Predictable multi-core implementation of multi-rate sensor fusion for high-precision positioning systems

Chaitanya Jugade, Sajid Mohamed, Dip Goswami, Andrew Nelson, Gijs van der Veen, and Kees Goossens

***Abstract*— The high-precision and high-speed positioning systems require position feedback with high accuracy at a higher frequency. As reported in recent literature, high accuracy and high operating frequency can be achieved by fusing multiple position sensor data, e.g., the linear encoder (less robust/accurate, but fast) and object detection using camera images (accurate, but slow due to heavy processing load). Typically, image-based object detection incurs a significant computational delay due to computationally intensive processes and is the main performance bottleneck. Moreover, the computation delay varies when implemented on industrial platforms and degrades the performance of the closed-loop control system. In this paper, we present scheduling techniques such as parallelism and pipelining considering predictable multi-core platforms for such a multi-sensor positioning system. On the one hand, the predictable platform nearly removes the variation in execution time, making the delay constant. On the other hand, the parallel and pipeline schedules reduce the computation delay, translating to a shorter sampling period and better closed-loop performance. Furthermore, we perform a design space exploration on various parameters and control performance considering an industrial case study of semiconductor die-bonding equipment.**

I. Introduction

The high-precision and high-speed positioning systems are the basic building blocks in many industrial systems, including semiconductor die-bonding equipment. Typically, such a system needs to bring the objects (e.g., semiconductor die, chip, any other object) to the *pick* (or *place*) location from where a pickup mechanism (e.g., vacuum nozzle) picks the object (or places the object). The performance of such a system is often determined by how accurately the object is brought to the pick (or place) location (i.e., reference position) such that the pickup (or place) operation happens without violating the system specifications or causing failure. This further implies that the precision of the positioning system depends on how accurately and fast the position of the object (feedback) is measured.

Fig. 1 shows a simplified schematic of an industrial positioning system in semiconductor die-bonding equipment [2], [3] that represents the class of systems under consideration in this work. The system in Fig. 1 utilizes high-precision linear motors with built-in industrial standard linear encoders to measure the positions of the motors as indicated in [4].

This work is supported by the ECSEL Joint Undertaking under grant agreement no. 101007311 (IMOCO4.E [1]).

C. Jugade, D. Goswami, A. Nelson, and K. Goossens are with the Department of Electrical Engineering, Eindhoven University of Technology (TU/e) (e-mail: {C.Jugade, D.Goswami, A.T.Nelson, K.G.W.Goossens}@tue.nl).

S. Mohamed and G. van der Veen are with ITEC B.V., The Netherlands (e-mail: {sajid. mohamed, gijs.van.der.veen}@itecequipment.com).

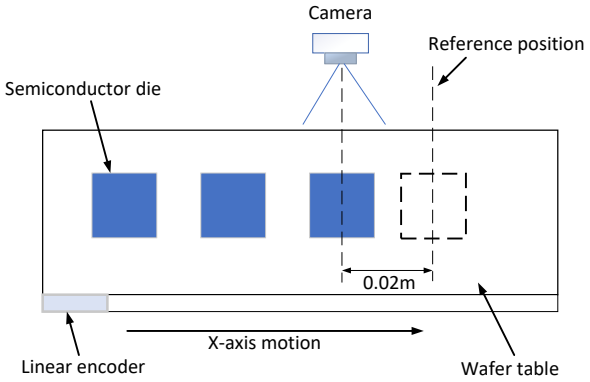

Fig. 1. Schematic of a typical industrial positioning system with camera and linear encoder as sensors.

The linear encoders exhibit high accuracy in measuring the positions at the point-of-control, such as the wafer table, but they do not provide the direct capture of the position-of-interest, such as the actual position of the semiconductor die. Therefore, in the presence of external disturbances (that influence the die positions), such a single-sensor system fails to meet the desired level of positioning accuracy. Multi-sensor fusion techniques are reported in literature [5], [6], [7] to improve the sensing accuracy of such systems. The idea is to measure the actual die/object position using object detection from the camera images and fuse with the measurement of the table location measured by the linear encoders. While such a multi-sensor fusion approach is promising, the performance of such an approach is heavily influenced by two important factors [8], [9] – (i) how large is the computation delay of the image processing and (ii) what is the range of variation in execution time of the sensor processing.

To deal with these challenges, in this paper, we consider the CompSOC platform [10], a predictable and composable multi-core system-on-chip (MPSoC) platform designed for developing embedded control applications with strict performance requirements. The platform offers deterministic execution of algorithms and supports composability for multi-application scenarios desired for the deployment of the control applications. Furthermore, we propose multiple scheduling approaches to reduce the overall delay in the closed-loop system. The key contributions of this paper are:

1) Predictable multi-core implementation of multi-sensor fusion imposing (nearly) constant delay.
2) Reducing sensing delay using predictable parallel implementation.
3) Reducing sensing delay using predictable pipelined

implementation.

4) Design space exploration (DSE) with various implementation choices and positioning performance on a dual-core platform.

II. Related Work

In this paper, our focus is on reducing long computational delays in the context of vision processing to improve positional precision. This goal is achieved through embedded implementation on the predictable multi-core CompSOC platform, using scheduling techniques such as parallelism and pipelining. In [11], authors explored the integrated design of control and scheduling. However, their focus is on networked control systems considering industrial platforms, not predictable platforms. Furthermore, in [8], [12], authors proposed a model-based design approach for the implementation of image-based control systems on multiple processors, effectively reducing computational delays. However, the idea of a model-based design approach is not verified on a predictable platform. The efficient implementation of vision-based perception for autonomous applications on resource-constrained embedded platforms is discussed in [13], but the primary focus is limited to vision-based perception. In addition, [14] presented the delay-based design technique for the feedforward controller, which targets the predictable embedded platform but does not explore the impact of different scheduling techniques. [15] discusses timing challenges associated with scheduling algorithms in control loops, addressing issues such as jitters in task executions. Authors in [16] introduced a task-parallel programming scheduling approach tailored to scientific computing applications. The existing literature lacks models or investigations into the impact of pipelining and parallelism scheduling techniques in the context of vision-in-the-loop for predictable implementation in industrial motion control systems. This gap in research is important for handling long computational processing times when aiming for precise and accurate motion control.

