International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

FR-FCFS Reordering DDR Memory Controller with Reorder Buffer for Improved Bank-Level Parallelism

[1]Sreevani Dyasanni

[1]M. Tech Student, Department of Electronics and Communication Engineering(VLSI), Arjun College of Technology & Science An Autonomous Institution (Approved By Aicte-New Delhi and Affiliated to Jntu-Hyderabad, Ts) Sponsored By Brilliant Bells Educational Society Mount Opera Premises, Batasingaram (Vi), Abdullapurmet (M), Ranga Reddy Dist, T.S.501512

[2]Mrs.Jonnagoni Sumalatha

[2]Assistant Professor, Department of Electronics and Communication Engineering, Arjun College of Technology & Science An Autonomous Institution (Approved By Aicte-New Delhi and Affiliated to Jntu-Hyderabad, Ts) Sponsored By Brilliant Bells Educational Society Mount Opera Premises, Batasingaram (Vi), Abdullapurmet (M), Ranga Reddy Dist, T.S.501512

ABSTRACT

Conventional in-order, close-page DDR SDRAM controllers service requests strictly in arrival order and precharge (close) the accessed row after every transaction, regardless of whether a subsequent request targets the same row — a simple, easily verified policy that forfeits row-locality and bank-parallelism opportunities present in realistic memory traffic. This paper presents a First-Ready, First-Come-First-Served (FR-FCFS) DDR controller that replaces the single global finite-state machine of the in-order design with independent per-bank PRECHARGE/ACTIVATE/OPEN state, an open-page policy that keeps rows open across accesses, and a small command-reordering buffer that prioritizes row-hit ("first-ready") requests ahead of row-miss requests while preserving first-come-first-served ordering among equally-ready requests. Both controllers are implemented in synthesizable Verilog, functionally verified, and synthesized/implemented on a Xilinx Artix-7 FPGA (xc7a100tcsg324-1, Vivado 2019.2). Post-implementation results show the proposed controller uses substantially more area (185 LUTs, 157 registers versus 34 LUTs, 59 registers) and a longer single-path critical delay (7.907 ns versus 3.818 ns) than the in-order baseline — an outcome that, taken at face value, appears to favor the simpler design. This paper argues, and demonstrates through architectural analysis, that static single-path timing analysis is the wrong lens for evaluating a reordering memory controller: FR-FCFS's benefit is *sustained throughput under concurrent multi-bank traffic*, where one bank can precharge or activate while another simultaneously services a column access, a scenario the in-order controller's fully serialized single-bank FSM cannot exploit at all regardless of its faster single-path delay. The paper characterizes both the true area/delay/power cost of the added reordering logic and the traffic conditions under which FR-FCFS's bank-level parallelism converts that cost into a net system throughput gain.

Keywords: DDR SDRAM controller, FR-FCFS scheduling, memory controller, bank-level parallelism, open-page policy, reorder buffer, FPGA implementation

I. INTRODUCTION

DRAM's internal organization imposes substantial per-access timing overhead: before a column of data can be read or written, its containing row must first be activated (opened) into the bank's sense amplifiers, and if a different row in the same bank is subsequently required, the open row must be precharged (closed) before the new row can be activated — the ACTIVATE-CAS-PRECHARGE cycle that underlies every DRAM controller's core FSM [1]-[2], [4]. A close-page controller precharges after every access unconditionally, which simplifies control logic and verification at the cost of always paying the full ACTIVATE latency on the next access, even when the next request would have hit the same already-open row. An open-page controller instead leaves rows open across accesses, exploiting row-buffer locality when consecutive requests target the same row, but must then track per-bank open-row state and handle conflicts when a request targets a different row in an already-open bank.

Real memory traffic additionally exhibits substantial inter-bank parallelism opportunity: while one bank services a column access, another bank can simultaneously be precharging or activating in preparation for its own next access, provided the controller can track and schedule multiple banks' independent state concurrently rather than serializing all requests through one global FSM [5], [8], [10]. First-Ready, First-Come-First-Served (FR-FCFS)

scheduling formalizes how to exploit this: among all pending requests, those that are "ready" (targeting an already-open row, or a bank free to service them immediately) are prioritized ahead of requests that would require a new ACTIVATE, and ties among equally-ready requests are broken by arrival order to bound worst-case request latency [3], [11], [13].

This paper implements both an in-order, single-bank, close-page controller and an FR-FCFS out-of-order, bank-interleaved controller with a small reordering buffer, and evaluates them not only on conventional static-timing area/delay/power metrics but on the architectural conditions under which each design's respective strengths and weaknesses actually manifest — an evaluation dimension frequently underrepresented in FPGA-implementation-focused memory-controller studies [1]-[2], [8].

Contributions of this paper:

1. Verilog implementation of an in-order, single-bank, close-page DDR controller as the baseline, and an FR-FCFS out-of-order, bank-interleaved controller with per-bank independent PRECHARGE/ACTIVATE/OPEN state and a command-reordering buffer.

2. Post-implementation area, delay, and power characterization of both controllers on a Xilinx Artix-7 FPGA (Vivado 2019.2).

