EIConRus49466.2020.9039108

Kuan K, Adegbija T (2018) Lars: logically adaptable retention time stt-ram cache for embedded systems. In: 2018 Design, automation test in Europe conference exhibition (DATE), pp. 461–466. Doi: https://doi.org/10.23919/DATE.2018.8342053

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 (VLSI) 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 Circuits 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

Long L, Ai Q, Cui X, Liu J (2018) TTEC: data allocation optimization for morphable scratchpad memory in embedded systems. IEEE Access 6:54701–54712

Park J, Seo H, Kong B (2017) Conditional-Boosting flip-flop for near-threshold voltage application. IEEE Trans Very Large Scale Integr (VLSI) Syst 25(2):779–782

Rudolf J, Strobel M, Benz J, Haubelt C, Radetzki M, Bringmann O (2019) Automated sensor firmware development—generation, optimization, and analysis. In: MBMV 2019; 22nd workshop—methods and description languages for modelling and verification of circuits and systems, pp. 1–12

Rumyantsev A, Krupkina T, Losev V, Maksimov A (2020) Development of a measurement system-on-chip and simulation on FPGA. IEEE Conf Russ Young Res Electr Electron Eng (EIConRus). https://doi.org/10.1109/EIConRus49466.2020.9039249

Sayed N, Bishnoi R, Oboril F, Tahoori MB (2018) A cross-layer adaptive approach for performance and power optimization in stt-mram. In: 2018 design, automation test in Europe conference exhibition (DATE), pp. 791–796

Sergey G, Daniil Z, Rustam C (2019) Simulated annealing based placement optimization for reconfigurable systems-on-chip. IEEE Conf Russ Young Res Electr Electron Eng (EIConRus). https://doi.org/10.1109/EIConRus.2019.8657251

Strobel M, Radetzki M (2019a) A backend tool for the integration of memory optimizations into embedded software. In: 2019 Forum for specification and design languages (FDL), pp. 1–7. Doi: https://doi.org/10.1109/FDL.2019.8876895

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

Strobel M, Radetzki M (2019c) Power-mode-aware memory subsystem optimization for low-power system-on-chip design. ACM Trans Embed Comput Syst 18(5):1–43. https://doi.org/10.1145/3356583

Strobel M, Führ G, Radetzki M, Leupers R (2019) Combined mpsoc task mapping and memory optimization for low-power. In: Proc. of the 2019 IEEE 15th Asia Pacific conference on circuits and systems (APCCAS)

Xu Y, Zhu W, Xiao J, Yang G, Hu J, Zhang S, Huang M, Kong W, Zou S (2018) A 280-KBytes twin-bit-cell embedded NOR flash memory with a novel sensing current protection enhanced technique and high-voltage generating systems. IEEE Trans Circuits Syst II Express Briefs 65(11):1569–1573

Zhao Z, Sheng Y, Zhu M, Wang J (2018) A memory-efficient approach to the scalability of recommender system with hit improvement. IEEE Access 6:67070–67081

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.