III. Background and Problem Statement

We consider a multi-sensor fusion strategy for position sensing in high-precision motion control systems as depicted in Fig. 1. The two sensors under consideration are a linear encoder and a camera (vision-based object detector). Typically, a linear encoder operates at a higher rate, e.g., $8kHz$ in die-bonding machines [2]. A linear encoder provides position feedback $z_{e,k}$ of the point-of-control, e.g., the wafer table in our case study. First, $z_{e,k}$ is available at a higher rate with sampling period $h_{encoder}$. Second, the encoder feedback $z_{e,k}$ does not capture the effect of disturbances at the point-of-interest. The overall control system operates at the base sampling period $h_{encoder}$.

On the other hand, a vision-based object detection algorithm (e.g., Hough transform) takes a camera image and provides the true location (including disturbances) of the object $z_{v,k}$. First, the processing of images to detect objects is compute-heavy and incurs significant computing delay τ_{vision}. The period of camera frames $h_{vision} \gg h_{encoder}$ and for a sequential implementation $h_{vision} \geq \tau_{vision}$. Given the control system runs at $h_{encoder}$, let

$$h_v = \lceil \frac{h_{vision}}{h_{encoder}} \rceil. \quad (1)$$

Second, τ_{vision} greatly varies depending on the workload of the specific image frame $img[k]$ on typical industrial platforms. Considering the computed or estimated worst-case delay τ_{vision}, the vision measurements are delayed by τ_v base samples, where

$$\tau_v = \lceil \frac{\tau_{vision}}{h_{encoder}} \rceil. \quad (2)$$

Therefore, the position information of the image frame captured at time-instance k, $img[k]$ produces feedback $z_{v,k}$ after τ_v samples.

The overall closed-loop system is shown in Fig. 2. The multi-rate Kalman filter block fuses the linear encoder and the vision processing data to produce the estimate of the position $\hat{z}_k$. The position estimate is used in computing the control input u_k. The following sections explain the various computational blocks and the parameters illustrated in Fig. 2.

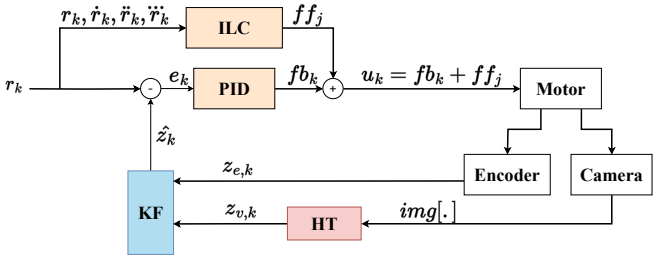

Fig. 2. Control system block diagram with multi-sensor multi-rate fusion. HT refers to the vision-based object detection algorithm i.e., Hough Transform, and KF refers to the multi-rate Kalman filter.

A. Multi-rate Sensor Fusion

The multi-rate Kalman filter is used for fusing the linear encoder and vision data. The linear encoder measurements $z_{e,k}$ are available with sampling period $h_{encoder}$. The camera captures images with period h_{vision}, and the object's true position is available after τ_v samples. That is, the vision measurement $z_{v,k}$ is produced from the image $img[k]$ captured τ_v sample time ago. Accordingly, there are 3 sample modes based on the sensor data availability at a time instance – Mode 1: when only linear encoder data $z_{e,k}$ is available; Mode 2: when $z_{e,k}$ is available and image frame $img[.]$ is being processed; and Mode 3: when $z_{e,k}$ and $z_{v,k}$ are available. The multi-rate Kalmar filter technique is used to fuse the above two types of data in different modes [17] and estimate position $\hat{z}_k$ with sampling period $h_{encoder}$. Fig. 3 shows the timing diagram of the multi-rate data fusion approach.

B. Control structure

The control law is composed of proportional-derivative-integrator (PID) based feedback controller and an iterative learning controller (ILC) for feedforward. As shown in

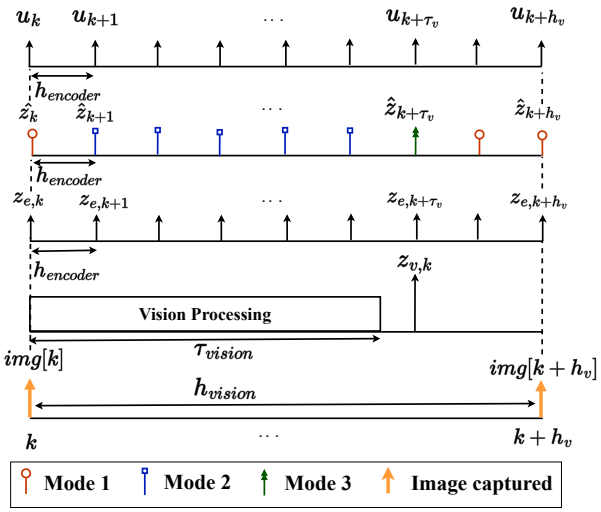

Fig. 3. Timing diagram of multi-rate sensor fusion algorithm. τ_{vision} is the worst-case execution-time of the vision-based object detector, $img[k]$ is the captured image, $z_{v,k}$ is the vision measurement, $z_{e,k}$ is the encoder measurement, and $\hat{z}_k$ is the sensor fusion output at time instance k.

