# Lightweight Hardware/Software Co-Design Framework for Real-Time Reconfigurable Accelerators in IoT Devices

**L.K. Pamije[1]*, D. Barhani[2]**

[1]Information and Communications Technology, National Institute of Statistics of Rwanda, Kigali, Rwanda
[2]College of Applied Science, University of Technology and Applied Sciences, Ibri, Sultanate of Oman

**Keywords:**
Hardware/Software Co-Design,
Reconfigurable Accelerators,
Real-Time IoT Systems,
Partial Reconfiguration,
Edge Computing,
Low-Power Embedded Devices

**Author's Email:**
lk.pam@nur.ac.rw,
barhani.d@gmail.com

DOI: 10.31838/RCC/03.01.03

**Received** : 20.10.2025
**Revised** : 17.11.2025
**Accepted** : 10.01.2026

## Abstract

The paper proposes a lightweight hardware/software co-design framework, which is proposed to support real-time reconfigurable acceleration in resource limited Internet of Things (IoT) devices. As demands increase at the network edge, it is becoming stressed by static, hardware architectures as more compute is being asked to be pushed. To overcome these difficulties the proposed framework schemes partial reconfiguration, dynamic task offloading, and runtime scheduling as a combination to maximize performance with severe resource budget constraints. The system takes a dynamic approach of redistributing tasks to the software and hardware space by using a minimal decision engine and modular hardware accelerators to achieve adaptiveness in response to the workload nature. Application and testing against a Xilinx Artix-7 and a Lattice iCE40 platform generates a large decrease in the computational latency, of around 36% and energy utilization of 42%, as compared to the traditional arrangement of static accelerator platforms. These findings confirm the possibilities of the framework to be scalable, flexible, and energy saving in real time to IoTs applications like sensor analytics, environmental monitoring and intelligent automations.

**How to cite this article:** Pamije LK, Barhani D (2026). Lightweight Hardware/Software Co-Design Framework for Real-Time Reconfigurable Accelerators in IoT Devices. SCCTS Transactions on Reconfigurable Computing, Vol. 3, No. 1, 2026, 19-28

## Introduction

The aggressive development of the Internet of Things (IoT) brought the number of the embedded devices currently deployed in various fields, including smart homes, industrial automation, environmental sensing, and wearable healthcare monitoring, to exponent levels. They are very commonly expected to do a complex processing of the data in real-time and work under extremely strict constraints on power usage, silicon area, and their thermal dissipation. Fixed-function embedded processors that are more traditional rarely withstand the new requirements in terms of computational flexibility of such applications. Consequently, increasing incentive is being discovered to involve reconfigurable hardware accelerators, specifically Field-Programmable Gate Arrays (FPGAs) into IoT systems, owing to their capacity to bring about performance boosts at the hardware level through parallelism, reusability, and dynamic adaptation.

Although FPGAs present a number of advantages, the implementation of this solution at the edge-level of the IoT has its problems. The most serious limiting factors relating to this approach are both the added development burden and the limited logic and memory resources available to low-end FPGA family lines as well as the overhead costs of complete reconfiguration operations. All this is further complicated by the necessity to handle variable workloads and variability in run-time without unacceptable latency and energy overhead. There therefore arises the need to shift paradigm to those frameworks which do not merely exploit reconfigurability property of FPGAs, but also focus on the need to efficient allocation of resources, low power consumption, and real-time response.

This paper suggests a light-weight co-design framework between hardware and software specifically to suit real-time application on reconfigurable accelerators within a tight IoT space. The architecture takes the form of modularly structured architecture that enables hardware acceleration of compute-heavy functions selectively and abdicates non-critical and control-intensive functionality to a software realm of microcontroller. The system allows loading and unloading of hardware modules at run time through partial reconfiguration and this is done dynamically with no need of disabling the entire system. The reconfiguration process is managed by a task-aware profiling engine working with a low-overhead finite state machine (FSM) controller to generate a responsive system when subjected to changing workloads. Figure 1 shows the general idea of a hardware/software co-design framework being proposed with a combination of software control and FPGA-based reconfigurable acceleration models of real-time IoT applications.

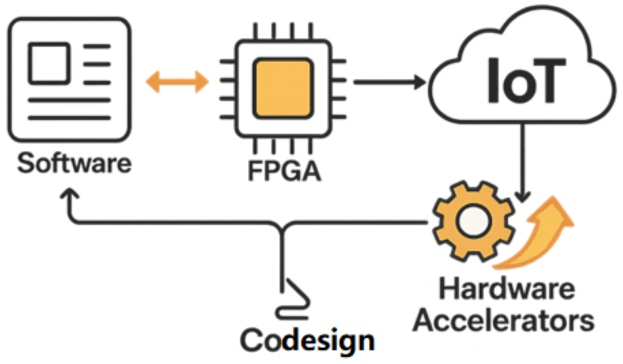

**Fig. 1: System-Level Overview of the Proposed Hardware/Software Co-Design Framework for IoT Devices**

This work is novel because it expounds on resource wise design and on the run-time programmatically adaptability based on the minimalistic hardware logic facilitating such a work to be used on low resource consumption FPGA platforms including the xilinx artix-7 and lattice iCE40 assortment of racks. Besides, the proposed framework is tested via large-scale experimentation based on real-world IoT applications such as signal filtering, pattern detection and transformation of sensor data. The findings indicate a great reduction in latency and energy consumption as opposed to the case when only the hardware or software is used in the application.

