

# MIQARA: Mixed-Criticality Queue-based Architecture for Reconfigurable Accelerator Platforms

Hassan Nassar<sup>\*†</sup>, Martin Rapp<sup>\*‡</sup>, Lars Bauer<sup>\*§</sup>, Mostafa Elshimy<sup>†</sup>, Zeynep Guelbeyaz Demirdag<sup>†</sup>, Jörg Henkel<sup>†</sup>

<sup>†</sup>Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany,

<sup>‡</sup>AI Research, Robert Bosch GmbH, Germany (work done while at KIT), <sup>§</sup>Ubitium GmbH

Corresponding Author: hassan.nassar@kit.edu

**Abstract**—Coexistence of safety-critical control functions and best-effort computations in mixed-criticality systems poses a challenge in resource allocation and scheduling, as high-criticality jobs must adhere to strict timing guarantees, while lower-criticality jobs should make effective use of available resources without compromising the system’s safety and predictability. This paper introduces MIQARA<sup>1</sup>, a mixed-criticality queue-based architecture designed for reconfigurable accelerator platforms. MIQARA efficiently combines software-programmable CPUs with reconfigurable hardware, utilizing a dynamic job pipeline, token-based dependency tracking, and out-of-order scheduling to optimize resource utilization. At the same time, MIQARA has been designed to satisfy real-time constraints. MIQARA is evaluated on four FPGA platforms: the Zed Board, DipForty board, ZCU102 board, all of which have ARM CPUs implemented on chip, and Arty A7 with a RISC-V soft-core processor, representing systems that rely on soft CPUs. Results demonstrate substantial performance gains, particularly in terms of execution speed, flexibility, and adaptability to mixed-criticality workloads. The integration of features such as a streaming network further illustrates MIQARA’s scalability to complex data-intensive applications, making it a compelling solution for embedded mixed-criticality systems. MIQARA requires a hardware overhead of 17.8% and achieves a speedup of up to 4 $\times$ .

**Index Terms**—Mixed-criticality, Reconfigurable Systems, Embedded Systems

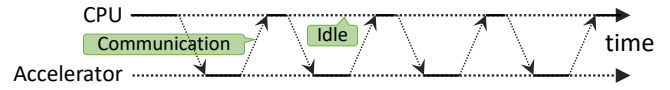
## I. INTRODUCTION

Many modern industrial and commercial applications demand hard or soft real-time guarantees while simultaneously managing multiple criticality levels [1]. In mixed-criticality systems, jobs of varying importance, often with different timing constraints and failure tolerances, must coexist, adding complexity to system design and scheduling. The worst-case execution time (WCET) of an application, which is required for real-time guarantees, is in general not attainable when executing on a high-performance CPU (e.g. a super-scalar out-of-order CPU) [1]. As an alternative, hardware implementations of an application’s compute-intensive kernels provide high performance while at the same time being analyzable and capable of accommodating different levels of criticality [2].

When implemented on a *reconfigurable fabric* such as an FPGA, there is no need to develop a dedicated application-specific integrated circuit (ASIC), as application-specific accelerators can be reconfigured on demand [3], e.g. when the application starts. Hardware implementations are not well-suited for control-flow dominant applications; therefore, a *reconfigurable processor* combines an (analyzable) CPU with a reconfigurable fabric. This allows executing control-dominant parts on the CPU and offloading computationally intensive parts, so-called *jobs*, to the reconfigurable fabric, while respecting the mixed-criticality constraints of the system [4].

The reconfigurable fabric executes a job using an *accelerator* that is configured into one of multiple reconfigurable *containers*, enabling

### (i) CPU manages the execution



### (ii) CPU issues jobs that are managed in hardware

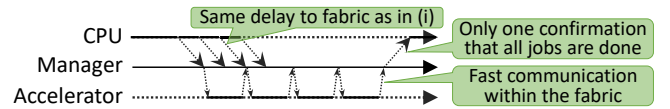


Fig. 1: Due to the high communication overhead between CPU and loosely-coupled accelerators, managing the accelerator execution from the CPU results in underutilization of the accelerators and high overhead on the CPU. In contrast, delegating this job to an autonomous manager (this is a part of our MIQARA) frees the CPU for other jobs, and at the same time maintains a high accelerator utilization.

different levels of criticality to share and dynamically utilize hardware resources. Altogether, this combines *software programmability* with *execution time guarantees* for both real-time and mixed-criticality jobs. Platforms that provide an ASIC-grade analyzable CPU along with a state-of-the-art reconfigurable fabric are commercially available. For instance, the Xilinx Zynq UltraScale+ MPSoC combines ARM Cortex-R5 cores for real-time workloads, ARM Cortex-A53 for best-effort workloads, and a reconfigurable fabric on a single chip. More details and background are provided in Section II.

Deploying reconfigurable platforms for mixed-criticality applications with some of them needing real-time guarantees faces notable challenges, including significant CPU-to-fabric communication overhead due to their *loose coupling* via on-chip buses. Data transfers can consume hundreds of CPU cycles, resulting in reduced gains for jobs with strict timing constraints [5]. Thus, hardware accelerators must not only compensate for this overhead but also accommodate jobs of varying criticalities. Figure 1 illustrates this communication challenge, highlighting the inefficiencies of CPU-driven job management and the benefits of autonomous reconfigurable processor handling, where jobs are executed based on their dependencies and criticality, with the CPU being notified upon completion or at defined checkpoints.