Fig. 2, the control input u_k is achieved through the combination of feedback control (fb_k) and feedforward control (ff_j), denoted as

$$u_k = fb_k + ff_j.$$

1) PID Controller: The discrete-time PID controller equation is given as follows:

$$fb_k = k_p e_k + k_i \sum_{m=0}^{k} e_m h_{encoder} + k_d \left(\frac{e_k - e_{k-1}}{h_{encoder}} \right), \quad (3)$$

where fb_k is the feedback control action, k_p, k_i, and k_d are the proportional, integral, and derivative controller gains respectively. e_k is the error signal,

$$e_k = \hat{z}_k - r_k, \quad (4)$$

where r_k is the reference at the k^{th} sampling instance.

2) Feed-forward ILC: The high-precision positioning system under consideration performs repetitive pick-and-place tasks of different objects, e.g., semiconductor dies. ILC is particularly beneficial for managing repetitive errors in a control system over time [18], [19]. We consider ILC design reported in [2] in this work. Essentially, the ILC algorithms provides the feedforward signal ff_j where j is the iteration (i.e., index of a particular pick-and-place task).

C. Problem Statement

The main problem tackled in this work is to achieve a shorter and (nearly) constant processing time of the vision-based object detection algorithm (for images with the same workload) towards improving control performance. τ_{vision} is the worst-case execution time of the object detection algorithm. As depicted in Fig. 3, if τ_{vision} is large, the vision feedback is provided less frequently which reduces the sensing accuracy and control performance. Moreover, a high variation in the computation time of vision processing results in a large and pessimistic τ_{vision}, which in turn negatively influences the control performance.

IV. Predictable Multi-core Platform

We consider CompSOC embedded platform featuring a tile-based architecture that enables configuration using multi-processor synchronized tiles, interconnects, local and shared memories. CompSOC uses a predictable and composable micro-kernel (CoMiK) [20] to create a virtual execution platform (VEP) [21]. In CompSOC, resources are space- and time-partitioned to create VEPs in which applications run independently. These partitions allow multiple applications to execute in their independent VEP, following the time-division-multiplexing (TDM) schedule. In the following, we describe the instance of the platform used in this work.

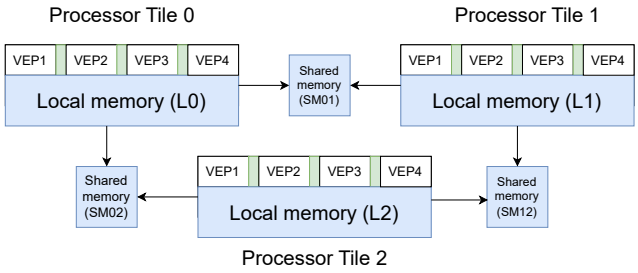

Fig. 4. CompSOC architecture with 3 processor tiles. The green boxes show the CoMiK slots.

A. Hardware Architecture

Fig. 4 shows the CompSOC hardware architecture under consideration. We focus on a commercial instance of CompSoC called Verintec as the hardware architecture, where the platform has three RISC-V processor tiles connected through shared memory. We denote the tiles as Processor Tile 0, 1, and 2. Each tile has a local memory denoted as $L0$, $L1$, and $L2$. Total local memory per tile ($L0/L1/L2$) is $128KB$ of which $96KB$ is available for the applications and $32KB$ is used by the system application (discussion in the following subsection). There is a shared memory of size $32KB$ between each pair of tiles. For example, Processor Tile 0 and 1 have a shared memory denoted by $SM01$. Therefore, we have three shared memories $SM01$, $SM02$, and $SM12$ for the 3-tile architecture under consideration.

B. Software Architecture

The platform utilizes CoMik to generate VEPs, which serve as dedicated resources. A TDM policy is used on all processor tiles. The platform achieves precise global synchronization by employing a periodic TDM policy. In the platform under consideration, the TDM table can be divided into maximum 4 VEPs and minimum 2 VEPs. The VEPs are denoted by $VEP1$ to $VEP4$ respectively. In between any two VEPs, there is a CoMik slot. The length of the CoMik slot is constant and denoted by w number of cycles and $w = 2000$. The length of each VEP is denoted by ψ_k number of cycles, where $k = 1,2,3,4$. A fixed number of cycles is allocated to the system application that runs on $VEP1$ of each tile. In our experiments, we consider $\psi_1 = 5000$ cycles and $32KB$ local memory for VEP1. Each VEP (i.e., VEP2, VEP3 or VEP4) might have local memory of size – the

maximum $64KB$ and the minimum $32KB$ and total memory size is $96KB$. Fig. 5 illustrates the TDM table that executes sequentially and periodically. The TDM period is calculated as $\sum_{k=1}^{4}(w+\psi_k)$.

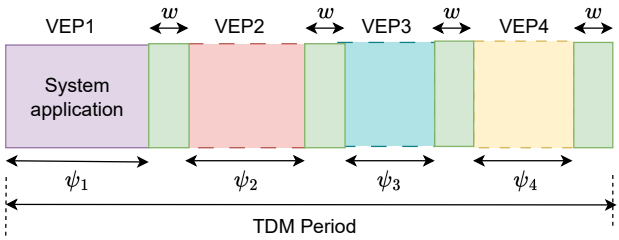

Fig. 5. The TDM table with 4 VEPs.

V. Predictable Sensor Fusion Implementation

We consider the application described in Section III and the platform described in Section IV for implementation. The question is how to deploy or map the application to the platform such that the timing behavior shown in Fig. 3 is realized.

