MASTER

Robustness analysis for distributed high-end servo control

Vaiyapuri, S.

Award date:
2014

Robustness analysis for distributed high-end servo control

Santhosh Vaiyapuri
0825851

5T746 Master thesis ES -E

Tutor: ir. Shreya Adyanthaya[1]

Supervisor(s): dr.ir. Jeroen Voeten[1,2]

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems Group

Eindhoven, 29th August 2014

[1]Eindhoven University of Technology
[2]TNO-ESI, The Netherlands

Acknowledgement

I would like to express my deepest gratitude to my advisor, Dr. Jeroen Voeten, for his support and guidance throughout the research. His mere presence in the team has inspired me to perform this project effectively.

My profound thanks to my tutor, ir. Shreya Adyanthaya for her invaluable advice during my research. In addition, I would like to express my sincere appreciation to Dr. Ramon Schiffelers and Dr. Arno Moonen for their support and help with the CARM 2G tools and planning of my thesis. Also, this project would not have been possible without the generous assistance of my colleagues at ASML.

I would like to dedicate this project to my father and mother. Without their encouragement, I would not have had a chance to study abroad. I would also like to thank my best friends, Vikram and Vignesh for their continued emotional support during the project.

Abstract

ASML is the world's leading provider of complex lithography systems for the semiconductor industry. Such systems consist of numerous servo control sub-systems. To design such control systems, a multi-disciplinary model-based development environment has been developed. It is based on a set of domain specific languages (DSLs) describing the servo application, the execution platform and the mapping of the application on the platform. These models are used to automatically schedule the servo tasks of a control application on a multi-processor, multi-core execution platform. Currently schedules are computed under the assumption that communication between servo tasks is timeless. In reality, this is not true since tasks communicate via a packet-switched communication network based on RapidIO technology. Due to communication contention, the communication times may vary significantly. This implies that deadlines that are met according to the scheduler, may be violated in reality when the schedules are executed on an ASML Twin-Scanner. To improve predictability, ASML is working towards robust scheduling, implying that schedules are able to tolerate communication-time variations, without affecting the variation in end-to-end latencies and thereby minimizing the probability of deadline misses.

A computation combined with simulation based approach has previously been developed at ASML assuming that communication is instantaneous. In this project, this approach is extended with the ability to predict the impact of communication time variations. To this end, the simulation part of the combined approach that transforms DSLs into an executable model in POOSL is extended such that the model incorporates the behavior of the packet-switched RapidIO network. This model allows the stochastic communication behavior to be simulated. The existing computational part of the combined approach computes completion time bounds assuming instantaneous communication. In order to take communication into account, we adopt another previously developed contention analysis algorithm. This algorithm computes the additional waiting time owing to communication contention on shared communication resources. However, this algorithm assumes infinite buffers in the RapidIO network and does not compute the waiting time due to a system wide phenomenon known as back-pressure that occurs in a RapidIO communication network. In this project, this algorithm is extended with a conservative way of estimating back-pressure. A combination of these two adaptations on the simulation part and computation part allows us to perform communication aware robustness analysis. This analysis technique is applied to various ASML stacks of motion control applications and validated against measurement data.

List of Figures

Contents

Chapter 1

Introduction

ASML is the world's leading provider of wafer scanners for the semiconductor industry. Wafer scanners carry out a crucial processing step in the manufacturing of Integrated Circuits (IC) manufacturing. A scanner is an optical machine that shines a light onto a mask (containing the pattern to be printed), reduces the image by a factor of four after which it is exposed to a silicon wafer. After the pattern is exposed to a photo-sensitive layer (resist) that is deposited on the wafer, the wafer is processed through various steps (ion implantation, etching, polishing, deposition) after which the lithographic step is repeated. There are about 20-90 lithographic steps depending on the dimensions and the type of the IC to be manufactured. The name *scanner* is a short version of *step and scan system* as there are stepping movements (to expose the mask multiple times) and scanning movements (to expose different portion of the masks).

Recent requirements by customers have put forward the need for unprecedented precision in those machines. In order to improve precision at higher speeds, the industry has turned towards active imperfection correction of components and (sub) systems. Active imperfection correction is realized in the form of real-time embedded servo control systems.

1.1 Servo Control

Control theory is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems with inputs and outputs. Feedback controls are widely used in modern automated systems. A feedback control system consists of five basic components: (1) input, (2) process being controlled, (3) output, (4) sensing elements, and (5) controller and actuating devices. These five components are illustrated in Figure 1.1. A typical example of a servo control system is an automated heating system. The input (also called reference) is the desired temperature setting for a room given by the user. The process being controlled is the heater. The output is the variable of the process that is being measured and compared to the input; in the heating system, it is room temperature. The sensing elements are the measuring devices used in the feedback loop to monitor the value of the output variable. The purpose of the controller and actuating devices in the

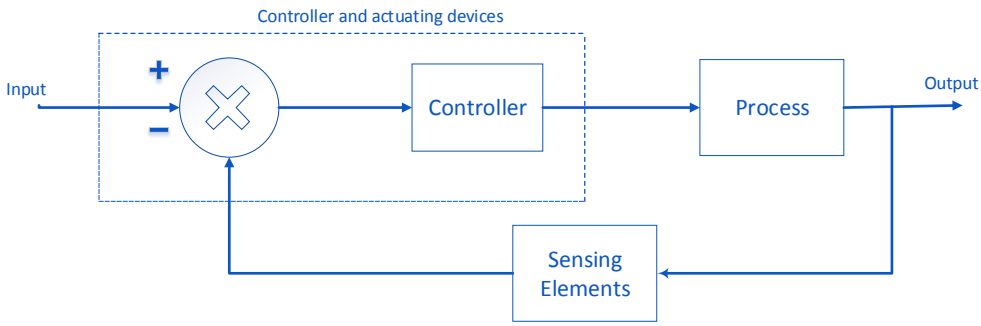


Figure 1.1: Servo Control

feedback system is to compare the measured output value with the reference input value and to reduce the difference between them. In general, the controller and actuator of the system are the mechanisms by which changes in the process are accomplished to influence the output variable. When the output (room temperature) is below the set point, the switch turns on the heater. When the temperature exceeds the set point, the heat is turned off. The term closed-loop feedback control or servo control is often used to describe this kind of control system.

1.2 Wafer Stage

In this project, the servo control of the wafer stage component inside AMSL's wafer scanners (see Figure 1.2) is analyzed. The wafer stage positions the wafer underneath the lens, such that areas on the wafer can be printed with the feature pattern present in the mask. The wafer stage consists of a fine positioning unit ("short stroke") and a coarse positioning unit ("long stroke"), plus various additional control units to support the positioning. Two wafer stages exist in the system. The first one measures the entire wafer. The second one positions the calibrated wafer for the actual exposure. In total, the control system for these two wafer stages comprises over 250 sensors and actuators, and over 4000 control tasks for tens of control networks. It forms a hard real-time system with computation latencies required to be in the microseconds range at most [1].

These servo control systems consist of control applications that are mapped onto an execution platform. Control applications read values from sensors, perform a particular computational task and send the results to the actuators. This is repeated in a periodic fashion. In reality, the ASML wafer scanners have hundreds of such applications, each with hundreds of control tasks, sensors and actuators. This complexity grows with each new generation of the scanners. To obtain the required performance needed by customers, control tasks have to run at high rates and have to satisfy stringent latency requirements. These requirements are increasingly tightened from one generation of machines to the next. Also, the number of dependencies between control applications have increased significantly due to an increased number of physical subsystem interactions. They often have late changes in

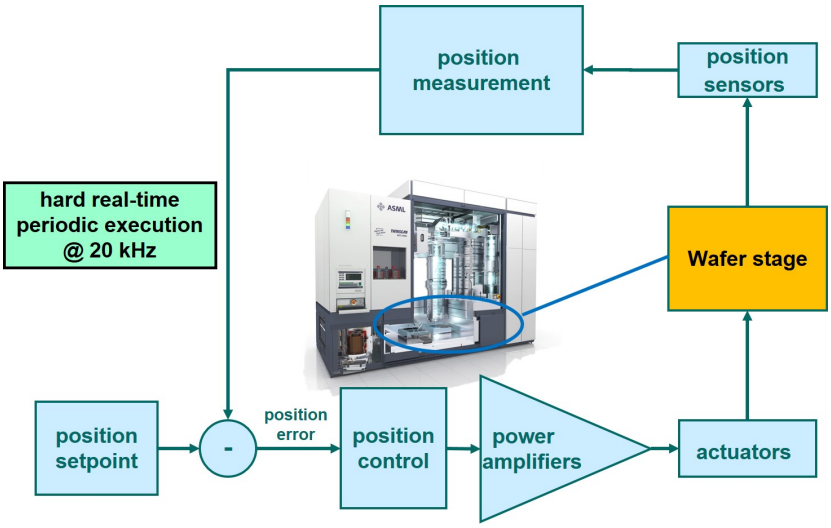


Figure 1.2: ASML TWINSCAN NXT wafer stages control

control requirements. These late changes and increase in complexity can result in timing performance problems that show up only during the integration phase, which threatens the time-to-market and time-to-quality constraints and also results in design iterations which are costly. To support late changes in the development process, execution platforms are desired to be reusable and reconfigurable. This leads to a need for a platform based design process that clearly separates the concerns between application and platform.

1.3 Model based design space exploration

Model-driven design-space exploration is an approach with which design engineers can predict the past and explore the future of an embedded system. This approach constructs executable models that separate the embedded system application from the execution platform on which it is mapped. The models can be calibrated with available measurements or approximations to validate and improve the model's predictive power.

The approach can be split into two steps,

1. Predict the past :
    - Model the system by decomposing it in an application, a platform and a mapping view.
    - Calibrate the model with available measurements and validate its predictive power.
2. Explore the future :
    - Explore different alternatives of application and platform.
    - Optimize application functionality, platform and mapping.

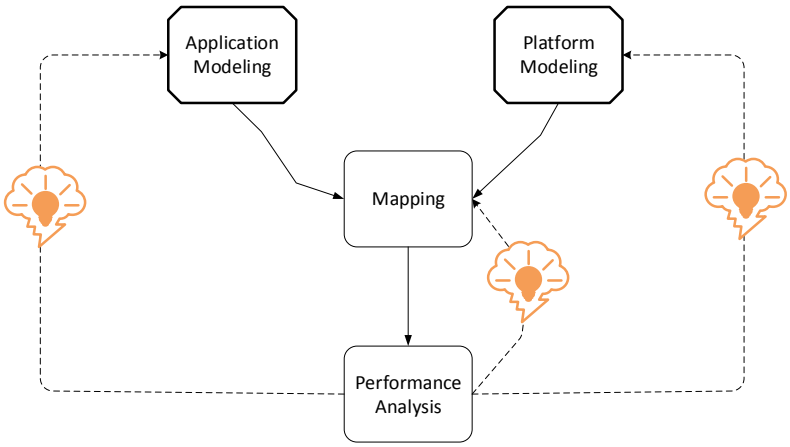


Figure 1.3: Y-Chart approach to design space exploration

Design space exploration often follows the Y-Chart approach [2]. The approach for Model-driven design-space exploration is shown in Figure 1.3. The concerns on functionality, platform, and mapping are separated. Models of applications and platforms are made and an explicit mapping step binds tasks and schedules in an application model to execution platforms in a platform model. The mapping can be evaluated in terms of performance, area, and power consumption. Results from the evaluation may trigger further iterations of the mapping. The design engineer has the choice to modify the application and the selection of platform building blocks, or the mapping strategy. After a series of iterations and modifications, the optimized application mapped on platform is found.

1.4 CARM framework

CARM (Controller Architecture Reference Model) framework [3] is a framework that is being used at ASML to design the servo control systems for the wafer stage and other sub-systems. To support model based design space exploration, domain formalization, detailed analysis and code generation a second generation of CARM is being actively developed. CARM based design-space exploration approach allows rapid exploration of alternatives for optimization of timing performances by separating the embedded control application from the execution platform on which it is deployed.

The core of CARM 2G relies on a set of domain-specific languages (DSLs) that formalize in a coherent and consistent unambiguous way the domain concepts governed by the different CARM layers. This framework is developed to accommodate the ability to design with multi-core processor boards and increasing sampling frequencies of the controllers. The design process using CARM framework relies on three phases:

1. **Specification** By means of a multi-disciplinary integrated development environment (IDE), formal models are developed that describe:

(a) *Control Logic :* The control logic in terms of servo networks and transducers (sensors and actuators).

(b) *Computation Platform :* Single/multi-core processors and FPGAs.

(c) *Mapping :* Deployment and scheduling of the control logic on the computational platform.

Incorporating different levels of abstraction into the DSL framework reduces the complexity of the (parallel) design process. The IDE provides the design engineers with feedback on the models early in the design process improving the quality of the designs.

2. **Analysis** Analysis models are used for making key decisions for which it has to be proven that a design will work. Verification of the designs reduces the risk on errors during integration, or eliminates the integration effort at all. In CARM, formal specification models are used to analyze worst-case and best-case timing. The formal specification models can also be transformed into executable models[1] which can be simulated to verify upfront whether the timing requirements are met and to predict the effect of control loop changes and/or platform changes. This thesis project contributes the analysis phase of the design trajectory.

3. **Construction** The formal specification models that were developed at specification phase and analyzed at the analysis phase are used for;

   (a) *Code Generation :* The formal specification models are used during the build by code generators to generate the actual software that is executed on the lithoscanners

   (b) *During Startup :* During start-up of the lithoscanners, the formal specification models are used to initialize the servo controllers and computation platforms, and to schedule control blocks on the processors.