Consider an example with independent jobs of types *A* and *B* executed on slots with matching accelerators, with the WCET of jobs *A* being double the WCET of jobs *B*. An optimal sequence, such as issuing one job *A* followed by two jobs *B*, minimizes overall execution by interleaving jobs efficiently. However, this static scheduling limits flexibility, as jobs executing faster than their WCETs cause idle periods. Separate FIFO queues for jobs *A* and jobs *B* can mitigate timing disparities but may lead to WCET and load distribution imbalances. A central queue with dynamic scheduling offers better load balance and predictable WCET.

Dynamic accelerator management is necessary to reduce both average and worst-case execution times, contrasting static scheduling.

<sup>1</sup>Source code available at: <https://github.com/hassanassar/miqaraDATE>

<sup>\*</sup>All three authors contributed equally. This work was supported in part by the Federal Ministry of Research Technology and Space, BMFT, as part of the DI-EDAI Project under Grant 16ME0990K

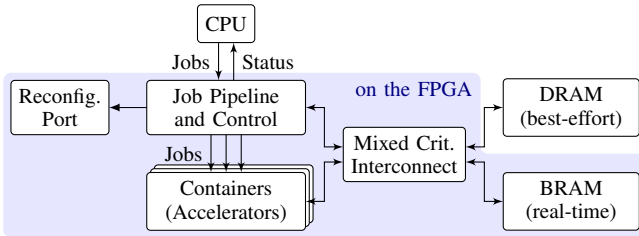


Fig. 2: Overview of the architecture of our MIQARA, showing the accelerator containers, the job pipeline, and the connections between them, to memory, and to the processors.

Applications must express job data dependencies for the management system to respect them. The main novel contributions of MIQARA are as follows:

- We address the challenges of *communication overhead* and *accelerator execution management* through an integrated hardware/software approach, where the software defines computational jobs with their dependencies, and the hardware autonomously manages their execution to ensure efficiency through high accelerator utilization and adherence to mixed-criticality requirements.
- We introduce a hardware queue featuring an *out-of-order dispatch mechanism* that achieves high container utilization regardless of the job issue sequence. This mechanism respects specified data dependencies and supports statically analyzable execution times, crucial for mixed-criticality guarantees.
- We perform a *precise analysis of all latencies* within the hardware queue and its management, enabling safe and predictable WCET bounds for sequences of issued jobs, which is essential for both real-time and best-effort jobs.

## II. BACKGROUND

Mixed-criticality systems in embedded computing handle jobs of varying criticality on a single platform [6]. FPGAs offer a robust solution by combining high-performance hardware with dynamic reconfiguration [7], adapting to workload changes, optimizing resources, and meeting performance demands. Although virtualization aids resource sharing, it introduces real-time, safety, and security issues, particularly with hypervisors [8]. Real-time architectures like RISC-V need optimized software and efficient context switching for these jobs, utilizing open instruction sets and adaptable stacks [9]. Reconfigurable platforms require advanced strategies such as dynamic scheduling and resource partitioning.

### A. Xilinx Zynq UltraScale+ MPSoC

The Xilinx Zynq UltraScale+ MPSoC integrates high-performance processing (PS) and programmable logic (PL) [10], creating a platform for mixed-criticality applications. The PS includes a quad-core ARM Cortex-A53 for general computing and a dual-core ARM Cortex-R5 for real-time, low-latency jobs. This setup efficiently manages diverse criticality levels. Communication between PS and PL is via the Advanced eXtensible Interface (AXI), ensuring high-throughput data transfer for seamless integration of custom accelerators and workload management. The Cortex-R5 cores, optimized for real-time, offer predictable execution with rapid data access. Offloading compute jobs to the PL complements the APU, forming a flexible environment for mixed-criticality workloads.

### B. Reconfigurable Processors

Reconfigurable processors enhance adaptability and performance by integrating programmable logic and processor cores. They efficiently manage jobs and adapt to workload changes [11]. Reconfigurable logic improves execution with specific optimizations and

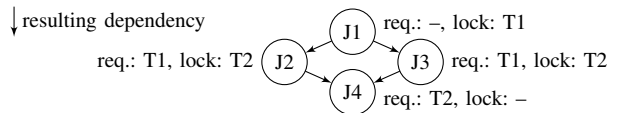


Fig. 3: Example of how dependencies can be specified using tokens.

resource partitioning [12]. RISC-V with embedded logic supports flexible instruction swaps for resource sharing [5]. High-performance reconfigurable processors increase efficiency and throughput, balancing performance, power, and real-time needs with noise-robust, low-power designs and strong compiler support [11, 13].

1) *Real-Time Processors*: Real-time processors ensure deterministic execution for strict timing for hard deadline applications, guaranteeing safety-critical job performance [14]. With deterministic cache and predictable memory access, they minimize job interference. Reconfigurable architectures enhance these processors through hardware accelerators, reducing software overhead and improving response times. Real-Time Operating Systems (RTOS) can offload components to hardware, minimizing jitter [15]. By adjusting power dissipation and using specialized RTOS scheduling, they maintain power efficiency and meet real-time constraints, critical for hard deadline applications. Frameworks combining real-time processing and dynamic partial reconfiguration boost adaptability in safety-critical systems by allowing flexible resource allocation based on job criticality while ensuring timing consistency [16]. This combination supports diverse mixed-criticality workloads [17].

2) *Mixed-Criticality Processors*: Mixed-criticality processors manage jobs of varying importance, ensuring high-criticality job safety and performance while sharing resources with lower-criticality jobs. The main challenge is avoiding interference through resource partitioning, highlighting the need for dynamic management [18]. These processors employ advanced scheduling and virtualization to allocate hardware according to job priority [19]. Reconfigurable hardware supports adaptability and system integrity by enabling job modifications. The design aims to balance predictability and performance, meeting high-criticality jobs' WCET and optimizing lower-criticality jobs' performance [20].

