Zero-Jitter Task Chains via Algebraic Rings

Enrico Bini, Paolo Pazzaglia, Martina Maggio
IEEE Transactions on Computers 2023

University of Turin

March 5th, 2024, Technical University Munich

Outline

1. Motivation
2. Model of tasks
3. Chains of LET tasks
4. Zero-jitter chains of LET tasks

Motivation

- Communication through **shared memory** is very popular in automotive applications
  - very efficient (just a `MOV`/`LOAD` instruction)
- When the communicated data is samples of a physical process, it is normally **unbuffered**
  - fresher values just overwrites old values
- The read/write instants are affected by the task schedule
  - *Logical Execution Time (LET)* mitigates this aspect: read/writes instants of each job are at pre-determined instants
- Still **chains** of LET tasks may lose the regular timing of LET tasks
- The presented research is about guaranteeing a regular timing (**zero jitter**) of chains of LET tasks

Outline

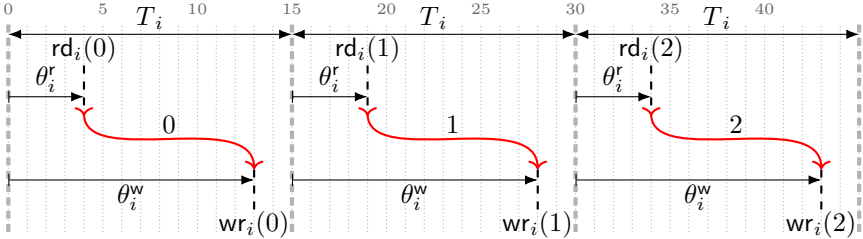
1 Motivation

2 Model of tasks

3 Chains of LET tasks

4 Zero-jitter chains of LET tasks

Model of a LET task τ_i



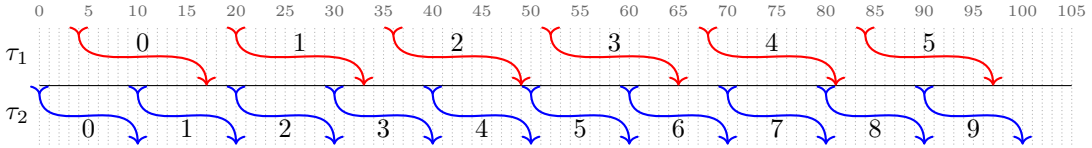
- A LET *task* τ_i is composed by *periodic jobs* (curvy arrows) indexed by $0, 1, 2, \ldots$
  - in LET, we only need the read and write instants

θ_i^{r}	*read phasing* of τ_i, relative to the *period* T_i (aka the *offset*)
θ_i^{w}	*write phasing* of τ_i, rel. to *period* T_i ($\theta_i^{\mathsf{w}} - \theta_i^{\mathsf{r}}$ is aka the *deadline*)
$\mathsf{rd}_i(j) = j\,T_i + \theta_i^{\mathsf{r}}$	*read instant* of job j of τ_i
$\mathsf{wr}_i(j) = j\,T_i + \theta_i^{\mathsf{w}}$	*write instant* of job j of τ_i

Outline

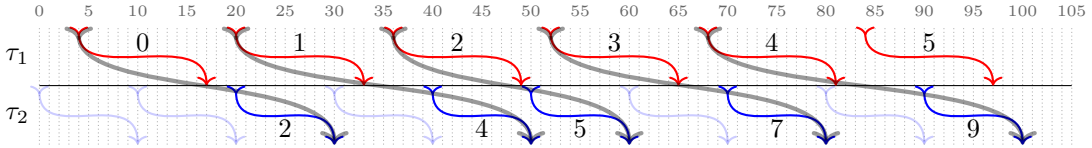
1. Motivation
2. Model of tasks
3. Chains of LET tasks
4. Zero-jitter chains of LET tasks

Chain of 2 LET tasks



- Chain of LET tasks
  - each task reads data written by the previous one
  - data is on **shared memory**
    - τ_1 may over-write data before τ_2 reads (if $T_1 < T_2$)
    - τ_2 may read again the same data (if $T_1 > T_2$)

Chain of 2 LET tasks



- Chain of LET tasks
  - each task reads data written by the previous one
  - data is on **shared memory**
    - ⋆ τ_1 may over-write data before τ_2 reads (if $T_1 < T_2$)
    - ⋆ τ_2 may read again the same data (if $T_1 > T_2$)
- Chains of jobs have **variable** read-to-write delay

			rd→wr delay
(j_1, j_2)	$\mathsf{rd}_1(j_1)$	$\mathsf{wr}_2(j_2)$	$\mathsf{wr}_2(j_2) - \mathsf{rd}_1(j_1)$
$(0, 2)$	4	30	$26 = 13 + \mathbf{3} + 10$
$(1, 4)$	20	50	$30 = 13 + \mathbf{7} + 10$
$(2, 5)$	36	60	$24 = 13 + \mathbf{1} + 10$
$(3, 7)$	52	80	$28 = 13 + \mathbf{5} + 10$
$(4, 9)$	68	100	$32 = 13 + \mathbf{9} + 10$
...	...	...	...