The Y-Chart approach is used to perform design space exploration in the CARM framework. The definition of the applications, the platforms on which they are deployed and their mappings are contained in DSLs. A transformation step is performed to generate executable models. The Software/Hardware Engineering method and accompanying tools with the underlying formal modeling language POOSL [4] was employed as basis of the executable model architecture. The POOSL language and underlying simulation engine allow for rapid analysis of timing performance through simulation of these models.

The results of the quantitative performance analysis yields valuable feedback on the adequacy of the platform, the performance of the application, or the effectiveness of the mapping strategy [1]. Based on the insights gained, the procedure can be repeated in an iterative way until a feasible platform for the complete set of applications is found.

[1] An executable model allows stochastic behavior of an embedded system to be analyzed by simulation.

1.5 CARM Layers

CARM enables specification of the control logic and the execution platform at different levels of abstraction by using DSLs. Figure 1.4 shows DSLs employed in CARM 2G by classifying them into application, platform and mapping layers.

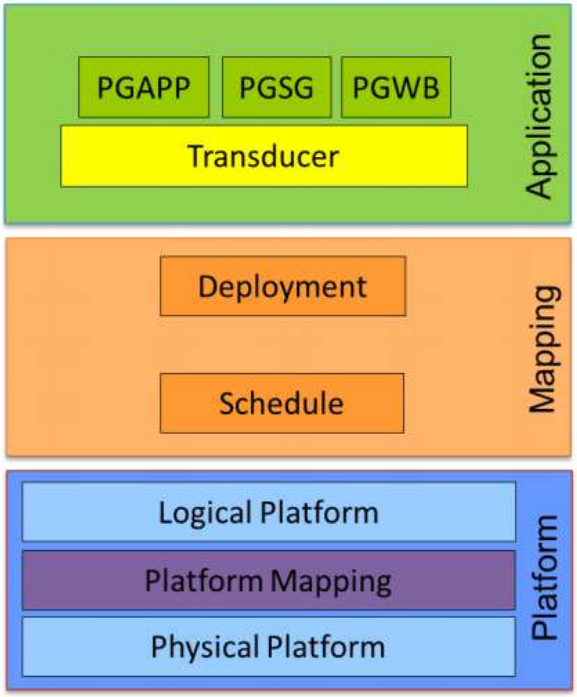


Figure 1.4: CARM Layers and their DSLs

1.5.1 Application layer

The application layer contains the description of the control application. It consists of the control logic described by means of the PGAPP, PGSG, and PGWB languages, and the description of the transducers in the transducer language. Networks of servo and transducer groups are defined in the PGAPP language, servogroups in the PGSG language, control blocks in the PGWB language and transducers in the Transducer language. By means of the transducer language, electrical and mechanical transducers can be defined. Transducers can be composed of multiple blocks, resulting in transducer groups.

1.5.2 Platform layer

In the platform layer, the execution platform of the lithoscanners is described. It consists of 3 domain-specific languages.

Physical Platform Language

The physical platform language contains a description of (a subset of) the hardware and their physical connections as present in the lithoscanners. They are;

- single and multi-core High Performance Process Controllers (HPPCs);
- input-output boards (IOBoards);
- electrical and mechanical transducers;
- network switches and connectivity;

A model in the physical platform language represents an instance of a platform. This language also contains the configuration data of the physical platform at hand. An example of configuration data for the physical platform is the information at which rate the IOBoards are triggered to send (sensor) data. This configuration data depends on the application that has to be executed as well as the physical limitations of the hardware. One example of a physical hardware limitation is the maximum frequency at which an IOBoard can acquire sensor data. This is also contained in this language.

Logical Platform

The logical platform language abstracts from the physical properties of the hardware such as location, IOboard types, HPPC processor types, network connections etc. Concepts contained in this language are;

- **Worker :** entity that can perform computations, abstracting from the real computing hardware (HPPC). A worker contains one or more processingUnits.
- **ProcessingUnit :** entity abstracting from a processor/core
- **IOWorker :** entity abstracting from IOBoard type, and the location of Transducers
- **Connection :** entity abstracting from network type and topology. It is used for data communication between workers and between workers and IOWorkers.

Platform Mapping

The platform mapping language contains the mapping from logical platform elements to physical platform elements by defining directed associations between them. Typical examples of platform mapping are,

- Worker(s) to HPPC;
- IOWorker to IOBoard(s);
- Connection to Network Elements.

1.5.3 Mapping layer

The mapping level describes the mapping of elements from the application language to elements from the logical platform language (shown in Figure 1.5). The *Deployment language* contains associations to the control application and the logical platform language. Typical mappings are;

- ServoGroup to Worker;
- ControlBlock to ProcessingUnit;
- Channel to Connection;

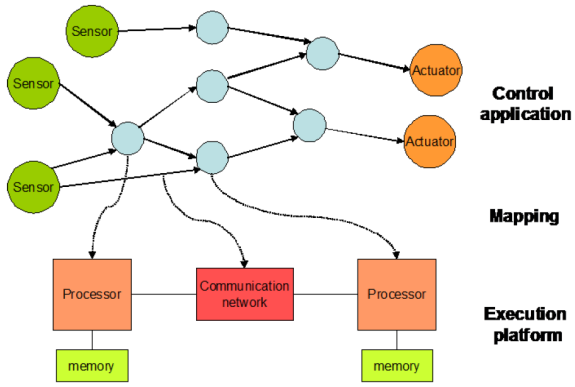


Figure 1.5: Mapping of elements from the application language to elements from the logical language

After scheduling, all controlblocks should be mapped to ProcessingUnits. Furthermore, the controlBlocks should be ordered in their calculation order that is constrained by the data dependencies present in the servo group model. This information is captured in the *Schedule language*. More information regarding scheduling is given in Section 1.6.

1.6 Scheduling

The computation of a schedule is based on the information in the models from the CARM framework. As a first step, essential scheduling information is extracted from the application and mapping DSLs. This is done by a model-to-model transformation that constructs a block dependency graph. The dependency graph specifies control blocks and their dependencies. Latency requirements of the application are transformed into corresponding deadlines of blocks. In addition, a control block is aware of the processor it is deployed on. It also aware of its execution time.

In the second step, the essential information in the dependency graph is used to compute schedules for each multi-core processor in the platform. The two steps are shown

in Figure 1.6. After computing a schedule, all controlBlocks should be mapped to ProcessingUnits. Furthermore, the control blocks should be statically ordered in their calculation order that is respecting the data dependencies present in the servogroup model. The scheduling is done using list Scheduling with Earliest Due-date First Heuristic [5].

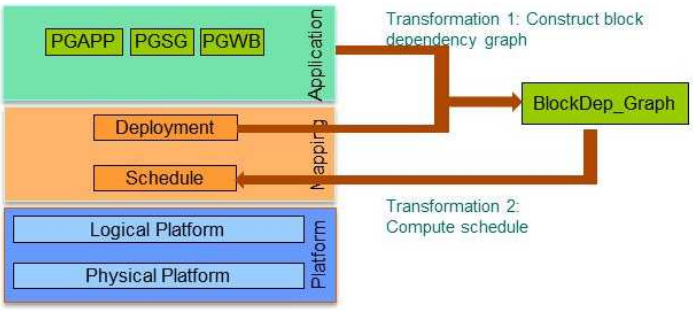


Figure 1.6: Scheduling

1.7 Preliminaries

For a set X, we use X^* to represent the collection of lists with elements from X.

- *Application :* An application is a directed acyclic graph (DAG) $G = (T, D)$ with a set of tasks T and a set of task dependencies $D \subseteq T \times T$.
- *Resources :* The multiprocessor platform that the application is bound to consists of processors called *resources*. R represents the set of resources of the platform.
- *Dependency :* $(a, b) \in D$ denotes that task b is allowed to start its execution only after the completion of task a. D is the collection of task dependencies.
- *Task :* A task $t \in T$ is defined by a tuple $t = (e_t, r_t, d_t)$ where e_t denotes the execution time of t, r_t denotes the resource that t is bound to and d_t denotes the deadline of t.
- *Schedule :* A (static-order) schedule S is a mapping $S : R \rightarrow T^*$ from the set R of resources to ordered lists of tasks from T. S is a schedule for application $G = (T, D)$ iff *a*) every task in T appears once in the ordered list of exactly one of the resources in S, *b*) S respects the task bindings and, *c*) dependencies.
- *Execution Time :* The execution Time is defined as the time that a task requires to complete its execution on the execution platform that it is mapped upon. For a platform with general purpose multi-core processors (such as the platform that exists in ASML machines), the execution times fluctuate. Typically the execution time of a task t can be characterized as a random variable (e_t) of a continuous probability distribution. The bounds within which the execution time of a task fluctuate can be expressed in terms of an interval. $E(t) = [a, b]$ denotes that the execution of task t requires at least a and at most b time units. $bc(E(t)), wc(E(t))$ represent the best-case execution time and the worst-case execution time of task t respectively.

- *Completion Time :* The completion Time is defined as the time at which a task will complete its execution. Typically the completion time of a task t can be characterized as a random variable (c_t) of a continuous probability distribution. The completion time of a task can also be represented as an interval ($C(t)$) similar to the execution time of a task. The best-case completion time and the worst-case completion time of the task t are represented as $bc(C(t)), wc(C(t))$ respectively.
- *Feasible schedule :* A feasible schedule is defined as a schedule in which all the tasks meet their respective deadlines ($\forall t \in T : C(t) \leq dl_t$)

1.8 Towards robust scheduling

The current scheduler assumes constant execution times for control tasks and also instantaneous communication times. High performance special purpose platforms such as FPGAs and GPUs have architectures that are designed to be specific to the applications running on them and have the advantage of high predictability. As such, there is little variation in the execution times of the applications running on them. In ASML, the high cost of adapting legacy software to application specific platforms have led to a trend of designing complex embedded applications on general purpose platforms. With the advent of multi-core general purpose platforms, the timing demands of complex embedded system can be satisfied. However, these general purpose platforms suffer from low predictability and exhibit fluctuations in execution timings. Also, the contention in accessing the shared communication resource has led to fluctuations in communication delay. Hence, there is a need to cope with these fluctuations and to be robust in nature.

Scheduling and analysis of applications in classical real time approaches mostly take the worst case execution timings into account. If a static order schedule of an application meets its latency requirements in the worst case, it is highly robust. However in the ASML wafer scanner applications, it has been proven that execution timings vary in such a way that most likely (nominal) execution times show a huge difference with the worst case and are lying close to the best case execution times [6]. Scheduling of applications for the worst case always requires excessive resources to meet latency requirements. If scheduling is done for the nominal case, tasks can violate their latency requirements due to execution time variations. This raises the concern that schedules running on general purpose platforms must be *robust* to be able to cope with these execution time fluctuations with low probability of resulting in failures. To produce schedules that are maximally robust against execution time fluctuations, we need to design *robust schedulers*. This requires three steps,

1. Defining a robustness metric for schedules.
2. Developing a method for analyzing the robustness of schedules.
3. Extend the current scheduler to use robustness analysis to steer scheduling decisions.

The first step has already been finished. This thesis project contributes to step two. The third step is a work in progress.

1.9 Outline

This report is organized as follows,

- Chapter 2 - Problem Definition: This chapter describes the challenges in the project and also about the prior work that was performed in order to tackle the challenges.
- Chapter 3 and Chapter 4 - Approach: This chapter presents the models and methods used to solve the problem. It also describes the motivation behind choosing those methods.
- Chapter 5 - Results: This chapter presents the results of the approach experimented on an ASML wafer stage stack. It also describes the insights gained regarding the robustness of a schedule.
- Chapter 6 - Calibration: This chapter describes about the calibration process. It describes how predicted results have been compared with measurements to perform the calibration process. It also describes other challenges that were triggered during this project.
- Chapter 7 - Conclusions: This chapter describes the validation of the approach under ASML industrial context and identifies the possible improvements to this project.
- Bibliography: Contains the references to relevant literature in this report.
- Appendix: This part contains the related work, terminology and software fragments that are relevant for the report.

Chapter 2

Problem Definition

2.1 Robustness Metric

2.1.1 Task completion time variability probability approximation

As nominal execution times of tasks are mostly closer to the best case execution time than the worst case execution time, the probablility density function of the task execution time is mostly not normally distributed but right skewed in nature. Apart from being skewed, we have information on the bounds of the distributions in the form of best-case and worst-case task execution times. The combination of the skewness and the boundedness requirements is met by the PERT distributions. A PERT distribution (Figure 2.1) is derived from the beta distribution and is defined by three parameters, namely the minimum (*min*), the mostly likely value (*mode*) and the maximum (*max*). A variant of the PERT distribution (*Modified PERT*) allows producing shapes with varying degrees of uncertainty by means of a third parameter, gamma (γ), that scales the variance of the distribution.

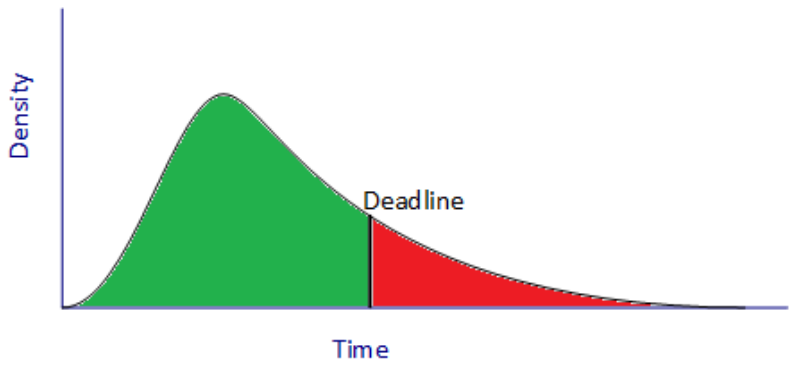


Figure 2.1: An example pert distribution showing completion time distribution of a task and its deadline

2.1.2 Deadline miss probability

The deadline miss probability of a task t is defined as the probability of the task missing its deadline. Given a completion time distribution for a task A, the probability of the task missing its deadline dl_A is given below,

$$c = \int_{dl_A}^{\infty} p_A^c(t)dt \tag{2.1}$$

where, p_A^c is the probability density function for the completion time of task A. sThis is the the red portion of the example distribution shown in Figure 2.1.