A. Application Model

As shown in Fig. 2, there are four computation blocks: HT, KF, PID, and ILC. Since the execution times of KF, PID, and ILC are negligible compared to that of HT, and are running at the same rate, we execute all three of them as a single task. This leads to two computation blocks: HT and KF+PID+ILC, with different execution times and execution rates. The application model should capture the dependency between various computation blocks, execution times, and rates. We use the synchronous dataflow (SDF) [22] to model the application behavior. A dataflow model is a directed graph with *actors* or nodes modeling computation blocks and platform-dependent execution time, and edges or *channels* capturing the dependencies between actors. *Tokens* represent data communicated through channels and have a memory requirement. Channels may contain *initial tokens*, denoted by dots. Initial tokens represent data availability in a channel at the start of analysis. A simulation of an SDF graph involves *firing* (execution) of actors. An actor executes for a fixed time specified by its execution time. An actor can fire iff sufficient tokens are available on all its input channels. These amounts are called the *rates*, indicated next to the channel ends (omitted if 1). Every time an actor fires, it consumes the same number of tokens from its input channels and produces the same number of tokens on its output channels, as modelled by the rates. Fixed rates allow iterative execution of an SDF, where the initial token distribution occurs after each iteration.

Fig. 6 shows the dataflow model of the sensor fusion implementation. Two computation blocks are modeled by two *actors* annotated by their worst-case execution times τ_{vision} and $\tau_{KF,PID,ILC}$ respectively. In addition, the encoder data generation and the camera image capture are modeled by two additional actors Encoder and Camera, with $h_{encoder}$ and h_{vision} execution times. Further, the actuation operation is assumed to take negligible time $\tau_{actuation}$ and is modeled by the Actuation actor. At $t=0$, the Camera, Encoder, HT and KF+PID+ILC actors use the initial tokens on their incoming channels and fires. The Encoder actor simulates the encoder data availability with period $h_{encoder}$, the base sample period of the sensor fusion implementation. The Camera actor produces one image frame in every h_v base samples (see Eq. 1). The actor HT executes for τ_{vision} time and produces h_v tokens on the outgoing channel. One firing of actor KF+PID+ILC requires only one token produced by the HT actor. An assumption in this model is that the control and actuation tasks complete within one base sample, i.e. $h_{encoder} \geq (\tau_{KF,PID,ILC}+\tau_{actuation})$. The above model captures the application behavior respecting the timing behavior and dependencies described in Fig. 3.

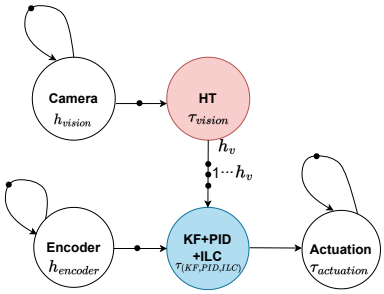

Fig. 6. Dataflow model for multi-rate sensor fusion algorithm shown in Fig. 2.

B. Application Profiling

To decide on the platform configuration and mapping, we need to measure the execution and memory footprint of the actors on the CompSOC platform under consideration. We use processor-in-the-loop (PIL) simulation to measure the WCET and memory utilization of each task. We prototyped the automatic code generation for the CompSOC platform on the PYNQ-Z2 FPGA board [23]. This tool generates the code specific to the target, and designers can choose which part of the MATLAB Simulink model runs on the platform.

The execution times of the tasks KF, PID and ILC are negligible compared to the HT actor and run at the same rate. Therefore, we run KF, PID and ILC tasks on one VEP while the HT actor runs on another VEP. Table I shows the memory footprint and execution time profiled on the above instance of the CompSOC platform.

TABLE I
Profiling the Sequential Implementation

	Memory Footprint	WCET
HT	47.88 KB	30.5 ms
KF + PID + ILC	21.62 KB	0.53 ms
Image (token)	12.20KB	-

C. Sequential Implementation

In the case of multi-rate sensor fusion, a sequential implementation refers to the sequential execution of the compute-heavy sensing task (modeled by the HT actor). To achieve independent task execution on the CompSOC platform, we implement it in such a way that, each task runs one after the other. The two tasks shown in Table I should run in parallel with different rates as captured in the application model. Therefore, they should be mapped to different tiles for parallel execution. Hence, we map the task or actor HT to VEP2 in Tile 1 and the task KF+PID+ILC to VEP2 in Tile 0. The task KF+PID+ILC should be executed with a sampling period $h_{encoder}$. We consider $h_{encoder} = 1ms$ for our implementation (in line with its execution time of $0.53ms$) which should be equal to the TDM period. The length of the TDM period i.e., $(\psi_1 + 2w + \psi_2)$ in Tile 0 is given by $1ms \times 40MHz = 40000cycles$. We configure the Tile 0 with two VEPs as shown in Fig. 7. VEP1 runs the system application and is 5000 cycles long. The length of VEP2 is given by

$$\psi_2 = (40000 - 2000 \times 2 - 5000)cycles = 31000cycles = 0.7ms.$$

From the execution time of task HT, $\tau_{vision} = 30.5ms$ and $\tau_v = \lceil \frac{\tau_{vision}}{h_{encoder}} \rceil = 31$. We consider $200fps$ with image frame arrival period $h_{camera} = 5ms$. As $h_{vision} \geq \tau_{vision}$,

$$h_{vision} = \lceil \frac{\tau_{vision}}{h_{camera}} \rceil \times h_{camera} = 35ms.$$

The length of the TDM period in Tile 1 should be equal to h_{vision} and is given by, $35ms \times 40MHz = 1400000cycles$. We configure the Tile 1 with two VEPs as shown in Fig. 7. VEP1 runs the system application and is 5000 cycles. The length VEP2 is given by

$$\psi_2 = (1400000 - 2000 \times 2 - 5000)cycles = 34.7ms.$$

Here, $h_v = \lceil \frac{h_{vision}}{h_{encoder}} \rceil = 35$.