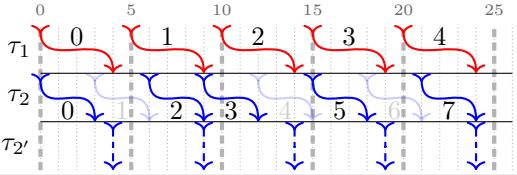
Outline

1. Motivation
2. Model of tasks
3. Chains of LET tasks
4. Zero-jitter chains of LET tasks

Making a zero jitter chain of 2 LET tasks: add a copier task

- if $T_1 \geqslant T_2$ then we add $\tau_{2'}$ after τ_2

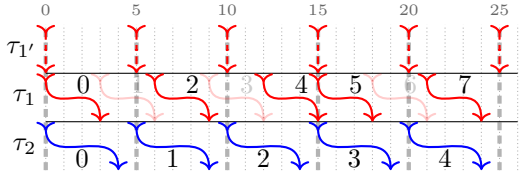
	T_i	θ_i^{r}	θ_i^{w}
τ_1	5	0	4
τ_2	3	0	3
$\tau_{2'}$	5	9	9



(j_1, j_2)	$\mathsf{wr}_2(j_2) - \mathsf{rd}_1(j_1)$	$\mathsf{wr}_{2'}(j_{2'}) - \mathsf{rd}_1(j_1)$
$(0, 2)$	9	9
$(1, 3)$	7	9
$(2, 5)$	8	9
$(3, 7)$	9	9

- if $T_1 \leqslant T_2$ then we add $\tau_{1'}$ before τ_1

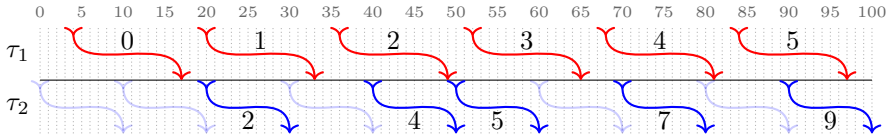
	T_i	θ_i^{r}	θ_i^{w}
$\tau_{1'}$	5	-5	-5
τ_1	3	0	3
τ_2	5	0	4



(j_1, j_2)	$\mathsf{wr}_2(j_2) - \mathsf{rd}_1(j_1)$	$\mathsf{wr}_2(j_2) - \mathsf{rd}_{1'}(j'_1)$
$(0, 1)$	9	9
$(2, 2)$	8	9
$(4, 3)$	7	9
$(5, 4)$	9	9

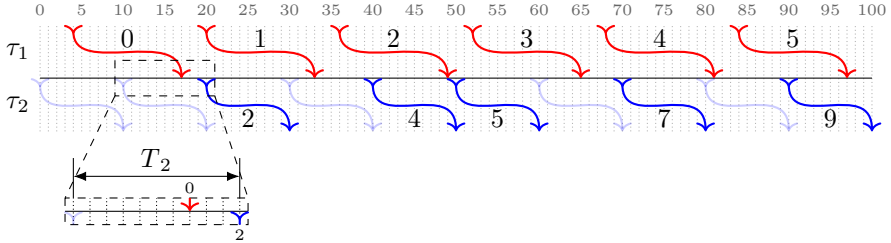
Variability of the input-output delay

- The source of variability is $\mathsf{rd}_2(j_2) - \mathsf{wr}_1(j_1)$. In the example: $3, 7, 1, 5, 9, 3, 7, \ldots$



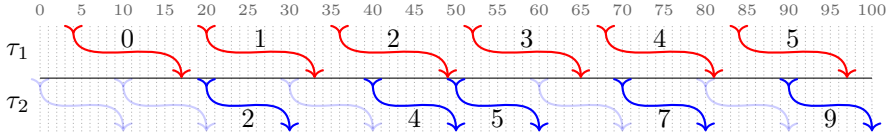
Variability of the input-output delay

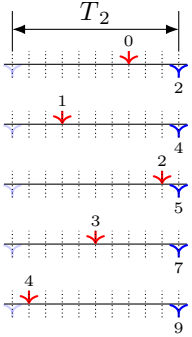
- $\mathsf{rd}_2(j_2) - \mathsf{wr}_1(j_1) < T_2$ ($= 10$ in the example), always. Let's zoom in ...