2.1.3 Robustness of a task

The robustness of a task in a schedule can be derived from the deadline miss probability. The higher the probability of a deadline miss, the lower is the robustness of the task. The robustness of a task is complementary to the deadline miss probability as given below,

$$R_A = \mathbb{P}[c_A < dl_A] = \int_0^{dl_A} p_A^c(t)dt \tag{2.2}$$

This is the green portion of the example distribution shown in Figure 2.1.

2.1.4 Expected number of task deadline misses

Given the probabilities of deadline misses per task, we define a random variable X to express the number of tasks that miss their deadline in a schedule. The probability distribution of this random variable is a discrete distribution with probability values for any x tasks missing their deadlines.

$$p(X = x) : \textit{Probability that x tasks miss their deadlines} \tag{2.3}$$

The expected value of this random variable gives the expected value of the number of tasks that miss their deadlines in a schedule S. It can be derived by taking the sum of the deadline miss probabilities of its constituent tasks. This is given in Equation 2.4. This Equation holds even if the tasks are dependent on each other.

$$EX = \sum_{A \varepsilon T} \mathbb{P}[c_A > dl_A] = 1 - \sum_{A \varepsilon T} (1 - R_A) \tag{2.4}$$

2.1.5 Robustness of a Schedule

The robustness of a schedule is a measure of tolerance of a schedule to variations in the execution times of tasks. A measure of the robustness of a schedule R_S is the normalized

expected number of tasks meeting their deadline in the schedule given in Equation 2.5.

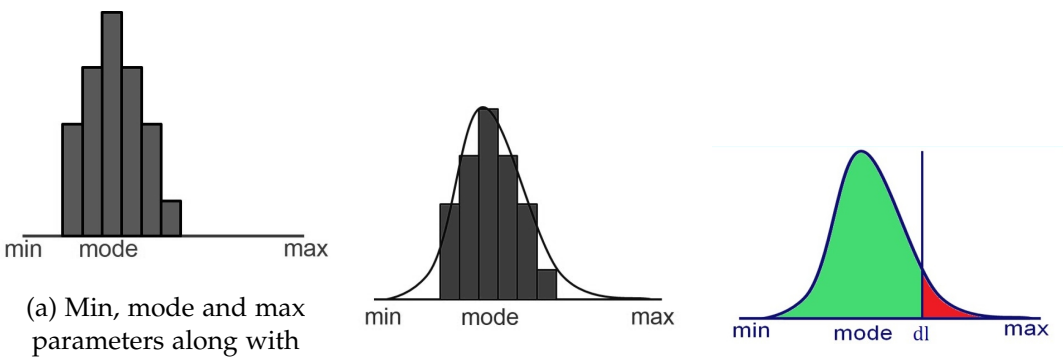
$$R_S = 1 - \frac{EX}{|T|} \tag{2.5}$$

Note that this metric generalizes task robustness. Hence, this measure can be applied to any subset of a schedule to find its robustness.

2.2 Robustness Analysis Overview

To compute the robustness of a schedule, we need to obtain the completion time distributions of all tasks. To this end, we require task execution time distributions. These typically need to be approximated using limited measurements. A curve fitting approach is used to fit a PERT distribution on histograms obtained from measurements. Once we have the execution time distributions, we need to compute the completion time distributions. Due to the computational complexity to perform max and plus operations on distributions [7], this cannot be done entirely analytically. On the other hand, simulation (using the PERT execution time distributions) produces insufficient mass in the tails to approximate the completion time distributions accurately.

Following [8] we use a combined analytical and simulation based approach to approximate the completion time distributions of tasks with the same curve fitting approach as used for task execution times, as shown in Figure 2.3. The histogram and bounds that results from the simulations and analytical computations respectively (Figure 2.2a) are used to fit a a PERT distribution by using a curve fitting approach (Figure 2.2b). Once the completion time distributions are obtained, task deadline miss probabilities are computed using Equation 2.1. Consequently, task robustness can be computed using Equation 2.2. Robustness of a task is the green shaded portion under the curve shown in Figure 2.2c. The schedule robustness metric is then found by computing the expected number of task deadline missing tasks of the schedule using Equation 2.5. Extending the current scheduler to use robustness analysis is a work in progress and is out of scope for this project. This work focuses on extending the combined analytical and simulation based approach with communication.



(a) Min, mode and max parameters along with histogram

(b) Curve fitting

(c) Finding robustness

Figure 2.2: Robustness Analysis approach

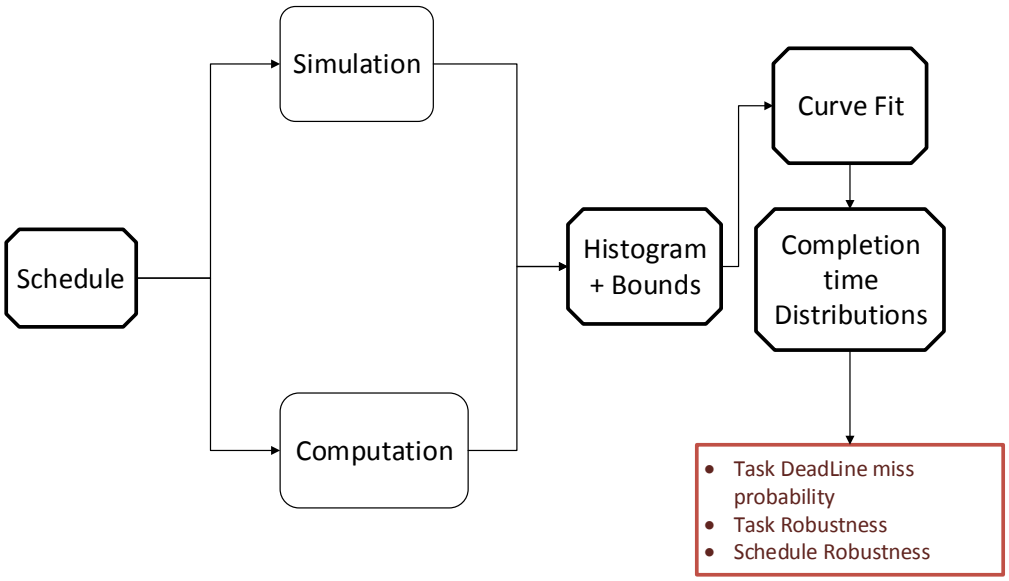


Figure 2.3: The complete Approach

2.3 Problem Statement

The statement of the problem being dealt with in this project is:

"Given static-order schedule S, its communication platform and execution platform, what are the completion time distributions of its constituent tasks?"

The focus of this work is to get the completion distribution of every tasks in a schedule taking communication time variability into account. Specifically, the communication timing obtained must include the waiting time due to contention of shared communication resources and the waiting time due to a system wide phenomenon called back-pressure that occurs on shared communication resources.

2.4 Prior Work

Prior work has been done inside ASML to develop the combined analytical and simulation based approach [8] to derive the completion time distributions of the constituent tasks of a schedule. However, this work assumes that communication is instantaneous. This section explains the prior work briefly,

2.4.1 Simulation

Running simulations with samples from individual task execution time distributions produce individual task completion time sample values, with which a histogram can be constructed. A histogram is a graphical representation of the distribution of data. It is an

approximation of the probability distribution of a continuous variable. For simulation purpose, a model is built for the servo schedule with all the necessary information. The DSL models in CARM are transformed to schedule languages represented by *ds_graph* and *ds_schedule* file extensions as explained in Section 1.5. This schedule model is transformed using a transformation algorithm to a executable POOSL model. Two types of POOSL models can be obtained using the algorithm. A brief explanation of them is given below.

Model with non-synchronized workers

Figure 2.4 shows the architecture of a POOSL schedule model with non-synchronized workers. Each block is a process. The connections between processes are the channels on which they communicate by sending messages. Tasks in the schedule are represented as Task processes in POOSL. Their dependencies in the schedule are transformed to channels with messages passing through. For each Worker, a Trigger process, a Queue process and a WorkerFinisher process are added. A Trigger process is positioned at the beginning of a Worker-schedule. Following the Trigger process is a Queue process. The Queue process connects to the first Task process of every sequence in the worker on the same Worker-schedule. A Schedule Finisher process is positioned at the end of a Worker-schedule, to receive output messages from the last task of every sequence on the Worker-schedule.

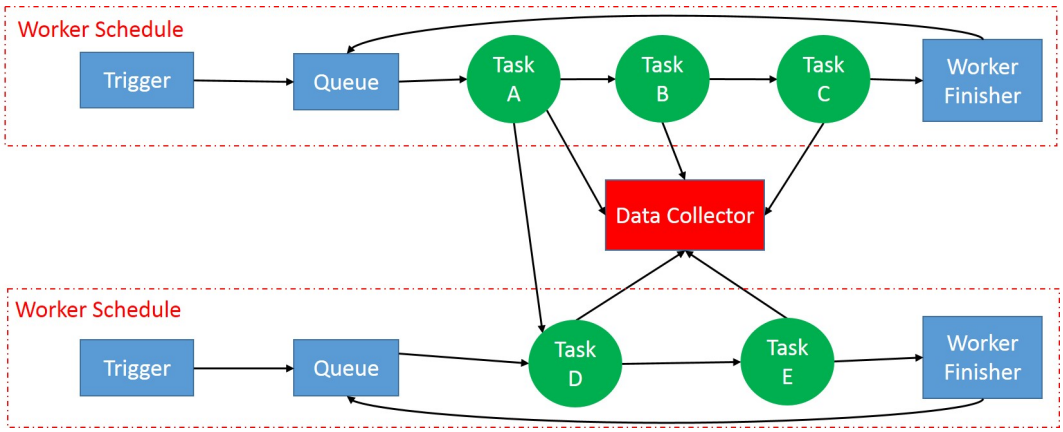


Figure 2.4: Architecture with non-synchronized workers

The working of a POOSL schedule model with non-synchronized workers for one sample period can be understood from Figure 2.4. The series of steps is explained below:

1. A Trigger process produces tokens periodically (the time period is specified in the transformation algorithm), to trigger the start of the sample round.

2. A Queue process receives a token from the Trigger process and stores the token in a queue.

3. A Task process receives one input token from each of its predecessors. Consequently a sample value is drawn from a PERT random generator. Then, it delays for an

execution time specified by the sample value. After the delay, the process produces one output token to each of its successor(s). The completion time is reported to the dataCollector class.

4. When a WorkerFinisher process has received a token from the last task of each sequence on this worker, it sends a message indicating to start the next sample round to the Queue process.

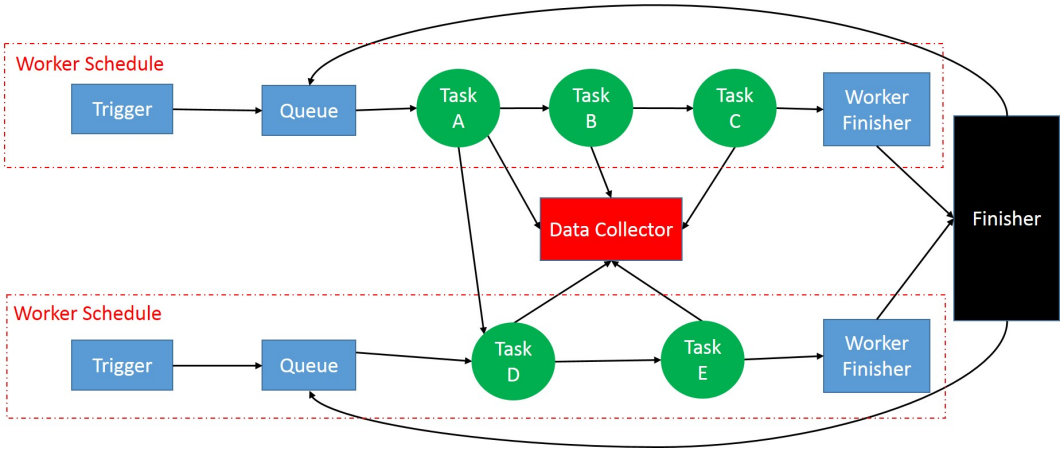


Figure 2.5: Architecture with synchronized workers

Model with synchronized workers

This model is similar to the previous model except that at the end of a servo schedule, a Finisher process is added. To be able to compute task deadline miss probabilities of each task, in this type of POOSL model all the worker sequences are made to start only when the previous sample run of all the worker sequences have been completed. However, in reality, the workers are triggered independently. If the last task of a worker over-run the triggering period in a sample run, the worker can start its next round only after the task completes its execution. This over-run time for the different workers is not the same under all circumstances. In such a situation, the best case and worst case completion times for a task computed without any over-runs may become invalid. As a result, task deadline miss probabilities can not be computed using this approach. Hence, the model with synchronized workers is used for analysis of robustness. The POOSL schedule model with non-synchronized workers is useful for visualizing the effect of deadline misses of a task in a sample period on its next period using a Gantt chart.

The working of a POOSL schedule model with synchronized workers for one sample period can be understood from Figure 2.5. The series of steps is explained below:

1. A Trigger process produces tokens periodically (the time period is specified in the transformation algorithm), to trigger the start of the sample round.

2. A Queue process receives a token from the Trigger process and stores the token in a queue. For the initial sample round, the Queue process checks if the token queue is not empty and if this is true, it sends a token to the first task of the sequence. Otherwise, the Queue process waits for a token from a Trigger process. For the other sample rounds, even if the Queue is not empty, the Queue process sends out tokens only when it receives the message from the finisher process indicating that the previous sample round is finished.