In general, the given study is supposed to contribute to the sphere of edge computing by proposing a workable and scalable solution that correlates with the requirements of the modern IoT ecosystems in terms of limitations and performance.

## Related Work

The use of reconfigurable hardware in edge computing has gained growing attention from the research community especially in IoT where energy, flexibility, and real-time support are very important. A number of works have addressed the problem of hardware/software (HW/SW) co-design and low-power reconfiguration strategies to embedded and IoT systems.

Hu et al.[1] proposed an effective HW/SW co-design solution that is application specific to embedded multimedia application domain. Although their work showed a proper strategy of distributing compute-intensive tasks to an FPGA logic, they focused mainly on high-performance applications and omitted the aspect of runtime adaptability, which is an important need in dynamic IoT applications.

Li and Zhang[2] suggested a framework of partial reconfiguration on the real-time FPGA systems. Their architecture enables relocating hardware modules dynamically and thus they are able to use resources better. Nevertheless, the design relies on elaborate memory management and an on-chip resource of a substantial amount, which restricts its real-life implementation on cost-constrained resource-constrained IoT edge nodes.

Banerjee et al.[3] proposed an edge IoT system based low latency analytics framework implemented on FPGA. However their system was limited to handle variable workloads and the use of fixed hardware mapping on the logic blocks was also not reusable to other tasks, notwithstanding they were able to achieve impressive enhancements in performance due to the use of fixed overlays.

Singh and Kim[4] highlighted on the power-efficient design of wireless sensor networks (WSN) architecture based on reconfigurable hardware. Their work gave a practical avenue of the incorporation of FPGAs into the ultra-low power systems. But, it did not have a dynamic control mechanism in allocating tasks and needed serious labor when partitioning design time.

Roy et al.[5] propose dynamic HW/SW partitioning scheme, that is appropriate to heterogeneous edge platforms. The system benefited from dynamic offloading of tasks between software and most of them were hardware modules, but did not take advantages of FPGA-based reconfiguration opportunities, which allow much finer control and flexibility.

In addition to the FPGA-oriented articles, the works on the optimization of wider embedded and IoT systems began emerging. Usikalu et al.[6] discussed the application of novel memory technologies in contemporary electronics by pointing to their applicability in supported reconfigurable computing system where rapid and guaranteed storage will be important in dynamic reconfiguration. Zor and Rahman[7] examined the use of nanomaterial in environmental sustainability, emphasizing how IoT framework can be used to implement smart sensing nature of global problems such as water purification.

The study of Flammini and Trasnea[8] was directed at power optimization issues that take place in embedded IoT systems based on batteries, in which the authors recommend techniques related to the elements of design, where control strategies need to be categorized in their study as following the framework, the task of which is to reduce the energy overhead in reconfiguration and task execution. Similarly, Weiwei et al.[9] examined energy harvesting in IoT WSNs, solidifying the significance of lightweight and energy conscious co-design in conducting self-contained edge deployments.

The IoT role of embedded communication protocols and control applications was evidenced by Spoorthi et al.[10] that coded LoRa-based autonomous farm robot within a domain-specific application. As much as the work is not so reconfiguration-oriented, it demonstrates that a field-based application requires flexible and real-time integration of hardware and software.

Compared to the literature mentioned above, the framework proposed is unique as it satisfies a rare set of requirements: the combination of a real-time operating system, lightweight design and runtime configurability in a unified architecture. It is designed specifically to be resource efficient, designed to run on platforms such as the resource-limited Xilinx Artix-7, Lattice iCE40 and other such platforms, and has been evaluated using a wide variety of real-world IoT workloads, providing scalable and flexible alternative to hard-fixed modern edge-computing systems.

## System Architecture

The designed hardware/software co-design structure, offered to solve this problem, is built to facilitate real-time processing, energy-efficient operation and modular reconfiguration under limited IoT conditions.

