Parallel Implementation of Iterative Learning Controllers on Multi-core Platforms

Mojtaba Haghi, Yusheng Yao, Dip Goswami, and Kees Goossens
Eindhoven University of Technology, the Netherlands
Email: {s.m.haghi, y.yao.1, d.goswami, k.g.w.goossens}@tue.nl

***Abstract*—This paper presents design and implementation techniques for iterative learning controllers (ILCs) targeting predictable multi-core embedded platforms. Implementation on embedded platforms results in a number of timing artifacts. Sensor-to-actuator delay (referred to as delay) is an important timing artifact which influences the control performance by changing the dynamic behavior of the system. We propose a delay-based design for ILCs that identifies and operates in the performance-optimal delay region. We then propose two implementation methods – sequential and parallel – for ILCs targeting the predictable multi-core platforms. The proposed methods enable the designer to carefully adjust the scheduling to achieve the optimal delay region in the resulting control system. We validate our results by the hardware-in-the-loop (HIL) simulation, considering a motion system as a case-study.**

***Index Terms*—Embedded control, Iterative learning control, Sensor-to-actuator-delay, Predictable multi-core platform.**

I. Introduction

High precision motion systems are an important part of industrial domains such as robotics [1], printing systems [2] and many more [3]. The purpose of such systems is an accurate and fast reference tracking [1], e.g., the trajectory for printer head. One of the state-of-the-art control structures for high precision control is iterative learning control (ILC) [4]. These controllers are the mature version of feedforward controllers with the emphasis of improving the control performance in presence of model uncertainty and repetitive tasks (periodic reference). In ILC, a feedback part stabilizes the closed loop system and a feedforward part finds a sequence of desired input signals over the period of the repetitive task, improving reference tracking performance [4].

Low-cost embedded implementation of motion systems are relevant in many industries [www.i-mech.eu]. Embedded platforms are cost and energy efficient implementation solutions [5]. These platforms perform dedicated computations as a part of a larger mechanical or electrical system [6]. However, the limitation of computation resources is the key constraint of using such platforms [7]. In control applications, this constraint may impose undesirable timing artifacts such as sensor-to-actuator delay and time-varying sampling period which degrade the control performance.

Among the possible existing platforms, composable and predictable platforms with no time-variation in execution time would be a suitable solution for control applications [7]. Using such platforms results cycle-accurate sampling period and execution times, which facilitate the control design by guaranteeing temporal assumptions such as uniform sampling period. Also, the jitter-free and deterministic execution times allows the designer to consider the delay as a variable, as well as accurately study the effect of temporal artifacts on the control performance [8]. In the absence of such platforms the designer may opt for robust controllers against uncertain delay and/or time-varying sampling period which degrades the control performance [9].

Sensor-to-actuator delay is one of the important timing implications of an embedded implementation. While it is considered in the design that the execution of a control loop is instantaneous, the implementation of a control loop takes a finite time from the start of sensing the plant output until actuating it with the new input value [10]. Considering such delay in the design step would improve the performance of feedback and feedforward controllers.

In this paper we propose a delay-aware design for ILCs which considers the sensor-to-actuator delay as a design parameter and studies its influence on control performance. Although, there is a mature literature in designing ILCs [4], their implementations on embedded platforms and considering the imposed timing constraints on the performance requires further studies. For ILC implementation on embedded platforms we start with a sequential implementation which offers an adjustable delay to be treated as a design parameter. We continue by proposing a novel parallel implementation which is a scalable and conservative solution for ILC implementation which can maximize the performance of the controller.

Our contributions:

1. Delay-based design of ILCs for motion systems, considering the delay as a design parameter and studying its influence on control performance.

2. Proposing two ILC implementation solutions on composable and predictable embedded platforms - sequential and parallel.

3. Parallel implementation of ILCs on multi-core embedded platforms, maximizing the control performance.

4. Performance evaluation and validation of the implemented controllers by performing hardware-in-the-loop (HIL) simulations.

