A Bandwidth Reservation Mechanism for AXI-Based Hardware Accelerators on FPGAs

Marco Pagani, Enrico Rossi, Alessandro Biondi,
Mauro Marinoni, Giuseppe Lipari, Giorgio Buttazzo

Scuola Superiore Sant'Anna, Pisa, Italy
Université de Lille, Lille, France

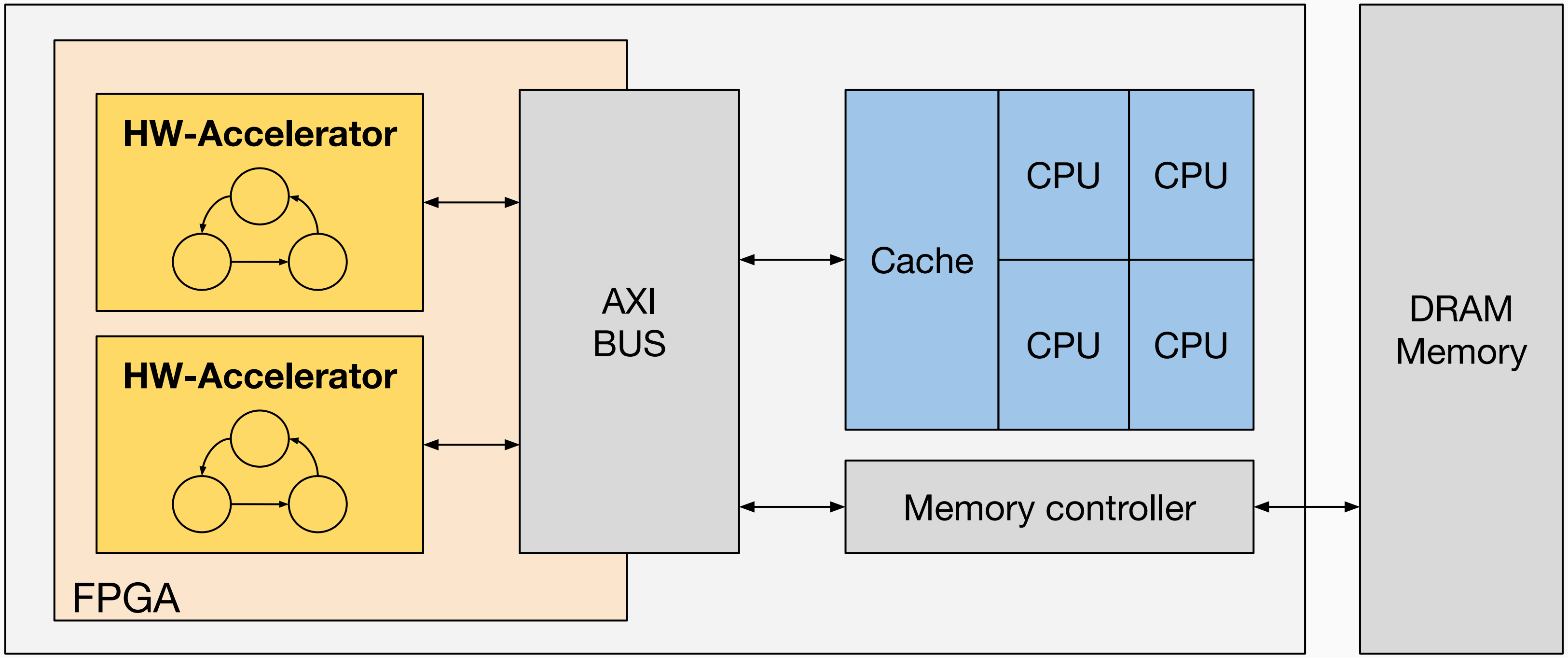
Real-Time Systems Laboratory

Centre de Recherche en Informatique, Signal et Automatique de Lille

Background

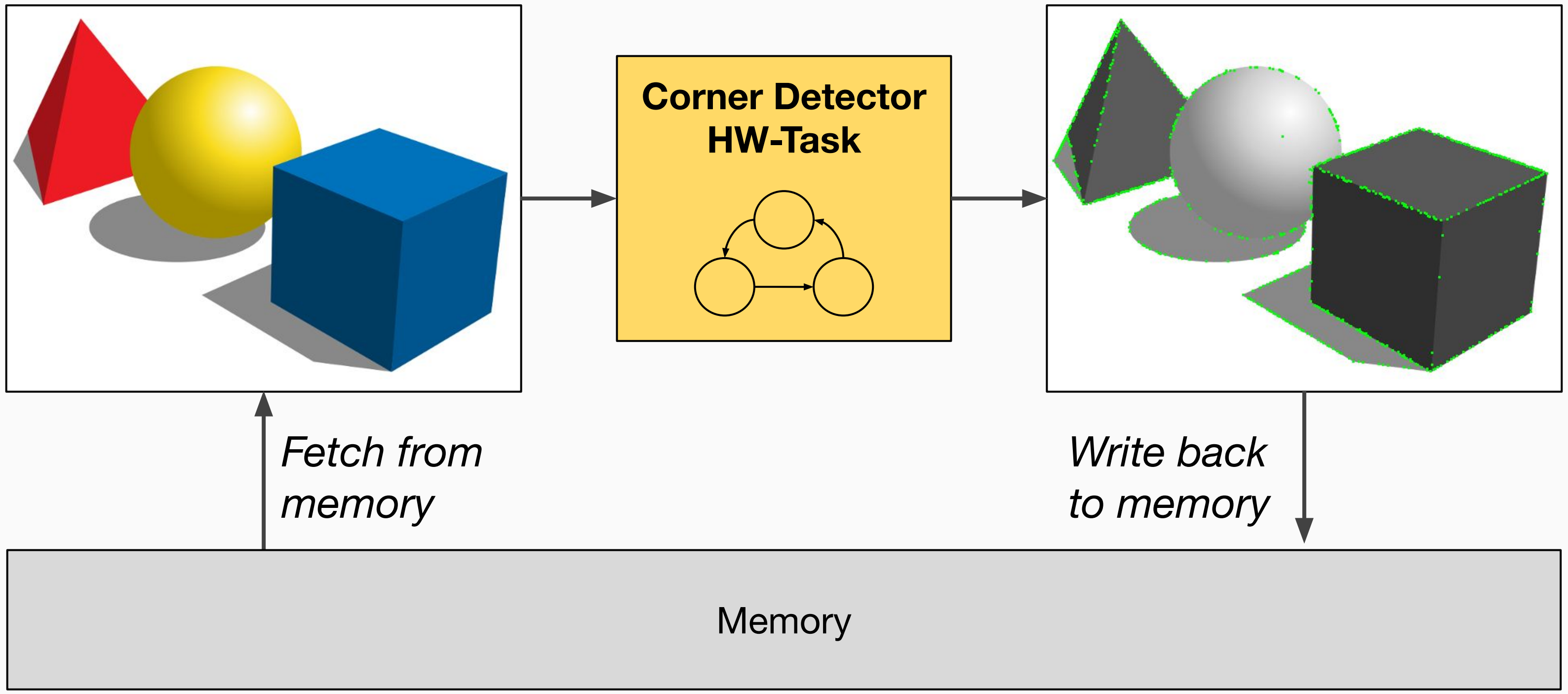
Background

- **Heterogeneous platforms** are popular to meet computational requirements while overcoming **scaling** and **energy issues**.
- **System-on-a-chip (SoC)** platforms including an **FPGA** are **extensible** systems:
  - **Allow offloading** computational activities from **CPUs** to **HW accelerators** deployed on the FPGA fabric.



DMA / bus mastering accelerators

- This work considers **HW accelerators** performing the same **computational activity** (e.g. processing a frame) at each run. Hence, they are referred to as **HW-tasks**.
- **High-performance HW accelerators** implement **bus mastering / DMA** to directly access data in the system memory;



Why FPGA-based acceleration for real-time systems?

Pros

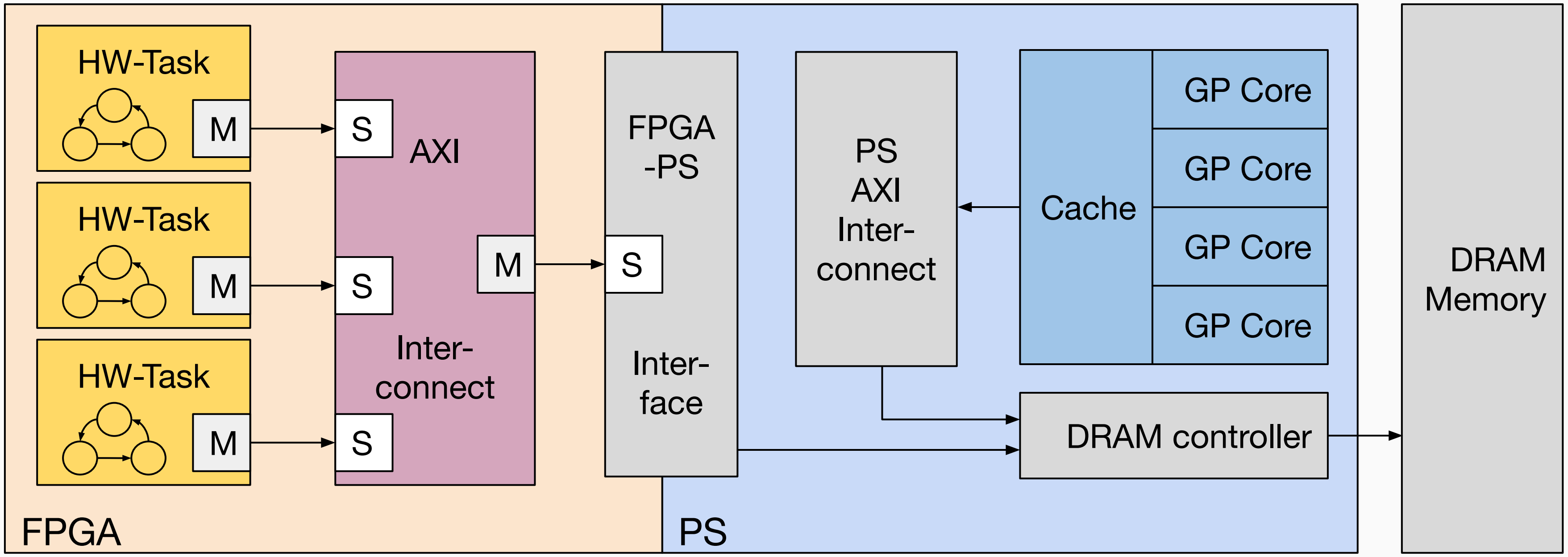
- **Very predictable** clock-level behavior;
- Possibility to **explicitly control** the behaviour of the accelerators;
- **High-performance** on SIMD / Dataflow operations;
- Can be **swapped** at **run-time** using partial reconfiguration.

Cons

- Developing HW accelerators for FPGAs requires **specific knowledge** that is typically **not** part of the **background** of SW programmers.
  - Even modern tools like **high-level synthesis** are **not so straightforward** to use.
- **Less libraries** and SW stacks are **available** with respect to other platforms (e.g., GPUs).

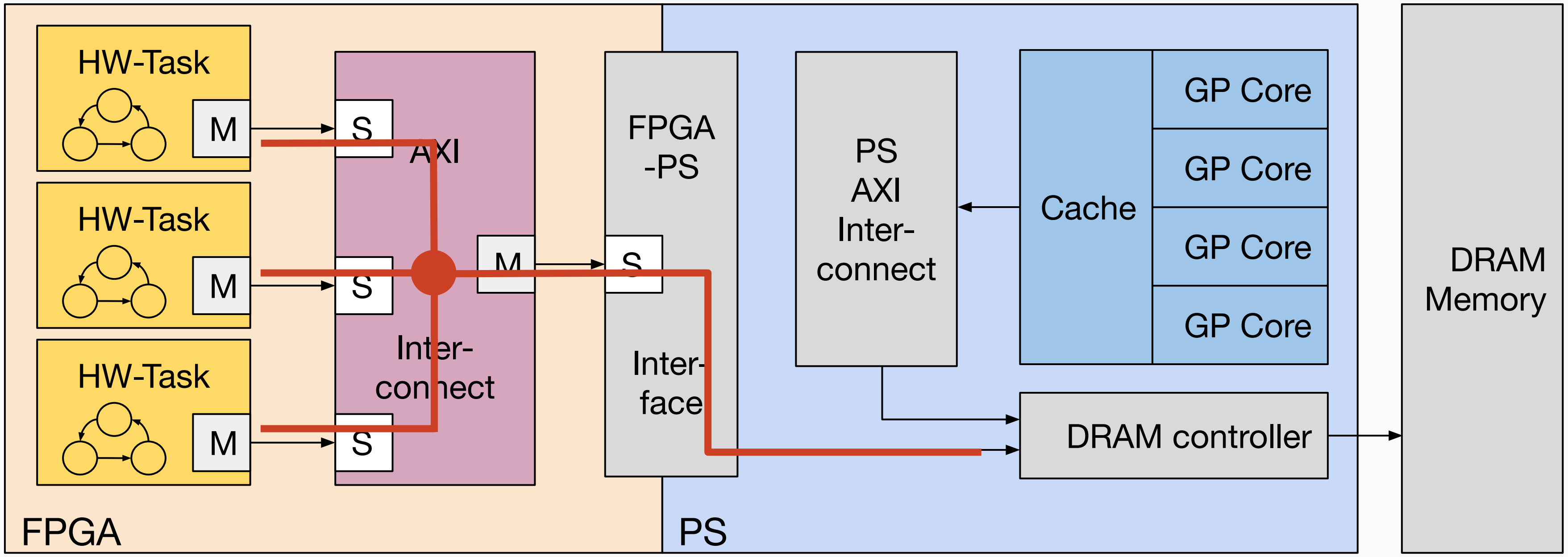
BUS contention problem

- To support **multiple HW-tasks**, an **AXI Interconnect** is required for arbitrating transactions **arbitration**.
  - **FPGA** to ***Processing System* (PS)** AXI ports provide a direct path to reach the DRAM memory controller.