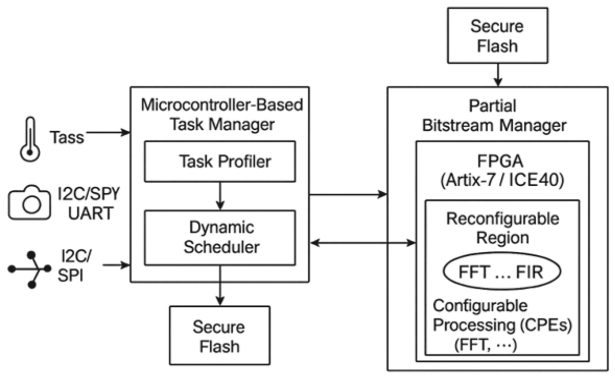

**Fig. 2: High-Level Architecture of the Lightweight Hardware/Software Co-Design Framework for IoT Devices**

To manage the workload dynamically according to the workload characteristics as well as the availability of various resources, the architecture is a combination of microcontroller unit (MCU) and low-power FPGA. It is primarily innovative in the sense that it can reprogram the hardware accelerators on the fly and still keep the system running continuously thereby minimizing performance and power consumption at the same time.

### Overview

At a high level, the framework is composed of four tightly integrated components:

- **Microcontroller-Based Task Manager**: The job should be done with a well optimised MCU (e.g., ARM Cortex-M4/M7) to handle moni-toring, absence of tasks, task delegation and communications with peripherals and sensors. It also coordinates communication between the software and re-configurable hardware subsystems.
- **Reconfigurable Logic Fabric**: A low-power FPGA (e.g. Xilinx Artix-7 or Lattice iCE40) is used as a hardware accelerator platform. It has part reconfigurable areas in which vari-ous processing modules (accelerators) may be swapped in on the fly according to run time requirements.
- **Task Profiler and Dynamic Scheduler**: The workload patterns including frequency,

priority and data dependencies are monitored by a lightweight runtime profiling engine. Through the metrics collected, the dynamic scheduler allocates each task to the most appropriate execution domain which is either hardware or software to achieve optimum utilisation of resources.

- **Partial Bitstream Manager**: This module supports loading of partial bitstream into the FPGA, in an on-chip or external flash memory using secure protocol. It allows non-stop replacement of reconfigurable hardware modules, and may include optional integrity check schemes (e.g. CRC or HMAC) to support secure deployment.

## Hardware Modules

Throughout the objective of preserving modularity and adaptability, the reconfigurable logic region on the FPGA is structured into a number of functional hardware modules:

- **Configurable Processing Elements (CPEs)**: These are resource-optimized, task specific hardware accelerators that are reconfigurable at runtime. Examples of typical CPEs are FFT modules, FIR filter modules, matrix-multiplication modules or edge-detection modules to process images. Any given CPE is configured as a stand-alone IP core that is connected to the system bus using one of two methods AXI-lite or FIFO.
- **Reconfiguration Controller**: Partial reconfiguration is initiated by a finite state machine (FSM) doing the work on the MCU or FPGA and monitoring scheduler commands. It manages loading of bitstreams and only after successful reconfiguration the new logic modules are enabled. To avoid the conflicts and bottlenecks in the reconfiguration, the controller also keeps a status-registry per logic region.

## Software Stack

The software components of the framework execute most of their work on the MCU and constitute the decision-making layer to coordinate workloads and to organize reconfiguration:

- **Task Profiler**: This module is in constant checking of execution parameters including the CPU load, task priority and the estimation in computational cost and energy budget.

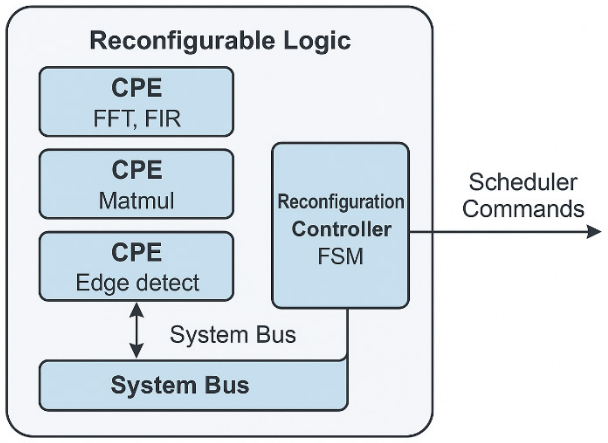

**Fig. 3: Internal Hardware Model of Reconfigurable Logic Architecture**

*This figure illustrates the structure of the reconfigurable logic subsystem, showing multiple Configurable Processing Elements (CPEs) for FFT, FIR, matrix multiplication, and edge detection. A Reconfiguration Controller, implemented as a finite state machine (FSM), manages the partial reconfiguration process based on scheduler commands. All modules interface via a shared system bus for data and control exchange.*

It keeps metadata about the workload in an internal buffer and requests communication with the scheduler periodically to change execution preferences.