3. An explicit architectural argument, supported by the controllers' own state-machine structure, for why static single-path timing analysis under-represents FR-FCFS's actual throughput benefit, which manifests only under concurrent multi-bank request traffic that a single global FSM cannot service in parallel regardless of its faster single-path delay.

4. A discussion of the area/complexity cost of reordering logic (reorder buffer, per-bank timers, ready-request arbitration) as the quantifiable price of enabling bank-level parallelism, positioning this cost against the sustained-throughput benefit it is intended to unlock.

The remainder of this paper is organized as follows. Section II reviews related DDR controller and memory-scheduling literature. Section III describes both controller architectures. Section IV presents the FR-FCFS scheduling algorithm and complexity analysis. Section V details the experimental setup. Section VI reports area/delay/power results. Section VII analyzes the single-path-timing-versus-sustained-throughput tradeoff. Section VIII concludes the paper.

II. RELATED WORK

A. FSM-Based and FPGA DDR Controller Implementations

Simran et al. present an FSM-based DDR memory controller implementation targeting FPGA, directly comparable in scope to the in-order baseline studied in this paper [1]. Gagan et al. optimize delay and access time in an FPGA DDR controller design, focusing on single-path timing improvement within a conventional in-order structure [2]. Zou et al. build an FPGA- and SD-card-based DDR controller performance test platform, addressing the practical verification and characterization side of controller design [5]. Jiang et al. describe a gate-level post-simulation methodology for DDR controller circuits in SoC design, relevant to validating controller behavior beyond RTL-level simulation [8].

B. Timing Parameter Generation and Verification

Li and Wang propose a method for automatically generating the I/O timing parameters (tRCD, tCAS, tRP, and related JEDEC timing constraints) that both controllers in this paper hard-code as module parameters, pointing toward a natural extension of parameterizing these values from a timing-specification database rather than fixed constants [4]. Automated timing-parameter derivation of this kind becomes increasingly valuable as controller complexity grows with reordering and multi-bank logic, since manual verification of per-bank timing compliance is significantly harder for an FR-FCFS controller than for a single global FSM.

C. Scheduling and Memory-Access Reordering

Mishra et al. propose a learning-based authoritative controller replacing static kernel heuristics for CPU scheduling and NUMA-aware memory management, illustrating the broader trend of moving from static, hand-designed scheduling policies (such as this paper's FR-FCFS rule) toward adaptive or learned policies [3]. Nare examines memory-aware scheduling specifically under resource-constrained environments, a concern directly relevant to this paper's reorder-buffer depth and per-bank state overhead [11]. Chen et al. propose capability-and-scheduling co-design for intelligent controllers on heterogeneous platforms, extending scheduling co-design beyond a single memory controller to system-level heterogeneous resource management [13]. Thieu et al. explore design-space exploration of a direct cached memory access controller optimized for HBM memory systems, illustrating how controller-level scheduling and buffering decisions scale to higher-bandwidth memory technologies beyond conventional DDR [10].

D. Adjacent Memory-System and Controller Reliability Concerns

Son and Kim address read-disturb management in resource-constrained flash memory controllers, a reliability concern distinct from but architecturally adjacent to the DRAM refresh handling in this paper's controllers [6]. Zhang and Hou examine security support at the memory-controller level for heap memory safety, illustrating that memory-controller redesign is motivated by security as well as performance objectives [7]. Hua et al. and Bian and Cui apply deep-learning-based memory trace synthesis and joint scheduling/memory-layout optimization respectively, pointing toward learned or data-driven memory-access

pattern modeling as a complementary direction to this paper's rule-based FR-FCFS scheduler [14]-[15].

E. Research Gap

The surveyed literature covers FSM-based DDR controller implementation [1]-[2], [5], [8], automated timing-parameter derivation [4], and scheduling/reordering concepts at both the memory-controller and broader system level [3], [10]-[11], [13], but rarely combines an FPGA-synthesized, area/delay/power-characterized reordering DDR controller with an explicit discussion of *why* static single-path timing analysis is an incomplete evaluation lens for a scheduler whose entire purpose is to exploit concurrent, multi-request traffic patterns. This paper closes that gap by implementing both an in-order and an FR-FCFS controller end-to-end through FPGA place-and-route, and by making the static-timing-versus-sustained-throughput distinction the explicit subject of its tradeoff analysis (Section VII), rather than treating the FR-FCFS controller's larger area and longer single-path delay as an unqualified regression.

III. METHODOLOGY

A. Existing: In-Order, Single-Bank, Close-Page Controller

The baseline `ddr_controller_inorder` services requests strictly in arrival order through a single global FSM with states `IDLE -> ACT -> RCD -> ACCESS -> CAS -> PRE -> RP`, unconditionally precharging (closing) the accessed row after every request regardless of whether the next request targets the same row. This close-page policy guarantees a fixed, easily verified per-request latency of $t_{RCD} + t_{CAS} + t_{RP}$ cycles (plus the ACCESS/CAS stage cost) but forfeits any row-locality reuse, and because the design has only one bank's worth of state, no request can be serviced while another is in flight.