3. A Task process receives one input token from each of its predecessors. Consequently a sample value is drawn from a PERT random generator. Then, it delays for an execution time specified by the sample value. After the delay, the process produces one output token to each of its successor(s). The completion time is reported to the dataCollector class.

4. When a WorkerFinisher process has received a token from the last task of each sequence on this worker, it sends the token to the Finisher process.

5. After the Finisher process has received a token, it sends a message to every Queue process, indicating that one sample round has been finished.

Execution time distributions of tasks

As described earlier, for each sample run a execution time sample value for every task is drawn from a PERT random generator. However the three parameters (min, mode, max) needed to construct a pert distribution is found from execution time histograms. Execution time histograms are constructed by performing limited measurements on a actual ASML wafer stage component. These parameters are stored in a database and the model transformation algorithm reads from this database and feeds the information to the POOSL models. Currently, the database do not contain the execution time histogram mass. Hence, we cannot perform a curve fit to obtain the execution time distributions. But the three parameters (min, mode, max) are present in the database. We approximate the task execution time distribution by considering the gamma parameter to be four. This is the default value for a standard PERT distribution.

2.4.2 Analytical computation

The analytical method is responsible for computing the best case(minimum) and worst case(maximum) completion times of a task. For the first task of each worker sequence, the best case and worst case completion times is equal to its best case and worst case execution times respectively if the task starts at time 0. If the worker sequence is delayed by an offset, then the the best case and worst case completion times is equal to its best case and worst case execution times added with the offset value respectively.

$$bc(C(t)) = Max(bc(C(t'))) + bc(E(t)) \tag{2.6}$$

$$wc(C(t)) = Max(wc(C(t'))) + wc(E(t)) \tag{2.7}$$

Equations 2.6 and 2.7 are used to compute the best case and worst case completion times for every task except the first task of a worker sequence. The operators $Max(bc(C(t')))$ and $Max(wc(C(t')))$ refer to maximum of best case and worst case completion times of all predecessor tasks of task t obtained using max-plus algebra.

2.4.3 Problem

We can see from Section 2.4.1 that in the architecture of the POOSL model, the communication delay is not taken into account. Also from Section 2.4.2 we can see that the equations to compute the bounds do not take the delay due to communication into account. Due to the assumption of instantaneous communication, the completion time intervals obtained from this approach are not conservative. Hence, the goal of this project is to obtain a *conservative combined analytical and simulation* based approach that is *communication aware* to predict the deadline miss probabilities such that they are closer to reality.

2.5 Communication Tasks

The communication interconnect used in ASML scanners consists of several connected packet switches. On extending the prior work, the goal of this project is to extend the

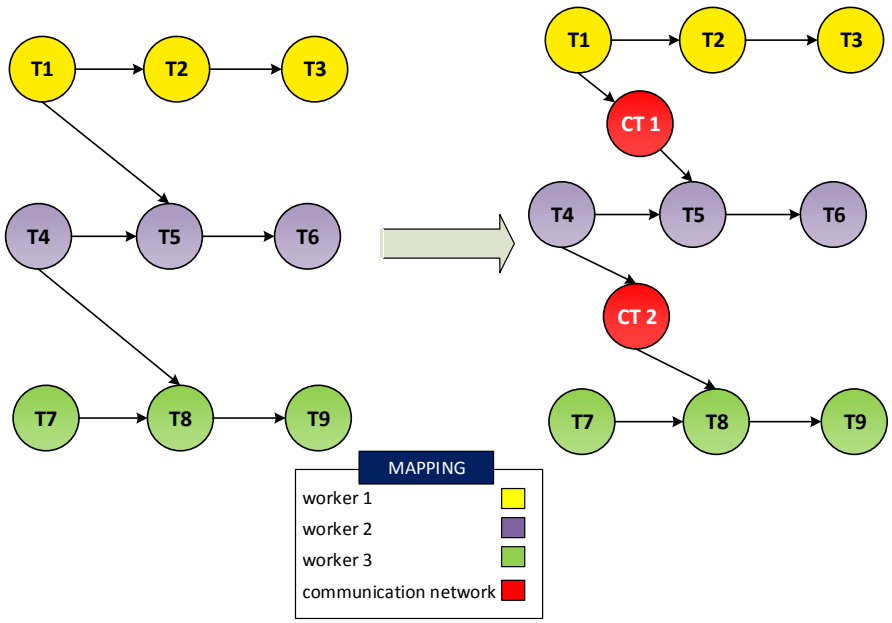


Figure 2.6: Adding Communication Tasks

current robustness analysis approach to be *communication aware*. This process is started by adding communication tasks (CT) for each worker-to-worker dependencies and worker-to-IOworker dependencies as shown in Figure 2.6. The different colors in the figure show

tasks mapped on different resources. The red colored tasks represent the communication tasks.

2.5.1 Challenges

The communication tasks are *not statically ordered* like the normal control tasks. They are mapped on a communication network that is an interconnect of several packet switches. The communication network is explained in detail in Section 2.6. These packet switches are switching systems that are shared and they provide *on-demand* communications technology. In particular, they work with non-monotonic arbitration policies like First-Come-First-Served (FCFS) scheduling. If we perform limited measurements of communication delay, the execution time distributions of communication tasks will not be valid because the measurements cannot represent all possible enabling instances of the communication task. An extension of the current robustness analysis approach is needed to accommodate these communication latencies.

Simulation Approach

As mentioned earlier, the measured execution time distribution of communication tasks are not valid. Hence, we cannot feed the communication tasks in the POOSL model with PERT random samples as it has been done in the prior work explained in Section 2.4.1. However, in POOSL models we can find the completion time of communication tasks directly by mapping the task to the appropriate communication resource and *simulating the communication behavior*. For this a complete communication network model must be built for the appropriate ASML wafer stage stack when analyzing robustness. To build a accurate model the communication resource must be studied in detail. This is presented in Section 2.6.

Analytical Approach

Analysis methods that conservatively analyze FCFS based communication systems are often based on state-space exploration, which is not scalable due to its inherent susceptibility to combinatorial explosion. For industrial applications a scalable timing analysis method is needed. Prior work (presented in Section 2.7) has been done to develop a scalable timing analysis method on periodically restarted directed acyclic graphs (DAG), that can provide conservative bounds on task timing properties when shared resources with FCFS scheduling are used. They work by expressing task enabling and completion times in intervals, denoting best-case and worst-case timing properties. Contention on the shared resources can be estimated using conservative approximations. But if we study the communication interconnect in detail (presented in Section 2.6), we can see that a communication task does not use a single shared resource, Instead the detailed communication path reveals a interconnection between switches. The interconnection between switches leads to a system wide phenomenon called as back pressure (presented in Section 2.6.4). Hence, this scalable contention analysis approach has to be extended to adapt to multiple shared FCFS resources such that back pressure is taken into account by the algorithm.

2.6 The communication resource

Communication switches are based on Serial RapidIO Specification [9]. RapidIO is a high performance, low latency packet-switched interconnect technology for embedded systems.

2.6.1 Overview of Internal Switching Fabric

The Internal Switching Fabric (ISF) is a crossbar-switching matrix at the core of the RapidIO switch. It transfers packets from ingress ports to egress ports and prioritizes traffic based on the RapidIO priority associated with packet and port congestion.

The ISF has the following features [10]:

- full-duplex, 16-port, line rate, non-blocking, crossbar-based switching fabric;
- 10 Gbit/s fabric ports;
- it manages head-of-line blocking on each port;
- buffers hold eight packets per ingress RapidIO port;
- buffers hold eight packets per egress RapidIO port;
- cut-through and store-and-forward switching of variable-length packets is supported with a maximum packet size of 256 bytes;

2.6.2 Functional behavior

When RapidIO packets arrive at the ingress ports, the switch performs several tests to ensure the packet is valid. If a packet passes these tests, the ingress port consults its Destination ID Lookup Table to determine the egress port for the packet. The ISF transfers entire packets without interruption.

The ISF is a crossbar switch, which means that an ingress port can only *send one packet at a time* to the ISF, and an egress port can only *receive one packet at a time* from the ISF. However, the ISF can *simultaneously transport packets from multiple disjoint ingress, egress port pairs*. Since many ingress ports can attempt to send a packet to the same egress port, queuing is required at the ingress ports. Special arbitration algorithms at both the ingress and egress sides of the fabric ensure that head-of-line blocking is avoided in these queues by packet overtaking. Queuing is also required at the egress ports. Packets can accumulate when an egress port has to retransmit a packet due to an error.

Since many ingress ports can attempt to send a packet to the same egress port, *queuing* is required at ingress ports. Arbitration algorithms are present at both ingress and egress port sides to ensure that head-of-line blocking is avoided. Queuing is also present in egress ports. Packets can accumulate when an egress port has to retransmit a packet due to a CRC

error for example. It is also needed when a higher-bandwidth ingress port sends traffic to lower-bandwidth egress port.

Figure 2.7 shows a conceptual diagram of a ISF. It shows the arbiters at each port, the connected mesh and the relationship between them.

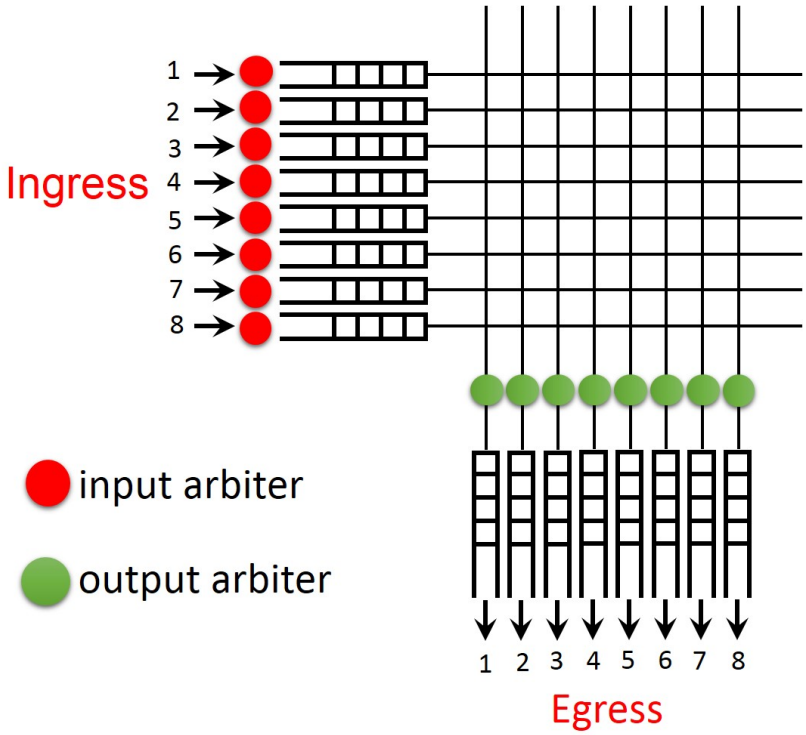


Figure 2.7: ISF conceptual diagram

2.6.3 Contention

Each switch has eight ingress port and eight egress port, as shown in Figure 2.7. Buffers hold eight packets per ingress and egress RapidIO port. When there is contention, the packets are queued in the ingress port. Congestion and contention can be understood by examples.

1. Contention example 1,
   - Ingress Port 1 is currently sending Packet-1 to Egress port 2
   - Ingress Port 4 wants to send Packet-6 to Egress port 2
2. Contention example 2,
   - Ingress Port 1 wants to send Packet-1 to Egress port 6
   - Ingress Port 4 wants to send Packet-5 to Egress port 6

- Ingress Port 7 wants to send Packet-3 to Egress port 6

In example 1, Packet-6 must wait for the Packet-1 transfer to finish before it has access to Egress port 2. In this case, Packet-6 must be queued inside the buffer of Ingress port 4. In example 2, since three ingress ports wants to send their respective packets to one port, only one packet can be sent and the other two have to be buffered. This packet selection is performed through a fair arbitration decision based on priorities.

The RIO switch handles four priority level. The arbitration based on priorities can stall transmission of lower priority packets due to the presence of higher priority packets. Currently, the ASML data packets are not assigned with priorities. Hence, contention analysis under priority based arbitration is out of scope for this project. Currently, all packets are sent with same priority and a round robin based selection is made. It should be noted that the abstraction of priorities make this a conservative approach even when priorities are used in the ASML applications. However, due to this abstraction, the predictions for data packets assigned with priorities will be over-conservative.

2.6.4 Back-pressure

The usage of buffers with finite capacities leads to system-wide queuing phenomenon called back-pressure. This happens when a buffer cannot queue any new packet, and hence the packet has to be buffered along the communication path. This queuing can ripple all the way back to the original input. This phenomenon is beneficial in the sense that it does not overwrite data packets and keeps the communication to be reliable (loss-less). But this creates dependencies and makes the timing analysis complex.

The existing contention analysis algorithm (Section 2.7) computes timing intervals in the presence of communication without taking back-pressure into account. In the subsequent chapters an extension to this algorithm including back-pressure is presented.

2.6.5 Odds of communication contention in ASML scanners

A typical ASML wafer stage platform includes,

- Eight RIO switches.
- Eight multi-core processors.
- About 250 sensors.
- Approximately 4000 control tasks.