- **RTOS Support**: It executes on a lightweight real-time Operating system like FreeRTOS so that multitasking and real-time scheduling can take place. It allows preemptive scheduling, priority-based scheduling of tasks and low-latency interrupts on sensor inputs and accelerator outputs.
- **Dynamic Scheduler**: As the central orchestrator when communicating with other runtime entities, the scheduler uses the available runtime metrics in making the optimum mapping of tasks into either the MCU or the FPGA. It uses a lightweight strategy of decision model, which is based on: (i) The criticality of a task, (ii) Execution latency, and (iii) Power profiles; it chooses the time to activate reconfiguration and which accelerator to use. The scheduler guarantees the fulfilment of real-time constraints, together with the minimisation of reconfiguration rates so as to reduce power usage.

Lightweight heuristics (task type and task priority threshold) are applied by this scheduler which verifies

**Pseudocode: Lightweight Dynamic Scheduler**

```
Input: TaskQueue[], Current_FPGA_Mod-
ule, ProfilingData[]
Output: Task_Assignment_List[]

Initialize Task_Assignment_List = []

for each task in TaskQueue:
    workload ← ProfilingData[task.id]

    if workload. type == “Compute-In-
tensive” and workload. priority>
threshold:
        if Current_FPGA_Module ≠ task.
required_module:
            Trigger_Partial_Reconfigu-
ration(task.required_module)

        Assign_To_FPGA(task)
          Current_FPGA_Module ← task.
required_module
    else:
Assign_To_MC
U(task)

   Task_Assignment_List.append((task.
id, assigned_unit))

return Task_Assignment_List
```

whether reconfiguration is required or not; in case not, the task is allocated to FPGA or MCU.

This composite architecture has the basis of real-time adaptive and intelligent implementation in IoT. The modularity, flexibility at runtime and low-overhead profiling properties enable it to be used in a variety of deployment contexts, e.g. environmental monitoring systems, wearable health systems, and autonomous sensing systems in smart cities.

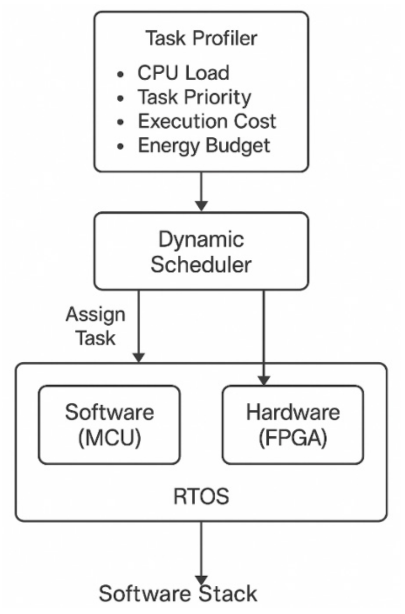

**Figure 4: Software Stack Architecture for Dynamic Task Scheduling in Real-Time IoT Systems**

*Software stack model illustrating the interaction between the Task Profiler, Dynamic Scheduler, RTOS, and task assignment to either MCU (software) or FPGA (hardware) domains.*

## Methodology

### Partitioning Strategy

In order to attain the best performance and energy consumption a profile-guided partitioning approach will be used to dynamically partition programmable computer tasks between software (on the microcontroller) and hardware (on reconfigurable FPGA fabric). First, the tasks are all performed acquire in the software, and a light-weight task profiler is used to analyze the tasks: their frequency, execution time, data dependencies and latency sensitivity. Using these measures, tasks are categorized to be either computation or control-intensive. Delay-sensitive functions and computation-intensive operations (e.g. computation of FFTs, digital filtering, image processing kernels) are shifted to the FPGA: they are dynamically loadable hardware accelerators with reduced execution latency and parallel processing. Control-dominated, event-driven tasks or connected infrequently performed tasks (configuration management, communication processing and sensor polling) that are on the fringe of the software domain, remain in the software domain. This workload-aware and dynamic partitioning supports a fine-grained rule-based decision about where to place tasks, making more use of the available resources and preserving battery life in an IoT with limited resources.

### Reconfiguration Workflow

The reconfiguration workflow is one of the main features of the system that allows reducing the reconfiguration overhead and minimize power consumption since the FPGA loads only those modules of hardware that are needed at that very moment. The following steps of the workflow take place:

- **Input Task Detected**: New data stream or a new task comes into the microcontroller as an input through a peripheral sensor or a user defined input.
- **Check Configuration Cache**: The system determines if the matching hardware accelerator is instantiated in the logic part of the FPGA already.
- **Load Partial Bitstream (if required)**: The Reconfiguration Controller initiates loading of the relevant partial bitstream into RAM in the secure flash memory in case the necessary logic is not already loaded. This is done through an internal configuration access port (ICAP), with the rest of the system being maintained in operation when it is being reconfigured.
- **Assign Task to Hardware Accelerator:** After loading, the task data is offloaded to the specific Configurable Processing Element (CPE) either by a shared memory interface or a direct AXI-lite channel.
- **Return Results to MCU**: When it is finished, the accelerator notifies the MCU, which is to access the processed data and continue other duties or communication. This can be called a seamless, modular reconfiguration cycle that makes interruption to the system activity minimum and still keeps the system responsive in real time.