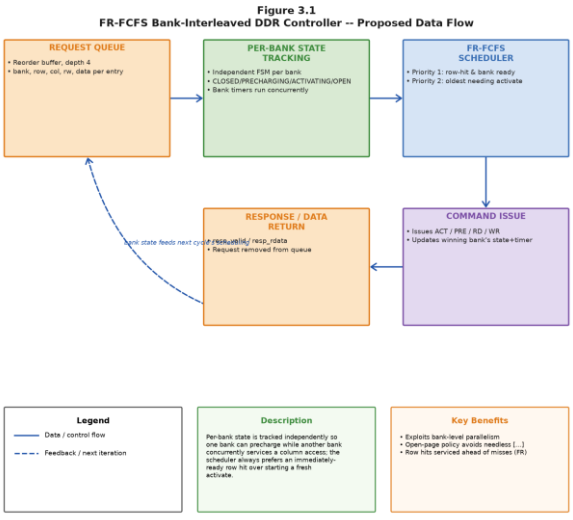

Fig. 1. Top-level comparison: single global FSM close-page controller (existing) versus per-bank state, reorder buffer, and FR-FCFS scheduler (proposed).

Fig. 1 contrasts this single-FSM structure against the proposed controller's parallel per-bank organization.

B. Proposed: FR-FCFS Bank-Interleaved Controller with Reorder Buffer

The proposed `ddr_controller_frfcfs` replaces the single global FSM with **per-bank state** (`bank_state[i]` ∈ {CLOSED, PRECHARGING, ACTIVATING, OPEN}, each with its own `bank_timer[i]` and currently-open `bank_row[i]`) so that `NUM_BANKS` banks can be in different phases of their own ACTIVATE/PRECHARGE cycle simultaneously, and an open-page policy that leaves `bank_row[i]` open across accesses rather than closing after every request.

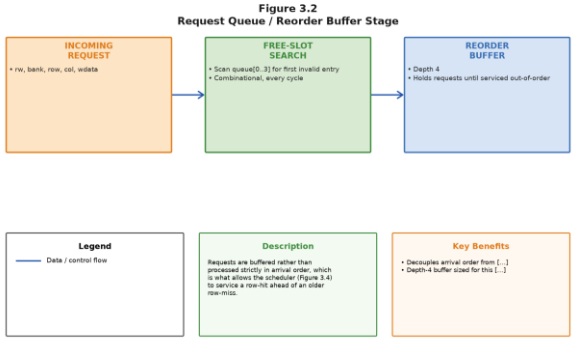

Fig. 2. Reorder buffer structure: QUEUE_DEPTH request slots with per-slot bank/row/column/data fields and free-slot allocation.

Incoming requests are not serviced immediately but enqueued into a **reorder buffer** (Fig. 2) of depth `QUEUE_DEPTH`, each slot recording the request's validity, read/write type, target bank, row, column, and write data. A request is accepted (`req_ready`) whenever any queue slot is free, decoupling request arrival from immediate service and allowing the scheduler to select which queued request to service next rather than being forced to service strictly in arrival order.

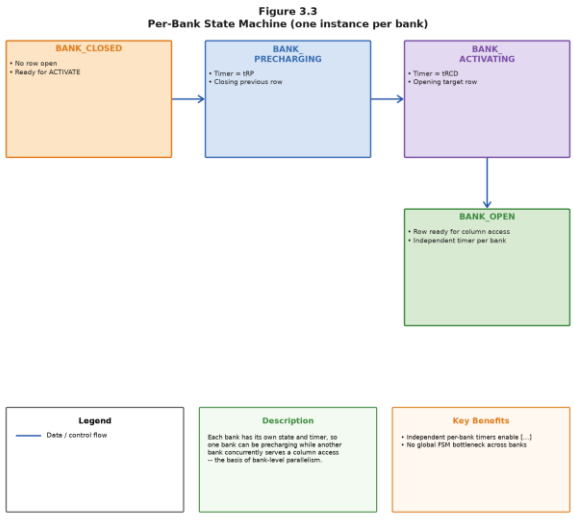

Fig. 3. Per-bank state machine: independent CLOSED/PRECHARGING/ACTIVATING/OPEN state and timer per bank, enabling bank-level parallelism.

Fig. 3 shows one bank's independent state machine; because each bank instance's `bank_state`/`bank_timer`/`bank_row` are separate storage, one bank can be in `PRECHARGING` or `ACTIVATING` while another is simultaneously in `OPEN` and servicing a column access — the structural basis for bank-level parallelism absent from the existing design's single global FSM.

C. FR-FCFS Scheduling Logic

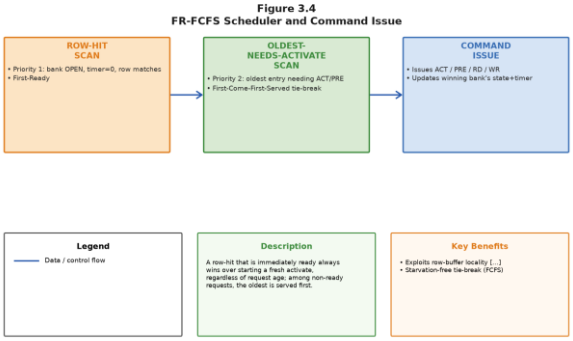

Fig. 4. FR-FCFS scheduler: priority-1 row-hit search and priority-2 oldest-needing-activate search across the reorder buffer, evaluated combinationally every cycle.