The paper is organized as follows. Section II defines the motion systems under the effect of delay and the case-study motion system. Section III describes the ILC structure and demonstrates the effect of delay on ILC performance. Section IV demonstrates the embedded platform, and the sequen-

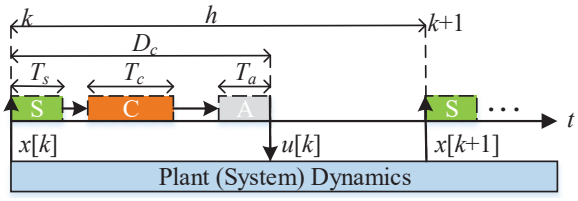

Fig. 1. Timing diagram of an embedded control system.

tial and parallel ILC implementations. Section V illustrates and discusses the HIL simulation results. The paper is concluded in Section VI with some future studies proposed.

II. Motion systems under the effect of delay

We consider a motion system which can be represented as linear time-invariant (LTI) systems. The continuous state-space of such a system is given by,

$$\begin{aligned} \dot{X}(t) &= AX(t) + BU(t), \\ Y(t) &= CX(t), \end{aligned} \tag{1}$$

where $X(t) \in R^n$ are the states of the system, $U(t)$ is the input, and $Y(t)$ represent the system output.

The purpose of a control loop is to actuate the system with the input signal $U(t)$ so its output follows a desired reference $r(t)$ accurately. A control loop performs three main operations of sensing, computation and actuation in a sequential and periodic manner. Sensing operation reads the corresponding sensor values of each state at equidistant time instances t_k. the sensed state are defined as:

$$x[k] := X(t_k), \quad k \in N_{\geq 1} \tag{2}$$

Based on the $x[k]$ and the control law, computation operation calculates the next control value $u[k]$. The control value is updated from $u[k-1]$ to $u[k]$ in the actuation. The interval between two sensing operations is the sampling period h.

The execution of operations and the communications between them requires a finite time. Fig. 1 illustrates the execution behavior of the control loop in a sampling interval . The delay from the start of the sensing until the end of the actuation, imposed by implementation, is sensor-to-actuator delay D_c. In this paper, we consider the case where D_c is shorter than the sampling period h, which means: $0 \leq D_c < h$.

A. Delay modeling

The discrete-time representation of the system in (1), by considering h and D_c is [10],

$$\begin{aligned} x[k+1] &= \phi x[k] + \Gamma_0 u[k] + \Gamma_1 u[k-1], \\ y[k] &= Cx[k], \end{aligned} \tag{3}$$

where $\phi = e^{Ah}$, and,

$$\Gamma_0 = \int_0^{h-D_c} e^{As} B ds, \quad \Gamma_1 = \int_{h-D_c}^{h} e^{As} B ds.$$

Referring to (3), in each sampling interval, the system is actuated by two control values of $u[k-1]$ before the actuation and $u[k]$ after the actuation. This is caused by D_c and impacts the system output and consecutively the performance. The effect of D_c on control performance is studied through the effect of the delay on zeros of the system.

B. Zero polynomial and zero loci

Zeros are the roots of zeros polynomial which is the nominator of the systems transfer function in z frequency domain. To derive zeros polynomial, we define augmented stated space [11] by defining augmented states as $\xi[k] = \left[x^T[k], u^T[k-1]\right]^T$,

$$\begin{aligned} \xi[k+1] &= \phi_{aug}\xi[k] + \Gamma_{aug}u[k], \\ y[k] &= C_{aug}\xi[k], \end{aligned} \tag{4}$$

where,

$$\phi_{aug} = \begin{bmatrix} e^{Ah} & \Gamma_1 \\ 0 & 0 \end{bmatrix}, \quad \Gamma_{aug} = \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix}, \quad C_{aug} = [C \quad 0].$$

The transfer function of the system is derived from the state-space (4) as [12]:

$$\frac{y[k]}{u[k]} = G(z) = \frac{\Delta(z)}{P(z)} = C_{aug}(zI_n - \phi_{aug})^{-1}\Gamma_{aug}, \tag{5}$$

where $G(z)$ is the transfer function of the system, $\Delta(z)$ is the zero polynomial and $P(z)$ is the characteristic polynomial. The zeros polynomial results from:

$$\Delta(z) = C_{aug}adj(zI_n - \phi_{aug})\Gamma_{aug}. \tag{6}$$

where $adj(.)$ is the adjugate of a matrix, and $I_{n\times n}$ is the identity matrix.