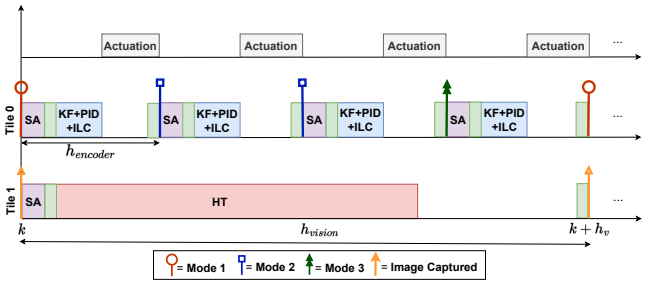

Fig. 7. Timing diagram of the sequential implementation.

VI. Adaptation for Parallel Implementation

As shown in Table I, the HT task is compute-heavy compared to the other tasks. The main idea of parallelism is to split the HT task into multiple subtasks and execute them in parallel. This is feasible if the subtasks are independent. The HT task executes the Hough transform which searches for a line over a range of angles $\theta = -5^\circ$ to $+5^\circ$. The search can be done independently over different ranges of θ and in essence, we can parallelize the HT task over different ranges of θ. In this work, the HT task is split into two ranges -5° to 0° and 1° to $+5^\circ$ which can be executed independently. We name these tasks HT_1 and HT_2. Internally, the Hough transform algorithm computes data peaks in these two tasks and merging task (M) merges those data peaks to obtain the location of the center of the object.

A. Application Model

The dataflow model of the parallel implementation is shown in Fig. 8. The only difference from the sequential case is that both HT_1 and HT_2 must be completed before the M task is executed. In the case of parallel implementation, the production rate of the sensing task is h_v^{par}, instead of h_v, and is calculated in Section VI-C.

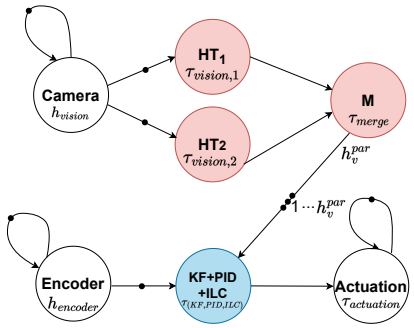

Fig. 8. Dataflow model for parallel implementation of multi-rate sensor fusion algorithm and control.

B. Application Profiling

Table II shows the memory footprint and execution time profiled on the instance of the CompSOC platform in PIL mode. HT_1 has a longer execution time since it handles a larger range of θ.

TABLE II
PROFILING THE PARALLEL IMPLEMENTATION

	Memory Footprint	WCET
HT_1	22.3KB	18.5ms
HT_2	25.58KB	15.5ms
KF+PID+ILC	21.62KB	0.53ms

C. Parallel Implementation

The tasks shown in Table II should run in parallel with different rates as captured in the application model and illustrated in Fig. 9. We map the task or actor HT_1 to VEP2 in Tile 1, HT_2 to VEP2 in Tile 2, and the task KF+PID+ILC to VEP2 in Tile 0. We implement both the M task and the HT_1 task on VEP2 in Tile 1. However, after the completion of the HT_1 task, the M task initiates its execution and awaits the output from both HT_1 and HT_2. The configuration for Tile 0 is identical to the sequential case with $h_{encoder} = 1ms$. From the execution time of tasks HT_1 and HT_2,

$$\tau_{vision} = \max(\tau_{vision,1}, \tau_{vision,2}) = 18.5ms$$

and $\tau_v = \lceil \frac{\tau_{vision}}{h_{encoder}} \rceil = 19$. With frame arrival period $h_{camera} = 5ms$ and $h_{vision} \geq \tau_{vision}$, $h_{vision} = \lceil \frac{\tau_{vision}}{h_{camera}} \rceil \times h_{camera} = 20ms$.

The length of the TDM period of Tile 1 and Tile 2 is given by $20ms \times 40MHz = 800000cycles$. Accordingly, we configure Tile 1 and Tile 2 with two VEPs. VEP1 runs the system application and is 5000 cycles long. The length of VEP2 is given by

$$\begin{aligned}\psi_2 &= (800000 - 2000 \times 2 - 5000)cycles \\ &= 791000cycles \\ &= 19.7ms.\end{aligned}$$

$$h_v^{par} = \lceil \frac{h_{vision}}{h_{encoder}} \rceil = 20.$$

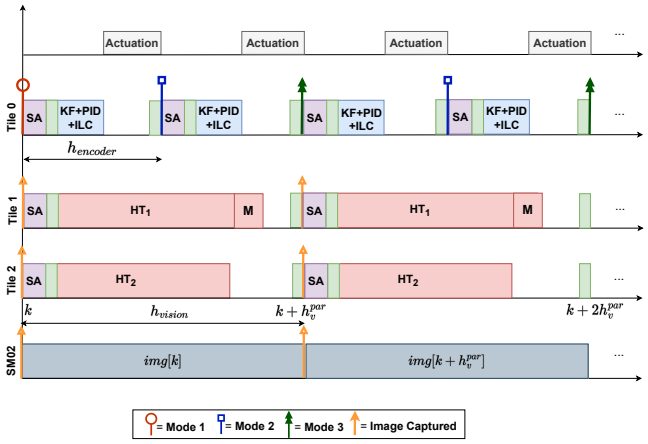

Fig. 9. Timing diagram of the parallel implementation.