## III. SYSTEM ARCHITECTURE

Figure 2 outlines the main components of our MIQARA architecture, designed for mixed-criticality workloads using reconfigurable resources efficiently. It consists of *reconfigurable containers* of uniform size, predetermined at design time, which can hold *accelerator functions*. At runtime, they can be reconfigured by submitting special *reconfiguration jobs* to the job pipeline, and execute computations by submitting *compute jobs*. Job submission, central to the MIQARA framework, involves the CPU submitting jobs to the *job pipeline* via the on-chip bus. The job pipeline schedules and maps jobs to containers while respecting job dependencies and ensuring priority execution, which is crucial for real-time jobs. The design of MIQARA involves two different memories. The real-time jobs use the on-chip BRAM to have predictable time delays. The best-effort jobs do not access the BRAM, but only use the off-chip DRAM. As a result, the best-effort and real-time jobs do not interfere with each other's execution.

### A. Job Dependencies

MIQARA schedules and maps jobs to containers by the job pipeline at runtime. However, (i) different compute jobs might have data dependencies that need to be expressed by the application and (ii) we need to ensure that a reconfiguration job does not replace an instance of a specific accelerator while it is still required by compute jobs that are waiting in the job pipeline.

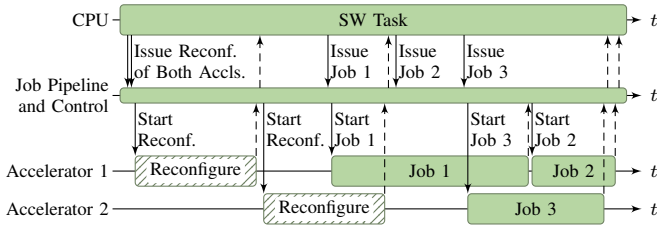


Fig. 4: Example of a job execution sequence within MIQARA. Job 3, despite being issued after job 2 has no dependency on neither job 1 nor job 2 can start executing immediately before job 2 that has a dependency on job 1 to remove unneeded wait times.

1) *Dependencies between Compute Jobs*: Our design makes use of the hardware efficiently and various dependencies like 1-to-1, 1-to-N, M-to-1, and M-to-N. We develop a *token* system where jobs define dependencies through *require* and *lock*. A token acts as a counter: locking a token  $t$  reduces  $t$ . Initially,  $t$  is zero; in M-to-1,  $t$  reaches  $-M$  when all  $M$  jobs lock the token. After a job  $j$  has finished, it increments  $t$ . Tokens are awaited by jobs needing them, until  $t = 0$  when all  $M$  jobs complete in M-to-1. Jobs can lock and require different tokens  $t_1$  and  $t_2$ . Multiple jobs can share tokens in 1-to-N, and an M-to-N dependency is represented by  $M$  jobs locking and  $N$  requiring the same token. The CPU often waits for several jobs using a token system and a *NOP job* to improve efficiency. A *NOP job* synchronizes depending on all CPU-related jobs using a token. Figure 3 illustrates the token system: Each job  $J$  can require a token  $T$  and executes only when the token is free. Moreover, each job can lock a token and frees it only after job completion. Overall, this token system supports a strict superset of series-parallel graphs [21]. Note that a job is not issued before its dependency jobs are also issued; otherwise, it has an unknown dependency. Hence, the token count is properly updated before any dependent job is eligible for execution, avoiding any violations of dependency constraints. To illustrate this, consider the following scenario involving jobs  $J_0$ ,  $J_1$ ,  $J_2$ , and  $J_3$  with two tokens,  $T_1$  and  $T_2$ :

- $J_0$  and  $J_1$  are independent and issued first.
- $J_2$  depends on both  $J_0$  and  $J_1$ , forming an *M-to-1* dependency using token  $T_1$ .
- $J_3$  depends only on  $J_0$ , forming a *1-to-1* dependency via token  $T_2$ .

When  $J_0$  and  $J_1$  are issued, they each decrement  $T_1$  by 1, resulting in  $T_1 = -2$ . In parallel,  $J_0$  also decrements  $T_2$  by 1, giving  $T_2 = -1$ . If  $J_0$  completes execution before  $J_1$ , then  $J_3$  can proceed (since its dependency on  $J_0$  is satisfied via  $T_2$ ), even though  $J_2$  must wait for  $J_1$  to finish. This form of execution allows for limited out-of-order behavior while still preserving correct dependency semantics.

2) *Dependencies between Compute and Reconfiguration Jobs*: As mentioned above, the job pipeline uses tokens to manage compute job dependencies and requires proper function configuration in containers. Two main requirements are: (i) a compute job for function  $f$  is scheduled only after a reconfiguration job sets  $f$  in a container, and (ii) a reconfiguration job cannot change the last container's function  $f$  until all dependent jobs for  $f$  are done. The pipeline manages (i), while (ii) is handled using *function counters*. These counters track active and pending jobs for function  $f$ . Reconfiguration on containers with function  $f$  occurs only when no pending jobs need  $f$ , preventing premature changes. Function counters and scheduling logic manage dependencies, reduce conflicts, and ensure function availability.