System zeros play a major role on transient behavior of the system and the control performance. In motion systems, the ideal case is when all the system zeros are stable which means they are in the unity circle in the z-plane. The systems with only stable zeros are called minimum-phase. Existence of unstable zeros (zeros outside of unity circle) causes undesirable degradation of the control performance. Referring to (6), Γ_{aug} and ϕ_{aug} are functions of D_c. Therefore, different values of D_c may change the position of the zeros and change the system behavior. The effect of delay on the performance is analyzed through the transition of system zeros for various delay values. This analysis is called *zero loci* [10].

C. Case-study motion system

In this paper, we study a forth-order dual rotary single-input-multiple-output motion system [13]. The rotary position of two masses are defined as θ_1 and θ_2 and their corresponding rotary speeds as ω_1 and ω_2. By defining these as system states, state space of the motion system is derived [14] as (7), where, $X(t) = [\theta_1, \theta_2, \omega_1, \omega_2]^t$ and,

$$\begin{aligned} A &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -7.08\times10^4 & 7.08\times10^4 & -1.1\times10^6 & 1.1\times10^6 \\ 7.08\times10^4 & -7.08\times10^4 & 1.1\times10^6 & -1.1\times10^6 \end{bmatrix}, \\ B &= \begin{bmatrix} 0 \\ 0 \\ 1.173\times10^4 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{7}$$

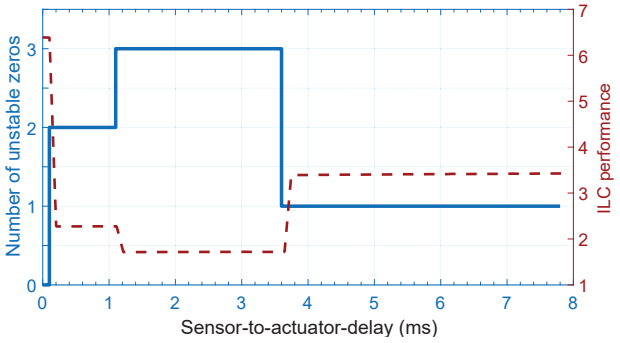

Fig. 2. Number of unstable zeros and ILC performance of the case-study motion system for $h = 8ms$ and $D_c \in [0, h)$.

The objective is to control the output θ_1 to perform a repetitive task. The repetitive task is modeled as a periodic reference $r(t)$ which should be followed accurately by the output. The periodic reference in our case study is defined as $r(t) = sin(\pi t)$ which has a period of $\tau = 2$ seconds.

Zero loci: To perform the zero loci analysis proposed in [8], we first discretize the system equations using our case study $h = 8ms$. Then, by considering each delay choice in the range of $0 \leq D_c < h$, we derive the roots of $\Delta(z)$ (6).

Fig. 2 demonstrates the number of unstable zeros for the delay range of $0 \leq D_c < 8ms$. There are four delay regions with different number of unstable zeros.

- For $0 \leq D_c < 0.1ms$, the system has no unstable zeros. For example, for $D_c = 0.05ms$ the zero polynomial (6) and the corresponding zeros of the system are,

$$\Delta(z) = z^4 + 1.7z^3 + 1.71z^2 + 0.93z + 7 \times 10^{-4},$$

$$z_1 \approx 0, \quad z_2 = -0.9391, \quad z_{3,4} = -0.3843 \pm 0.9186i,$$

  where all the zeros are inside the unit circle and stable.
- For $0.1ms \leq D_c < 1.1ms$, the system has 2 unstable zeros.
- For $1.1ms \leq D_c < 3.8ms$, the system has 3 unstable zeros.
- For $3.8ms \leq D_c < 8ms$, the system has 1 unstable zero. For example, for $D_c = 5ms$ the zeros of the system are,

$$z_1 = -5.6, \quad z_2 = -0.6875, \quad z_{3,4} = -0.4573 \pm 0.6i,$$

  where z_1 is the only unstable zero of the system.

In the next section we describe the ILC design and the impact of D_c on its performance.