Figure 2.8 shows a very simple example of contention that can happen in an servo application. Since the data packets from processor 1 and processor 2 have to reach the same egress port of switch-2, contention can occur. If more data packets from other processors

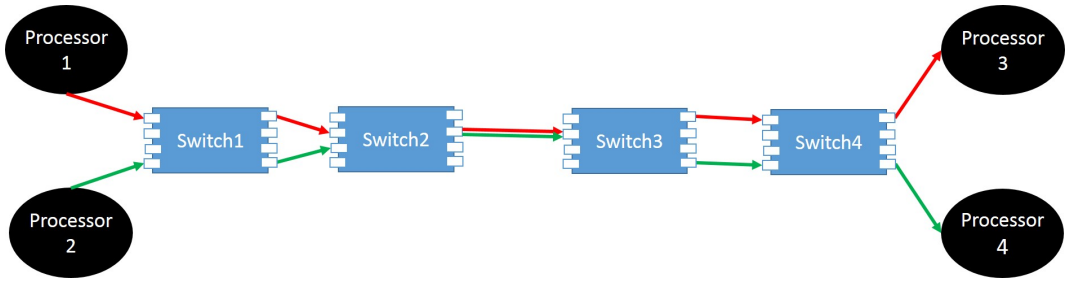


Figure 2.8: Contention in ASML wafer stages platform

also share the same egress port, the queue can get filled up and back-pressure will occur as explained in Section 2.6.4.

The transfer between sensors and workers take place at the start of a sample run. Hence the enabling times for their communication tasks are at the same time. Under such circumstances, the odds of communication contention in the platform is indeed considerably high. Appendix C shows a detailed conceptual view of a typical ASML wafer stage platform.

2.7 Prior Work - Contention analysis for shared FCFS resources

In this Section, the scalable contention analysis method [11] on periodically restarted directed acyclic graphs (DAG) that was previously done at ASML is described.

2.7.1 Preliminaries

- *Interval Bounds :* The lower bound L of an interval $[a, b]$ is given by $L([a, b]) = a$, the upper bound U is given by $U([a, b]) = b$.
- *Execution interval :* The execution interval of a communication task is the communication delay when there is no contention.
- *Enabled interval :* Enabled interval of a task provides min/max bounds of its predecessor completion.
- *Busy interval :* Busy interval of a task reflects the delay between its enabling and its completion time, including both waiting time to get access to its resource, and its execution time.

2.7.2 Timing Analysis

The relation between the enabled interval of a task $t \in T$ of G and its completion interval is:

$$En(t) = \begin{cases} [0,0], & \text{if } pred(t) = \phi \\ max_{t' \in pred(t)} C(t'), & \text{otherwise} \end{cases} \tag{2.8}$$

where $pred(t) = \{t' \in T | (t', t) \in D\}$ denotes the set of predecessors of t.

The completion interval of any task $t \in T$ in terms of its busy interval is given by:

$$C(t) = En(t) + B(t) \tag{2.9}$$

- **No Contention :** when there is no contention, the timing of a DAG is calculated by propagating the completion and enabled intervals of Equations 2.8 and 2.9 through the nodes of the graph in topological order using min-max propagation as shown in Section 2.4.2.
- **With Contention :** for DAGs with contention, the algorithm starts with a B assuming no contention. The additional task delay in B due to contention can be estimated by analyzing the relation between enabled intervals of different tasks on the same resource. These enabled intervals in turn depend on B. This recursive dependency is dealt with by using a fixed-point iteration on B. In each iteration more contention is taken into account, until a fixed-point is reached.

2.7.3 Contention Model

Given an initial B, the enabled intervals of the tasks in the DAG can be calculated by evaluating Equations 2.8 and 2.9 on tasks in the graph in topological order. If the enabled intervals of all tasks are known, the completion interval of a task t is estimated by analyzing the possible delay caused by tasks mapped to the same resource that can be queued in the execution queue of $t's$ resource before the enabling of t. There can be no contention between t and some other task t' if t and t' are dependent. Figure 2.9 shows two similar DAGs. In the DAG of Figure 2.9a, $t3$ precedes $t4$ in any execution of the DAG, even if their enabled intervals would overlap, since tasks in $pred(t4)$ are dependent on all tasks in $pred(t3)$. The DAG of Figure 2.9b has no such precedence relation between $t3$ and $t4$. Tasks that are enabled strictly earlier than t will precede t in any concrete execution. Thus the tasks that are enabled strictly earlier than t will always be affecting t in both best-case and worst-case. Tasks with an enabled interval that overlaps with that of t will precede t in only some concrete executions.

Let $ee(t)$ denote the set of tasks independent of t, which are mapped to the same resource and which are enabled strictly earlier than t, either based on a strictly earlier enabled interval, or because of the dependencies between predecessors of t and t'. Similarly, the set $oe(t)$ denotes all tasks independent of t, which are mapped to the same resource, whose enabled interval overlaps with that of t and which are not in $ee(t)$.

- **Best Case :** The earliest possible completion of t occurs when it is enabled as early as possible, and the start of its execution is delayed as little as possible. This is the case

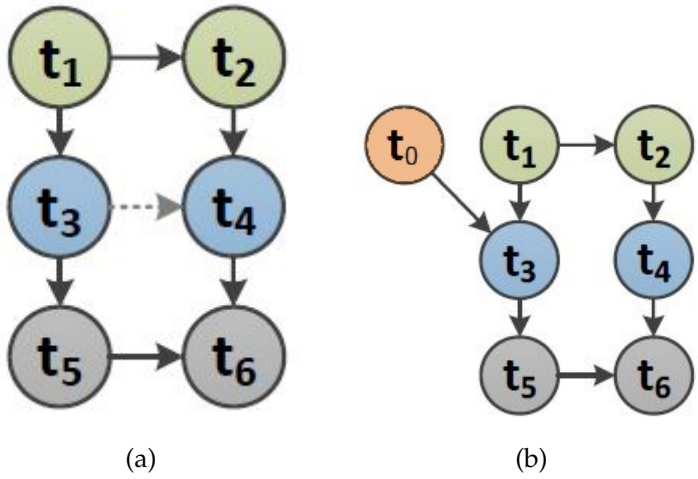


Figure 2.9: DAG examples

when t is enabled at $L(En(t))$, and all tasks in $ee(t)$ complete as soon as possible, and all tasks in $oe(t)$ (except for t itself, which executes at its best-case execution time) are enabled later then t. So, in the best-case, t can start executing after its best-case enabling and the best-case completion of the last completing task in $ee(t)$.

- **Worst Case :** The latest possible completion of t occurs when it is enabled as late as possible, and the start of its execution is delayed as much as possible. Given $t's$ worst-case enabling, t is delayed most if all tasks in $ee(t)$ complete at the upper bound of their completion interval, and all tasks in $oe(t)$ execute at the upper bound of their execution interval, while they are enabled just before the upper bound of the enabled interval of t.

With this contention model, the completion interval of a task t given some B is given by:

$C(B)(t) = max(\gamma, \mu)$, where

$$\gamma = En(B)(t') + B(t') + \left(\sum_{t'' \in oe(t) \setminus oe(t')} E(t'') \right) \cup E(t) \tag{2.10}$$

with t' the last completing task in ee(t)

$$\mu = En(B)(t) + \left(\sum_{t'' \in oe(t)} E(t'') \right) \cup E(t) \tag{2.11}$$

The algorithm is started with a B assuming no contention. Using the contention model a new B is found. An iteration on B is performed until a fixed-point is reached. It has been proved that a fixed point will always be found in a finite number of steps [11] and that this approach is conservatively.

2.8 Other Related Work - Literature Survey

The Synchronous Dataflow (SDF) model of computation is commonly used in performance analysis and synthesis of general purpose platforms. With SDF analysis, the impact of mapping and scheduling on application timing and platform memory requirements can be analyzed. SDF is a an expressive model of computation. In [12] they show how SDF models can be used to model data-flow applications that contain shared resources with finite buffer sizes. By using this approach, the communication dependencies can be abstracted and the bounds can be found in the switch as given below,

Best-case delay at a switch:

- No contention at ingress port
- No contention at egress port
- No waiting time due to contention
- Switch execution time = $\frac{DataSize}{BandWidth} + SwitchLatency$

Worst-case delay at a switch:

- Maximum contention at ingress port: By modeling the case where 8 packets have arrived before the concerned packet and the ingress port buffer is filled.
- Maximum contention at egress port: By modeling the case where 8 packets have arrived before the concerned packet and the egress port buffer is filled.
- Waiting time due to contention: $(8 + 8) * \frac{DataSize}{BandWidth}$
- Switch execution time: $\frac{DataSize}{BandWidth} + SwitchLatency$

For example, if we assume a datasize of 256 bytes, bandwidth of 8Gbps and switch latency of 140ns. The best-case delay per switch becomes 0.396us[1] and worst-case becomes 4.492us[2]. In a Typical ASML platform, the number of switches in a RIO communication interconnect path is between 1 and 4. Hence the worst-case delay for a typical communication task that uses four switches becomes 17.968us[3].

This communication delay is indeed conservative and handles back-pressure. The disadvantage is that the bound is over-conservative. For the use-case of ASML scanners, the deadline of the tasks that send computational data to the actuators is approximately between 15-25us for a machine that has sampling frequency of 20Khz; so the conservative upper-bound of 17.968us is not sufficiently tight.

[1] $\frac{256bytes*8bitsPerByte}{8Gbps} + 140nS$
[2] $\frac{16*256bytes*8bitsPerByte}{8Gbps} + \frac{256bytes*8bitsPerByte}{8Gbps} + 140nS$
[3] $4.492us * 4$

In the case of contention analysis method shown in Section 2.7, the communication dependencies are not abstracted. This gives a sufficiently tight bound as proved in Section 5.2. Due to this reason we did not opt for SDF techniques.

Timing analysis based on model checking [13] is also a widely used approach in timing performance analysis. The timing properties are verified by analyzing a set of Timed Automata. These techniques however do not scale well due to their underlying state-space explosion problems. This makes it unusable under the ASML industrial context.

2.9 Refined problem statements and deliverables

In order to study the impact of communication-time variations on ASML servo control systems various incremental steps are to be made. This section describes the steps and the problems prevalent in each of these steps.

Transformation from domain models to an executable model

The automated transformation from the formal domain models to executable POOSL model must be extended such that it includes detailed communication behavior.
For this purpose, the executable POOSL model must include the communication platform detail. Hence, the automated transformation must generate a POOSL model that includes details of the communication platform for that particular instance of the domain model. In addition, it must be ensured that the transformation algorithm scales with respect to the size of the schedule.

Computing best/worst case timing along with communication contention

To compute the best case timing and worst case timing of all tasks in a schedule, the best/worst case completion times of each of the control task and communication task must be computed. However, to find the best/worst case timing bounds of the communication tasks, the *waiting times due to communication contention* in shared resource must be computed. For this purpose, the contention analysis algorithm presented in Section 2.7.2 must be applied to the RIO communication network of the ASML scanners. However, this algorithm does not compute the waiting time due to back-pressure. Hence, an extension must be made to the algorithm to make it capable of computing the waiting time due to back-pressure in the RapidIO interconnect.

Validation and calibration

It is desirable that the timing analysis resulting from the simulation of POOSL models are closer to the timing values that are measured on an actual ASML scanner. In other words, a high predictive power of the robustness analysis approach is desired. The predictive power can be checked by validating the completion time distributions derived from the robustness

analysis approach against the completion timing measurement distributions from an actual ASML scanner. A crucial way to improve predictive power is by calibrating the execution time disstributions used by the robustness analysis approach with available measurements from an actual ASML scanner.

Chapter 3

Analytical Approach

The analytical approach is responsible for computation of the best-case and worst-case completion time bounds of all the tasks in the schedule. The prior work in Section 2.4.2 based on max-plus algebra does not take the delays due to communication into account. In this chapter, the extensions made to the scalable contention analysis algorithm (presented in Section 2.7) such that it is can be applied to a shared interconnect of serial RIO switches is presented.

3.1 Requirements

The following points are the requirements of the completion time intervals of all tasks that are computed in the analytical approach,

- The completion time intervals must be conservative
- The completion times must include the waiting time due to communication contention
- The completion times must include waiting time due to back-pressure

As shown in Section 2.6, in the RIO network of ASML wafer scanners, every worker-to-worker or IOWorker-to-worker communication is performed by a data transfer between switches and their constituent links.

The contention analysis algorithm explained in Section 2.7 is a generic algorithm that can be applied to any shared FCFS resource. This algorithm must be extended such that it is able to to be applied to a shared interconnect of serial RIO switches. To this end, it has to be extended such that it can take the waiting time due to back-pressure into account while computing the worst-case timing bounds.

3.2 Single resource mapping approach

The most simple way to use the algorithm shown in Section 2.7 is by considering that all the communication tasks are mapped on to a single shared FCFS resource. Hence, in this approach all the communication tasks are assumed to be dependent if their enabled times overlap. In this way all the requirements mentioned in Section 3.1 are met. However, there are several crucial disadvantages in this approach, which is mentioned below,

- **It does not take the independent communication paths into account** We have already seen that the ISF of a RIO switch can simultaneously transfer packets from multiple disjoint ingress, egress port pairs. Hence, if the communication tasks have overlapping enabled intervals but use disjoint ingress-egress port pairs of the same switch or if they use different switches altogether, they must be considered independent. However, in this approach this independence is ignored as all the communication tasks are considered to be mapped on a single resource.

- **Overly conservative** Since all the communication tasks are mapped to the same resource, this approach assumes that all tasks are dependent on each other. Hence, some communication time intervals that is derived using this approach are expected to be over-estimations. The validation of this expectation is a work to be done in the future. Over estimations of this approach makes it unusable for our use-case scenario.