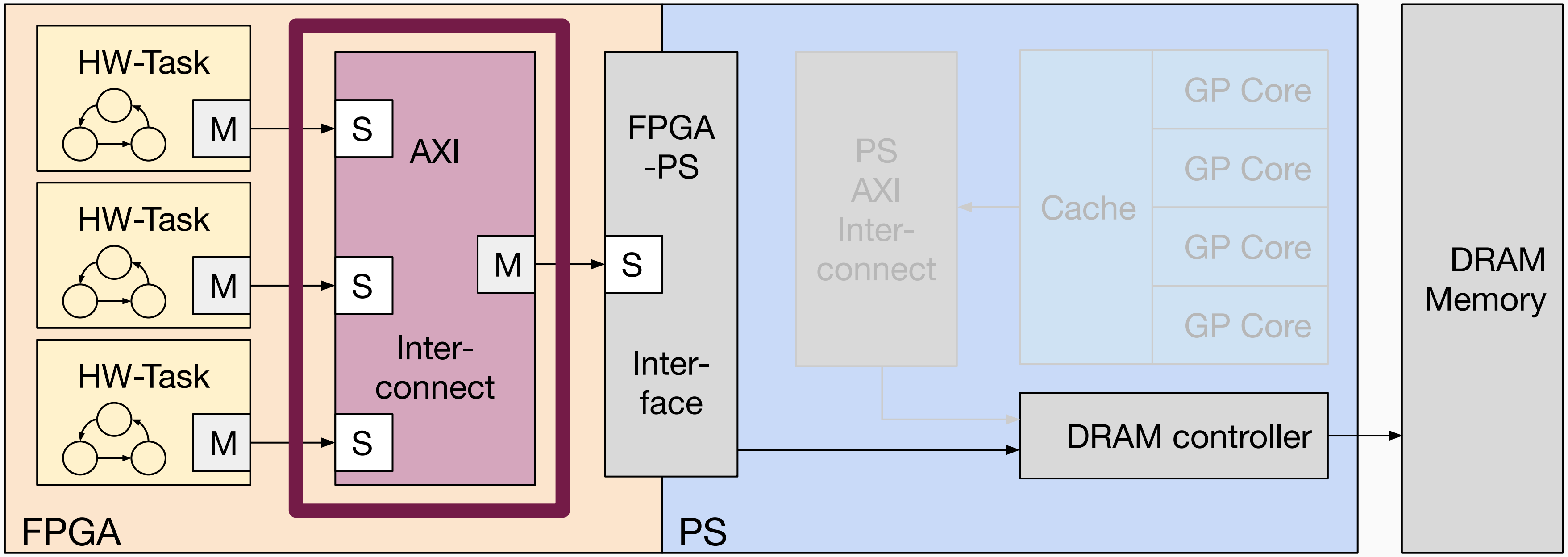
BUS contention problem

- To support **multiple HW-tasks**, an **AXI Interconnect** is required for arbitrating transactions **arbitration**.
  - **FPGA** to ***Processing System* (PS)** AXI ports provide a direct path to reach the DRAM memory controller.
- To enable a **sound timing analysis** of **HW-tasks** on FPGA, it is crucial to pay attention at how **bus transaction** are managed.



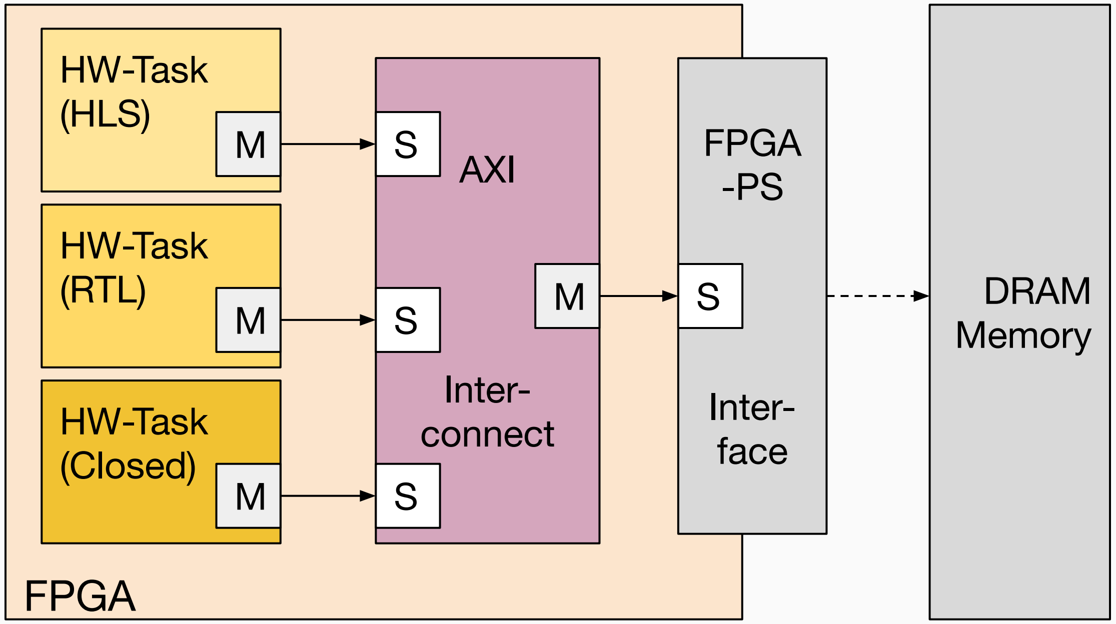
BUS contention problem

- Stock **Interconnects** implement **Round Robin** arbitration.
  - Designed for **throughput**.
  - **Difficult** to **explicitly control** the **response times** of **HW-Tasks** (e.g., priority-based arbitration is deprecated in Xilinx platforms).
  - **No protection** form **misbehaving HW-tasks**.



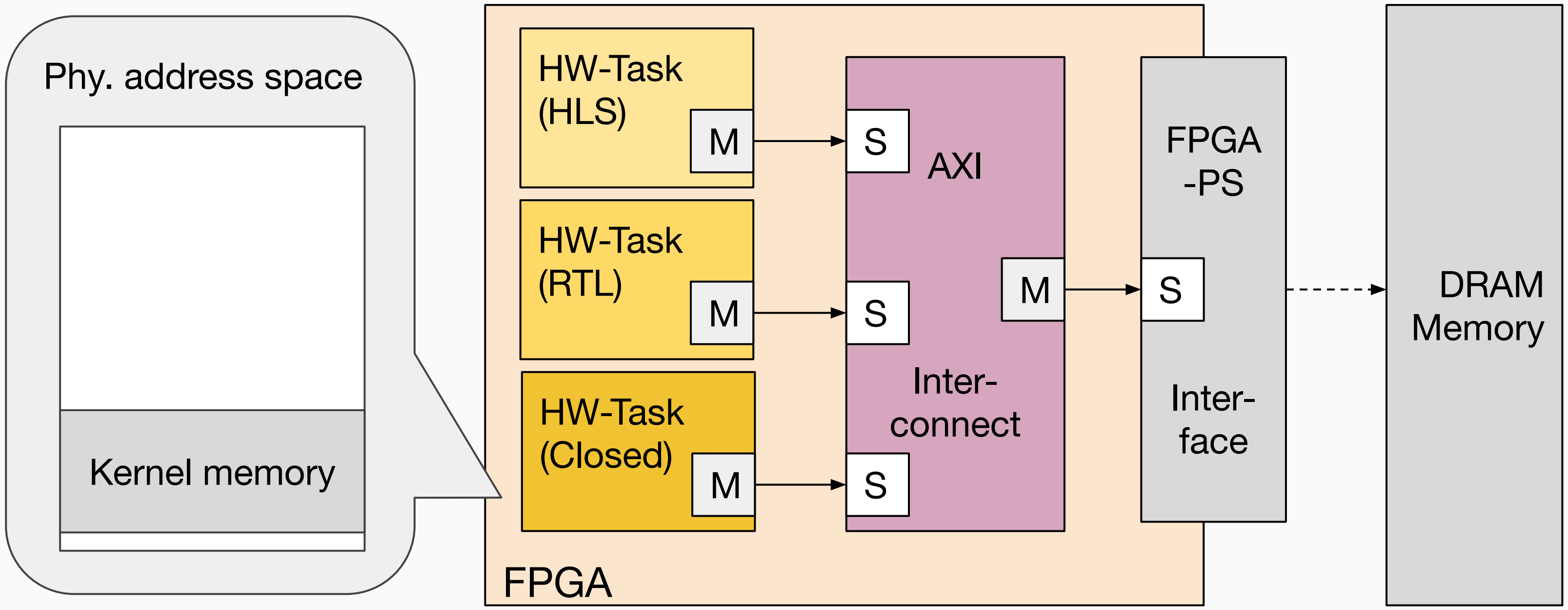
Other issues with HW-tasks: heterogeneous sources

- In practice, **HW-tasks** may come from **different sources**:
  - Some **HW-Tasks** may be designed with **HLS**
    - Implicit configuration of bus transactions;
  - Some others may be **closed-source** (no access to HDL code).
- Even if we assume that no **HW-tasks** can misbehave, it may be **difficult** to **control** the **bus bandwidth demanded** by each **HW-Task** at **integration time**.



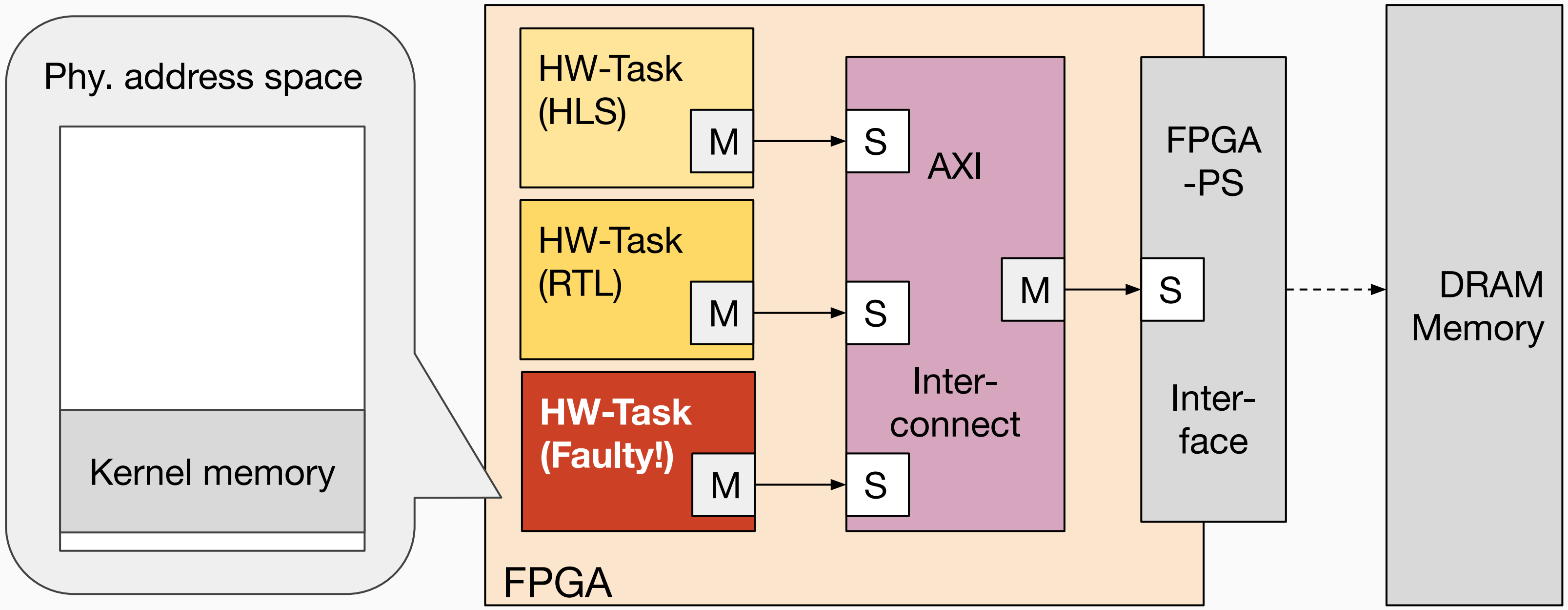
Other issues with HW-tasks: address space protection

- Due to the AXI master interface, **HW-Tasks** may **access** the **whole** physical **memory without any restriction** (including OS kernel mem).
- This poses **serious threats** to system safety.



Other issues with HW-tasks: address space protection

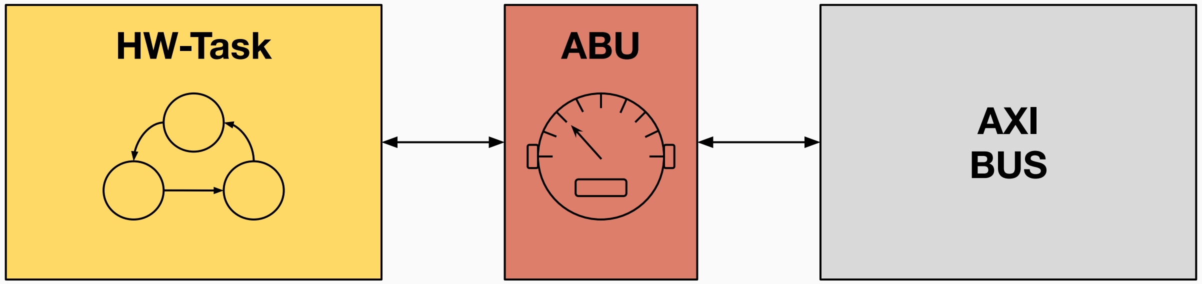
- Due to the AXI master interface, **HW-Tasks** may **access** the **whole** physical **memory without any restriction** (including OS kernel mem).
- This poses **serious threats** to system safety.
  - What if a **faulty/bugged HW-Task** skews OS/proc mem?
  - A **malicious HW-Task** could be purposely designed to do so!
  - A **protection/containerization** mechanism could be useful.