III. ILC Design and Delay Analysis

ILCs are mostly used where a motion system executes a repetitive task for multiple times. An example of such system is the inject head of a printing system, surveying the paper for every page printed [2]. While the non-learning feedforward controllers yield to the same tracking error for each task iteration, ILC yield to iteration improving performance by using the information in error signals from the previous iterations [4]. An iteration is a complete execution of a cycle of $r(t)$. The number of samples per iteration is defined as $m = \tau \times h$ ($m = 250$ in our case-study). By defining the notion of j as the task iteration, the calculation of ILC values

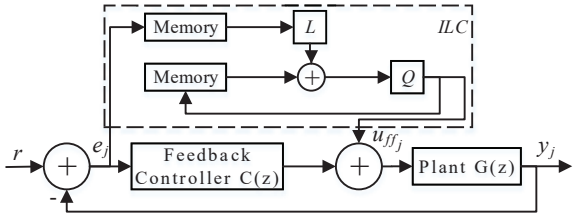

Fig. 3. The architecture of the ILC.

for $(j+1)^{th}$ iteration requires the information of j^{th} iteration, which are the error values e_j and ILC input values u_{ff_j}.

Fig. 3 illustrates the block diagram of an ILC. The control input $u[k]$ consists of two parts of feedback and ILC values:

$$u[k] = u_{fb}[k] + u_{ff_j}[k]. \tag{8}$$

The feedback value $u_{fb_j}[k]$ is the result of applying the controller ($C(z)$) on error signal $e_j[k]$:

$$u_{fb}[k] = C(z)e_j[k], \tag{9}$$

where $e_j[k] = r[k] - y_j[k]$. The feedback design technique is *lead-lag* [12] (which can be replaced by any state-of-the-art design techniques) which aims to stabilize the system.

The ILC input of each iteration consists of a constant-size set of input values $u_{ff_j}[k]$, where:

$$u_{ff_{j+1}}[k] = Q(q)(u_{ff_j}[k] + L(q)e_j[k+1]), \tag{10}$$

where Q and L are robustness and learning filters respectively and q is the forward time-shift operator $qx[k] = x[k+1]$. By transforming the ILC equation (10) to frequency domain and replacing the $e_j[k]$ by its definition, we realize:

$$U_{ff_{j+1}}(z) = Q(z)[U_{ff}(z)_j + zL(z)(R(z) - Y_j(z))]. \tag{11}$$

Among various approaches to design ILC, we focus on model inversion. In this design, the $L(z)$ filter is designed using a model inversion technique. The output can be defined as a product of the reference and the feedforward input as [12]:

$$Y_j(z) = \underbrace{\frac{C(z)G(z)}{1 + C(z)G(z)}}_{\text{Process Sensitivity}} R(z) + \frac{G(z)}{1 + C(z)G(z)} U_{ff_j}(z). \tag{12}$$

We define $L(z)$ as the inverse of process sensitivity, and $Q(z)$ as unity gain. By replacing $Y_j(z)$, $L(z)$ and $Q(z)$ in (11) by their definition and replacing the resulted $U_{ff_{j+1}}(z)$ in (12), we realize for $(j+1)^{th}$,

$$Y_{j+1}(z) = R(z). \tag{13}$$

where the output asymptotically tracks the reference [4].

Although the inversion technique leads to perfect tracking of the reference, it may not be a suitable solution for non-minimum phase systems (system with unstable zeros) since the inverse of the process sensitivity leads to an unstable filter, which results in undesirable large control signals [4]. In this case the $L(z)$ is designed by an stable approximate inversion. Considering ZPETC as one of the approximate methods [15], the $L(z)$ filter is designed as,

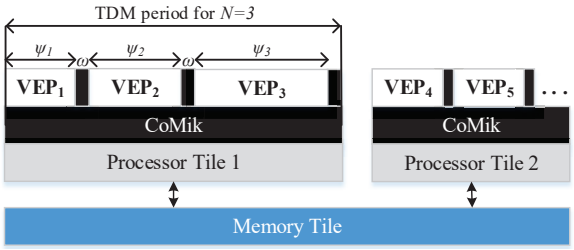

Fig. 4. Predictable embedded platform under consideration.

$$L(z) = \tilde{P}S^{-1}(z) = \frac{P(z)}{\Delta_s(z)\Delta_u^*(z)}, \tag{14}$$

where $P(z)$ and $\Delta_s(z)$ are the characteristic function and the stable zeros polynomials of the process sensitivity respectively. $\Delta_u^*(z)$ is derived by flipping the coefficients of $\Delta_s(z)$ which is the unstable zeros polynomial of the process sensitivity.