3.3 Communication task splitting

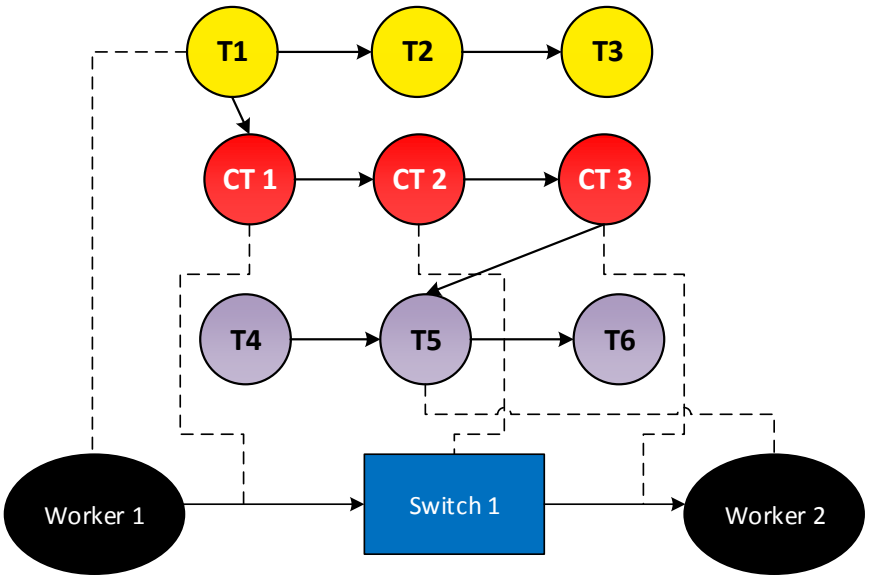


Figure 3.1: An approach to consider the detailed communication path

To find the independence between communication tasks we have to consider the detailed communication path for each communication task. One way to consider the detailed communication path is to split the communication tasks such that for each communication resource component (link/switch) in the detailed communication path a task is created. This approach is explained in the next section.

An example for a communication involving a single switch is shown in Figure 3.1. We can split every communication task into several tasks and map them on their appropriate RIO resource and use the contention analysis algorithm shown in Section 2.7. Now when the tasks are mapped on the same switch or link and have enabled times overlapping each other, their waiting time due to contention will be added to the completion time bounds.

The characteristics of this approach are:

- **It takes the independent communication paths into account** Since each component of a communication path is taken as a resource, when two worker-to-worker communication paths do not share any switch or link their tasks are mapped on independent resources.
- **It does not take the waiting time due to back-pressure into account** As explained in Section 2.6.4, back-pressure is a system wide phenomenon where queuing can ripple all the way back to the original input. When communication tasks are split and mapped to separate resources in the communication paths, even if the enabled intervals of the tasks that are mapped on the links and switches don't overlap, they may be dependent on other tasks due to back-pressure. This approach adds waiting time due to contention only on a single resource component (switch/link) and the dependency due to back-pressure is not taken into account.
- **Non-conservative:** As the splitting approach does not take back-pressure into account, the communication time intervals that is derived are non-conservative estimations. This property renders the splitting approach as an useless approach.
- **Scalability:** This approach may also not be scalable because it can lead to the creation of many communication tasks, as the schedule size increases.

In the ASML industrial use-case scenario, we need a approach that is both scalable and conservative. The approach does not satisfy both these conditions. We next present an approach that does not split the communication tasks but considers their detailed communication path. This approach is explained in the next section.

3.4 Overlapping Switch Port based Resource Mapping (OSPRM)

In this approach, we use one communication task for each worker-to-worker communication but we also consider the detailed communication path for each of them. The main steps of the algorithm are as follows:

1. *For each task, find all the switch ports that the communication path of the task uses.*
2. *Map all communication tasks which have shared switch ports on same resource.*
3. *Communication tasks which do not share any switch ports must be mapped to different resources.*

3.4.1 The algorithm

We can determine the switch ports that a communication task uses from the physical platform model.

Definitions

- *Communication tasks and resources :* Let C denote a set of communication tasks and R denote a set of possible resources for these tasks. The size of R is initially chosen to be equal to the size of C. The OSPRM algorithm then informs us which of these resources will be used.
- *Ports :* Let P denote a set of all switch ports in the communication network. It contains both the ingress ports and egress ports.
- *Mapping relations :* $M_{CR} : C \rightarrow R$ maps any communication task to a resource. $M_{CP} : C \rightarrow P$ maps any communication task to its set of ports. M_{CP} can be derived from a physical platform model. $M_{RP} : R \rightarrow P$ maps any resource to a set of ports.

Detail

Algorithm 1 shows the complete algorithm to find M_{CR}. This can then be used with the approach presented in Section 2.7 to get the completion time bounds. The mapping relations M_{RP} and M_{CR} are initialized to $NULL$ in lines 3-8. In lines 9-15, for each communication task it is checked whether it shares one or more switch ports with any resource in R. If yes, the communication task is mapped onto this resource. In case there is no such resource, the communication task is mapped onto a resource in R that has no communication task previously mapped onto it. In line 19, M_{CR} is updated with this resource mapping. In line 20, M_{RP} is updated with all the switch ports of the communication task that are mapped to it.

3.4.2 Predicting Back-pressure

Figure 3.2 shows four communication tasks and the switch ports that they use by showing the communication task and their communication path in the same color. Each switch is assumed to have four ingress and four egress ports. The ingress ports are named as A, B, C and D. The egress ports are named as E, F, G and H. There are eight workers and four

Algorithm 1 Find M_{CR}

```
Input: M_CP, R, P, C
Output: M_CR
for all r ∈ R do
    M_RP(r) = NULL
end for
for all c ∈ C do
    M_CR(c) = NULL
end for
for all c ∈ C do
    r_Chosen = NULL                                  ▷ The chosen resource r for task C
    for all r ∈ R do
        if M_RP(r) ∩ M_CP(c) then
            r_Chosen = r                             ▷ Choose a existing resource
        end if
    end for
    if r_Chosen == NULL then
        r_Chosen := {r ∈ R | M_RP(r) = ϕ}            ▷ Choose a new resource
    end if
    M_CR = M_CR ∪ {c, r_Chosen}
    M_RP(r_Chosen) = M_RP(r_Chosen) ∪ M_CP(c)
end for
```

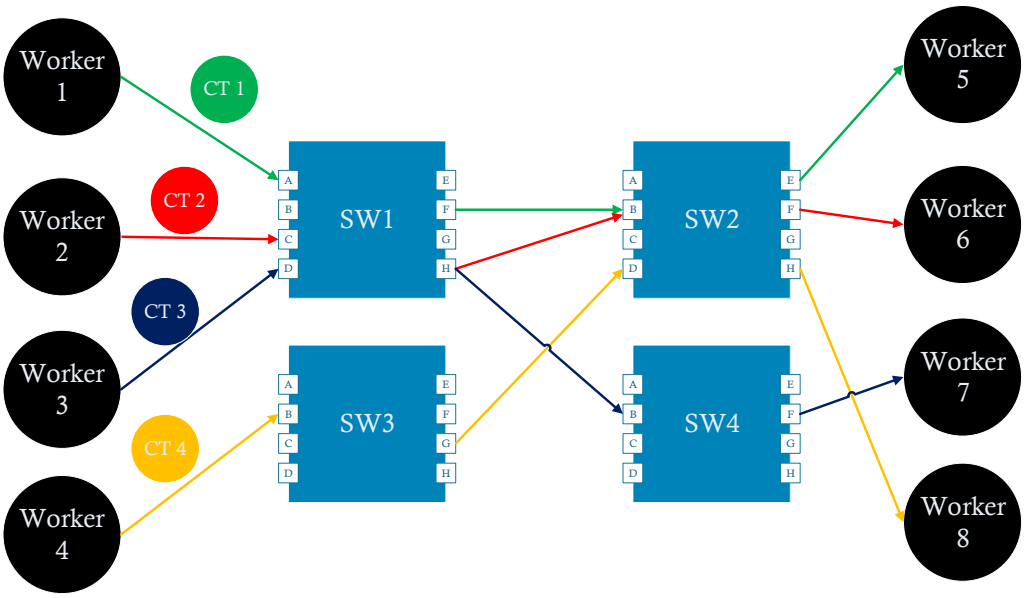


Figure 3.2: An example of four communication tasks depicting back-pressure

switches. We can see from the communication path of *CT1* and *CT2* that they share the ingress port B of switch 2. This means that these two communication tasks can contend

with each other. Hence they must be mapped to the same resource in order to compute the waiting time due to contention using the algorithm presented in Section 2.7. Consider a situation when the buffer of ingress port B is full. This forces the packets of *CT2* to stay in the buffer of egress port H in switch 1 exhibiting back-pressure. However, *CT3* also uses the same switch port. Although *CT3* does not share the ingress port B in switch 2, if *CT1* and *CT3* have overlapping enabling times then back pressure may affect *CT3* as well due to sharing the buffers of egress port H in switch 1. Hence, mapping *CT1* and *CT3* to the same resource and using the algorithm presented in Section 2.7 allows to us to compute the waiting time due to back-pressure. If we consider *CT4* it does not share any other switch port with either of the other three tasks and hence must be mapped to a new resource. This is the essence of the OSPRM approach that adds the overlapping switch port based resource mapping as a pre-processing step to the prior contention analysis algorithm presented in Section 2.7.

3.4.3 Features

The features of this approach are:

- **It accounts for the independent communication paths** The communication tasks that do not share the same switch ports are mapped onto different resources. This means that this algorithm is able to find the independent communication paths.
- **Transitive closure** Consider an example of three communication tasks c_1, c_2 and c_3. If c_1 is found to share a single switch port as c_2, they are mapped to the same resource. If c_3 is found to share a single switch port as c_2, they are mapped to the same resource. Since they all are mapped to the same resource by this algorithm, this implies that,

$$((M_{CR}(c_1) == M_{CR}(c_2)) \wedge (M_{CR}(c_2) == M_{CR}(c_3)) \implies (M_{CR}(c_1) = M_{CR}(c_3))$$

- **It takes waiting time due to back-pressure into account** Since the communication tasks that share any one switch port is mapped onto one resource, the dependencies due to back pressure is taken into account.
- **Conservative** As both waiting time due to contention and waiting time due to back-pressure is taken into account. This approach is a conservative approach.
- **Scalable** This approach does not need to split the communication tasks into several tasks, this makes the approach scalable in terms of memory usage and computations needed.

3.4.4 Possible Optimizations

The OSPRM approach does not compute back-pressure with the knowledge of buffer occupancy. If buffer occupancy is known, then we can reduce several cases where back-pressure will not occur due to sufficient capacity in the buffers. If this approach is optimized with

the ability to compute back-pressure with the knowledge of buffer occupancy, it can lead to computing tighter and more accurate worst case estimates that are closer to the true worst case time.

In Figure 3.2 we can see that the links from the egress switch ports of switch 2 and switch 4 onwards are completely independent of each other. In this case an optimization of splitting the communication tasks and mapping resources such that this independence can be found will help in reducing overestimations.

Chapter 4

Simulation Approach

In Section 2.5.1 it was mentioned that by using POOSL models we can find the completion time of control tasks along with the waiting time due to communication by mapping the communication task to the appropriate communication resource and simulating the communication behavior. In this chapter, the architecture of the communication network model that must be built for the appropriate ASML wafer stages stack to analyze their schedule robustness is shown. Also, the architecture of the complete POOSL model that includes RapidIO networks along with the control tasks is shown.

4.1 Communication resource POOSL model

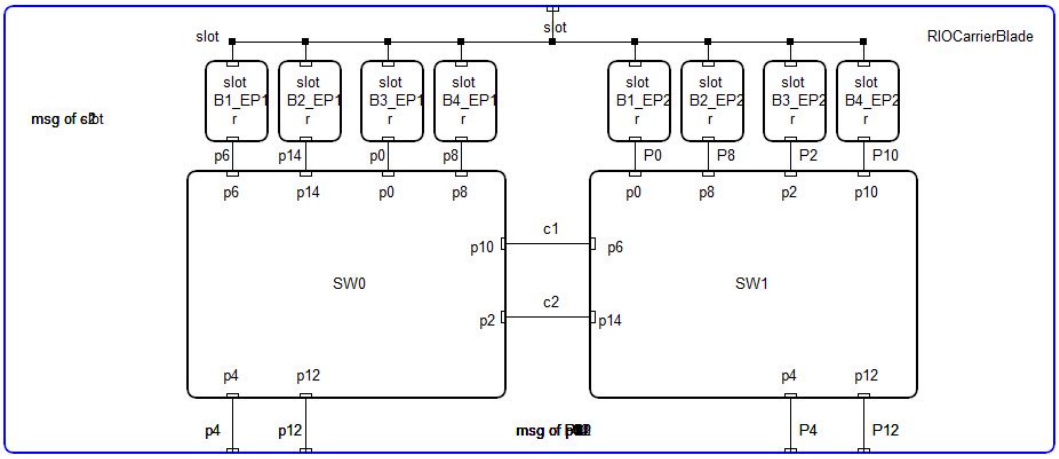


Figure 4.1: A blade containing two RioSwitches view in poosl shesim GUI

The model transformation algorithm has been adapted to generate a RapidIO (RIO) model according to a particular ASML machine stack by reading the network parameters and network topology from a physical platform model. The model includes all the major components within the interconnect (links and switches) and all internal components of a switch

(input/output ports and associated queues). Some of the features have been abstracted. For example, link level error control is not modeled, we assume that there is no packet loss.