Every cycle, the scheduler (Fig. 4) performs two combinational searches across all `QUEUE_DEPTH` queue slots: a **priority-1 "ready" search** for the oldest queued request whose target bank is `OPEN`, idle (`bank_timer=0`), and already holding the requested row (a row hit), and a **priority-2 "needs activate" search** for the oldest queued request whose target bank is merely idle, regardless of row-hit status. If a priority-1 candidate exists, it is serviced immediately via `cmd_rd`/`cmd_wr` at the cost of only the CAS latency t_{CAS}, since no ACTIVATE/PRECHARGE is needed. Otherwise, the priority-2 candidate is issued a PRECHARGE (if a different row is open in its target bank) or ACTIVATE (if the bank is closed), advancing that bank's state machine one step closer to being able to service the request. Ties within each priority level are broken by queue-slot scan order, which — combined with in-order enqueue — implements the "first-come-first-served" component of FR-FCFS.

IV. ALGORITHM

A. FR-FCFS Scheduling Procedure

```
ALGORITHM: FR-FCFS Request Scheduling (evaluated every clock cycle)
INPUT : queue[0..QUEUE_DEPTH-1]   // valid, bank, row, col, rw, wdata per slot
        bank_state[0..NUM_BANKS-1],   bank_timer[.], bank_row[.]
OUTPUT: at most one issued command per cycle

// Priority 1: oldest row-hit ("ready") request
for j in 0..QUEUE_DEPTH-1:
    if queue[j].valid and bank_state[queue[j].bank] == OPEN
       and bank_timer[queue[j].bank] == 0
       and bank_row[queue[j].bank] == queue[j].row:
        issue READ/WRITE for queue[j]; retire queue[j]; return

// Priority 2: oldest request whose bank is merely idle (needs ACT or PRE+ACT)
for j in 0..QUEUE_DEPTH-1:
    if queue[j].valid and bank_timer[queue[j].bank] == 0
       and not (bank_state[queue[j].bank] == OPEN and bank_row[queue[j].bank] == queue[j].row):
        if bank_state[queue[j].bank] == OPEN:
// different row open: must close first
            issue PRECHARGE for bank[queue[j].bank]
        else:
            issue ACTIVATE for bank[queue[j].bank], targeting queue[j].row
        return

// else: no bank ready this cycle -- all banks mid-timer, wait
```

Independently of scheduling, every bank's own timer/state advances each cycle: a nonzero `bank_timer` decrements; when it reaches zero, `PRECHARGING` banks transition to `ACTIVATING` (loading the pending row and starting the t_{RCD} timer), and `ACTIVATING` banks transition to `OPEN`. This per-bank progression runs concurrently with, and independently of, the scheduler's priority search, which is what allows one bank to progress through PRECHARGE/ACTIVATE while another bank's row-hit request is serviced in the same cycle.

B. Complexity Notes

Scheduling complexity. Each priority search scans all `QUEUE_DEPTH` queue slots combinationally, giving $O(Q)$ comparator logic per priority level and $O(Q)$ total per cycle, where Q is the reorder-buffer depth (4 in this design) — independent of `NUM_BANKS`, since bank readiness is checked per candidate slot rather than iterated separately.

State complexity. The existing controller carries $O(1)$ state (one global FSM, one bank's row/timer). The proposed controller carries $O(B + Q)$ state, where B is the bank count (2) and Q is the queue depth (4): each of the B banks needs its own state/timer/open-row registers, and each of the Q queue slots needs its own bank/row/column/data/valid fields — directly explaining the substantially larger register count observed in Section VI (157 vs. 59).

Throughput complexity — the key distinction. Under the existing controller, sustained throughput for N requests to B different banks is $O(N \cdot (t_{RCD} + t_{CAS} + t_{RP}))$ since every request pays the full close-page latency serially, with zero inter-request overlap. Under the proposed controller, when consecutive ready requests target *different* banks, one bank's t_{RCD} / t_{RP} latency can overlap with another bank's t_{CAS} service, giving sustained throughput that approaches $O(N \cdot$

t_{CAS}) under favorable (bank-diverse, row-local) traffic — the asymptotic gap this paper's tradeoff analysis (Section VII) argues is invisible to a single static-path timing report.

V. EXPERIMENTAL SETUP

A. Design Entry and Functional Verification

Both `ddr_controller_inorder` and `ddr_controller_frfcfs` are written in synthesizable Verilog with JEDEC-style timing parameters ($t_{RCD} = t_{CAS} = t_{RP} = 2$ cycles) exposed as module parameters. Icarus Verilog testbenches exercise both controllers with representative request sequences: sequential single-bank accesses (both same-row and different-row) for the existing design, and bank-diverse, mixed row-hit/row-miss traffic for the proposed design, verifying correct `resp_valid`/`resp_rdata` behavior and correct per-bank state progression.