VII. Adaptation for Pipelined Implementation

We refer to Table I. The HT task takes a longer execution time compared to the KF+PID+ILC task, resulting in a significant increase in sensor-to-actuator delay. The concept behind a pipelined implementation is to process camera frames by the HT task in a pipeline across multiple cores, thereby reducing the effective vision sampling period.

A. Application Model

Fig. 10 shows the data flow model of the pipelined implementation with two pipes (derived from [8]). We pipelined the HT task on two cores, processing the two pipes. We represent the two pipes as HT^1 and HT^2.

B. Application Profiling

We consider the timings and memory footprints obtained from the sequential implementation profiling (see Table I). The HT^1 and HT^2 task operates over the same range $\theta = -5^\circ$ to $+5^\circ$ as the sequential implementation.

C. Pipelined Implementation

Fig. 11 shows the Gantt chart for the pipelined implementation to illustrate the workflow of each task. We map the HT^1 on VEP2 of Tile 1 and HT^2 on VEP2 of Tile 2. Each HT processes a distinct image. In the example illustrated in

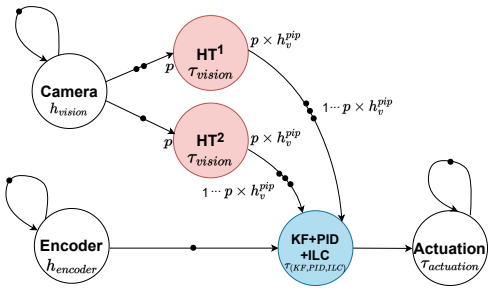

Fig. 10. Dataflow model for pipelined implementation with two pipes, i.e., p=2.

the figure, the HT^1 on Tile 1 processes the $img[k]$ (stored in SM01), while the HT^2 task runs on Tile 2 and processes the $img[k+2]$ (stored in SM02). The KF+PID+ILC task is mapped to VEP2 of Tile 0, the same as the sequential implementation. The execution time of the HT task is $\tau_{vision} = 30.5ms$, and $\tau_v = \lceil \frac{\tau_{vision}}{h_{encoder}} \rceil = 31$. The camera frame rate is $200fps$, therefore h_{vision} is given by

$$h_{vision} = \lceil \frac{\tau_{vision}}{p \times h_{camera}} \rceil \times h_{camera} = 20ms.$$

where p is the number of pipes i.e., $p = 2$. The HT^1 and HT^2 runs on Tile 1 and Tile 2, processing distinct images at intervals of $h_{\text{vision}} = 20ms$. Therefore, the length of the TDM period for Tile 1 and Tile 2 is given by $35ms \times 40MHz = 1400000cycles$. As shown in Fig. 11, we configure Tile 1 and Tile 2 with two VEPs. VEP1 runs the system application and requires 5000 cycles. The length of VEP2 is given by

$$\psi_2 = (1400000 - 2000 \times 2 - 5000)cycles = 34.7ms.$$

Since, the $h_{vision} = 20ms$ and $h_{encoder} = 1ms$, the $h_v^{pip} = \lceil \frac{h_{vision}}{h_{encoder}} \rceil = 20$.

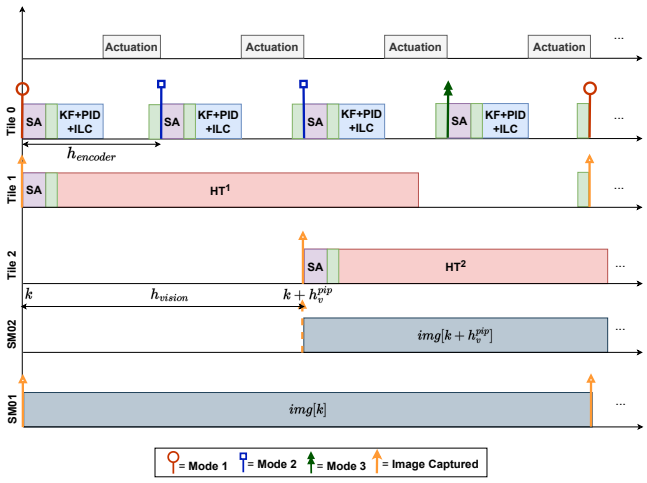

Fig. 11. Timing diagram of the pipelined implementation with two pipes.

VIII. Results

A. Predictable Implementation Tool Flow

Fig.12 illustrates the complete tool flow for *synchronous* lock-step simulation of sequential, parallel, and pipelined

implementations on CompSOC. We model the industrial positioning system including actuators and sensors in Fig.1 using the physics simulation engine CoppeliaSim [24]. The multi-rate sensor fusion and control algorithm computation are performed on the CompSOC platform. CoppeliaSim and CompSOC are interfaced through MATLAB using the MATLAB remote API.

In CoppeliaSim, the linear encoder sensor measures the position of the wafer table and sends the position information $z_{e,k}$ to the MATLAB Simulink **Get Encoder** block. The vision sensor sends the captured image $img[k]$ to the **Get Image and Edge Detection** block in MATLAB Simulink. Furthermore, the **Monitoring Application** is responsible for synchronization between the tasks running on CompSOC (in real-time) and the plant simulation (in logical simulation time). It receives the $img[k]$ and $z_{e,k}$ as inputs and stores them in the CompSOC shared memory that is further accessed by other tasks for control signal computation and actuation.

For example, in the case of the sequential implementation explained in Fig. 7, the monitoring application stores the $img[k]$ and $z_{e,k}$ in $SM01$ shared memory and sends the $img[k]$ to the HT task every h_{vision} period. The HT task runs on CompSOC Tile 1, processes the $img[k]$, and sends the vision measurement $z_{v,k}$ to the *KF+PID+ILC* task on Tile 0 which also receives $z_{e,k}$ from the monitoring application every $h_{encoder}$ period and computes the control action u_k. The resulting control action u_k is written back to shared memory $SM01$, which is then read by the monitoring application to provide to the **Update actuator** block responsible for actuating the plant. Similar deployment is realized for some variants of our implementations.