The AXI Budgeting Unit (ABU)

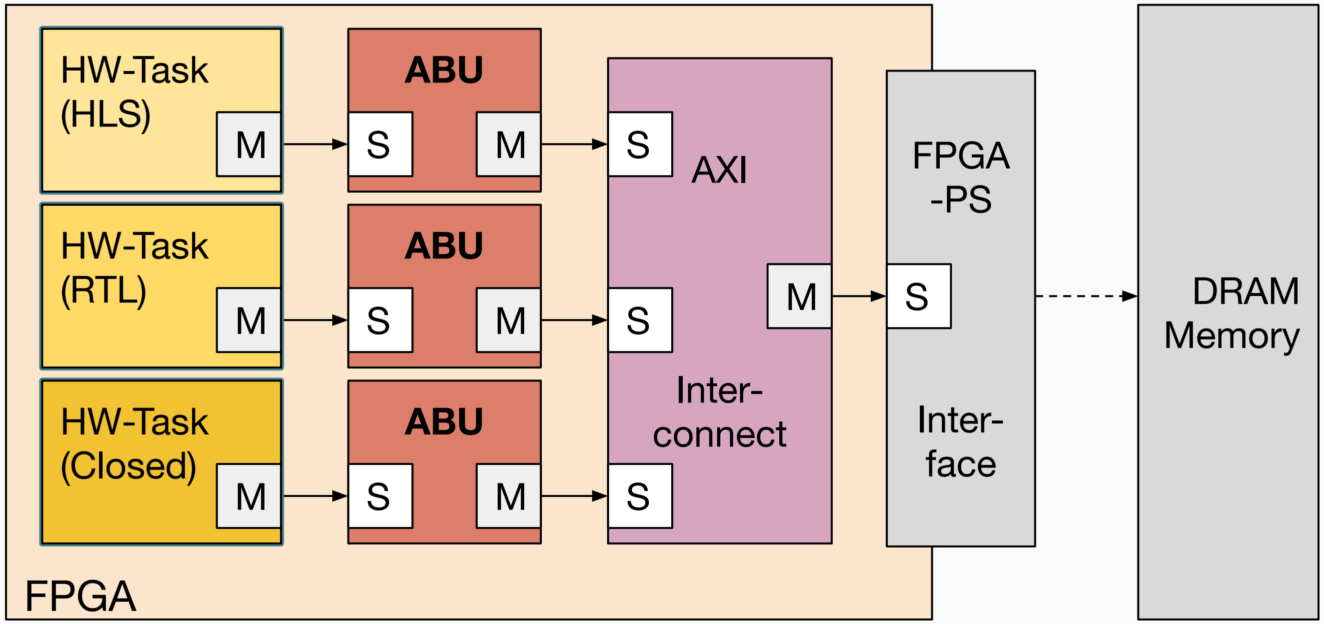
Contribution of this work: the AXI Budgeting Unit (ABU)

- The **ABU restores** the system **predictability** by regulating the **BUS** bandwidth **contention**.
  - Provides a **BUS bandwidth reservation** mechanism for **HW-tasks** on FPGA;
  - **Supervises** BUS **transactions** for accesses control.
  - **Ensures temporal** and **spatial isolation** for **HW-tasks**.



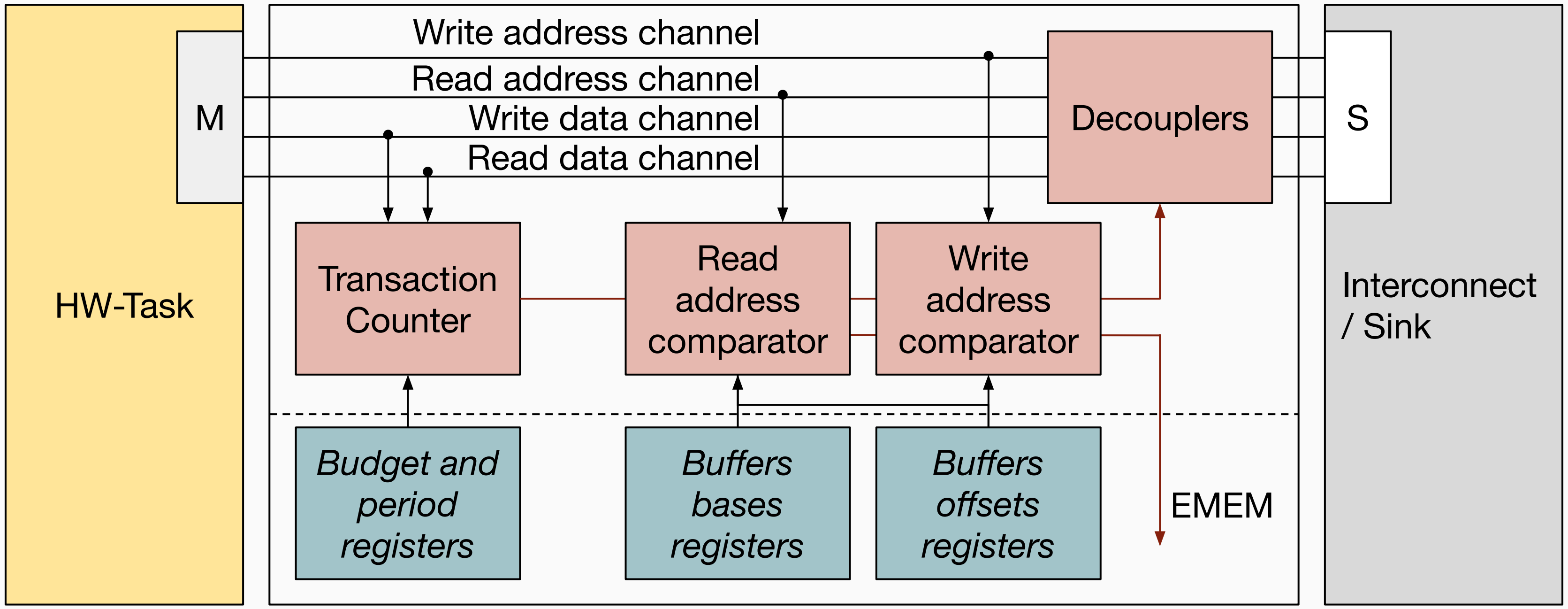
The ABU is a supervision mechanism for HW-tasks

- Provides a **confined environment** allowing for a **safe integration** of first- and third-party **HW-Tasks**.
  - Allows to **explicitly control** the **bus bandwidth reserved** to each **HW-Task**;
  - **Shields the system** from possible **misbehaving HW-Tasks**.



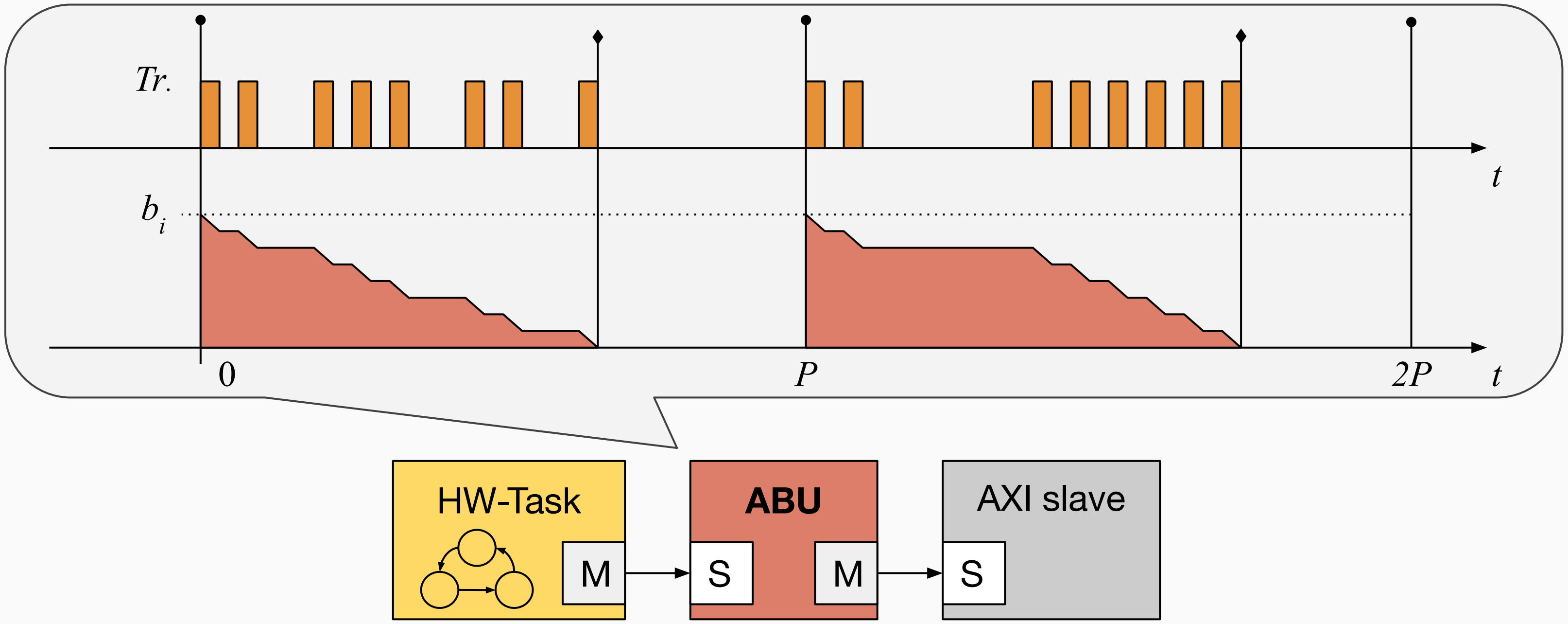
How does it work? Internals

- The **ABU monitors** all **AXI channels** (in parallel):
  - **Checks** and compares the **address** of transactions;
  - **Senses** and **counts** transactions;
- **AXI channels** are routed through **decoupler blocks** that can **stop** the **HW-task** from issuing transactions.



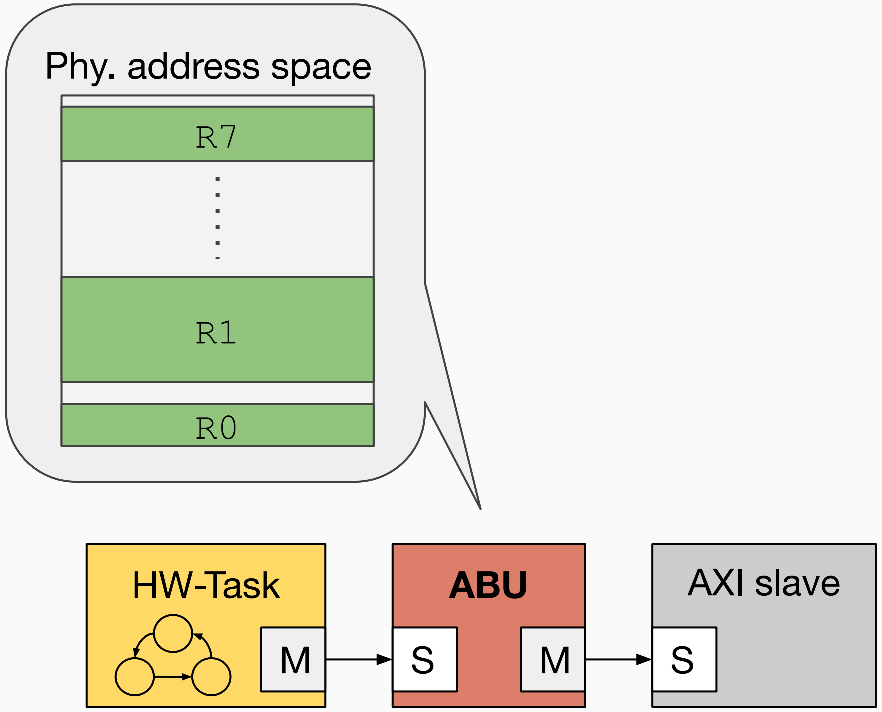
How does it work? Temporal isolation

- Each **ABU** has a **budget** b_i
  - **Each transaction** passing on the AXI link **consumes** a budget unit.
  - The **budget** is **periodically replenished** to the maximum value B_i every P clock cycles.
- When the **budget** reaches zero, the **HW-Task** is **interdicted** from issuing transactions (<u>disconnecting</u> the handshake signals valid/ready).



How does it work? Spatial isolation

- Each **ABU** allows specifying up to 8 **address segments.**
  - Each **segment** is defined by a **base** address and a **size**;
  - The **ABU** monitors the transactions issued by the **HW-Task**.
- If the **HW-Task** issues a transaction **outside** of the 8 segments, the **ABU blocks** the **HW-Task** and notifies the processor (interrupt).



Overheads and Resource consumption

- The **ABU** does **not** introduce any additional **latency**.
    - Temporal **overhead free** (no additional delay).

Overheads and Resource consumption

- The **ABU** does **not** introduce any additional **latency**.
  - Temporal **overhead free** (no additional delay).
- Key observation: the budget **replenishment period** can be **arbitrarily small** (a few clock cycles) **without particular penalties**
  - Differently from software reservation servers!

Overheads and Resource consumption

- The **ABU** does **not** introduce any additional **latency**.
  - Temporal **overhead free** (no additional delay).
- Key observation: the budget **replenishment period** can be **arbitrarily small** (a few clock cycles) **without particular penalties**
  - Differently from software reservation servers!
- **Low** FPGA **resource consumption**.
  - Described in VHDL: **ABUs** can be integrated into any design.