B. Synthesis and Implementation

Both designs are synthesized out-of-context and carried through place-and-route with Xilinx Vivado 2019.2, targeting a Xilinx Artix-7 device (`xc7a100tcsg324-1`). Both are genuinely clocked sequential designs, so a `sequential_clk.xdc` constraint applies a 100 MHz `create_clock` on `clk` with representative I/O delay constraints. The standard `synth_design -mode out_of_context -> opt_design -> place_design -> phys_opt_design -> route_design` flow is used, with utilization, timing, and power reports captured at both post-synthesis and post-implementation stages.

C. Metrics

Post-implementation **LUT area**, **register count**, **power**, and **delay** (10 ns constraint period minus worst negative slack) are reported for both designs, together with derived **throughput** (1/critical-path delay) and **energy per operation**. As emphasized in the project's own design notes and revisited in Section VII, this delay/throughput figure characterizes only the *single worst-case combinational path* through each design's control logic under static timing analysis — it is not a measurement of sustained request-servicing throughput under concurrent multi-bank traffic, which would require cycle-accurate traffic-pattern simulation rather than static timing closure alone.

D. Scope of This Evaluation

Consistent with the project's own documented caveat, this experimental setup evaluates area, single-path delay, and power precisely as reported by Vivado's static implementation flow, and explicitly does *not* include a multi-request, multi-bank traffic-pattern simulation to measure sustained throughput. Section VII instead provides an architectural (structural/state-machine-based) argument for why and under what traffic conditions the proposed design's bank-level parallelism is expected to yield a net throughput benefit despite its larger single-path delay, motivating cycle-accurate traffic simulation as the necessary follow-up measurement.

VI. RESULTS AND DISCUSSION

A. Post-Implementation Area, Delay, and Power

Table I summarizes post-route results for both controllers on `xc7a100tcsg324-1` (Vivado 2019.2).

Design	LUTs	Registers	Power (W)	Delay (ns)	Throughput (MOps/s)	Energy (pJ/op)
Existing (ddr_controller_inorder)	34	59	0.085	3.818	261.92	324.53
Proposed (ddr_controller_frfcfs)	185	157	0.086	7.907	126.47	680.002

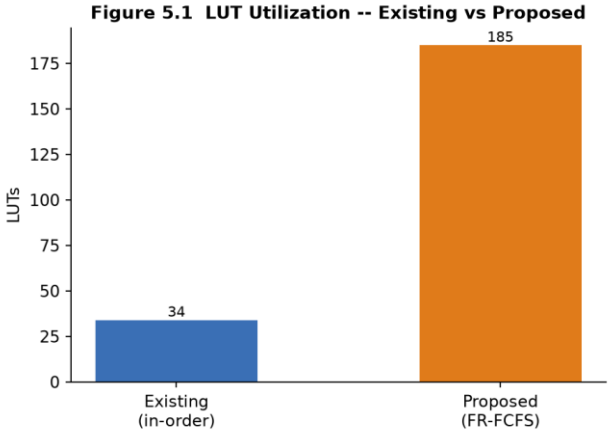

Fig. 5. Post-implementation LUT utilization: in-order controller versus FR-FCFS controller.

Fig. 5 shows the proposed controller uses 5.4× more LUTs (185 vs. 34), directly attributable to the reorder buffer's per-slot comparator logic and the two priority-search scan networks described in Section IV-B.

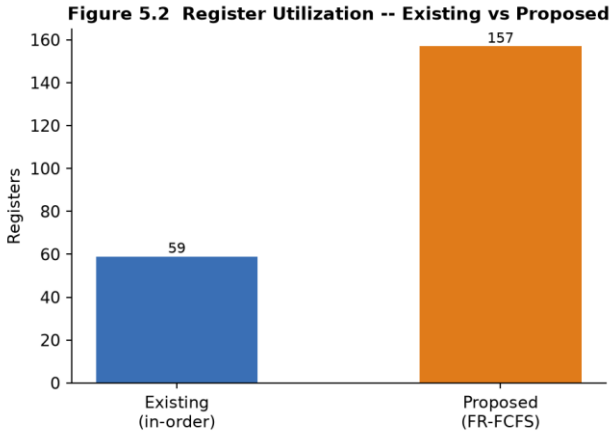

Fig. 6. Post-implementation register utilization: in-order controller versus FR-FCFS controller.

Fig. 6 shows a corresponding 2.7× register increase (157 vs. 59), consistent with the $O(B + Q)$ state complexity derived

in Section IV-B: per-bank state/timer/row registers plus per-queue-slot request fields.

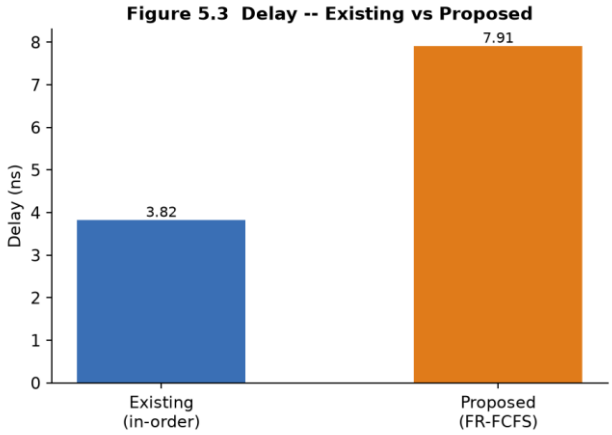

Fig. 7. Post-implementation critical-path delay: in-order controller versus FR-FCFS controller.