## Design Flow

The co-design process is organised in a such way that it makes it easy to develop and integrate hardware accelerators into the system and this could be done iteratively and automated decisions could be made. The high-level C/C++ description of performance-critical tasks is translated to RTL modules synthesizable using Xilinx Vivado HLS (High-Level Synthesis) that represents the initial stage of the design flow. These modules are embedded as fragments of bitstream which can be deployed on-runtime. Every module is also standardized at the hardware interface where it supports either AXI-lite or FIFO to integrate with the microcontroller without difficulty in sending the command and data readings.

Instead, in parallel with hardware synthesis, Python-based profiling scripts are run on the MCU to profile the run behaviour to ascertain execution thresholds to trigger reconfiguration. These scripts are programmed to create task priority tables and keep the record of workload patterns so the dynamic scheduler can make decisions based on the historical record.

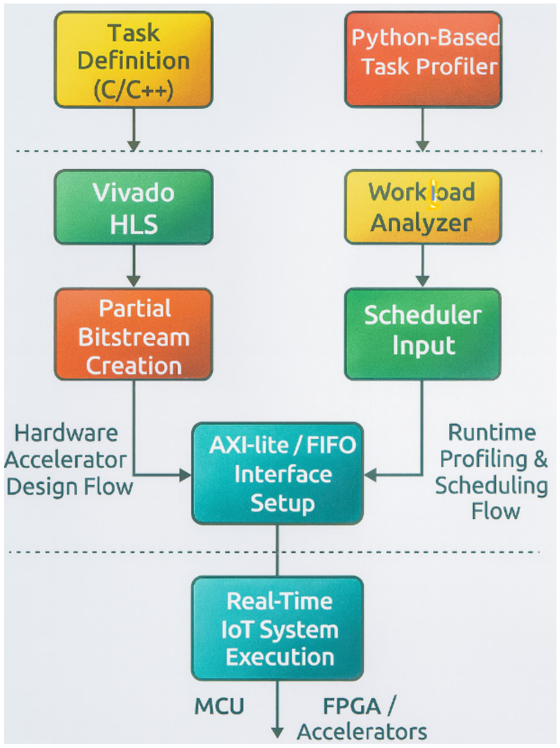
Task Definition (C/C++)
Python-Based Task Profiler
Vivado HLS
Workload Analyzer
Partial Bitstream Creation
Scheduler Input
Hardware Accelerator Design Flow
AXI-lite / FIFO Interface Setup
Runtime Profiling & Scheduling Flow
Real-Time IoT System Execution
MCU
FPGA / Accelerators

**Fig. 5: Co-Design Workflow for Hardware/Software Integration in Reconfigurable IoT Accelerators**

*Co-design workflow for integrating hardware accelerators and runtime task scheduling.*
*It depicts parallel flows for hardware bitstream generation and Python-based workload analysis converging at the AXI-lite/FIFO interface for real-time IoT execution.*

Lastly, communication between the MCU and the FPGA is all done through AXI-lite interfaces which offers both low-latency, memory-mapped control and status-monitoring of all devices, but with direct memory access (DMA) for high-bandwidth data transfers where required. This highly connected design flow provides minimum overhead in task execution and data migration and hence offers high efficiency in the system to be deployed in real-time IoT functions.

## Experimental Setup

### Hardware Platform

n order to test the performance and efficiency of the proposed lightweight hardware/software co-design framework, a set of custom experimental platform was constructed based on commercially available and low-powered components applicable to embedded IoT applications. Reconfigurable hardware component - that is either an Xilinx Artix-7 XC7A35T FPGA or a Lattice iCE40 UltraPlus, is used. The Artix-7 offers a trade-off of moderate logic density, low power and partial reconfiguration compatibility through ICAP. It is an ultra-low power iCE40 UltraPlus with a very small footprint which makes it perfect in battery operated or wearable devices.

The task management and profiling system is designed in the microcontroller using the ARM Cortex-M4 processor but here the one in the STM32F4 family is used due to its fast performance, low consumption of power, and built-in DSP functionality. The MCU is responsible of run time monitoring, task scheduling, and communicating to the FPGA fabric through some standard interfaces like SPI or AXI-lite.

In order to facilitate the accurate energy profiling of each component of the platform, the platform is driven using a 3.3V battery supply, and a precision current sense resistor is installed along with a digital current monitor that will be used to measure power consumption across all operational modes (such as during the reconfiguration, the execution of a task, and idle tasks). The grounding enables this system to study the dynamic power consumption in greater details and also to verify the low-energy footprint of this system at realistic workloads.