As an illustrative example, consider a system with three containers, where container  $C_1$  is currently configured with function  $f$ . Two jobs,  $J_1$  and  $J_2$ , are issued to execute on function  $f$ , incrementing the function counter for  $f$  to 2. While  $J_1$  is still running and  $J_2$  is pending,

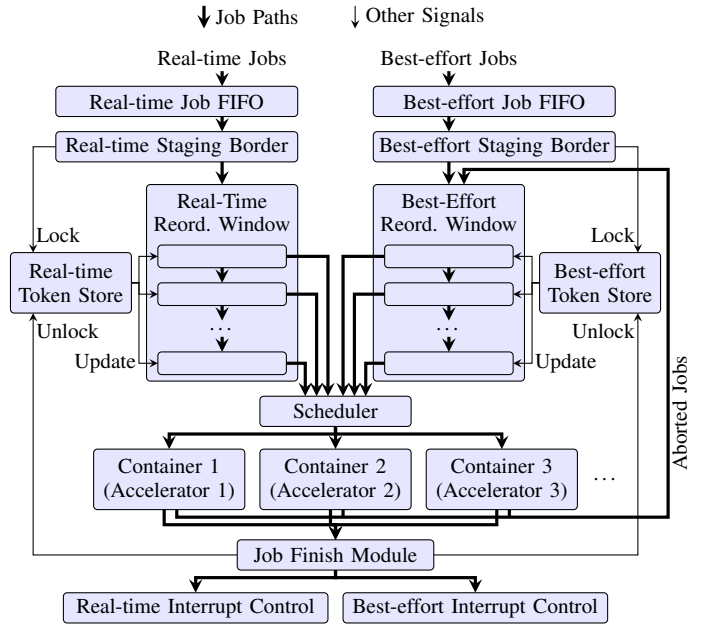


Fig. 5: Detailed view of the MIQARA job pipeline showing job scheduling across accelerators. Two queues exist, one for real-time jobs, the other is for best-effort jobs. The scheduler selects jobs from the reorder windows based on priority and dependencies and dispatches them to the accelerators inside the containers. After job completion, the job finish module manages dependencies and potential interrupts to the CPU.

a new reconfiguration job  $R$  arrives, requesting to reconfigure  $C_1$  to a different function  $g$ . Since the function counter for  $f$  is not yet zero, job  $R$  is delayed. Once both  $J_1$  and  $J_2$  complete, the function counter is decremented accordingly and reaches zero, allowing  $R$  to proceed. This mechanism ensures that no job requiring function  $f$  is blocked indefinitely by a premature container reconfiguration.

### B. Queue-Based Scheduling and Real-Time Management

The MIQARA architecture uses a hardware-managed queue system for scheduling both critical and non-critical jobs, dispatching them out-of-order based on priority and availability. This ensures prompt processing of real-time jobs while maximizing resource use in reconfigurable containers. Efficient scheduling is achieved through dependency tracking and resource allocation, respecting job dependencies and mixed-criticality constraints.

The job scheduling and dispatch model in MIQARA is illustrated by an execution sequence in Figure 4. The job pipeline continuously assesses and dynamically reorders active and pending jobs to meet dependencies and criticality levels. This ensures real-time jobs get resources immediately, while best-effort jobs are scheduled as resources allow e.g. when a real-time job is waiting because of a dependency. This maximizes resource use and ensures predictable real-time behavior for mixed-criticality workloads.

### C. Job Pipeline Details

The job pipeline in MIQARA plays a crucial role in scheduling and allocating jobs across accelerators while respecting dependencies and reconfiguration needs. This centralized system manages the job order and assigns jobs to reconfigurable containers to ensure that dependencies and system constraints are met. MIQARA employs two distinct job queues: one for real-time jobs and another for best-effort jobs. The real-time jobs queue has the highest priority, ensuring that time-critical jobs are scheduled and executed promptly, with minimal and, importantly, bounded delays caused by low-priority workloads.

TABLE I: Resource Utilization on Zed Board

Component	LUTs
Accelerator container Size (per container)	3,200
Baseline Design	8,459
Job Pipeline (MIQARA)	14,529

As shown in Figure 5, the job pipeline handles job submission, dependency tracking, scheduling, and execution. Jobs enter a centralized queue system where they are prioritized based on their dependencies, criticality, and container availability. The pipeline monitors job progress and resource allocation, optimizing resource utilization and reducing idle times. This dual-queue approach effectively manages scheduling conflicts and ensures that real-time jobs are prioritized, allowing MIQARA to adapt dynamically to changing workloads. This flexibility is essential for mixed-criticality systems, simplifying dependency management, maintaining predictable job timing to meet worst-case execution time (WCET) requirements, and delivering robust performance for both real-time and best-effort jobs.

The job pipeline ensures that interference between real-time and best-effort jobs is minimal. Interference occurs in the memory interconnect, the scheduler, the job finish module, and the reconfiguration. The memory interconnect uses AXI crossbars to support full parallel memory transfers. Their interference occurs due to arbitration which is bounded as the crossbar can start new memory transactions every 3 cycles. Therefore, memory accesses may be delayed by up to  $3b + 2$  cycles, where  $b$  is the number of parallel best-effort jobs. The scheduler and job finish module prioritize real-time jobs; the interference is limited to a few cycles if the module started processing a best-effort job just before a real-time job could have been processed. For instance, the scheduler requires two cycles to start a job, i.e., a real-time job can be delayed by up to one cycle in the scheduler by parallel best-effort jobs. To reconfigure accelerators, we use partial reconfiguration. Hence, the remaining system remains fully operational during a reconfiguration, i.e., ongoing real-time jobs are not delayed. Finally, the accelerators themselves operate in isolation and do not interfere with each other.