Figure 4.1 shows one blade of a ASML platform modeled in POOSL. A blade is one unit of the execution platform in ASML wafer stage component on which processors, IOBoards and RIOSwitches are placed(explained in Detail at Appendix C). A blade is modeled as a cluster class in POOSL. There are currently five blades in one execution platform. Figure **??** shows the complete RIO interconnect modeled in terms of interconnected clusters[1]. Here, each cluster is a blade. From the Figure 4.1 we can see that there are processes named as EP. These represent end points of processors and IOBoards that connect to RIOSwitches. There are eight EPs in one blade cluster. RIOSwitches are also modeled as a process. There are two RIOSwitches in every blade cluster. The links between the endpoints and RapidIO switches are constructed by reading the network topology from the physical platform model. The links are modeled as channels with messages passing through them. The network parameters such as switch latency, bandwidth are also read from the physical platform model. The switches model the functional behavior of a cross bar switch as mentioned in Section 2.6.2. The switches contain buffers of depth eight in all their ports. Using conditional statements from POOSL, the switch ports have been made to accept packets depending on the fill-level of an input buffer (also depending on the priority of the packet). If a packet is not accepted, it will stay in the buffer of the connected switch. In this way back-pressure is modeled in a natural manner.

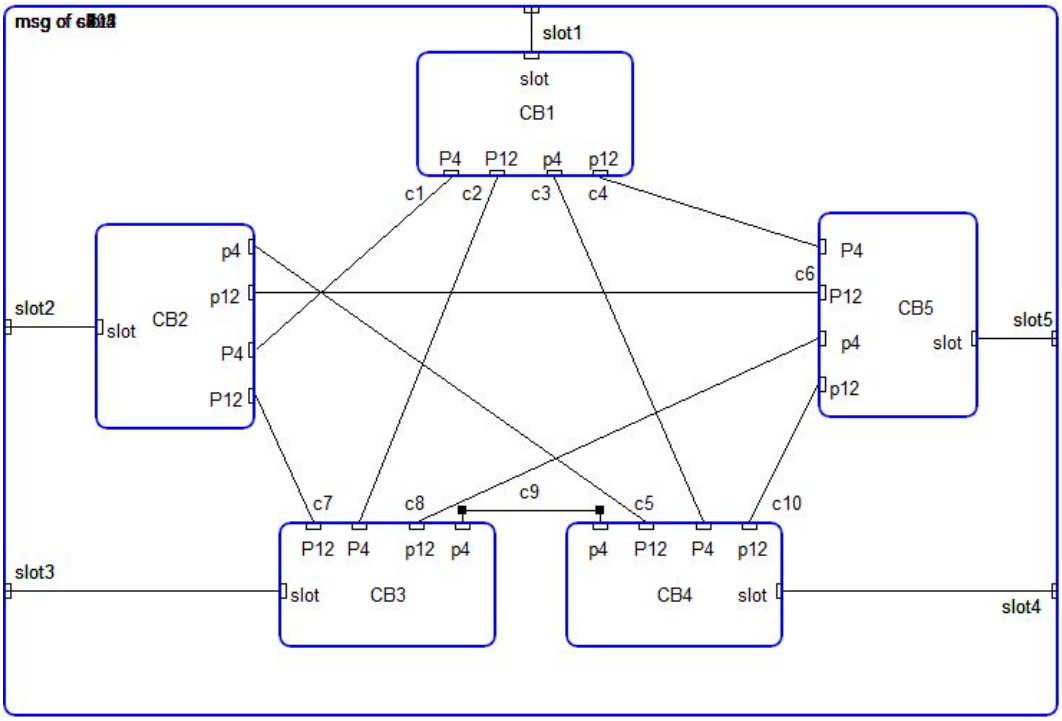


Figure 4.2: A complete communication resource POOSL model

[1]Clusters are instances of cluster classes and group a set of processes and clusters (of other cluster classes)

4.2 Schedule POOSL model

The models with synchronized workers and non-synchronized workers described in Section 2.4.1 are extended with the RIO network models described in the previous section. Additionally another process called service is added. Figure 4.3 shows the architecture of the extended model. Each block is a process. The connections between processes are the channels on which they communicate by sending messages. Communication tasks in the schedule are represented as communication task processes in POOSL. Their dependencies in the schedule are transformed to the channels with messages passing through. The behaviour of the task process, worker finisher process, finisher process are the same as described in Section 2.4.1.

The working of the communication task process in the POOSL model for one sample period can be understood by the series of steps explained below:

1. A communication task process receives one input token from its predecessor. The token contains the information about the data size to be communicated to the destination worker. Then, it sends the token to a service process.

2. The service process splits the data into packets by using the data size from the token. Typically data size of one packet is 256 bytes. Consequently, it finds the endpoints that the packet has to travel in the RIO network through the source and destination information in the token. Then the packet is sent to the appropriate end point in the RIO network model along with the destination endpoint address.

3. When the packets are received by EP process it sends the token to RIOswitch process through a link. One of the ways to calculate link delay is by the formula $delay = \frac{datasize}{bandwidth}$.

4. Then the switch process checks if the destination egress port is free and if so it sends it to the port and switch delay is added. Else, it stays in the buffer and a buffer delay is added. When the port is egress free to be accessed, it travels to the egress port and switch delay is again added. This happens until the packet reaches the destination endpoint. The destination endpoint returns the token to the service process.

5. The service process sends the token back to the communication task process instantaneously. The communication task process reports the completion time to the data-collector.

4.2.1 Modular extension

This architecture is formed on the basis of modular programming[2]. The architecture separates different abstraction levels. It separates the application and platform and the process

[2]In the order of nano-seconds

service provides the mapping between them. When a modeler wants to abstract communication in his design space exploration he can just make the communication time instantaneous by adding a execution time of 0 in the service process and directly sending the token back to the communication task process. This type of architecture was beneficial to easily extend the prior work in Section 2.4.1 with only minor changes to their process classes. This type of architecture also facilitates to debug and maintain the complete POOSL model transformation algorithm.

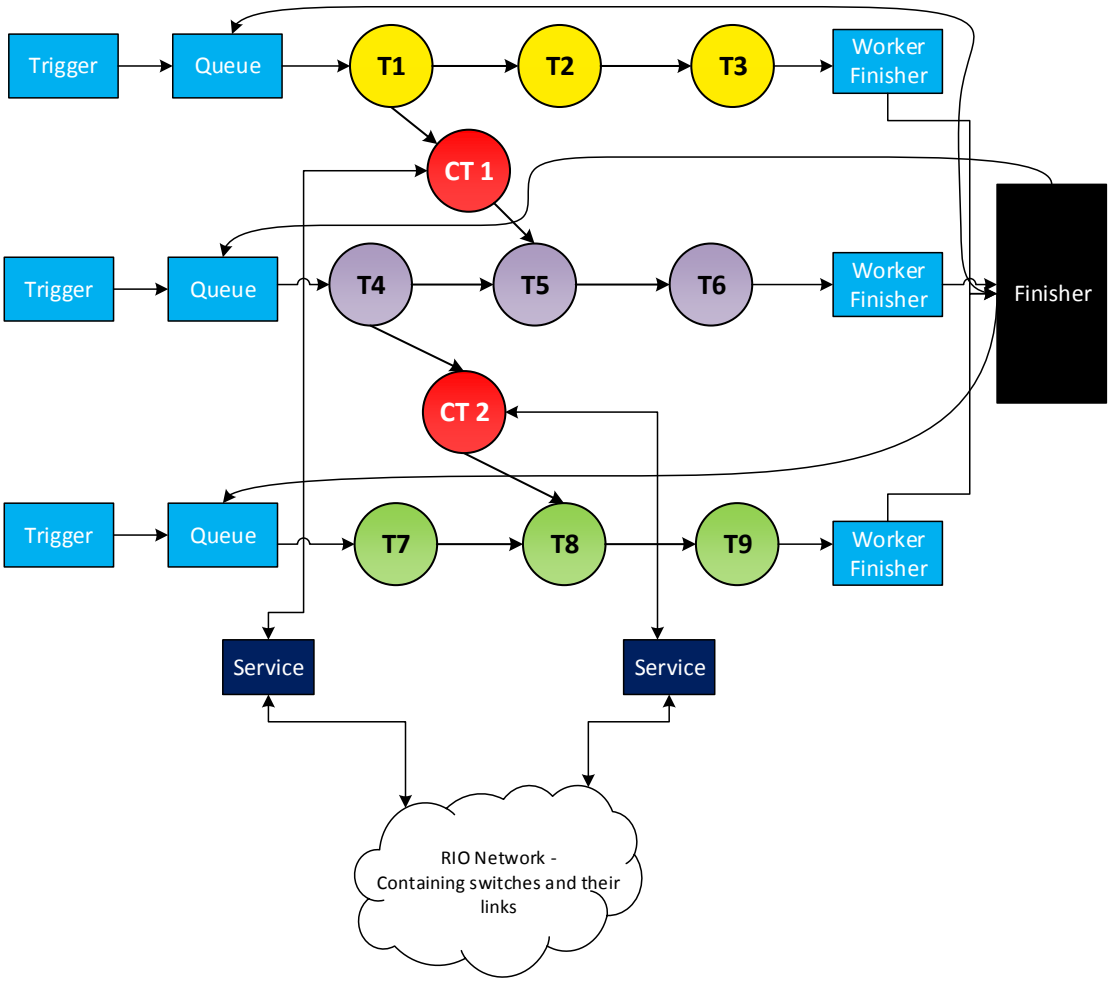


Figure 4.3: Architecture of a schedule POOSL model with RIO

[2]It is a software design technique that emphasizes separation of concerns. It is the act of designing and writing programs as interactions among functions that each perform a single well-defined function, and which have minimal side-effect interaction between them.

Chapter 5

Results

In this chapter, the robustness analysis approach has been applied to one ASML wafer stage stack and the results will be presented.

5.1 The Wafer Stage Stack

The wafer stage stack that is used to perform the experiments consists of,

- 7 multi-core processors;
- 4520 control tasks;
- 39 worker-to-worker dependencies;
- 10 RIO Switches.

However, the schedule does not contain information about sensors and actuators. Hence, IOWorker to worker communication which represents the communication between the sensors/actuators and control tasks cannot be analysed. On the other hand, worker-to-worker communication can be analyzed. There are 39 worker-to-worker dependencies in the application. Consequently, while starting the robustness analysis approach, 39 communication tasks were added. The deadline was tightened to 70% of the processor budget. The results presented are from all the workers that have sampling frequency of 20 kHz. Worker 6 is a single core worker with sampling frequency of 1 kHz. For sampling frequency of 1 kHz is so high that the latency requirements are always met. Due to this reason, Worker 6 has been skipped has been skipped during analysis. A software tool (Appendix A) to prototype the approach was made. This tool was used to perform these experiments with the above mentioned stack to obtain the results presented below.

5.2 Comparison of worst-case completion times

5.2.1 Instantaneous communication vs OSPRM

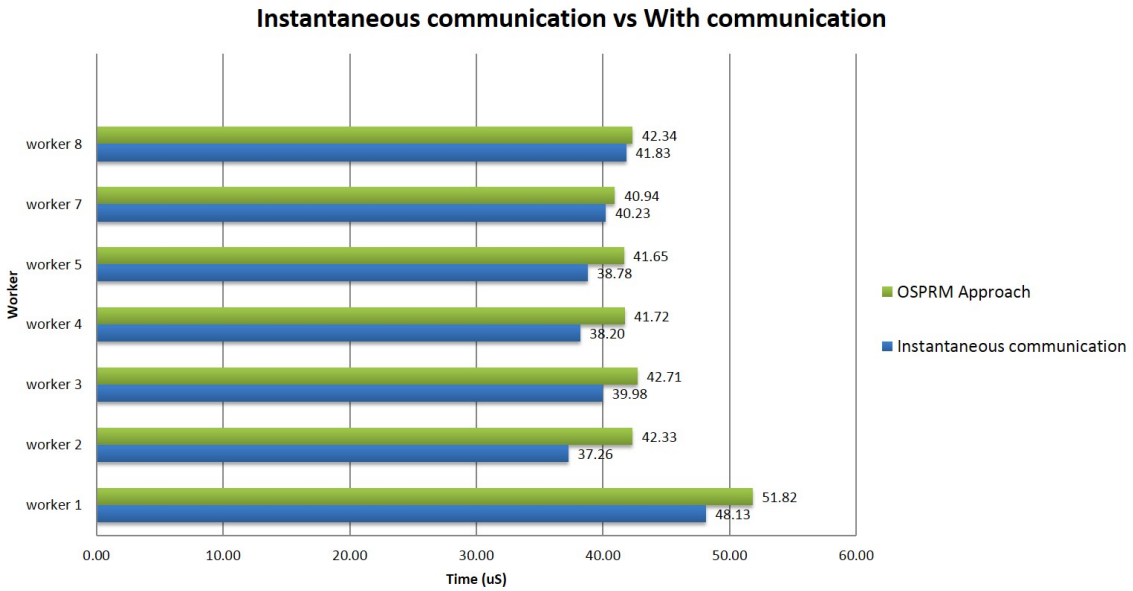


Figure 5.1: Comparison between instantaneous communication solution and OSPRM solution

A comparison of computed worst case timings for the last task in every worker under two approaches is made. The first approach was to use the prior work in which communication is assumed to be instantaneous. The second approach is the OSPRM approach in which communication is taken into account. The comparison results are shown in Figure 5.1.

We can see that extending the prior work with communication times have indeed increased the worst case completion times. The average percentage increase of the worst-case communication times is 6.82%. The individual percentage increase for each worker is shown in Figure 5.2. The increase in completion time is owing to the fact that the approach proposed in this project have taken the communication delays into account. It has to be checked in the future whether this predicted worst case timing is closer to the real timing measurements from the machine.

5.2.2 Single resource mapping vs OSPRM

A comparison of the computed worst case timings for the last task in every worker using OSPRM and the single resource mapping approach was made. The results are shown in Figure 5.3.

We can see that in worker 1, worker 2, worker 4 and worker 5 the worst case times predicted using single resource mapping approach is higher than in case of the OSPRM approach.