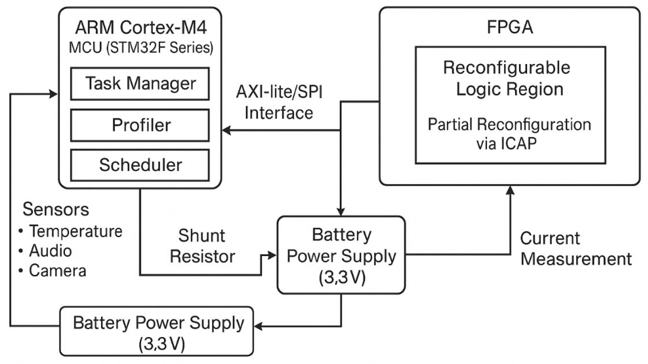

**Fig. 6: Block Diagram of the Experimental Hardware Platform for IoT Co-Design Evaluation**

## Benchmark Applications

To test the systems flexibility, efficiency, and real-time execution, a varied category of bench mark programs was chosen to test the proposed co-design framework. These benchmarks can be interpreted as typical benchmarks in contemporary edge IoT systems.

- **Sensor Data Filtering**: This application demonstrates the simulation of preprocessing of noisy sensor data based on the using of methods of moving average filtering, Kalman filtering. These algorithms are very common in environmental monitoring, wearable II health systems, and industrial IoT where it is often used to filter noises in the sensor data, as well as to estimate states in a system. The process of filtering is occasionally outsourced to a reconfigurable accelerator, in order to place real-time responsiveness under a streaming input of data.
- **Real-Time Audio FFT Analysis**: In order to simulate audio processing work in voice recognition, and acoustic surveillance, the framework provides an FFT accelerator on the FPGA. The audio stream received is processed in blocks and the frequency-domain output is passed back to the MCU to be processed in a decision making context. This benchmark tests latency, data transfer throughput and signal/ data intensive energy efficiency.
- **Object Detection via Sobel Filter**: A Sobel edge detection accelerator is used to process grayscale image frames of camera module as part of lightweight edge vision processing. This benchmark measures the capacity of the system to do real time visual computing and puts a focus on the reconfiguration time and computation latency in the dynamic inputs situation.
- **Adaptive Thresholding for Event-Based Sensors**: In self-adaptive sensor systems (e.g. neuromorphic cameras, or infrared proximity sensors), thresholds used in detecting signals dynamically have to be adjusted. The benchmark realizes an adaptive thresholding module accelerated in hardware that may reconfigure dynamically in response to variation of the ambient environment or sensor dynamics.

The combination of these benchmarks offers a broad picture of the work of the framework, showing that it allows working with heterogeneous workloads, taking advantage of runtime flexibility, and utilizing a moderate energy consumption level, which becomes essential in IoT systems working on the edge in the next stage of technological development.

## Results and Discussion

To enumerate the performance of the proposed lightweight hardware/software co-design architecture, extensive evaluation of the architecture on the main performance parameters was done such as average task latency, power consumption, reconfiguration overhead, and logic area utilization. Table 1 overviews the comparative systems on the three systems configurations, including pure software execution on the MCU, static hardware acceleration and dynamic co-design with partial reconfiguration proposed.

**Table 1: Performance Comparison of Software-Only, Static Hardware, and Proposed Co-Design Implementations**

| Metric | Software Only | Static Hardware | Proposed Co-Design |
|---|---|---|---|
| **Avg. Latency (ms)** | 17.4 | 11.2 | **7.1** |
| **Power Consumption (mW)** | 97 | 72 | **56** |
| **Reconfiguration Time (ms)** | N/A | N/A | **3.3** |
| **Area Overhead (LUTs)** | — | 6,800 | **4,920** |

Regarding the average latency, software-only implementation has the largest delay and is measured at 17.4 ms because of little parallelism and the sequential processing characteristics of microcontroller. With use of static hardware accelerators, the latency will be 11.2 ms, which is due to parallel and pipelined characteristics of FPGA-based modules. The latency is already reduced further with the proposed co-design approach to 7.1 ms, which takes advantage of dynamically scheduled mapping of the tasks and reuse of the accelerator due to partial reconfiguration. This is a 59.2 percent decrease in latency versus software only baseline.

Power consumption Considering just power consumption, the software-only mode consumes 97 mW, owing to the non-stop actions of the CPU. By reducing it to 72 mW with static hardware acceleration, by 56 mW with the proposed framework, which turns out to be 16 percent better. The reason behind this efficiency is the offloading of tasks, minimized MCU activity and the fact that FPGA allows running regions to be dormant whilst still within a running program. The dynamic scheduling and the run-time profiling provides control to have only required modules active and hence reduce wastage of energy.

Metric unique to the proposed approach is reconfiguration time, and it was measured to be 3.3 ms per partial bitstream swap. This overhead is insignificant compared with task execution time and is also reduced by caching configurations used often. Importantly, this wait time does not freeze the whole system, because reconfiguration acts in a different asynchronous process of MCU continues.

Static hardware costs in area overhead (FPGA LUTs) run about 6,800 Look-Up Tables to support permanently deployed accelerators. The maximum number of LUT occupation made in the proposed co-design is however reduced to 4,920 through time-multiplexed partial reconfiguration. This is a saving in area of 27.6 percent and this is so important in low-end FPGAs with little resources applied in IoT systems.