Resource type	4 ABUs consumption
LUT	2023 / 53200 **(3.80 %)**
FF	2045 / 106400 **(1.92 %)**
DSP	0 / 140 **(0 %)**
BRAM	0 / 220 **(0 %)**

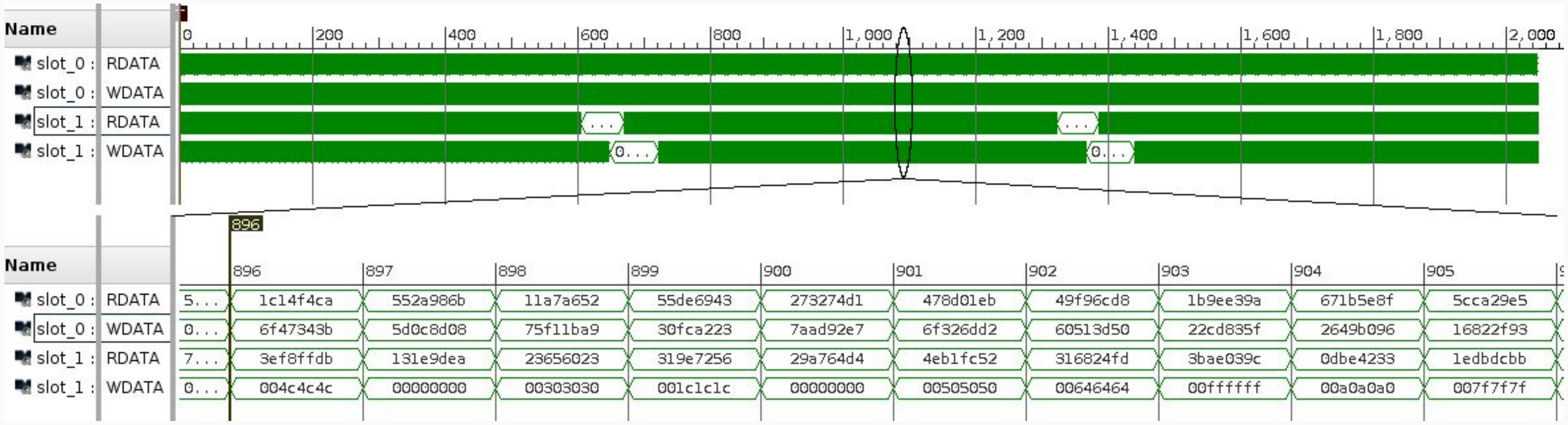
Resource for four ABUs consumption on the Zynq-7020.

Bandwidth-driven analysis

FPGA dataflow accelerators

Real-world HW-tasks are very different from software tasks!

- **Hardware activities** implemented using **programmable logic**;
- Internal **control logic** is typically based on **state machines;**
- **Clock-level regular and predictable** patterns of **BUS transactions.**



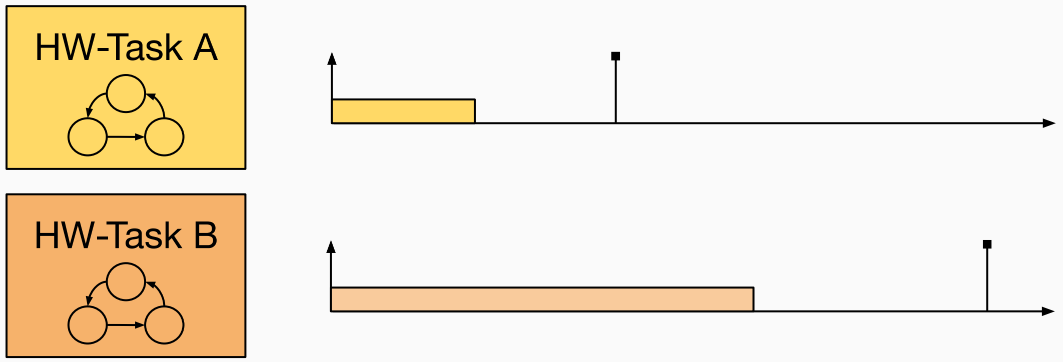
Name			896	897	898	899	900	901	902	903	904	905
slot_0 :	RDATA	5...	1c14f4ca	552a986b	11a7a652	55de6943	273274d1	478d01eb	49f96cd8	1b9ee39a	671b5e8f	5cca29e5
slot_0 :	WDATA	0...	6f47343b	5d0c8d08	75f11ba9	30fca223	7aad92e7	6f326dd2	60513d50	22cd835f	2649b096	16822f93
slot_1 :	RDATA	7...	3ef8ffdb	131e9dea	23656023	319e7256	29a764d4	4eb1fc52	316824fd	3bae039c	0dbe4233	1edbdcbb
slot_1 :	WDATA	0...	004c4c4c	00000000	00303030	001c1c1c	00000000	00505050	00646464	00ffffff	00a0a0a0	007f7f7f

Execution trace of a **FIR** and **Sobel HW-tasks** on the Zynq-7020
(Screenshot from Xilinx Vivado 2017.4)

Bandwidth-driven analysis

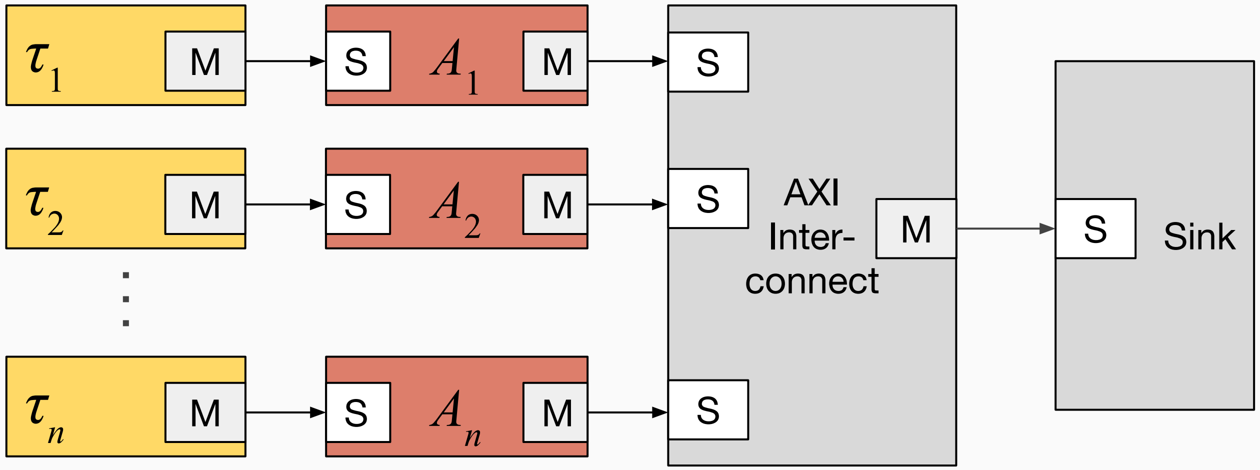
Analyzing the BUS contention experienced by HW-tasks is simpler than analyzing the memory contention experienced by SW-tasks on multicores

- **HW-Tasks** can be treated as **parallel computational activities** that **fluidly contend** the bandwidth supplied by the **AXI BUS**.
- **AXI sink modules** on FPGAs typically return **transactions in order**.
  - On many SoC FPGA platforms each port of the **FPGA-PS** interface return AXI transactions **in order**;
  - On-fabric **AXI BRAMs** memories return transactions **in order**.



Model of an AXI system

- A set of n periodic **HW-Tasks** $\Gamma = \{\tau_1, \ldots, \tau_n\}$.
  - τ_i is characterized by: a bandwidth **demand** D_i ;
  - A number of per-job **transactions** N_i ;
  - and a **period** T_i ;
- A set of n **ABUs** $A = \{A_1, \ldots, A_n\}$;
  - A_i is characterized by a **budget** B_i and a **period** P_i
- A slave sink S;
  - Characterized by a **supply** S (E.g., *FPGA-PS* ports or *BRAM mem*).



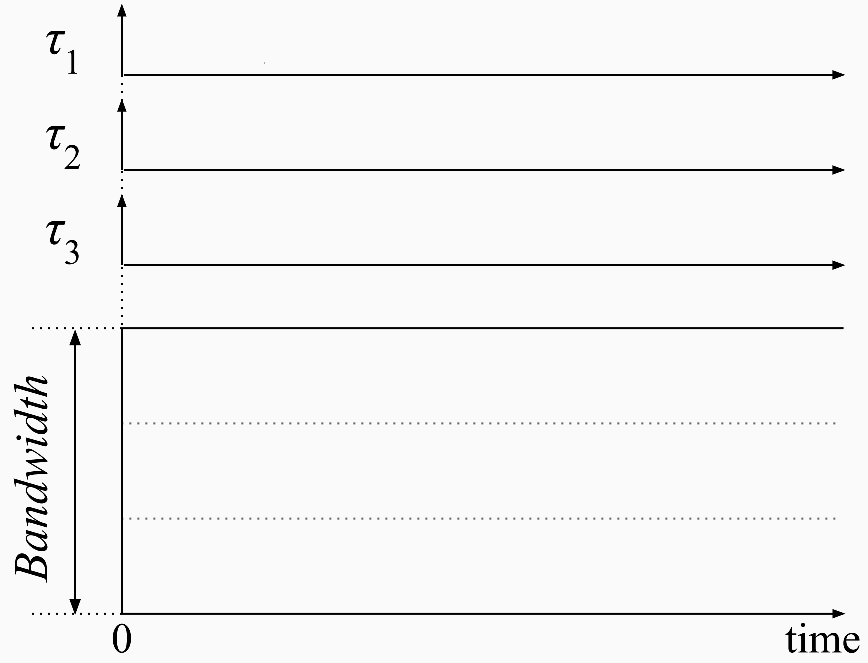
Bandwidth-driven response-time analysis

- The **BUS bandwidth** is “elastically” **shared** between **HW-Tasks**;

Bandwidth-driven response-time analysis

- The **BUS bandwidth** is “elastically” **shared** between **HW-Tasks**;
- An example without the **ABUs**:

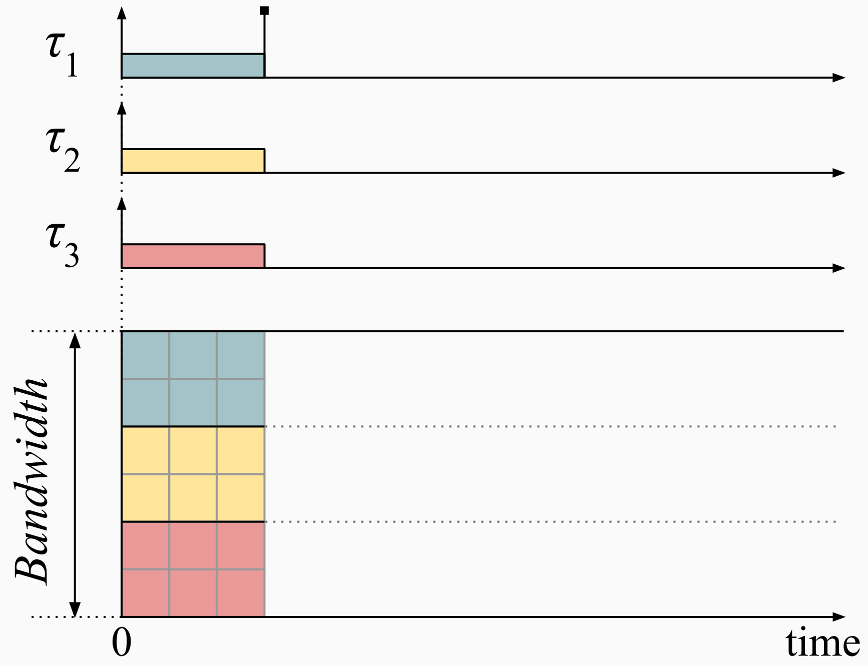
	D_i / S	N_i
τ_1	4 / 6	6
τ_2	4 / 6	18
τ_3	4 / 6	34



Bandwidth-driven response-time analysis

- The **BUS bandwidth** is “elastically” **shared** between **HW-Tasks**;
- An example without the **ABUs**:

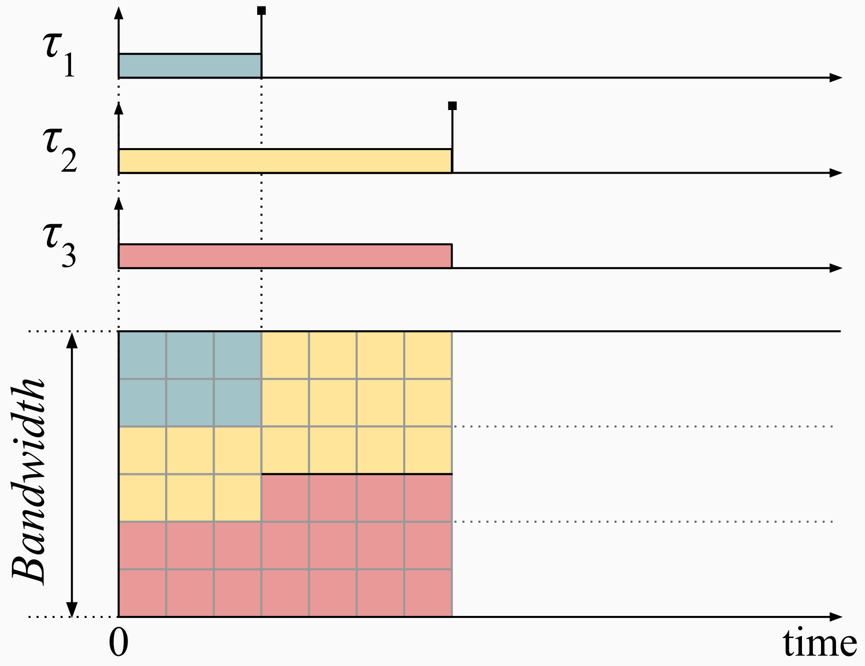
	D_i / S	N_i
τ_1	4 / 6	6
τ_2	4 / 6	18
τ_3	4 / 6	34



Bandwidth-driven response-time analysis

- The **BUS bandwidth** is “elastically” **shared** between **HW-Tasks**;
- An example without the **ABUs**:
  - When a **HW-Task** terminates, the spare bandwidth can be “reclaimed” by other **HW-Tasks**;