#### D. Implementation and Benchmarking

We evaluate the MIQARA architecture’s viability and scalability by implementing it on three Zynq FPGA-SoC platforms of different sizes: large, medium, and small. This approach assesses the capabilities of MIQARA with varied hardware resources and demands, showing the adaptability of MIQARA across FPGA sizes and potential for mixed-criticality needs. Moreover, we evaluate MIQARA on a normal FPGA chip that does not contain any hard processors but has a soft RISC-V core on it to show our design’s portability.

1) *Implementation on the medium sized FPGA board:* The main evaluation is done on the medium sized board to avoid evaluating on an extreme case. The chosen board is the ZedBoard, which features a Xilinx Zynq-7000 SoC. The implementation on this platform utilizes eight accelerator containers. The reconfigurable containers are designed to accommodate a variety of accelerators, with each container capable of dynamic reconfiguration based on incoming job requirements. The job pipeline, managed by the PS, is responsible for submitting reconfiguration and compute jobs to the PL, leveraging the token system to enforce dependencies and maintain consistent scheduling policies. The implementation focuses on demonstrating the adaptability of the architecture, showing how MIQARA can dynamically reallocate resources and meet critical timing constraints under varying workload conditions. The benchmark used on this platform is AES.

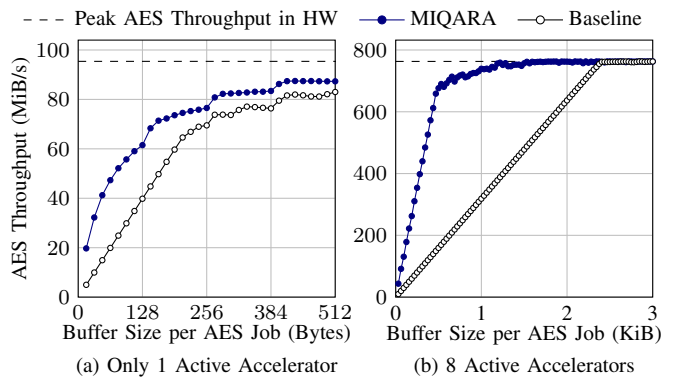


Fig. 6: Performance evaluation of MIQARA using AES jobs with varying buffer sizes compared to the baseline design.

2) *Implementation on the large-sized FPGA board:* The large-sized platform selected for evaluating MIQARA is the ZCU102 board, featuring a Xilinx Zynq UltraScale+ MPSoC. This board offers extensive reconfigurable logic resources and a high-performance processing system, making it well-suited for complex mixed-criticality applications. The implementation on this platform utilizes eight accelerator containers larger in size than on the ZedBoard.

Leveraging the larger capacity of the ZCU102, a streaming network is integrated into the architecture. This addition allows for efficient data flow between accelerators, reducing communication bottlenecks and enabling high-throughput data processing. The streaming network is beneficial for applications requiring large data transfers, as it minimizes latency and enhances overall system performance by optimizing data dependencies and reducing contention. To demonstrate the capabilities of this implementation, a video processing pipeline with an edge detection feature is used as the benchmarking application. This application benefits from the streaming network’s capability of handling data-intensive jobs.

3) *Implementation on the small-sized FPGA board:* To evaluate MIQARA on a constrained hardware platform, the DipForty1 board, featuring the smallest Zynq-7000 system, is selected. The implementation on this platform utilizes two accelerator containers, which are significantly smaller in size compared to the containers used in the ZedBoard and UltraScale+ versions. The benchmarking application used for this implementation is an object detection neural network workload with 5 layers, specifically designed for low resource scenarios. In this scenario, the accelerator processes one output channel at a time, with each output channel being processed independently of the others. Moreover, one input image can be divided in two halves, allowing parallel execution of the neural network.

4) *Implementation on the Arty A7 FPGA board:* To demonstrate the applicability of MIQARA on FPGA platforms that do not include a hardened CPU, we deploy the system on a Digilent Arty A7 board featuring an Artix-7 FPGA and a RISC-V soft-core processor. The soft-core is instantiated using the LiteX framework [22] with the VexRiscv processor [23]. All components, including the job pipeline and accelerators, are described in Migen [24]. In contrast to the Zynq-based platforms where communication relies on AXI, here the RISC-V compliant open source Wishbone protocol is employed for memory-mapped interactions between the CPU and accelerator containers. The implementation uses two accelerator containers, each capable of being dynamically reconfigured with lightweight compute kernels. To evaluate this setup, a set of representative integer-based benchmarks is used, namely Fibonacci number generation, addition, Hamming distance (HD) calculation, and a simple hash function.

TABLE II: Worst-Case Cycle-by-Cycle Breakdown of Job Completion Process

Cycle	Component Description
0	Job completion is signaled by the accelerator.
1	Accelerator control: The job is latched as finished job.
2	Job finish module: The job is latched as finishing job.
3	Job finish module: Decreasing the function counter is requested. Unlocking the token is requested. AWVALID and WVALID are asserted to update the job status. The job is passed to the interrupt control.
5	AXI crossbar 1: Potential wait cycles after best-effort request.
6	Token store: Acknowledge decrease of function counter. Interrupt control: IRQ is asserted.
7	AXI crossbar 1: AWVALID and WVALID are asserted.
8	BRAM controller: BVALID is asserted. Token store: Update of the token values in the reorder window.
9	AXI crossbar 1: BVALID is asserted to the job finish module.

#### IV. EVALUATION

To evaluate MIQARA comprehensively, its performance is benchmarked across the three FPGA boards to assess the architecture’s adaptability and efficiency under different conditions. To compare against MIQARA, we implement a baseline design that uses a simplified pipeline and omits mixed-criticality features such as token-based dependency tracking, job abortion, and real-time prioritization. This baseline reflects a general-purpose, non-criticality-aware reconfigurable. As no existing system offers complete hardware support for mixed-criticality execution with dynamic pipelining and job control, this baseline serves as a controlled reference to isolate and evaluate the contributions of our proposed architecture.