A. Effect of sensor-to-actuator delay on ILC performance

As discussed in Section II, different values of sensor-to-actuator delay results different number of unstable zeros. In the case of ILC design this means a different design of $L(z)$ for different D_c values. Therefore, it is important to consider the delay as a design parameter and find an optimal value for the delay which offers the least number of unstable zeros.

Fig. 2 depicts the control performance of the ILC designed for the case-study against different delay values, where the P is the performance metric and:

$$P^{-1} = \sum_{k=1}^{N}(r[k] - y[k])^2. \tag{15}$$

$N = 1000$ is the number of samples for 4 iterations. The maximum performance is achieved for $D_c < 0.1ms$ where the system has no unstable zeros and the direct inversion for $L(z)$ is realized. In the case where this D_c is not feasible in the implementation, the sub-optimal delay region is $3.8ms < D_c < 8ms$ where the system has only one unstable zero.

In the next section we discuss the implementation of delay-based ILC design on the embedded platform.

IV. Embedded ILC implementation

In this paper we target the CompSOC embedded platform [7]. CompSOC is a tile-based embedded platform which consists of a number of processor tiles, local and shared memories and their interconnections. Fig. 4 demonstrates a possible configuration of the platform with two soft-core MicroBlaze as the processor tiles.

The platform can perform a composable execution of multiple applications on each processor tile, isolating the processor, memories and their interconnections in the application level. This enables an interference-free implementation of multiple applications. The composable property is the result of a predictable and composable micro-kernel (CoMik). CoMik creates virtual execution processors (VEPs) as processing resources. It dedicates a specific portion of underlying physical processors and their interconnections to each VEP.

To execute the VEPs on each processor, a periodic time-division-multiplexing (TDM) policy is applied on all processors and interconnections. Executing the VEPs by the TDM order, enables the platform to achieve the real-time performance and cycle-accurate time granularity. Executing the TDM table periodically, enables the periodic and sequential execution of VEPs. Predictability of the platform ensures constant execution times for the tasks which are periodically executed within the VEPs.

Each TDM is the execution of a table which consists of N partition slots. These slots can have specific lengths in clock cycles, indicated by ψ_i, and are separated by N CoMik slots with a constant length of $\omega = 4096$ clock cycles. The CoMik slots enable the jitter-free context switching between partition slots. Each VEP is addressed to one or more partition slots on (possibly) multiple processors and they are swapped in and out periodically and transparently by CoMik. In Fig. 4 the TDM scheduling of processor tile 1, demonstrates an example of a TDM table with 3 different-sized partition slots.

We are interested in the implementation and scheduling of the ILCs. The ILC application is a sequential execution of three operations of sensing, computation and actuation on each sampling interval. The sampling interval is executed periodically with a sampling period of $h = 8ms$ (for our case-study). The sampling period must be kept constant in the implementation. One way to achieve constant/uniform h is to design a TDM table of length h and hence,

$$h = (N \times \omega + \sum_{n=1}^{N}\psi_i)/F_p, \tag{16}$$

where $F_p = 100MHz$ is the operating frequency of the platform.

To implement the ILC operations, we first measure their corresponding execution times based on which design the TDM scheduling. For the controller in our case-study, the measured (constant) execution times in clock cycles are,

$$T_s = 35, \quad T_{cFB} = 125, \quad T_{cFF} = 900, \quad T_a = 50, \tag{17}$$

where the execution of computation is divided into two parts. T_{cFB} is the time to calculate $u_{fb}[k]$ using (9), and T_{cFF} is the time to calculate $u_{ff_j}[k]$ using (10). Since the platform is predictable, the resulting execution times are constant for every sampling interval. This is an important property since it results a cycle accurate value of D_c in every sampling interval. The value of D_c depends on the scheduling method.