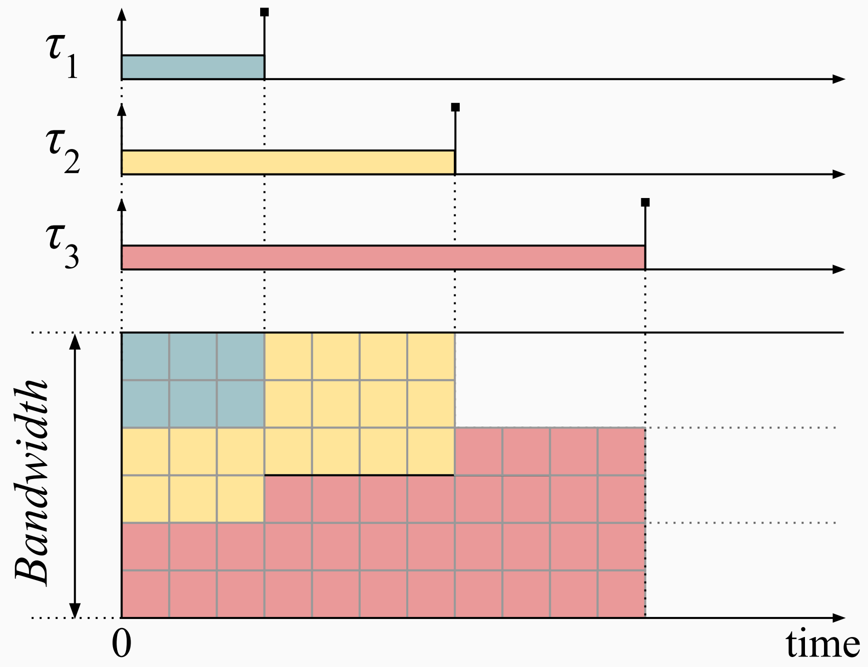
	D_i / S	N_i
τ_1	4 / 6	6
τ_2	4 / 6	18
τ_3	4 / 6	34



Bandwidth-driven response-time analysis

- The **BUS bandwidth** is “elastically” **shared** between **HW-Tasks**;
- An example without the **ABUs**:
  - When a **HW-Task** terminates, the spare bandwidth can be “reclaimed” by other **HW-Tasks**;
  - **HW-tasks** may be **unable** to fully utilize the bus bandwidth when they reach their maximum bandwidth demand.

	D_i / S	N_i
τ_1	4 / 6	6
τ_2	4 / 6	18
τ_3	4 / 6	34



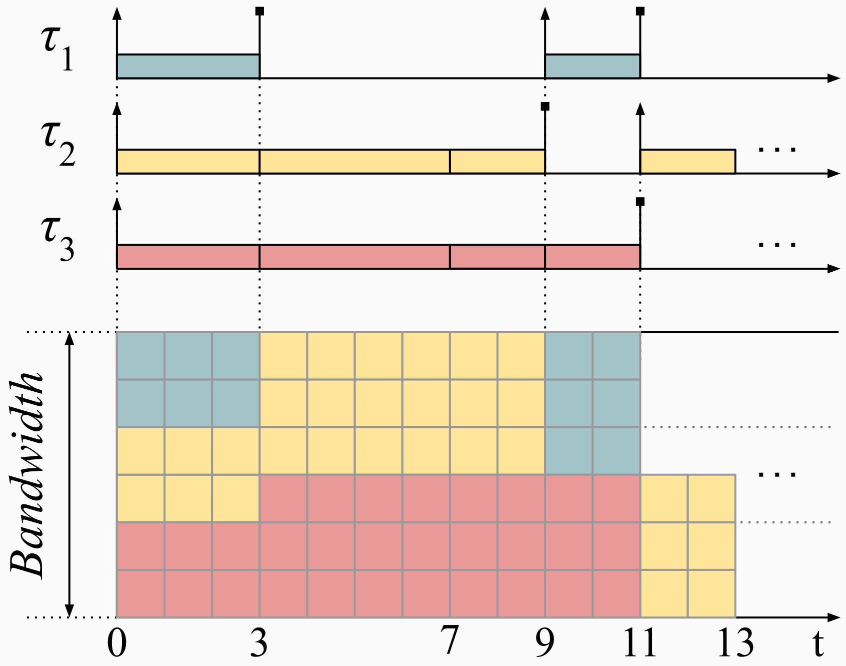
Analysis issues

- A **Bandwidth-driven** response-time **analysis** for **HW-tasks cannot be** accomplished by leveraging **classical techniques** used for periodic real-time tasks;
    - **Critical instant** of a **HW-task may not occur** when it is **synchronously** released together with all other HW-tasks.

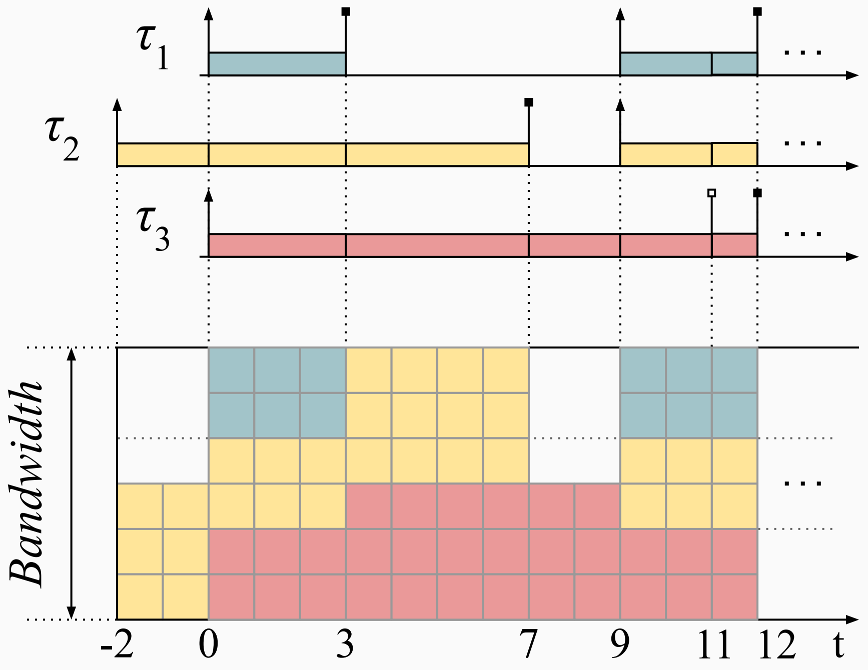
Analysis issues: critical instant

- Consider the following example:

	D_i	N_i	T_i
τ_1	3 / 6	6	9
τ_2	3 / 6	24	11
τ_3	3 / 6	30	15



Synchronous release.



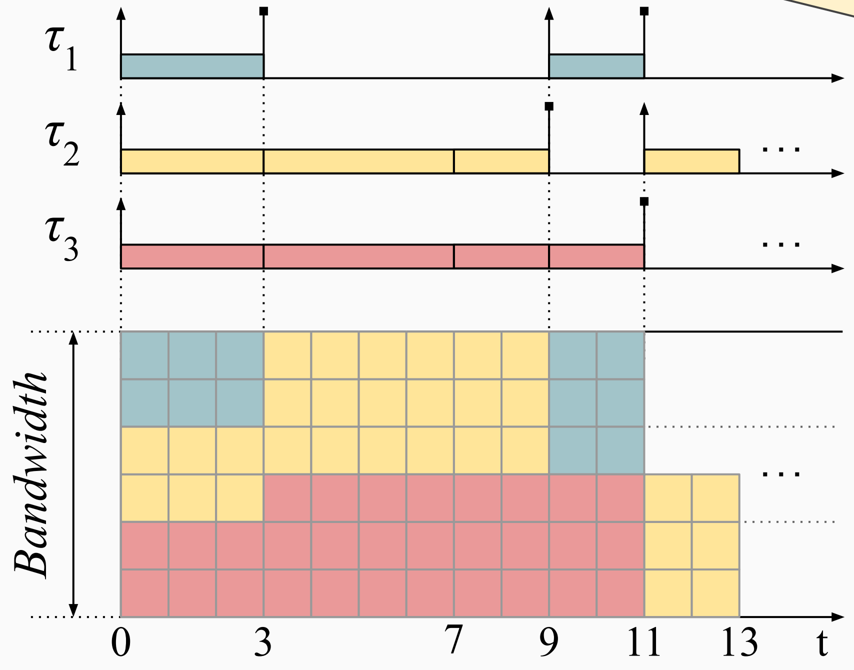
τ_2 is **released** 2 time units **earlier**.

Analysis issues: critical instant

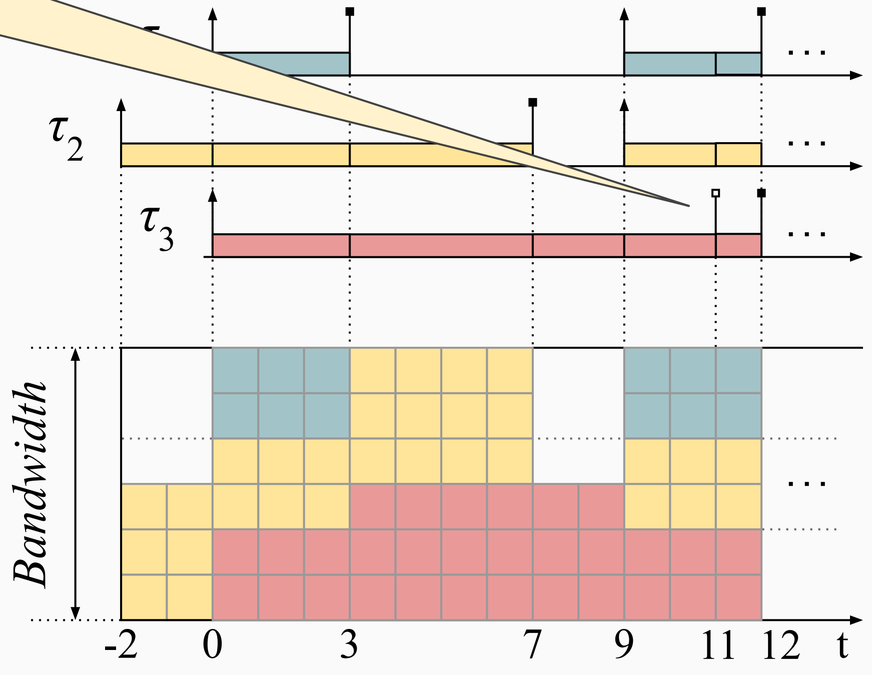
- Consider the following example:

	D_i	N_i	T_i
τ_1	3 / 6	6	9
τ_2	3 / 6	24	11
τ_3	3 / 6	30	15

if τ_2 is **released** 2 time units **earlier** τ_3 response time increases!



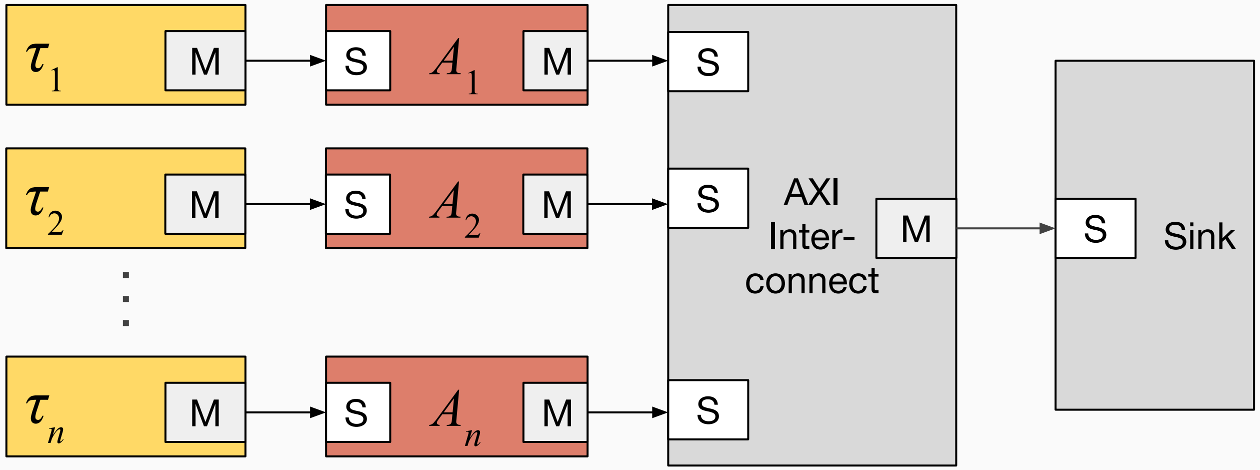
Synchronous release.



τ_2 is **released** 2 time units **earlier**.