##### A. Evaluation on ZedBoard

The evaluation of MIQARA on the ZedBoard is conducted with a PL frequency of 100 MHz and a PS frequency of 666 MHz. MIQARA’s job pipeline is implemented with a reorder window depth of 16 jobs. As Table I shows, it requires a total of 14,529 LUTs. In contrast, the baseline design, which utilizes a simpler interface, occupies 8,459 LUTs, mainly for the high-throughput data interconnect. The architecture is implemented with eight accelerator containers, each having a size of 3,200 LUTs. The full system of the baseline uses 34,059 LUTs and the full system of MIQARA uses 40,129 LUTs with an overhead of 17.8%.

1) *Evaluation of the Execution of MIQARA*: To assess the performance of MIQARA, we use an AES accelerator, which is configured either on only one or all eight containers. AES uses 88% of LUTs within each accelerator container. We measure the encryption throughput using 1,000,000 encryption jobs with varying buffer sizes from 16 Bytes to 3 KiB, representing various workload characteristics. We compare the attainable performance with MIQARA to the performance on the baseline design, which requires the CPU to launch new jobs as soon as a previous job is finished.

Figure 6a illustrates the performance results across different buffer sizes when a single accelerator is used. MIQARA manages to efficiently utilize the accelerator, increasing the throughput by a factor of 4× for small buffer sizes (16 Bytes), decreasing with larger buffer sizes (e.g., 1.5× for 128 Bytes, 1.05×) for 256 Bytes. As already motivated in Fig. 1, this is as MIQARA mitigates the high communication overhead between CPU and accelerator. To show the full potential of MIQARA, Fig. 6b shows the attainable throughput with all eight accelerators. MIQARA achieves a speedup of around 4.5× for all buffer sizes below around 700 Bytes, decreasing with larger buffer sizes. Across all experiments, further increasing the buffer size results in higher utilization of the AES accelerator, eventually causing the throughput to approach an upper limit defined by the accelerators themselves.

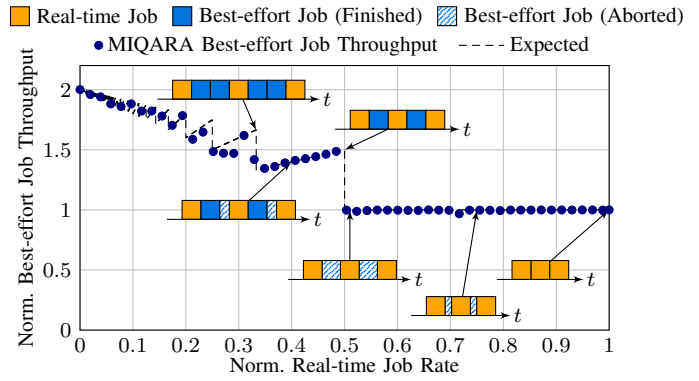


Fig. 7: Best-effort job throughput when reserving an accelerator for soft real-time jobs. The plot demonstrates the impact of increasing real-time job rates on best-effort job performance.

2) *Evaluation of Real-Time Guarantees*: To evaluate MIQARA’s ability to provide real-time guarantees, a detailed latency analysis is conducted to determine the worst-case execution times (WCET) for key stages of job execution. The evaluation focused on various stages such as job initialization, data transfer, job completion, and inter-job dependency handling. The results highlight the ability of the job pipeline to efficiently manage and dispatch real-time jobs, ensuring predictable execution and adherence to timing constraints. As an example, Table II outlines the key cycle components involved in job completion and their respective descriptions. In addition to the job completion process detailed in Table II, the worst-case delays for earlier pipeline stages are as follows: up to 1 cycle for scheduler arbitration, 2 cycles for job issue, and 23 cycles for BRAM arbitration (7 other accelerators). These values are fixed and account for all control-path delays affecting real-time job execution.

3) *Mixed-Criticality Evaluation*: In this evaluation, the performance of MIQARA under mixed-criticality workloads is assessed, focusing on the simultaneous handling of real-time and best-effort jobs. The experiment utilizes a system configuration with two active accelerators. Two CPUs were employed, with CPU 0 running a real-time application and CPU 1 executing a best-effort application. Both applications execute AES-128 jobs with a buffer size of 4 KiB.

To evaluate the sharing of resources, one accelerator is reserved for soft real-time jobs, while the second accelerator is shared between real-time and best-effort jobs. Real-time jobs are sent at a constant rate, ensuring timely processing on the reserved accelerator. Best-effort jobs, on the other hand, are allowed to execute on the second accelerator whenever it is idle or not needed for real-time jobs. If a real-time job becomes available, any ongoing best-effort job on the second accelerator is aborted and later restarted, ensuring prioritized real-time response.

The performance impact of this mixed-criticality setup is illustrated in Figure 7. The figure shows how best-effort throughput decreases as real-time job rates increase. For low rates of real-time jobs, best-effort jobs make efficient use of idle cycles. As the real-time job rate rises, however, the frequency of interruptions increases, impacting overall best-effort job throughput. Despite this, MIQARA effectively balances the execution of both criticality levels, leveraging its dynamic scheduling capabilities to maintain system performance.