A. Sequential Implementation

In the sequential implementation, each of the control operations are mapped to one of the partition slots and are treated as a separate VEP. The size of each partition slot is defined to be bigger than the worst case execution time of their corresponding operation. Also, defining partition slots less than 1024 clock cycles requires additional programming considerations which is avoided in this work. Referring to (17) the sizes of partition slots in clock cycles are defined as:

$$\psi_s = \psi_a = 1024, \quad \psi_c = 1536, \tag{18}$$

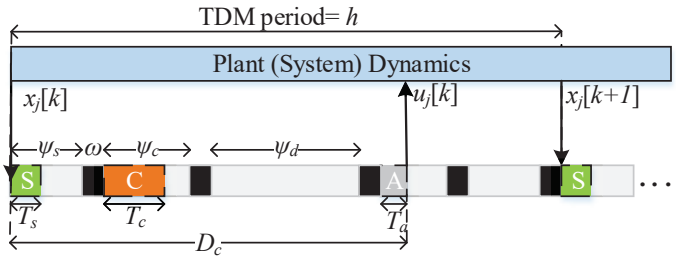

Fig. 5. Scheduling of the sequential implementation where S:*sensing*, C:*computation*, A:*actuation*. The black blocks are the CoMik slots and white blocks are the partition slots.

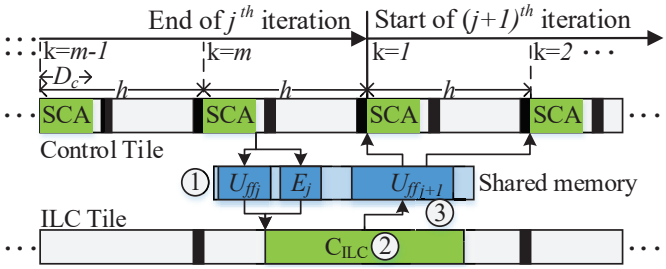

Fig. 6. Scheduling of the parallel implementation. *SCA* is the merged operation of sensing, feedback computation, and actuation, executed on the control tile. C_{ILC} (ILC computation) is mapped to the ILC tile. At the final step of j^{th} iteration ($k = m$), $U_{ff_{j+1}}$ is calculated by the ILC tile and handed over to the control tile to be used in $(j+1)^{th}$iteration.

where ψ_s, ψ_c, and ψ_a are the size of partition slots of sensing, computation, and actuation respectively.

The D_c is the least when we schedule three operations sequentially in the first three partition slots. In this case the value of D_c is:

$$D_c = (2 \times \omega + \psi_s + \psi_c)/F_P + T_a \approx 0.11ms, \qquad (19)$$

Referring to Fig. 2, D_c is in the $0.1ms \leq D_c < 1.1ms$ region, which results in two unstable zeros. Since the performance-optimal delay of $0 \leq D_c < 0.1ms$ is not achievable, we propose a scheduling to derive D_c in the region $3.8ms \leq D_c < 8ms$ which results in one unstable zero. Fig. 5 depicts the TDM table of this scheduling. By adding a *delay* slot with a size of $\psi_d = 385000$ in clock cycles, between computation and actuation, we change the value of D_c to:

$$D_c = (3 \times \omega + \psi_s + \psi_c + \psi_d)/F_p + T_a = 4ms, \qquad (20)$$

which the best achievable ILC performance.

The sequential implementation is a solution for implementing ILCs when the designer wants to accurately choose the values of D_c. It is suitable for a single-core implementation. However, small values of D_c, if desired (like our case-study), may not be achieved by using such an implementation. Therefore we propose the parallel implementation which offers smaller values of D_c and better performance.

B. Parallel Implementation

By studying the definition of the ILC and the sequential implementation, one can realize two important properties which would improve the implementation for multi-core scenarios:

1. Referring to (10), the ILC input in each sample $u_{ff_{j+1}}[k]$ is a function of the error and the ILC input of the previous iteration. Therefore, all of the ILC input values for the current iteration can be calculated at once, at the end of the previous iteration. This calculation also can be done outside of the control scheduling supposedly in parallel, in a separate tile.

2. Consider the execution times in (17) and D_c in (19). A considerable part of D_c are CoMik slots. To remove this, we combine all three operations in a single slot and dedicate a partition slot (long enough to fit all three operations) to the whole control application.

Therefore, we propose a parallel implementation, where a separate core is dedicated to ILC calculation that calculates all the ILC input values of the current iteration at the end of the previous iteration and provide them to the control tile. Fig. 6 demonstrates the parallel implementation.