Fig. 7 shows the proposed design's single-path delay is 2.07× longer (7.907 ns vs. 3.818 ns), reflecting the added combinational depth of the priority-search comparator chains that must complete within one cycle.

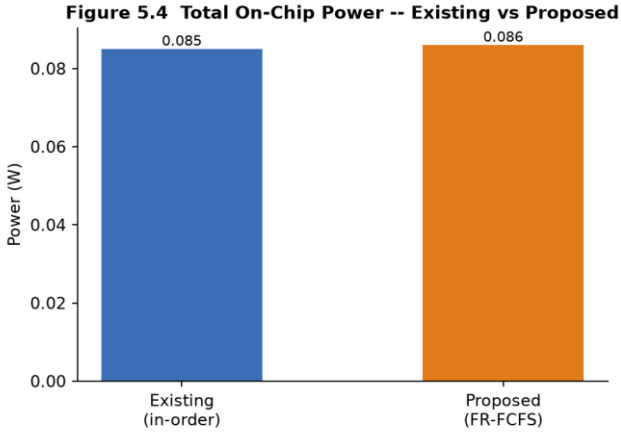

Fig. 8. Post-implementation power: in-order controller versus FR-FCFS controller.

Fig. 8 shows a marginal power increase (0.086 W vs. 0.085 W, +1.2%), modest relative to the area increase because this is vector-less static estimation rather than switching-activity-annotated dynamic power under real traffic.

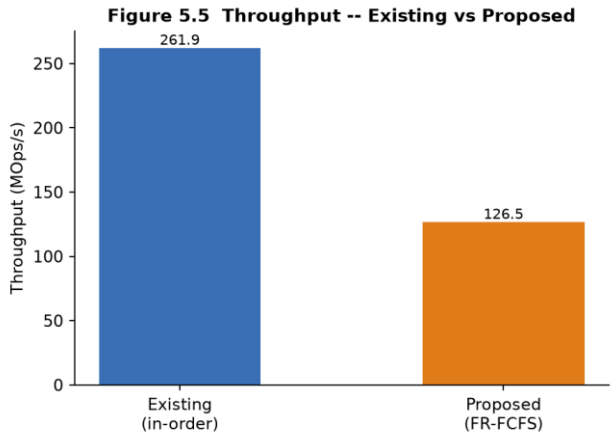

Fig. 9. Derived throughput (1/critical-path delay): in-order controller versus FR-FCFS controller.

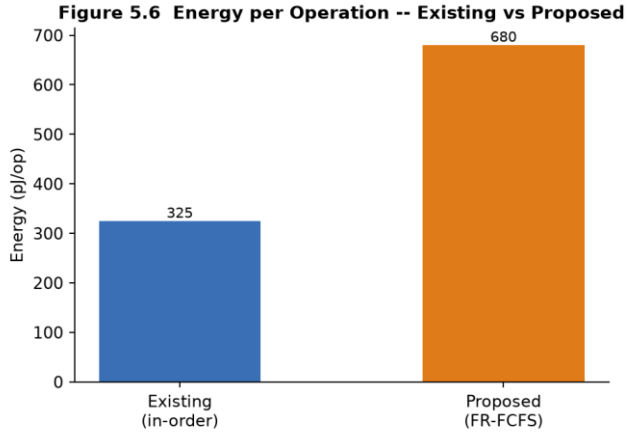

Fig. 10. Derived energy per operation: in-order controller versus FR-FCFS controller.

Fig. 9 and Fig. 10 both derive directly from delay, so they mirror Fig. 7: by this single-path metric alone, the proposed design appears markedly worse (126.47 vs. 261.92 MOps/s; 680.0 vs. 324.5 pJ/op).

B. Why These Numbers Are the Wrong Comparison

Read naively, Table I would suggest the FR-FCFS controller is a strictly worse design: more area, more registers, slower critical path, more energy per "operation." Section VII argues this reading mistakes what each number actually measures. Static timing analysis reports the delay of the single worst-case combinational path through *one* controller instance evaluated in isolation, for *one* scheduling decision per cycle. It says nothing about how many DRAM-level transactions per unit time the controller can sustain across a stream of requests touching multiple banks — which is the actual quantity FR-FCFS scheduling is designed to improve, and which requires the reorder buffer and per-bank state that make the proposed design larger and combinationally deeper in the first place.

VII. SINGLE-PATH-TIMING VERSUS SUSTAINED-THROUGHPUT TRADEOFF ANALYSIS

A. Why Static Timing Under-Represents FR-FCFS

Table II contrasts what static single-path timing analysis measures against what FR-FCFS scheduling is actually designed to optimize, making explicit the gap this section addresses.