The setup verifies the timing properties of the designed application executed on the CompSOC platform as well as it allows for functional verification on MATLAB.

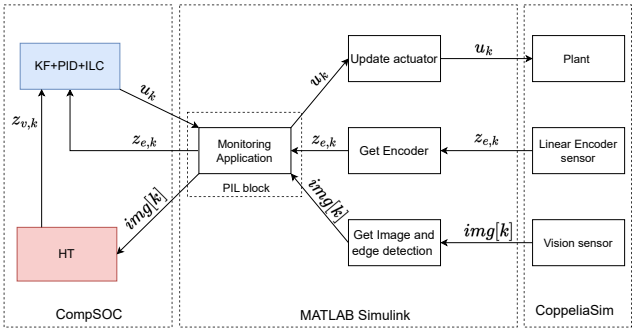

Fig. 12. Predictable implementation tool flow for synchronous lock-step simulation and experimentation.

B. Validation of Predictable Implementations

Fig. 13 illustrates the closed-loop performance achieved with a sequential, parallel, and pipelined predictable implementation on CompSOC. The region-of-interset (RoI) of the camera is 142×88 pixels – see Fig. 1. The HT task uses 142×88-pixel images. One pixel is calibrated to the dimension of $1mm$. A semiconductor die has a dimension of 25×25 pixels (or $25mm \times 25mm$). The reference position is $0.020m$ which is equivalent to $20pixel$. We evaluate the closed-loop performance of the control system considering the mean absolute error (MAE) (m) i.e., $MAE = \frac{1}{N}\sum_{k=1}^{N-1}|(z_k - \hat{z}_k)|$ where z_k is the true die position, $\hat{z}_k$ is the estimated position, i.e., multi-rate sensor fusion output and the settling time is the time to reach within 2% of the reference position.

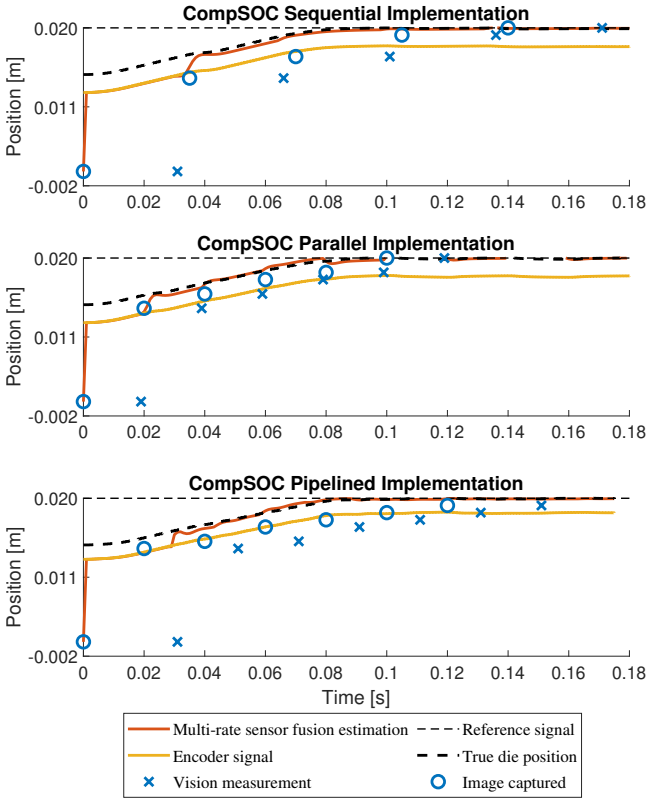

Fig. 13. Closed-loop performance with sequential, parallel, and pipelined implementation.

We consider a system explained in Fig. 1 with an external disturbance of $0.0025m$. In the case of the sequential implementation, the output of the multi-rate sensor fusion converges to the true die position in $0.18s$ with an MAE of $0.00023m$. In the parallel implementation, the system converges to the reference in $0.15s$ with an MAE of $0.00020m$. In the pipelined implementation, the settling time is reduced to $0.13s$ with an MAE of $0.000211m$. Clearly, the parallel and pipelined implementations significantly reduce the sampling period which leads to an improved performance compared to the sequential implementation.

C. Design space exploration

We perform design space exploration (DSE) by considering various implementation choices and evaluating their effectiveness across various θ ranges in the HT task. Table III presents the results, which show how execution timings and sample periods change with respect to different θ ranges. Table IV presents the closed-loop performance for different predictable implementations based on various ranges of θ with different tuned PID controller gains.

TABLE III
DESIGN SPACE EXPLORATION

θ range	Sequential		Parallelism		Pipelined
	WCET	h_{vision}	WCET	h_{vision}	h_{vision}
Case 1	30.5ms	35ms	18.5ms	20ms	20ms
Case 2	25.5ms	30ms	16.5ms	20ms	15ms
Case 3	20.5ms	25ms	13.5ms	15ms	15ms

$^{\star}\theta$ range – Case 1:-5° *to* 5°, Case 2:-4° *to* 4°, and Case 3: -3° *to* 3°

TABLE IV
CLOSED-LOOP PERFORMANCE FOR DIFFERENT IMPLEMENTATIONS

Scheduling technique	Case	K_p	K_i	K_d	Performance metrics	
					Settling Time	MAE
Sequential	Case 1	1500	500	100	0.18s	0.00023m
	Case 2	1500	500	100	0.21s	0.00028m
	Case 3	1500	500	100	0.28s	0.00031m
Parallelism	Case 1	1500	320	185	0.15s	0.00020m
	Case 2	1500	320	185	0.15s	0.00022m
	Case 3	1500	320	185	0.17s	0.00026m
Pipelined	Case 1	1500	450	230	0.13s	0.000211m
	Case 2	1500	450	230	0.15s	0.00024m
	Case 3	1500	450	230	0.16s	0.000254m