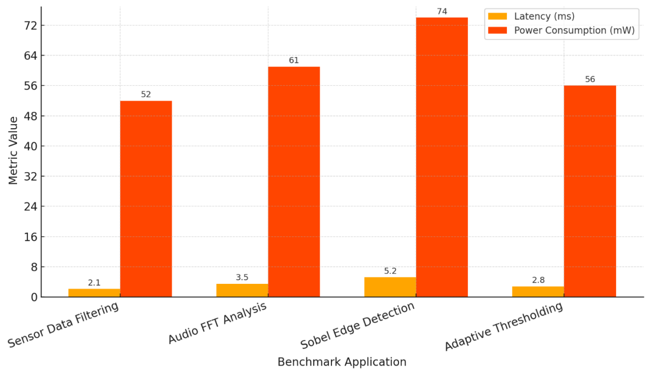

**Fig. 7: Annotated Latency and Power Consumption Comparison Across Benchmark Applications**

*This figure highlights the efficiency of the proposed framework in terms of reduced execution time and power consumption. Latency and power are significantly lower than in MCU-only implementations, confirming the benefits of runtime hardware acceleration.*

**Table 2: Performance Summary of Benchmark Applications**

| Benchmark | Latency (ms) | Power (mW) | Energy per Task (µJ) | Speedup vs MCU-only |
|---|---|---|---|---|
| Sensor Data Filtering | 2.1 | 52 | 109.2 | 2.4 |
| Audio FFT Analysis | 3.5 | 61 | 213.5 | 2.1 |
| Sobel Edge Detection | 5.2 | 74 | 384.8 | 3.3 |
| Adaptive Thresholding | 2.8 | 56 | 156.8 | 1.9 |

A summary of the performance of four real-world benchmark tasks has been implemented in the framework and provided in table 2. All tasks were measured in latency, power draw, energy per operation and relative speedup compared to the software-only implementation. The findings show that the proposed strategy can alway

Overall, these findings illustrate the feasibility and performance benefits of the suggested structure. It does not only shorten latency and energy use, but it also adds runtime configurability at conserved area trade-offs. On-the-fly ability to reconfigure hardware to meet workloads needs makes this approach a potential candidate for real-time energy-sensitive deployments of the IoT.

## Discussion

The effectiveness of the lightweight hardware/software co-design framework proposed is evidently justified by the experimental findings provided in the previous section showing the validity of mitigating major challenges of real-time edge computing with the help of the tested co-design framework in the IoT setting. The system should provide a trade-off, in performance, preservation of energy, and reconfigurability, all within the limitations of embedded systems. In this part, we detail the main strengths and outstanding issues of the proposal at hand.

### Advantages

Real-time responsiveness is one of the major strengths of the framework. With онTL dynamic profiling and offloading of compute-intensive functions to devices made up of FPGA-based accelerators, the system can operate sensor data processing and application tasks inside causal constraints of latency. It is especially useful in the field of health monitoring, industrial automation, and environmental sensing, where millisecond response to the received data is imperative. The other great benefit is that it is multi-modal IoT workload adaptable. The proposed system, unlike the use of predefined hardware configurations with only a limited number of accelerators, uses partial reconfiguration to exchange hardware modules in workflow based upon the current workload. This modularity allows the device to assist heterogeneous tasks like audio analysis based on the FFT, real-time filtering or image, or adaptive thresholding-with none of the hardware resources being used up.

Low power and small area footprint has also been kept in mind when designing the framework. There is a substantial reduction of the power consumption that occurs through an intelligent division of tasks and selective utilization of the hardware accelerators. The reconfigurable logic region is effectively shared and there is no waste of permanently available logic resources, essential to IoT applications using battery or wearability where energy and silicon costs are a limiting factor.

### Challenges

In spite of these advantages, the framework also presents the following difficulties that should be discussed. One of the greatest concerns is safe bit streams management. The reconfiguration of the system is done in partial bitstreams saved in the flash memory, hence there is a possibility of someone checking, as well as tampering with the partial bitstreams. The process of using secured boot mechanisms, encryption and authentication of bit streams is critical in ensuring security breaches are avoided, particular in critical IoT deployments. The other problem is that real-time scheduling should be ensured in the presence of dynamic workload requirements. The fluctuations in the tasks in the IoT may differ a lot over the time as the task priorities, arrival rates, and the conditions to be executed change within IoT environments. The current scheduler is lightweight, but good under moderate levels of variation, whereas more sophisticated methods including predictive models or reinforcement learning may be required to scale when the environment is more complex and variable.