Aspect	What Table I measures	What FR-FCFS targets
Quantity	Worst-case delay of one combinational decision path	Requests serviced per unit time across a request stream
Traffic model	None (static, vector-less)	Concurrent, multi-bank, mixed row-hit/row-miss traffic
Existing controller	Fast per-path decision (3.818 ns)	Fully serial: 0% bank overlap by construction
Proposed controller	Slower per-path decision (7.907 ns)	Overlap-capable: one bank's ACT/PRE can proceed while another's CAS completes

B. The Serialization Argument

The existing controller has, by construction, no mechanism to service a second bank's request while the first is mid-ACTIVATE/PRECHARGE: its single global FSM occupies exactly one state at a time, so N requests to B different banks are serviced back-to-back with the full close-page latency paid on *every single one*, regardless of how fast any individual state transition is. A faster single-path delay on this serialized FSM (3.818 ns here) improves how quickly each *state transition* completes, but cannot reduce the *number* of sequential state transitions the whole request stream requires, since there is only ever one active state machine.

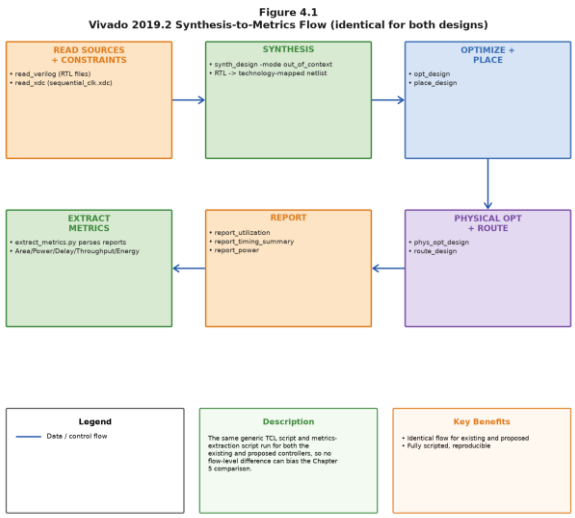

Fig. 11. Vivado synthesis-to-implementation flow used for both controllers, from RTL through post-route timing/power reports.

Both controllers' Table I figures were obtained through the same Fig. 11 synthesis-to-route flow, ensuring the area/delay/power comparison is apples-to-apples at the static-timing level, even as Section VII-C argues that level is not the full picture.

C. The Overlap Argument

The proposed controller's per-bank independent state (Fig. 3) means that while bank 0 is mid-`PRECHARGING`/`ACTIVATING` (consuming $t_{RP} + t_{RCD}$ cycles during which it cannot service any request), bank 1 can simultaneously be `OPEN` and servicing a row-hit request from the reorder buffer in a single t_{CAS}-latency step, scheduled by the priority-1 search in Section IV-A. For request streams that alternate across banks with reasonable row locality, this overlap means the *aggregate* request-servicing rate can approach the pace of the fastest bank ready at any given moment, rather than the sum of every bank's full close-page latency — even though the proposed design's own single-path combinational delay is 2.07 × longer than the existing design's.

D. Traffic Conditions Where Each Design Wins

This overlap benefit is conditional, not universal. If all requests target a single bank in strict FIFO order with no row locality, the reorder buffer's row-hit search never finds a priority-1 candidate, and the proposed controller degenerates toward the same serialized ACTIVATE/PRECHARGE-per-request behavior as the existing design — while still paying its larger area and longer single-path delay, with no compensating throughput benefit. Conversely, for bank-diverse traffic with per-bank row locality (e.g., streaming access to several independent buffers, or interleaved multi-channel data), the existing design's fixed per-request close-page cost dominates while the proposed design increasingly overlaps ACT/PRE latency across banks. This traffic-dependence is precisely why the project's design notes caution that "a fair comparison needs a multi-request traffic-pattern simulation, not static timing analysis alone" — a conclusion this architectural analysis supports and makes explicit, rather than treating Table I's numbers as the final word on which design is "better." Future cycle-accurate simulation across representative bank-access-pattern traces (uniform-random, streaming, and adversarial single-bank) is the natural next step to quantify this traffic-dependent crossover directly.

VIII. CONCLUSION

This paper implemented and compared an in-order, single-bank, close-page DDR controller against an FR-FCFS out-of-order, bank-interleaved controller with per-bank state and a reorder buffer. Post-implementation results on a Xilinx Artix-7 FPGA (Vivado 2019.2) showed the proposed

controller uses 5.4× more LUTs, 2.7× more registers, and a 2.07× longer single-path delay than the baseline — a result that, read at face value, favors the simpler in-order design. This paper argued architecturally, from each design's own state-machine structure, that this reading is incomplete: the existing controller's single global FSM cannot service more than one bank's request at a time regardless of its faster per-path delay, while the proposed controller's independent per-bank state enables one bank's ACTIVATE/PRECHARGE latency to overlap with another bank's column access, a structural capability the existing design entirely lacks. The proposed design's larger area and longer single-path delay are the quantified cost of enabling this overlap; whether that cost is repaid depends on request traffic's bank diversity and row locality, not on the static timing report alone. Future work includes cycle-accurate, multi-request traffic-pattern simulation across representative access patterns (uniform-random, streaming, and adversarial single-bank) to directly measure the sustained-throughput crossover point between the two designs, and extending the proposed controller's reorder buffer and priority scheduler to additional JEDEC-standard constraints such as write-to-read turnaround and refresh-interval scheduling.