In every sampling interval, the merged operation *SLC* senses the system state, calculates the $u_{fb}[k]$ using (9), combines it with $u_{ff_j}[k]$, provided by the ILC tile, and actuate the system with $u[k]$. In the final sampling interval of each iteration where $k = m$ ($k = 250$ in our case-study), the control tile gives two vector sets of error values $\{E_j\}_{m\times 1}$ and control values $\{U_{ff_j}\}_{m\times 1}$ of the whole iteration to the ILC tile through the shared memory (1 in the Fig. 6). Next, the ILC tile calculates all the ILC input values for the next iteration $\{U_{ff_{j+1}}\}_{m\times 1}$ (2 in Fig. 6), and hand it over to the control tile through the shared memory (3 in Fig. 6). When the new iteration starts, in each sampling interval, SLC operation reads the corresponding ILC input value $u_{ff_{j+1}}[k]$ from the memory and calculates $u[k]$.

The scheduling of the control tile has two partition slots. The first one is dedicated to SCA operation and has a length of $\psi_{SCA} = 1024$ considering the execution times in (17). The second one is *idle* and is defined to keep the size of the TDM table equal to h. This slot can be dedicated to another time-critical application in multi-application scenarios. The ILC tile has a different schedule. In this tile, there is a single partition slot dedicated to C_{ILC} which is executed once per iteration. The execution time for C_{ILC} is:

$$T_{cILC} = m \times T_{cFF}. \qquad (21)$$

In our case-study $m = 250$, and using (17), $T_{cILC} = 225000$.

Using such scheduling, the value for D_c is:

$$D_c = T_s + T_{cFB} + T_a = 2.5ns. \qquad (22)$$

This is because T_{cILC} does not contribute to D_c due to parallel execution. The resulting D_c is in the performance-optimal region with no unstable zero. In this case, $L(z)$ in (11) is the direct inversion of the process sensitivity and referring to (13) the perfect tracking of the reference is achieved.

An alternative single-core solution similar to parallel implementation is to use the *idle* slot of the control tile at the end of each iteration (see Fig. 6) to perform the ILC computation. However, referring to (21) in the case of higher T_{cFF} (complex ILC designs which require more computation) or higher number of samples per iteration (e.g $m = 2500$ in

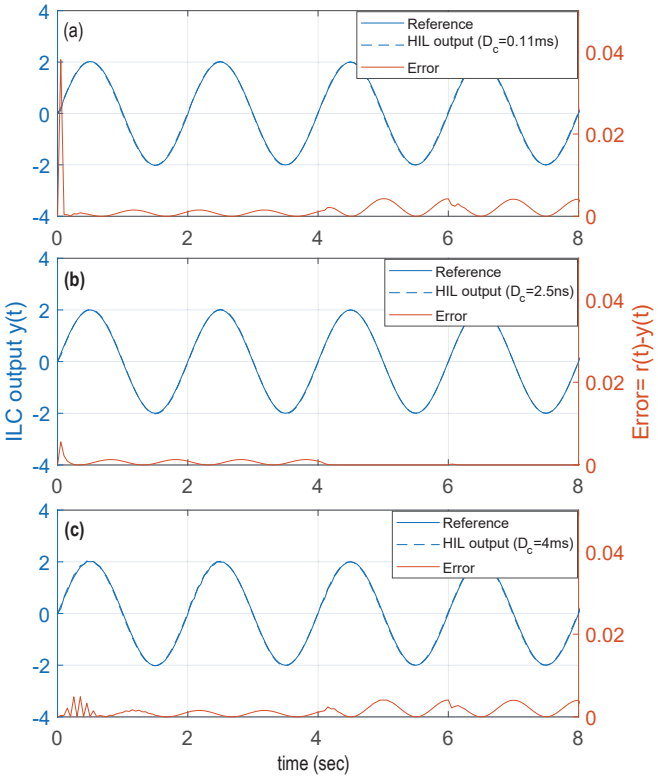

Fig. 7. ILC output and error for $D_c = 0.11ms$ (a), $2.5ns$ (b), and $4ms$ (c).

TABLE I
PERFORMANCE MEASUREMENT WITH DIFFERENT DELAY VALUES

Implementation (Delay)	P**(HIL)**
$Sequential(0.11ms)$	3.0776
$Parallel(2.5ns)$	16.1933
$Sequential(4ms)$	3.4231

our case-study), the *idle* slot may not be sufficient for the ILC computation, since the size of the *idle* slot is bounded by the chosen sampling period. On the other hand, the in parallel implementation utilizes a separate core for C_{ILC} and can execute it at the same time with the execution of the control loop on the other tile without any limitation imposed by h.