Variability of the input-output delay

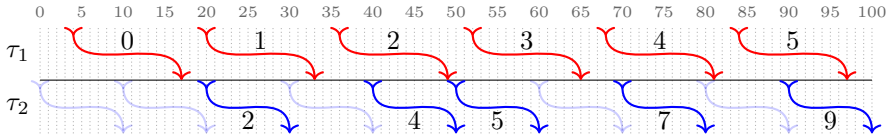
- ... and take $\mathrm{rd}_2(j_2)$ as reference for all $\mathrm{wr}_1(j_1) \to \mathrm{rd}_2(j_2)$ delays.

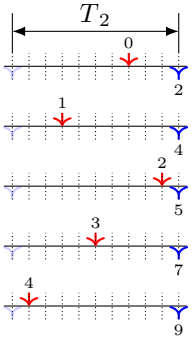


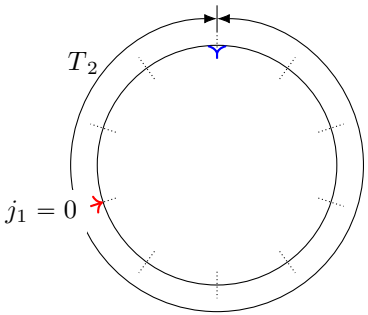


Variability of the input-output delay

- A repetitive requence: let's wrap all the T_2-long segments over the (algebraic) ring $\mathbb{Z}/T_2\mathbb{Z}$.

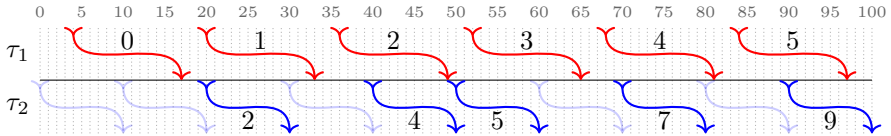


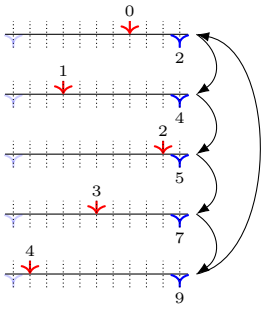


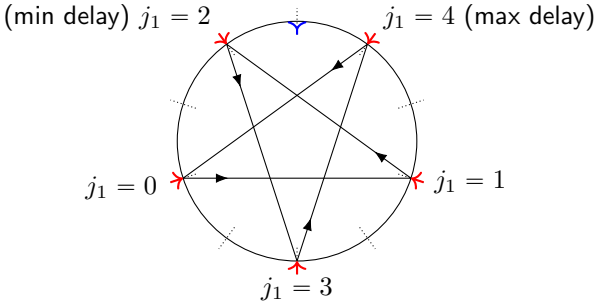


Variability of the input-output delay

- By inverting over the ring $\mathbb{Z}/T_2\mathbb{Z}$, max/min delay is found **without** unrolling the schedule.

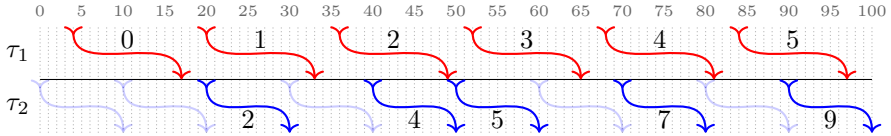


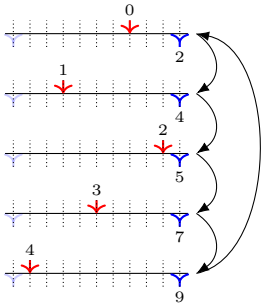


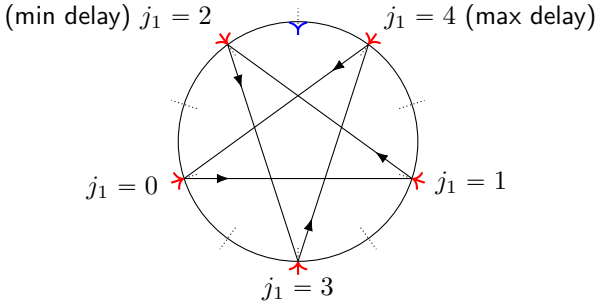


Variability of the input-output delay

- The parameters of the copier task follows from the maximum delay.







Conclusions

- The LET model allows a zero-jitter timing **for the tasks**
- Chains of LET tasks do **have non-zero jitter**
- By adding a copier, we can **eliminate the jitter**
- The parameters of the copier can be found with operations over an **algebraic ring**
- Published on IEEE Transactions of Computers, 2023
- Some Python code implements the whole machinery