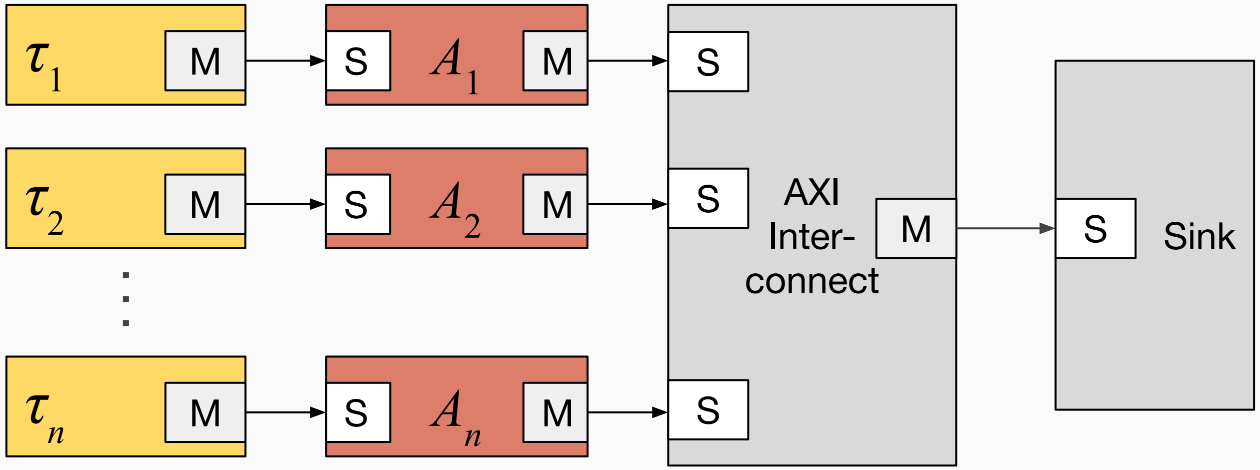
Bandwidth-driven response-time analysis

- The **ABUs** can be **leveraged** to improve predictability and help bounding **HW-tasks' response times**.
  - Under the assumption that $P_i \ll min\ \{T_i\}$, **ABUs** act as **fluid bandwidth regulators**.
    - e.g., 128 FPGA clock cycles (1.28 μs) vs 10 milliseconds;



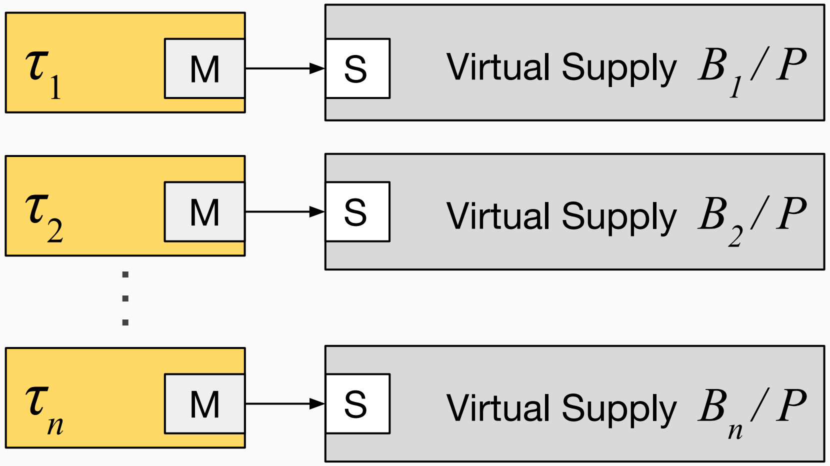
Bandwidth-driven response-time analysis

- The **ABUs** can be **leveraged** to improve predictability and help bounding **HW-tasks' response times**.
  - Under the assumption that $P_i \ll min\ \{T_i\}$, **ABUs** act as **fluid bandwidth regulators**.
    - e.g., 128 FPGA clock cycles (1.28 µs) vs 10 milliseconds;
    - As the **ABUs** do not introduce extra latencies, there is no particular penalty in using a small period P_i;
  - All **ABUs** are synchronized (same clock).



Bandwidth-driven response-time analysis

- As long as the **ABU budgets** are **guaranteed**:
  - Each **ABU** offers to the corresponding **HW-Task** τ_1 a **virtual bandwidth supply** of B_i / P_i **irrespectively** of the behaviour of the other **HW-Tasks**.
  - As the **ABU periods** can be small, there's no relevant benefit in selecting heterogeneous periods, hence $P_i = P$



Bandwidth-driven response-time analysis

- Hence, the problem of analyzing a set of **HW-Tasks** supervised by **ABUs** can be decomposed in **two steps**:

Bandwidth-driven response-time analysis

- Hence, the problem of analyzing a set of **HW-Tasks** supervised by **ABUs** can be decomposed in **two steps**:

1 **Assign** to each **HW-task** τ_i the **minimum budget** B_i to complete within its deadline.

- Easy, for each τ_i assign B_i such that:

$$B_i = \frac{N_i \cdot P}{T_i}$$

Bandwidth-driven response-time analysis

- Hence, the problem of analyzing a set of **HW-Tasks** supervised by **ABUs** can be decomposed in **two steps**:

1 **Assign** to each **HW-task** τ_i the **minimum budget** B_i to complete within its deadline.

- Easy, for each τ_i assign B_i such that:

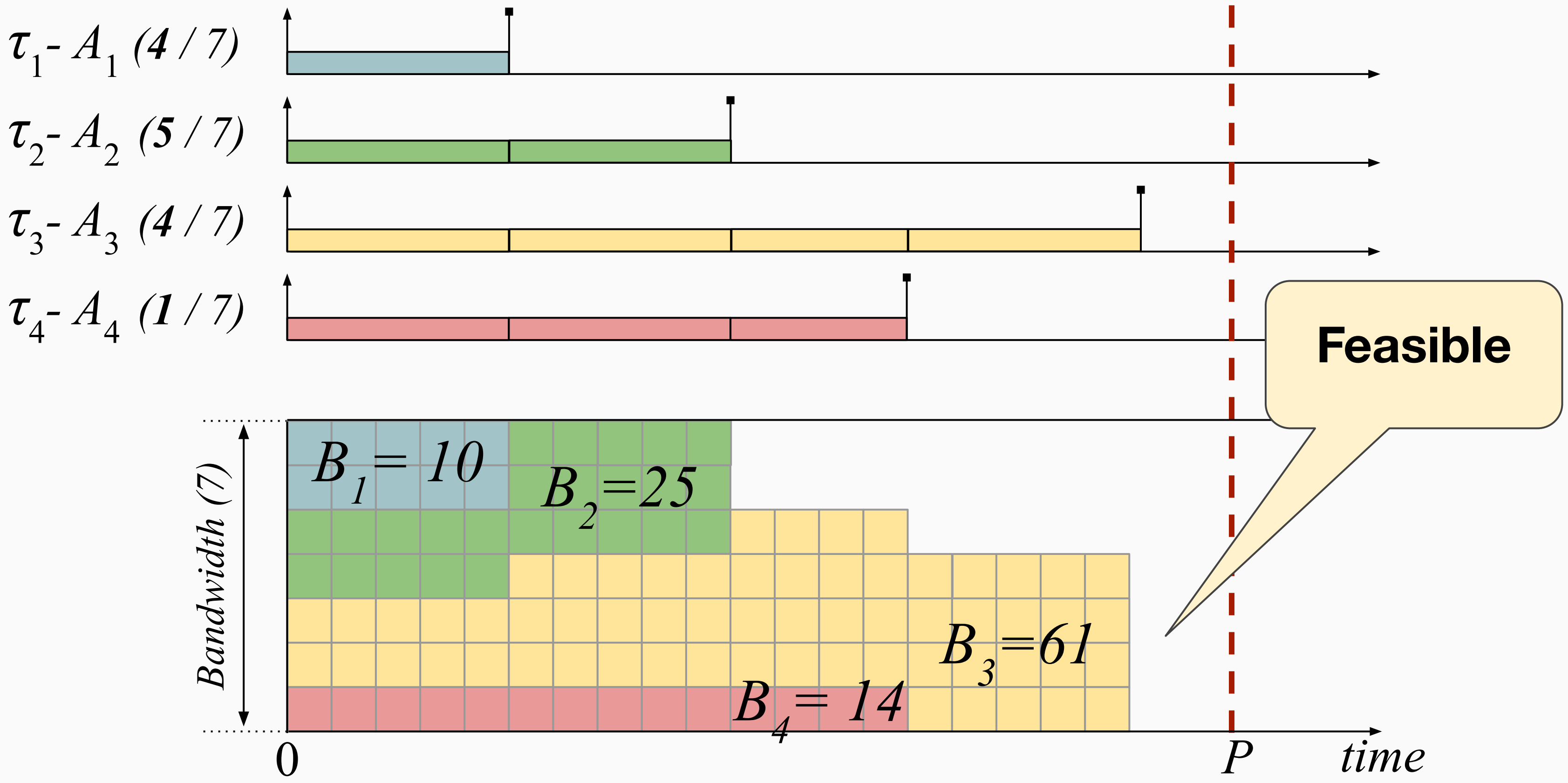
$$B_i = \frac{N_i \cdot P}{T_i}$$

2 **Check** that system (Interconnect and Sink) can **provide** to each **HW-Task's enough bandwidth** to exhaust all **ABUs** budgets $\{ B_i \}$ within the period P.

- It's necessary to perform a **bandwidth-driven analysis** within the scheduling window of one **ABU** period $[0, P]$.
- **Schedulability** can be tested using an **iterative procedure**.

Analysis with ABUs

- The **schedulability** test procedure “unrolls” the **execution** of the **HW-Tasks** just within one **ABU** period.
  - Check if the Sink can provide to each **HW-Task’s enough bandwidth** to exhaust all **ABUs** budgets $\{ B_i \}$ within the period P.



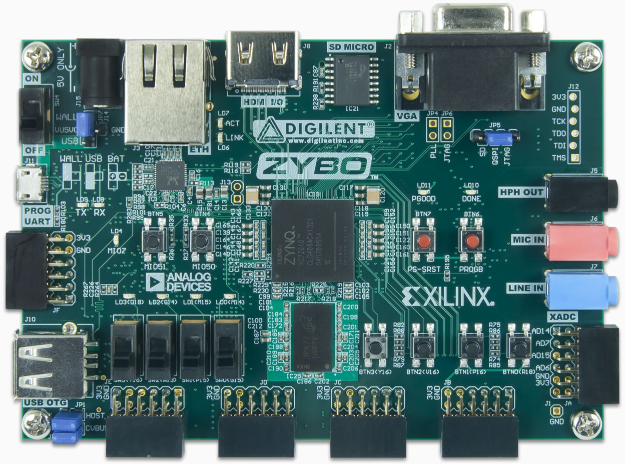
Experimental results

Experimental evaluation

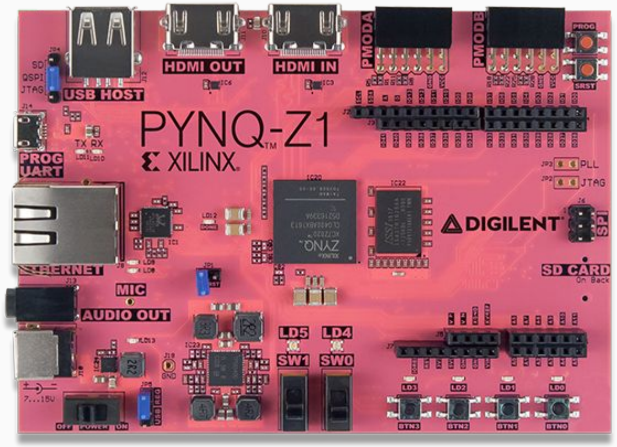
- The **ABU** has been **experimentally evaluated** on **real platforms** such as the **Zynq-7020** (and **Zynq-7010**) using realistic workload.
    - HW-Tasks from **Xilinx IP library**, HLS-generated workload, etc.
- **Objectives** of the experimental evaluation:

    1) Show that the **ABU works** on a **real hardware**;

    2) Show that the proposed **analysis** is experimentally **tight**.



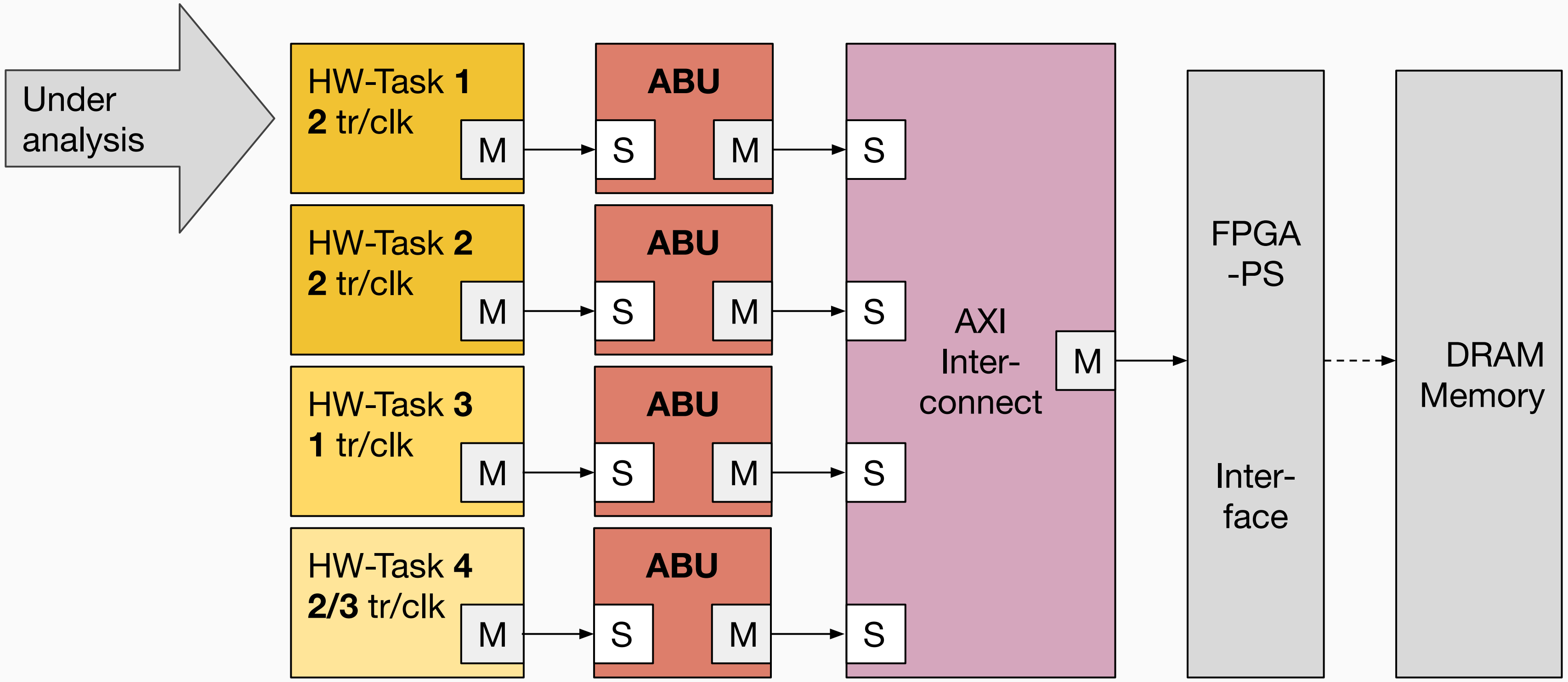
ZYBO board (Zynq-7010)



PYNQ board (Zynq-7020)