##### B. Evaluation on DipForty Board

Next we evaluate MIQARA on the small sized FPGA. The implementation of MIQARA on the DipForty board is carried out with a reduced reorder window depth of two, resulting in a job pipeline

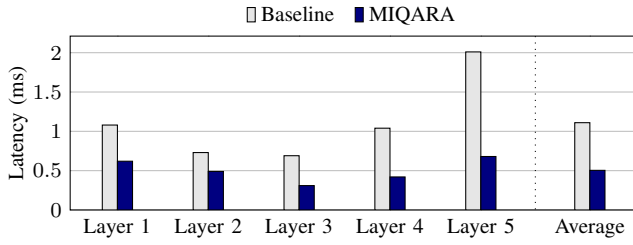


Fig. 8: Performance evaluation of object detection CNN on the DipForty board. The figure shows speedup achieved by MIQARA with job reordering and additional parallelism compared to the baseline implementation.

size of 5,832 LUTs. This setup utilized two accelerator containers, each with a size of 1,929 LUTs. The evaluation focused on the performance of an object detection CNN applied to images, where each image is divided in half for parallel processing. Each layer of the network is treated as an individual job.

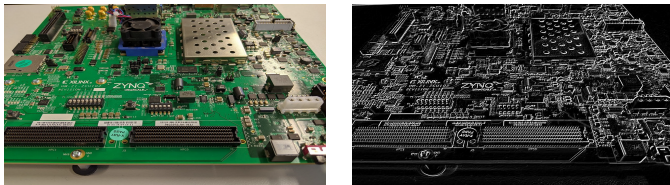
Figure 8 shows the evaluation results. In comparison to the baseline system, MIQARA demonstrated improved performance due to its ability to execute jobs out of order, leveraging its dynamic job reordering capabilities. The baseline implementation, which lacks these capabilities, is substantially slower as it adhered to a strict sequential execution order. Additional performance gains were achieved by introducing extra parallelism through the concurrent processing of channels within each layer. This allowed not only for independent processing of jobs from different halves of the image but also for parallel execution of channels per layer. This leads to significant speedups (maximum  $3.01\times$  and average  $2.17\times$ ), proving the adaptability of MIQARA in optimizing workloads.

### C. Evaluation on ZCU102 FPGA Board

The next evaluation of MIQARA is on the ZCU102 board. It uses its extensive reconfigurable logic resources to demonstrate the architecture’s adaptability and scalability. Each accelerator container is configured with a size of 6,720 LUTs, while the integration of a streaming network consumed 18,391 LUTs. The streaming network facilitated efficient data transfer and coordination between accelerators. To show the capabilities of MIQARA on this platform, the edge detection algorithm is used. Figure 9 demonstrates the results. In Fig. 9a, the ZCU102 board is shown. In Fig. 9b the output after applying the edge detection algorithm on the ZCU102 board.

### D. Evaluation on Arty A7 Board

The Arty A7 board is used to evaluate the applicability of MIQARA on FPGA platforms with soft-core processors. The system runs at a clock frequency of 100 MHz, with the RISC-V core. Two accelerator containers are instantiated in the programmable logic, each configured with an accelerator corresponding to the benchmark under execution. For each benchmark, we define three jobs,  $J_1$ ,  $J_2$ , and  $J_3$ , which are issued sequentially by the CPU. In the baseline design, a strict in-order policy enforces that  $J_2$  always depends on



(a) ZCU102 FPGA board.

(b) Edge detection output using MIQARA for the ZCU102 board.

Fig. 9: Evaluation of edge detection application on ZCU102 board.

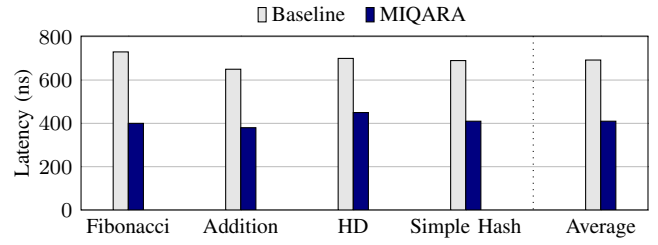


Fig. 10: Performance evaluation of several benchmarks on the Arty A7 board with RISC-V softcore. The figure shows speedup achieved by MIQARA across all benchmarks.

$J_1$ , which also prevents  $J_3$  from executing until  $J_1$  completes. As a result, no reordering occurs and accelerator utilization remains low. In contrast, MIQARA enables dynamic reordering of independent jobs by leveraging its hardware-managed job pipeline and token-based dependency tracking. This allows  $J_3$  to execute concurrently with  $J_1$ , while still respecting the true dependency of  $J_2$  on  $J_1$ . The evaluation results, illustrated in Fig. 10, show that MIQARA achieves an average speedup of  $1.69\times$  compared to the baseline design across the set of benchmarks. This demonstrates the effectiveness of MIQARA’s reordering mechanism even in soft-core FPGA environments, where resources are constrained and CPU overhead can otherwise dominate.

### E. Comparison to related work

Table III shows the comparison of MIQARA to state-of-the-art reconfigurable processors. MIQARA stands out in comparison to HiPreP [11] and FlexBex [5]. It supports mixed-criticality workloads, unlike HiPreP and FlexBex, which lack strong mechanisms for this. FlexBex supports dynamic scheduling and optimizes the communication to reduce the overhead but lacks a prioritization scheme for mixed-criticality workloads like MIQARA.