V. Performance Analysis with HIL Simulations

To validate the ILC design and implementation, we performed HIL simulations. The platform depicted in Fig. 4 is synthesized on PYNQ FPGA board [www.tul.com.tw]. The architecture consists of three MicroBlaze tiles where one is the control tile and one is the ILC tile (*idle* in the sequential implementation). The third tile is used to implement the system model in (1) with a sampling period of $100\mu s$ to mimic the continuous behavior of the system.

Fig. 7 illustrates the results of the HIL simulations for the first 4 iterations. Fig. 7(a) and Fig. 7(c) are the results of sequential implementation with $D_c = 0.11ms$ (the least achievable delay in the sequential implementation in (19)) and $D_c = 4ms$ (the sub-optimal delay) respectively. Fig. 7(b) is the result of the parallel implementation. Table I provides the performance values for different implementations where P is defined in (15). The results validate that the parallel implementation offers the highest performance, followed by the sequential implementation with $D_c = 4ms$.

VI. Conclusions

In this paper, we proposed design and implementation of ILCs targeting multi-core and predictable embedded platforms. We demonstrated that considering the sensor-to-actuator delay in ILC design and choosing the optimal delay range improves the tracking performance. To implement the ILC, we proposed the sequential and parallel implementations on TDM-based, predictable embedded platforms. While sequential implementation is suitable to carefully tune the delay in a single-core implementation, parallel implementation, as a multi-core solution, enables the designer to realize negligible delay and maximize the control performance. We validated our approach by HIL simulations on a predictable multi-core platform. Possible extensions of this work are the delay-based design of motion controllers similar to ILC, and the implementation of ILC with techniques other than model inversion.

Acknowledgments

This work was partially supported by the H2020 project I-MECH (GA no.737453).

References

[1] K. Ohnishi *et al.*, "Motion control for advanced mechatronics," *IEEE/ASME Transactions on Mechatronics*, pp. 56–67, March 1996.
[2] J. Bolder, T. Oomen, S. Koekebakker, and M. Steinbuch, "Using iterative learning control with basis functions to compensate medium deformation in a wide-format inkjet printer," *Mechatronics*, vol. 24, no. 8, 2014.
[3] Purtojo and Wahyudi, "Integral anti-windup scheme of full-state feedback control for point-to-point (ptp) positioning system," in *2008 International Conference on Electronic Design*, Dec 2008, pp. 1–6.
[4] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE control systems magazine*, pp. 96–114, 2006.
[5] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewycz, S. Chakraborty, H. Voit, and A. Annaswamy, "Challenges in automotive cyber-physical systems design," in *SAMOS, 2012*.
[6] M. Barr, "Embedded systems glossary," *Neutrino Technical Lib.*, 2007.
[7] K. Goossens *et al.*, "NOC-based multiprocessor architecture for mixed-time-criticality applications," *Handbook of Hardware/Software Codesign*, pp. 491–530, 2017.
[8] M. Haghi, F. Wenguang, D. Goswami, and K. Goossens, "Delay-based design of feedforward tracking control for predictable embedded platforms," in *ACC*, 2019.
[9] H. Yan *et al.*, "H_∞ output tracking control for networked systems with adaptively adjusted event-triggered scheme," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–9, 2018.
[10] K. J. Åström and B. Wittenmark, *Computer-controlled systems: theory and design*. Courier Corporation, 2013.
[11] M. B. Cloosterman *et al.*, "Stability of networked control systems with uncertain time-varying delays," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1575–1580, 2009.
[12] K. Ogata, *Discrete-time control systems*. Prentice Hall Englewood Cliffs, NJ, 1995, vol. 2.
[13] W. Geelen *et al.*, "The impact of deadline misses on the control performance of high-end motion control systems," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1218–1229, 2016.
[14] J. Boot, *Frequency response measurement in closed loop: brushing up our knowledge*, ser. DCT rapporten 2003.059. TU/e.
[15] M. Tomizuka, "Zero phase error tracking algorithm for digital control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 109, no. 1, pp. 65–68, 1987.