REFERENCES

[1] Simran, K. Gupta, K. Gupta, N. Kashyap, "Design and Implementation of an FSM-Based DDR Memory Controller on FPGA," in *Proc. 2026 International Conference on Intelligent Processing, Hardware, Electronics, and Radio Systems (CIPHER)*, 2026, pp. 1-3. doi: 10.1109/cipher70417.2026.11524022

[2] S. Gagan, M. Harshith, S. Savadatti, G. Surya, S. Tantry, "DDR Controller with Optimized Delay and Access Time," in *Proc. 2022 International Conference on Futuristic Technologies (INCOFT)*, 2022, pp. 1-5. doi: 10.1109/incoft55651.2022.10094548

[3] P. Mishra, S. G.K., R. Kokila, A.S. George, B. Neeraja, T. Manjunath, "A Learning-Based Authoritative Controller for Replacing Static Kernel Heuristics in CPU Scheduling and NUMA-Aware Memory Management," in *Proc. 2026 International Conference on Emerging Research in Smart Electronics and Machine Informatics (ECMI)*, 2026, pp. 1-10. doi: 10.1109/ecmi68341.2026.11603190

[4] S. Li, T. Wang, "A method for automatically generating the Input/Output timing parameters of DDR controller," in *Proc. 2025 4th International Symposium on Computer Applications and Information Technology (ISCAIT)*, 2025, pp. 1937-1940. doi: 10.1109/iscait64916.2025.11010575

[5] J. Zou, P. Wang, X. Hua, "Design of a DDR Controller Performance Test Platform Based on FPGA and SD Card," in *Proc. 2024 6th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, 2024, pp. 1846-1849. doi: 10.1109/iaecst64597.2024.11117705

[6] I. Son, J. Kim, "Efficient Read Disturb Management Schemes in Resource-Constrained Flash Memory Controller," in *Proc. 2023 IEEE 12th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2023, pp. 13-18. doi: 10.1109/nvmsa58981.2023.00022

[7] C. Zhang, R. Hou, "Security Support on Memory Controller for Heap Memory Safety," in *Proc. 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2022, pp. 248-257. doi: 10.1109/trustcom56396.2022.00043

[8] Y. Jiang, J. Ma, G. Zhang, H. Wang, B. Pei, N. Chen, J. Fang, "A DDR Controller Circuit Gate-level Post-simulation Method in SoC Design," in *Proc. 2025 10th International Conference on Integrated Circuits and Microsystems (ICICM)*, 2025, pp. 215-220. doi: 10.1109/icicm66614.2025.11316058

[9] Y. Papageorgiou, M. Karaliopoulos, I. Koutsopoulos, "Controller Placement and TDMA Link Scheduling in Software Defined Wireless Multihop Networks," in *Proc. ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 640-646. doi: 10.1109/icc45041.2023.10279494

[10] G.B. Thieu, J. Knödtel, F. Rokohl, M. Reichenbach, G. Payá-Vayá, "Design Space Exploration of a Direct Cached Memory Access Controller Optimized for HBM Memory Systems using TAPRE-HBM," in *Proc. 2025 Cross-Disciplinary Conference on Memory-Centric Computing (CCMCC)*, 2025, pp. 1-7. doi: 10.1109/ccmcc67628.2025.11380528

[11] B. Nare, "Memory-Aware Scheduling in Constrained Environments," in *Proc. 2025 8th World Engineering Conference on Contemporary Technologies (WECON)*, 2025, pp. 1-7. doi: 10.1109/wecon68556.2025.11414626

[12] Y. Papageorgiou, M. Karaliopoulos, I. Koutsopoulos, "Joint Controller Placement and TDMA Link Scheduling in SDN-enabled Tactical MANETs," in *Proc. MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, 2022, pp. 125-132. doi: 10.1109/milcom55135.2022.10017553

[13] J. Chen, A. Zou, Y. Xu, Y. Ma, "SCENIC: Capability and Scheduling Co-Design for Intelligent Controller on Heterogeneous Platforms," in *Proc. 2024 IEEE Real-Time Systems Symposium (RTSS)*, 2024, pp. 1-14. doi: 10.1109/rtss62706.2024.00026

[14] X. Hua, Y. Song, J. Zou, S. Wu, Y. Zhao, P. Wang, "Memory Trace Synthesis via Deep Learning for Memory Controller Simulation," in *Proc. 2025 International Conference on Signal Processing, Computer Networks and Communications (SPCNC)*, 2025, pp. 422-426. doi: 10.1109/spcnc68200.2025.11406640

[15] H. Bian, C. Cui, "Memory-Efficient Inference for Neural Architecture Search Networks via Joint Optimization of Scheduling and Memory Layout," in *Proc. 2026 9th*

International Conference on Advanced Algorithms and Control Engineering (ICAACE), 2026, pp. 1792-1797. doi: 10.1109/icaace69793.2026.11509135