TABLE III: Qualitative Comparison of MIQARA with State-of-the-Art Reconfigurable Processors

Feature	MIQARA	HiPreP [11]	FlexBex [5]
Mixed-Criticality	✓	✗	✗
Dynamic Scheduling	✓	✗	✓
Dependency Management	✓	✗	✗
Real-Time Guarantees	✓	✗	✗
Optimized Communication	✓	✗	✓

No other reconfigurable processor supports mixed-criticality by offloading the job scheduling to the hardware itself. The works in [19, 20] rely on software-level job scheduling and OS modifications on Zynq platforms. For example, the scheduler in [20] incurs up to 180 clock cycles of overhead per job, while MIQARA launches jobs in 2 cycles with memory arbitration constrained to 23 cycles, enabling tighter and more predictable real-time behavior. Unlike prior work, MIQARA offloads all scheduling and dependency tracking to hardware, providing deterministic execution guarantees.

## V. CONCLUSIONS

We present MIQARA, a queue-based architecture for mixed-criticality reconfigurable accelerator platforms. Through a dynamic job pipeline, token-based dependency management, and out-of-order scheduling, MIQARA ensures high resource utilization and strict timing guarantees. Evaluations showed its scalability and adaptability on various FPGA platforms, with significant performance enhancements in execution speed, flexibility, and efficiency over baseline designs. The MIQARA architecture effectively manages both real-time and best-effort jobs in mixed-criticality environments, making it suitable for diverse embedded applications. Compared to a baseline design, MIQARA requires a hardware overhead of 17.8% and achieves speedups of  $4\times$  while being able to maintain real-time guarantees.

## REFERENCES

- [1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The worst-case execution-time problem—overview of methods and survey of tools”, *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 5 2008.
- [2] M. Damschen, L. Bauer, and J. Henkel, “Extending the WCET Problem to Optimize for Runtime-Reconfigurable Processors”, *ACM Trans. on Archit. and Code Optim.*, vol. 13, no. 4, pp. 45:1–45:24, Dec. 2016.
- [3] A. Podlubne and D. Göhringer, “Reconfigurable computing systems as component-oriented designs for robotics”, in *Int. Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 412–415.
- [4] J. Kim, K. Park, and T.-H. Kim, “A reconfigurable inference processor for recurrent neural networks based on programmable data format in a resource-limited FPGA”, in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2022, pp. 470–475.
- [5] N. Dao, A. Attwood, B. Healy, and D. Koch, “FlexBex: A RISC-V with a reconfigurable instruction extension”, in *Inf. Conf. on Field-Programmable Technology (FPT)*, 2020.
- [6] A. Burns and R. Davis, “Mixed criticality systems—a review”, *Department of Computer Science, University of York, Tech. Rep.*, 2022.
- [7] C. Wulf, M. Willig, and D. Göhringer, “A survey on hypervisor-based virtualization of embedded reconfigurable systems”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 404–407.
- [8] C. Wulf and D. Göhringer, “Virtualization of embedded reconfigurable systems”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 460–461.
- [9] R. Balas and L. Benini, “RISC-V for real-time mcus: Software optimization and microarchitectural gap analysis”, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
- [10] Xilinx Inc., *Zynq UltraScale+ MPSoC Technical Reference Manual*, November 2024, uG1085 (v1.11).
- [11] P. S. Käsgen, M. Messelka, and M. Weinhardt, “HiPReP: High-performance reconfigurable processor - architecture and compiler”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 380–381.
- [12] A. Kamaleldin and D. Göhringer, “Design for agility: A modular reconfigurable platform for heterogeneous many-core architectures”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 408–411.
- [13] B. Liu, Z. Shen, L. Huang, Y. Gong, Z. Zhang, and H. Cai, “A 1D-CRNN inspired reconfigurable processor for noise-robust low-power keyword recognition”, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
- [14] A. Bansal, R. Tabish, G. Gracioli, R. Mancuso, R. Pellizzoni, and M. Caccamo, “Evaluating the memory subsystem of a configurable heterogeneous MPSoC”, in *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, Barcelona, Spain, July 2018.
- [15] G. Akgün and D. Göhringer, “Power-aware real-time operating systems on reconfigurable architectures”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 402–403.
- [16] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, “A framework for supporting real-time applications on dynamic reconfigurable fpgas”, in *Real-Time Systems Symposium (RTSS)*, 2016.
- [17] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke, and C. Rochange, “WCET and mixed-criticality: What does confidence in WCET estimations depend upon?” in *OASiCS-OpenAccess Series in Informatics*, vol. 47. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- [18] R. Ernst and M. Di Natale, “Mixed Criticality Systems—A History of Misconceptions?” *IEEE Design & Test*, 2016.
- [19] C. Wulf, N. Charaf, and D. Göhringer, “Virtualization of reconfigurable mixed-criticality systems”, in *Int. Conf. on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 54–60.
- [20] C. Wulf, N. Charaf, and D. Göhringer, “Scheduling of hardware tasks in reconfigurable mixed-criticality systems”, in *Int. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022.
- [21] A. S. Arefin and M. A. Kashem Mia, “NP-completeness of the minimum edge-ranking spanning tree problem on series-parallel graphs”, in *Int. Conf. on Computer and Information Technology*, 2007.
- [22] F. Kermarrec, S. Bourdeauducq, H. Badier, and J.-C. Le Lann, “Litex: an open-source soc builder and library based on migen python dsl”, in *OSDA 2019, colocated with DATE 2019 Design Automation and Test in Europe*, 2019.
- [23] *VexRiscv: an FPGA-friendly RISC-V CPU implementation*, <https://github.com/SpinalHDL/VexRiscv>, SpinalHDL, accessed: 2025-09-15.
- [24] *Migen: A Python domain-specific language for hardware description*, <https://github.com/m-labs/migen>, M-Labs, accessed: 2025-09-15.