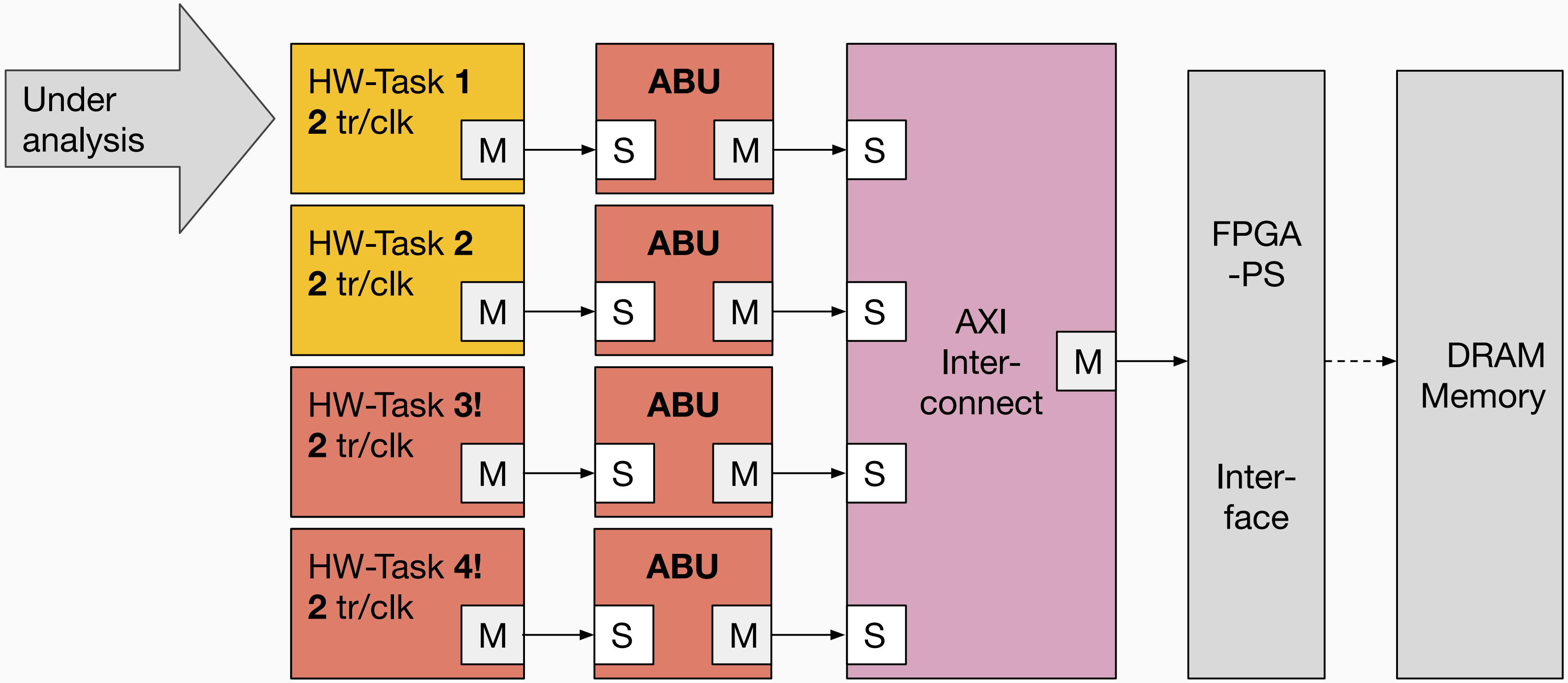
Experimental evaluation: bandwidth reservation

- Goal: test the effectiveness of the reservation mechanism.
- Setup: **four** DMA-like **HW-tasks** with **different demand** rates:



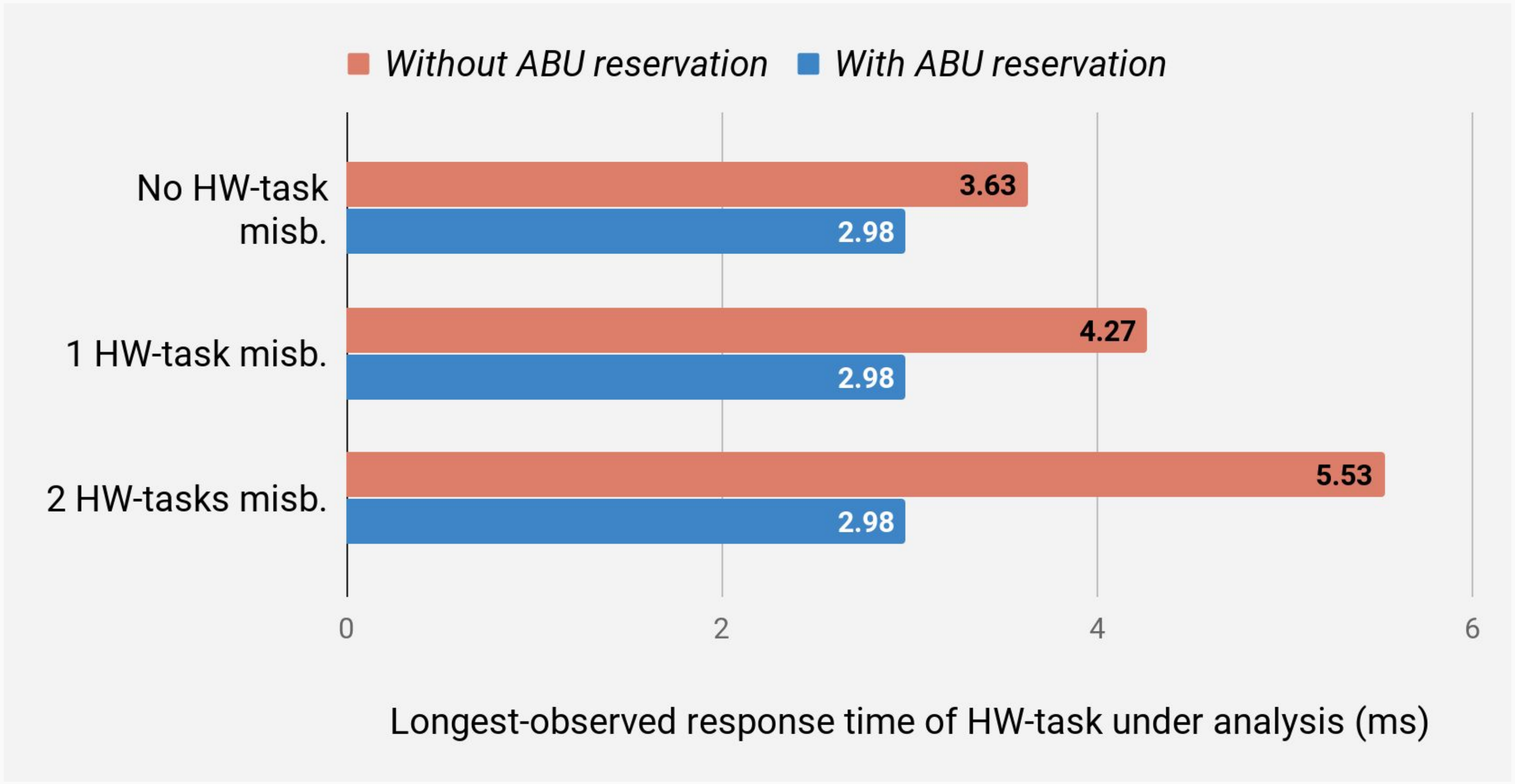
Experimental evaluation: bandwidth reservation

- Goal: test the effectiveness of the reservation mechanism.
- Setup: **four** DMA-like **HW-tasks** with **different demand** rates:
    - **Swap** one or more DMA-like **HW-tasks** with a more demanding version to **simulate** a **misbehaving HW-Task**.



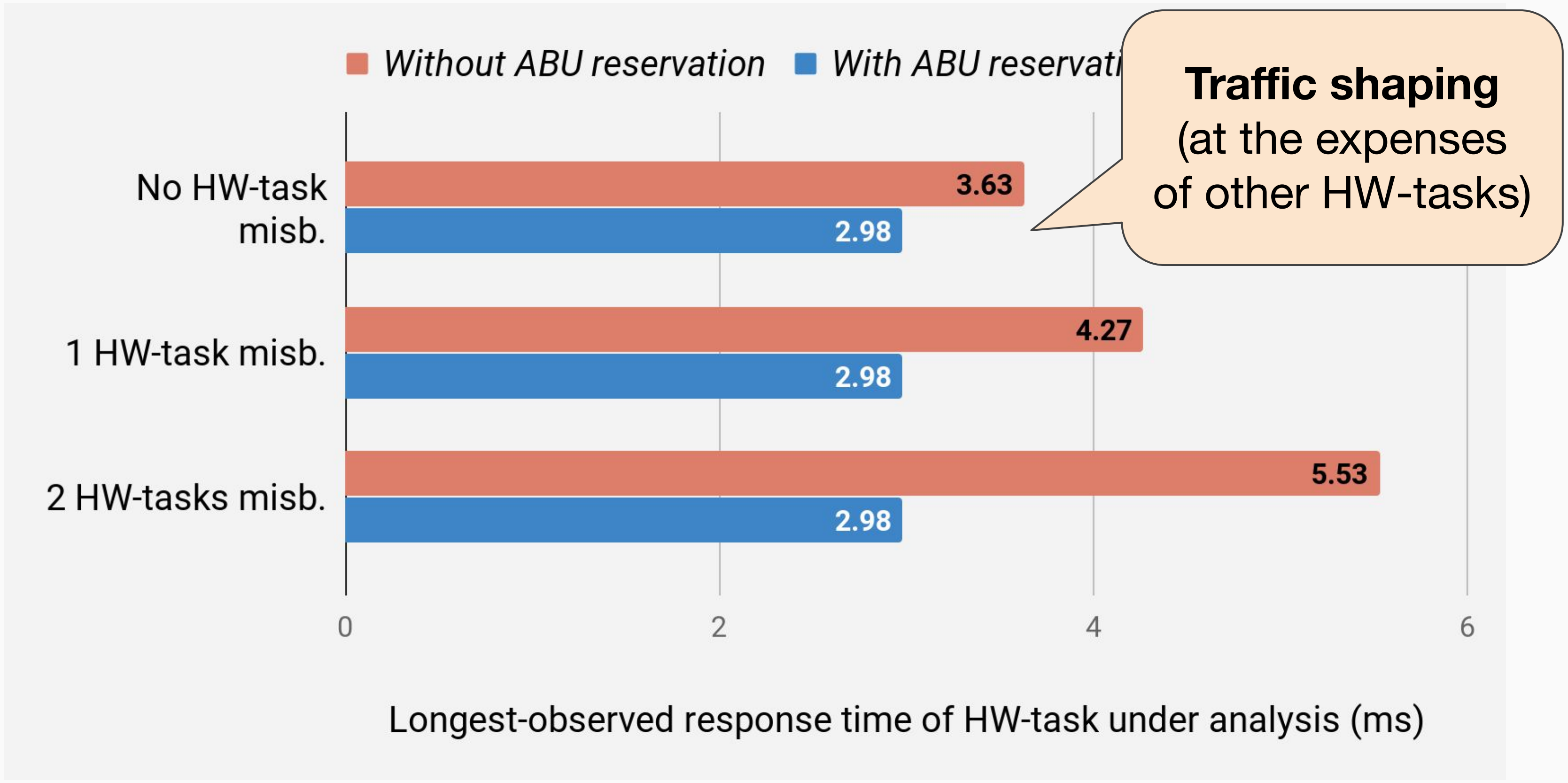
Experimental evaluation: bandwidth reservation

- Effect of the **misbehaving** HW-tasks on the HW-task under analysis.



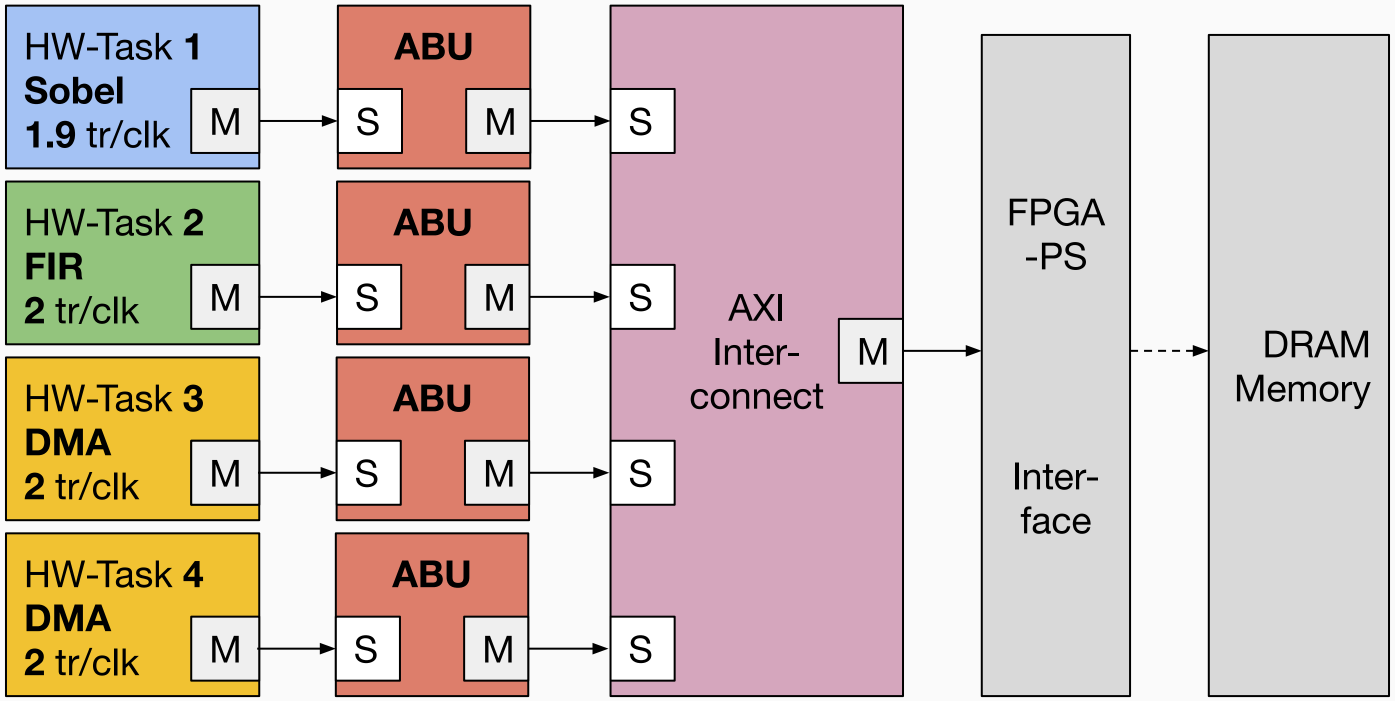
Experimental evaluation: bandwidth reservation

- Effect of the **misbehaving** HW-tasks on the HW-task under analysis.