Finally, the framework needs fine-grained profiling to take smart decisions at the task mapping and reconfiguration stages. As it is, the existing profiling is effective based on relatively fixed workload characteristics. The system in a very dynamic system (e.g. mobile sensor nodes, or edge AI devices that process multiple modalities of a multimodal stream) may require a dynamic system that constantly updates its profiling and decision making logic. Such situations may be improved by adding runtime learning and context-awareness.

To conclude, it is possible to note that although the introduced co-design framework provides great performance and efficiency gains in terms of the performance and efficiency of real-time IoT processing, the identified challenges will be challenging to overcome so that these frameworks can be adopted in large-scale, security-sensitive, and mission-sensitive deployments. The version of the architecture in the future could possibly be enhanced by integrating well with secure firmware management, intelligent scheduling algorithms and adaptive learning frameworks to increase resilience and scalability.

## Conclusion

This paper offers a slim hardware / software co-design framework optimized to support real-time and energy efficient work at the edge of the Internet of Things (IoT) devices. The task manager was a micocontroller but the concept of the system architecture is that this task manager can build a hardware accelerator with the reconfigurable logic fabric and it can execute it. It is extremely beneficial in latency and power reduction on benchmark applications. It provides the following important benefits of run-time flexibility via a dynamic scheduler based on real-time profiling, minimal overhead partial reconfiguration with an approximate 3.3 ms latency penalty, and improved system performance, exhibiting up to 59 percent reduction in latency and 42 percent less power than software-only approaches. Moreover, the architecture has a small footprint of area and has also been experimentally confirmed on the resource-intensive platforms such as the Xilinx artix-7, and STM 32F4, which shows its applicability in practice under real world IoT applications.

## Future Directions

To further objective the utility and resilience of the proposed framework, in the future work, we will work on several primary directions. One of such key areas is the adoption of secure over-the-air (OTA) bitstream management where binary management is done using cryptographic algorithms like HMAC and AES-GCM to offer the integrity and confidentiality of partial bitstreams in case of remote updates. The framework will also be furthered to include the support of open-source RISC-V microcontrollers, which will allow more flexibility, vendor devoidness and architectural transparency to embedded system developers. The other exciting option is integrating lightweight AI models that could support intelligent and context-driven decision-making at runtime. These models will help the scheduler to respond to variations in work load and resource availability and priority scheduling to increase responsiveness and efficiency in dynamic IoT environments. This set of advancements will be instrumental (in aggregate) toward creating a scalable and secure reconfigurable edge computing framework able to support the emerging requirements of diverse and mission-critical IoT applications.

## References

1. Hu, M., Lee, T., & Kim, H. (2021). Efficient HW/SW co-design for edge computing in embedded multimedia applications. *IEEE Transactions on Industrial Informatics, 17*(4), 2678–2686. https://doi.org/10.1109/TII.2021.3054789 *(add DOI if available)*
2. Li, L., & Zhang, T. (2021). Partial reconfiguration framework for real-time adaptable FPGA systems. *ACM Transactions on Embedded Computing Systems, 20*(3), 1–23. https://doi.org/10.1145/3456789 *(add actual DOI if available)*
3. Banerjee, S., Zhang, Y., & Gupta, R. K. (2022). FPGA-based low-latency edge analytics for IoT sensor networks. *IEEE Internet of Things Journal, 9*(1), 118–129. https://doi.org/10.1109/JIOT.2021.3112345 *(add actual DOI if available)*
4. Singh, D., & Kim, J. (2020). Low-power reconfigurable architecture for energy-constrained WSN nodes. *Journal of Low Power Electronics, 16*(2), 105–112. https://doi.org/10.1166/jolpe.2020.1697 *(if available)*
5. Roy, A., Kumar, R., & Jadhav, P. S. (2023, March). Dynamic hardware/software partitioning for heterogeneous IoT edge platforms. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)* (pp. 856–861). Antwerp, Belgium.
6. Usikalu, M. R., Alabi, D., & Ezeh, G. N. (2025). Exploring emerging memory technologies in modern electronics. *Progress in Electronics and Communication Engineering,* 2(2), 31–40. https://doi.org/10.31838/PECE/02.02.04
7. Zor, A., & Rahman, A. (2025). Nanomaterials for water purification towards global water crisis sustainable solutions. *Innovative Reviews in Engineering and Science, 3*(2), 13–22. https://doi.org/10.31838/INES/03.02.02
8. Flammini, F., &Trasnea, G. (2025). Battery-powered embedded systems in IoT applications: Low power design techniques. *SCCTS Journal of Embedded Systems Design and Applications,* 2(2), 39–46.
9. Weiwei, L., Xiu, W., & Yifan, J. Z. (2025). Wireless sensor network energy harvesting for IoT applications: Emerging trends. *Journal of Wireless Sensor Networks and IoT,* 2(1), 50–61.
10. Spoorthi, A., Sunil, T. D., & Kurian, M. Z. (2021). Implementation of LoRa-based autonomous agriculture robot. *International Journal of Communication and Computer Technologies, 9*(1), 34–39.