$^{\star}\theta$ range – Case 1:-5° *to* 5°, Case 2:-4° *to* 4°, and Case 3: -3° *to* 3°

From Tables III and IV, it can be observed that as the θ range decreases, the WCET of the HT task also decreases, leading to a reduced h_{vision}. However, the reduction in the θ range increases the MAE which is undesirable. This is a trade-off. It is crucial to select the appropriate predictable implementation technique for a given position precision. Generally, the sampling period is reduced with parallel and pipelined implementation. The control performance further depends on the tuning or design of the controller. In this case, the PID gains further play an important role in translating a shorter sampling period to a better performance. Under the dual-core parallel and pipelined implementations, the sampling periods are nearly comparable. Depending on the PID gains, the performance of the closed-loop controller changes. With the manual gain tuning, Case 1 of the pipelined implementation gives the shortest settling time while Case 1 of the parallel implementation gives the smallest MAE. Generally, Case 1 has the widest θ range compared to other cases and leads to a lower MAE due to higher sensing accuracy.

IX. CONCLUSIONS

We presented a predictable multi-core implementation of multi-rate sensor fusion aiming to reduce the delay and make it constant. We used an instance of the CompSOC platform with three processing tiles, and our implementations are dual-core or three-core. Our results show that the parallel and pipelined implementations of fusion algorithms significantly reduce the effective delay/sampling period which can be translated to an improved performance by appropriate design of the controller. Future research involves expanding our design-space exploration by considering a greater number of cores and employing model-based design methods to optimize the computational delay of the vision processing workload.

REFERENCES

[1] S. Mohamed, G. v. d. Veen *et al.*, "The IMOCO4.E reference framework for intelligent motion control systems," in *ETFA*, 2023.

[2] G. van der Veen, J. Stokkermans *et al.*, "How learning control supports industry 4.0 in semiconductor manufacturing," in *ASPE Design and Control of Precision Mechatronic Systems*, 2020, pp. 1–5.

[3] D. D. Manickam, S. Mohamed *et al.*, "A structured inference optimization approach for vision-based DNN deployment on legacy systems," in *ETFA*, 2023.

[4] A. Teimel, "Technology and applications of grating interferometers in high-precision measurement," *Precision Engineering*, 1992.

[5] C. Jugade, D. Hartgers *et al.*, "Improved positioning precision using a multi-rate multi-sensor in industrial motion control systems," in *European Control Conference (ECC)*, 2023.

[6] S. Jeong, S. An, and S.-H. Hwang, "Multi-rate sensor fusion localization algorithm using camera and GPS in urban environments," in *Electric Vehicle Symposium (EVS)*, 2023.

[7] M. Čech, A.-J. Beltman, and K. Ozols, "Digital twins and ai in smart motion control applications," in *IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022.

[8] S. Mohamed, D. Goswami *et al.*, "Optimising multiprocessor image-based control through pipelining and parallelism," *IEEE Access*, 2021.

[9] P. M. Sharkey and D. W. Murray, "Delays versus performance of visually guided systems," *IEE Proceedings-Control Theory and Applications*, vol. 143, no. 5, pp. 436–447, 1996.

[10] K. Goossens, M. Koedam *et al.*, "Noc-based multiprocessor architecture for mixed-time-criticality applications," in *Handbook of Hardware/Software Codesign*. Springer, 2017.

[11] A. Cervin, D. Henriksson *et al.*, "How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime," *IEEE control systems magazine*, 2003.

[12] S. Mohamed, "Multiprocessor image-based control: Model-driven optimisation," *Eindhoven University of Technology*, 2022.

[13] S. Chakraborty and Q. Rao, "Introduction to the special issue on embedded systems for computer vision," *Leibniz Transactions on Embedded Systems*, vol. 8, no. 1, pp. 00–1, 2022.

[14] M. Haghi, F. Wenguang *et al.*, "Delay-based design of feedforward tracking control for predictable embedded platforms," in *American Control Conference (ACC)*. IEEE, 2019, pp. 3726–3733.

[15] P. Martí, R. Villa *et al.*, "On real-time control tasks schedulability," in *European Control Conference (ECC)*. IEEE, 2001.

[16] T.-W. Huang and L. Hwang, "Task-parallel programming with constrained parallelism," in *High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022.

[17] Z. Zhu, J. Lu, and S. Zhu, "Multi-rate kalman filtering for structural dynamic response reconstruction by fusing multi-type sensor data with different sampling frequencies," *Engineering Structures*, 2023.

[18] D.A. Bristow and M. Tharayil, and A.G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, 2006.

[19] Wijdeven, van de, J.J.M. and O.H. Bosgra, "Using basis functions in iterative learning control : analysis and design theory," *International Journal of Control*, vol. 83, no. 4, pp. 661–675, 2010.

[20] A. Nelson, A. B. Nejad *et al.*, "Comik: A predictable and cycle-accurately composable real-time microkernel," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.

[21] K. Goossens, A. Azevedo *et al.*, "Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow," *ACM SIGBED Review*, vol. 10, no. 3, pp. 23–34, 2013.

[22] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[23] "Xup pynq-z2," accessed: 14-02-2022. [Online]. Available: https://www.xilinx.com/support/university/xup-boards/XUPPYNQ-Z2.html

[24] A. C. Robotics, "Robot simulator CoppeliaSim: create, compose, simulate, any robot-coppelia robotics," 2020.