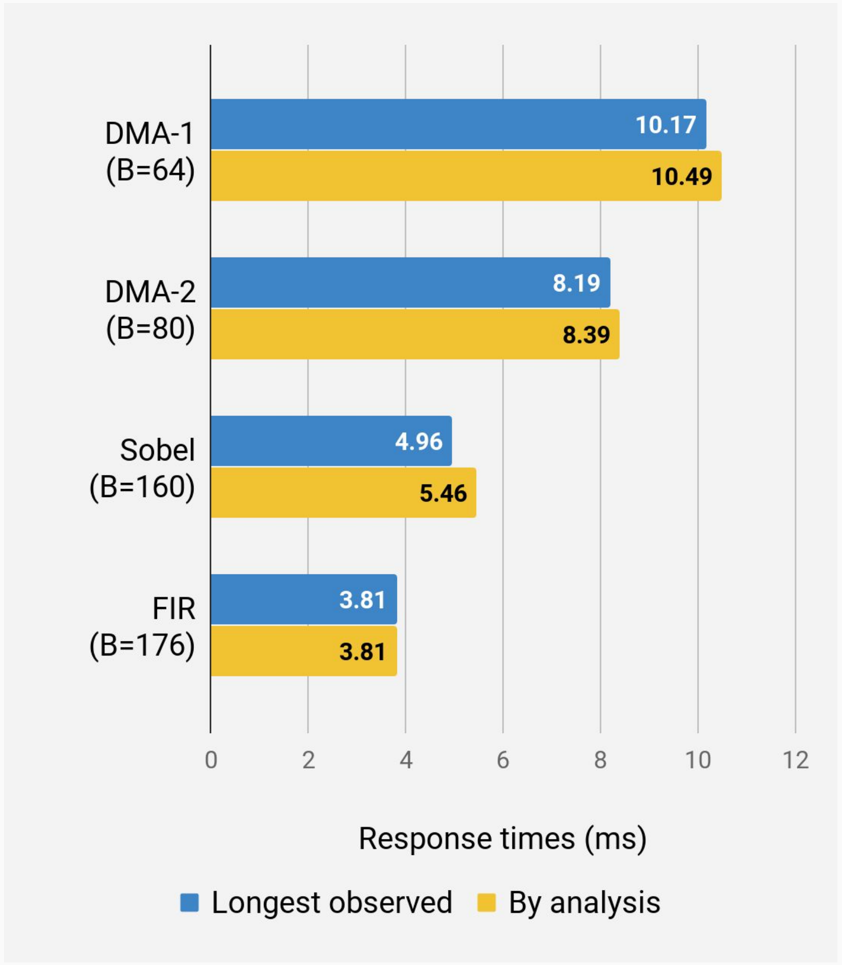
Experimental evaluation: case study

- Goal: **test** the **analysis** using realistic HW-tasks with **different budget configurations**.
- Setup: one **FIR** (filter) HW-task, one **Sobel** (image filter) HW-task, and 2 **DMA-like** HW-tasks.

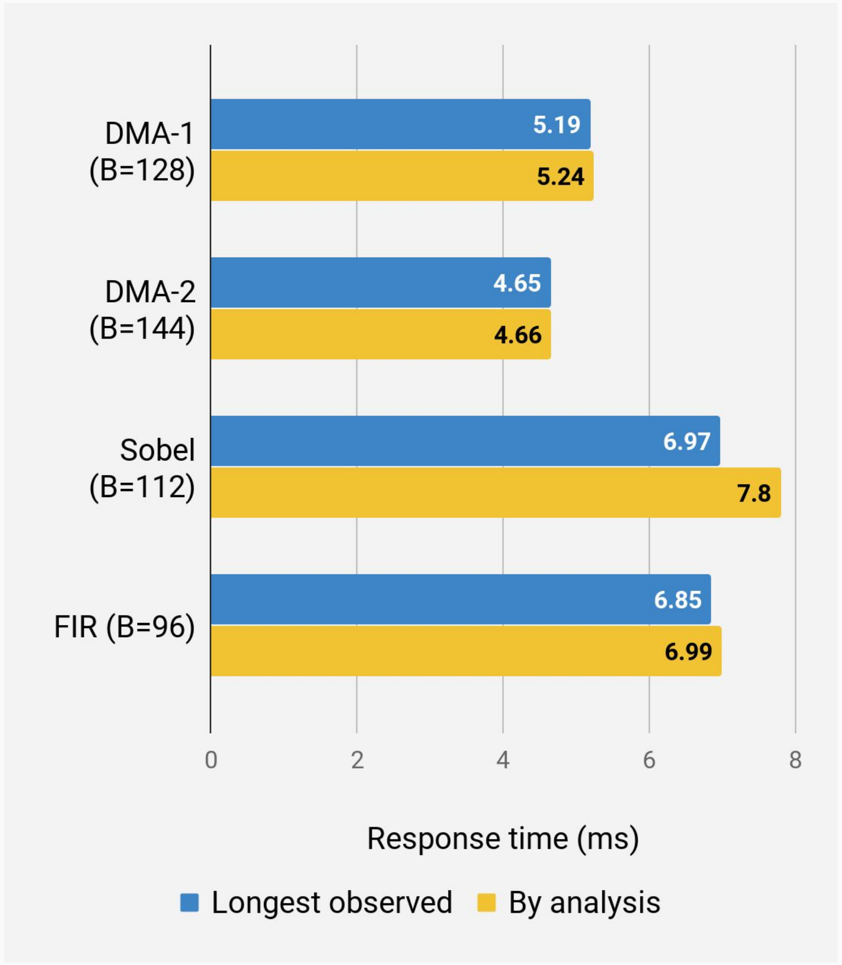


Experimental evaluation: case study

Configuration 1:
more bandwidth to **Sobel** and **FIR**.



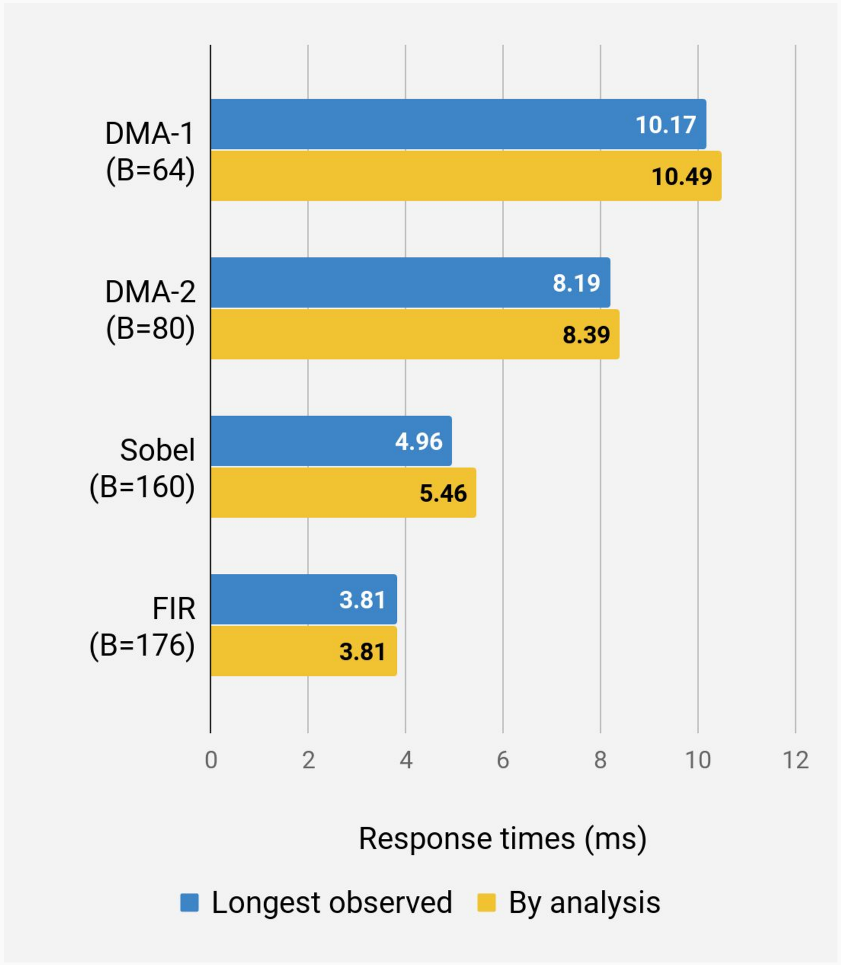
Configuration 2:
more bandwidth to **DMA-1** and **DMA-2**.



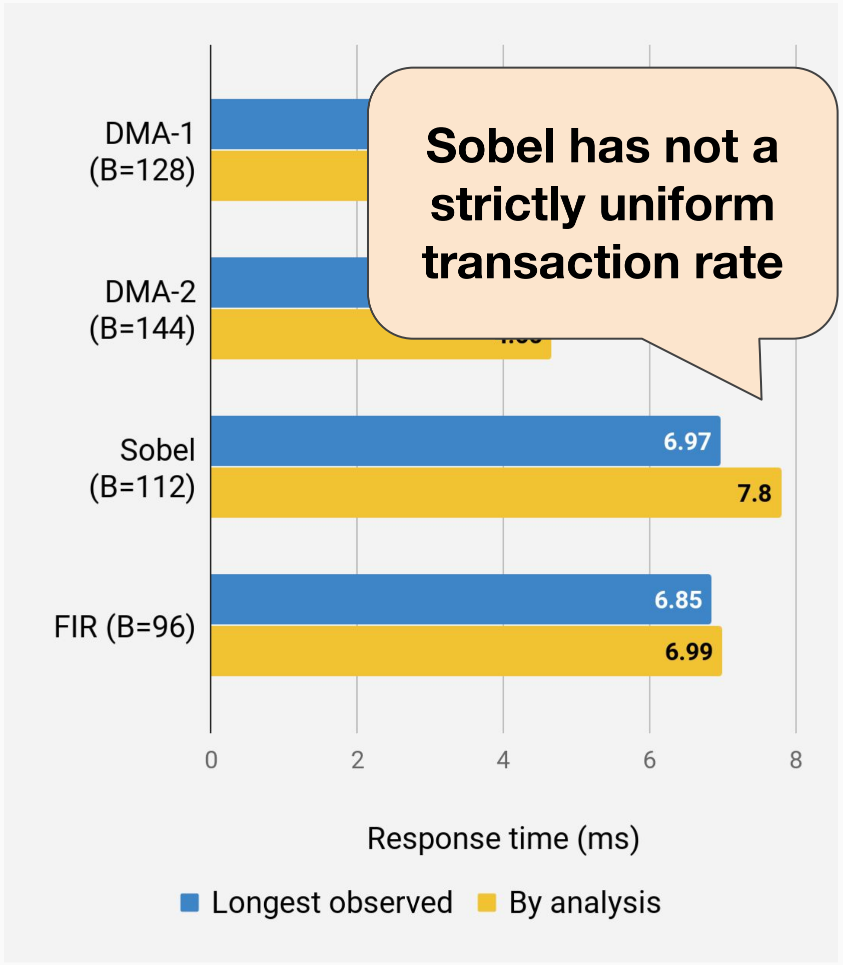
More budget configurations in the paper...

Experimental evaluation: case study

Configuration **1**:
more bandwidth to **Sobel** and **FIR**.



Configuration **2**:
more bandwidth to **DMA-1** and **DMA-2**.

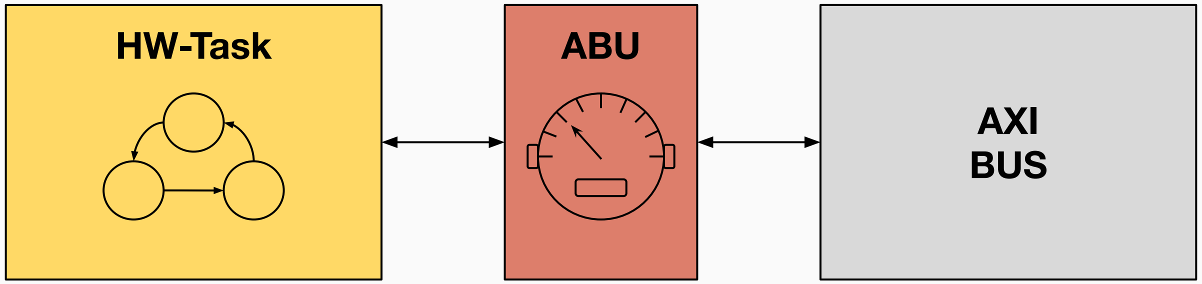


Sobel has not a strictly uniform transaction rate

More budget configurations in the paper...

Conclusions

- The **ABU** is a **hardware-based reservation mechanism** for the **AMBA AXI bus** aimed at **isolating hardware accelerators** implemented on FPGAs.
- **Leveraging the ABU**, a set of **HW-tasks** can be **analyzed** using the proposed response-time in the bandwidth domain.
- The **ABU** has been **implemented and validated** on the Xilinx **Zynq-7020** and **Zynq-7010** platforms to demonstrate its **practical applicability**.



Thank you for your attention

marco.pagani@sssup.it