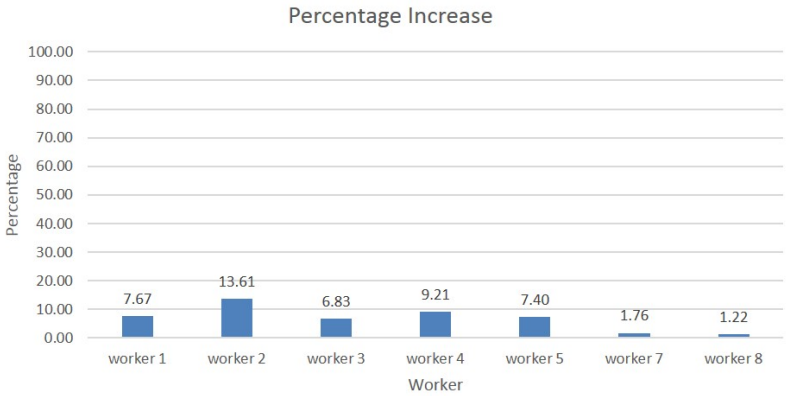


Figure 5.2: Percentage increase in worst case completion times between instantaneous communication solution and OSPRM solution

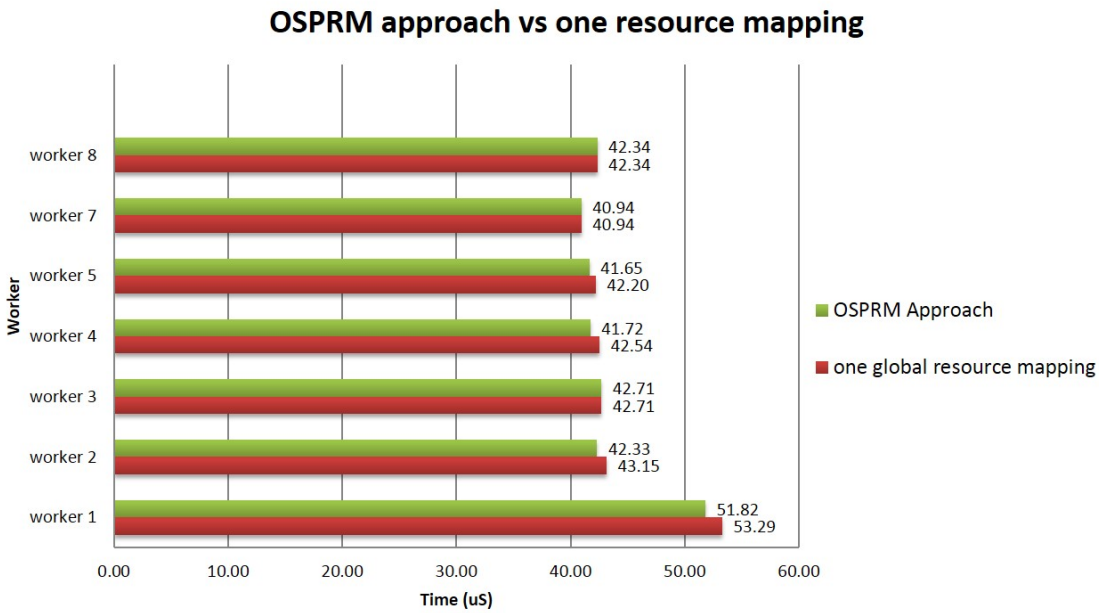


Figure 5.3: Comparison between one resource mapping solution and OSPRM solution

The overestimation is reduced in OSPRM approach. The percentage decrease for all workers is shown in Figure 5.4. The average percentage decrease of four workers was found to be 1.97%.

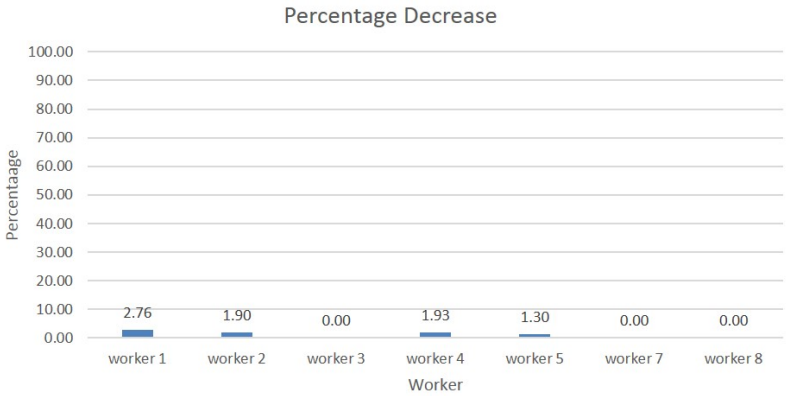


Figure 5.4: Percentage increase in worst case completion times between one resource mapping solution and OSPRM solution

5.3 Comparison of schedule robustness

We performed short simulations (1000 sample runs) on the schedule and combined the statistical results with the computed bounds to perform the robustness analysis. We computed the expected value of the number of tasks that miss their deadlines in the schedule by using Equation 2.4. Figure 5.5 shows the results. As mentioned earlier, the deadline was tightened to 70% of the processor budget. We see that after taking the communication delay into account, the expected value of the number of tasks that miss their deadlines in the schedule increases significantly. The expected value of the number of tasks increased approximately 8 times more, which amounts to reduction in schedule robustness by 13.4% (Equation 2.5).

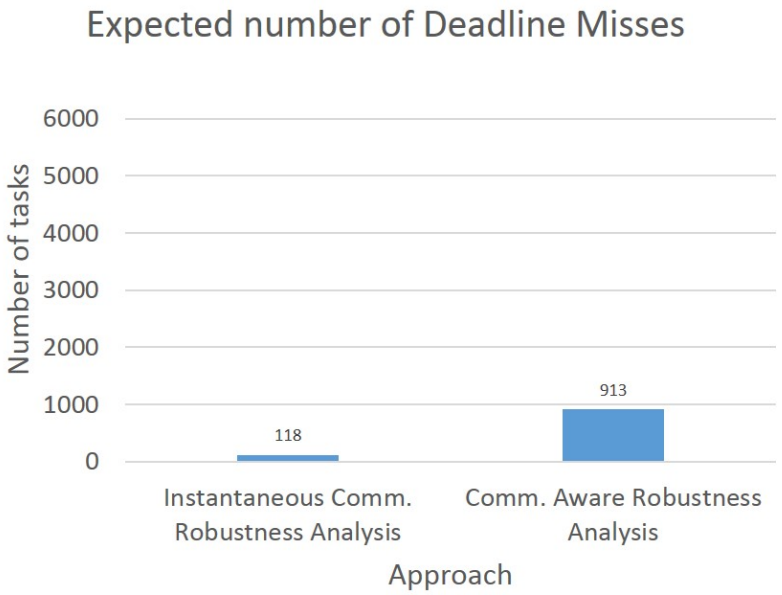


Figure 5.5: Comparing the excepted number of deadline misses

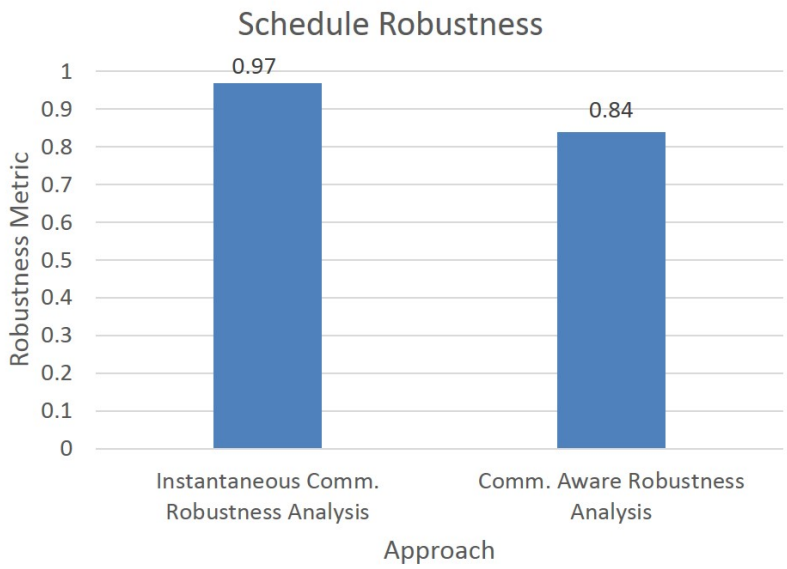


Figure 5.6: Comparing the schedule robustness

Chapter 6

Calibration

6.1 Validation and calibration of the tool-chain

This project aims to develop a clearly defined validation method for the complete tool-chain. This is because the predicted completion time distributions may vary significantly from the actual completion time distributions derived from measurements performed on the wafer scanner. The executable POOSL models are validated with available completion time measurements from a particular ASML stack. If the prediction is very far off, this must be investigated. For example, deviation in the execution time distributions could be due to measurement errors; incorrect network parameters may be used in the POOSL model. If this is found and the tool-chain is appropriately calibrated, then the predictive power of the robustness analysis approach will be increased. Hence, validation and calibration is a very important process to obtain a highly predictive robustness analysis approach.

6.2 Measurements and calibration

In order for the executable POOSL model to simulate the behavior of the wafer stage system that exists in reality, it has to incorporate the proper execution timing information of the tasks. Such information is retrieved by means of measurements in the real system which is highly prone to errors. Because of the presence of errors, the performance properties that exist in a model may be different than those that exist in reality. As shown in Section 1.3, in phase 1 of the Y-chart approach *Predict the past*, the model has to be properly calibrated with available measurements and its predictive power must be validated in order to closely follow the real system performance. Calibration and validation of the model can be obtained by considering an estimation of the error between the real system and the model analysis.

6.3 Validation and calibration of execution time distributions

6.3.1 The process

The calibration process is performed with a simple test on an ASML stack that contained two single core workers. The motivation behind choosing this stack was the absence of worker-to-worker communication tasks and no core-to-core synchronization. This helps in validating the execution time distributions used by the tool chain. During every calibration step we perform the robustness analysis approach and inspect the completion time distributions. In each worker, the two tasks we inspect are CDR (critical data ready)[1] and EOS (end of sample)[2]. CDR task appears approximately halfway during the sample.

6.3.2 Before calibration

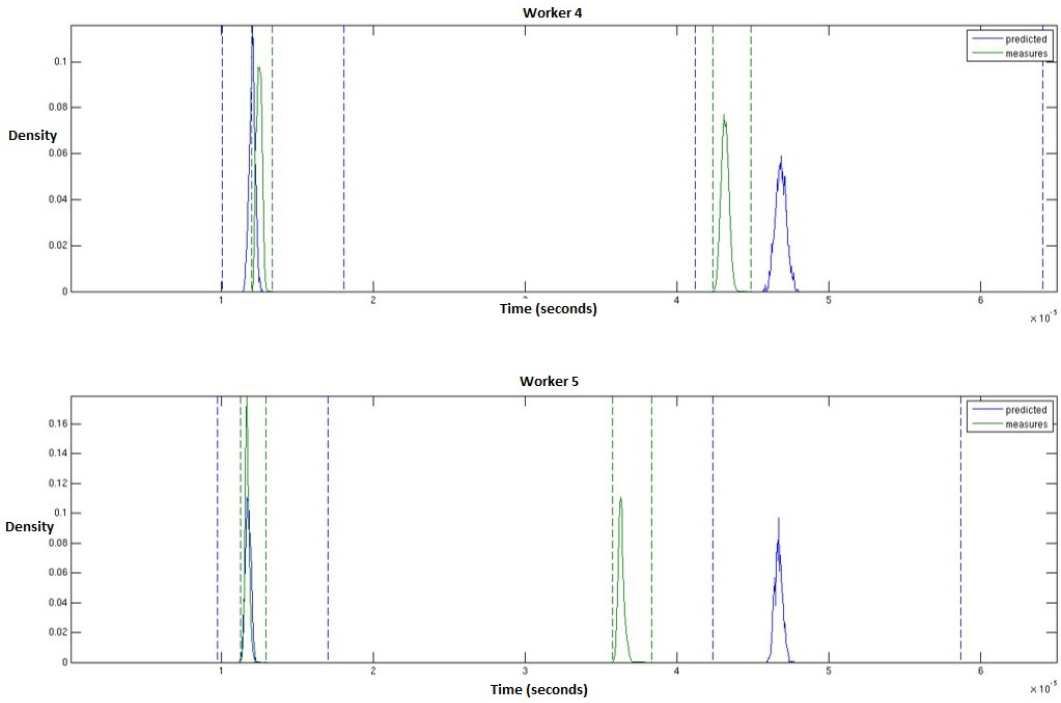


Figure 6.1: Single core results before calibration

Figure 6.1 shows a plot that represents the completion time distributions of the two tasks per worker. The x-axis represents time and y-axis represents probability density. The lines represents the min/max bounds. As mentioned earlier, there are two workers in this test stack. From the figure we can understand that there is a significant offset between the measured completion time distribution and predicted completion time distribution. The

[1]task responsible for sending computed data from control tasks to actuators
[2]last task in the schedule of a worker

difference between the mode value of EOS distribution found for the first worker is about 4 uS and for the second worker it is about 11 uS.

6.3.3 Calibration steps

After careful evaluations of the database that contains execution time distributions, it was found that the execution time of some certain tasks were set by default to a much higher value compared to the measurements by the tool chain. The reasons causing this were found and fixed by running the measurements again and updating the execution time database. After this calibration, this test was run again. Figure 6.2 shows the results of that. Even though there is significant improvement, we can see that the predicted completion time distributions are marginally different from the measured execution time distributions. The reason was due to the fact that instead of using the most likely value (mode) from the execution time distribution in the POOSL models, the mean value was used. In the current testing, we have fixed this. Also, instead of using a single gamma value of 4, finding gamma value of each task using the curve fitting approach is also currently being tested.

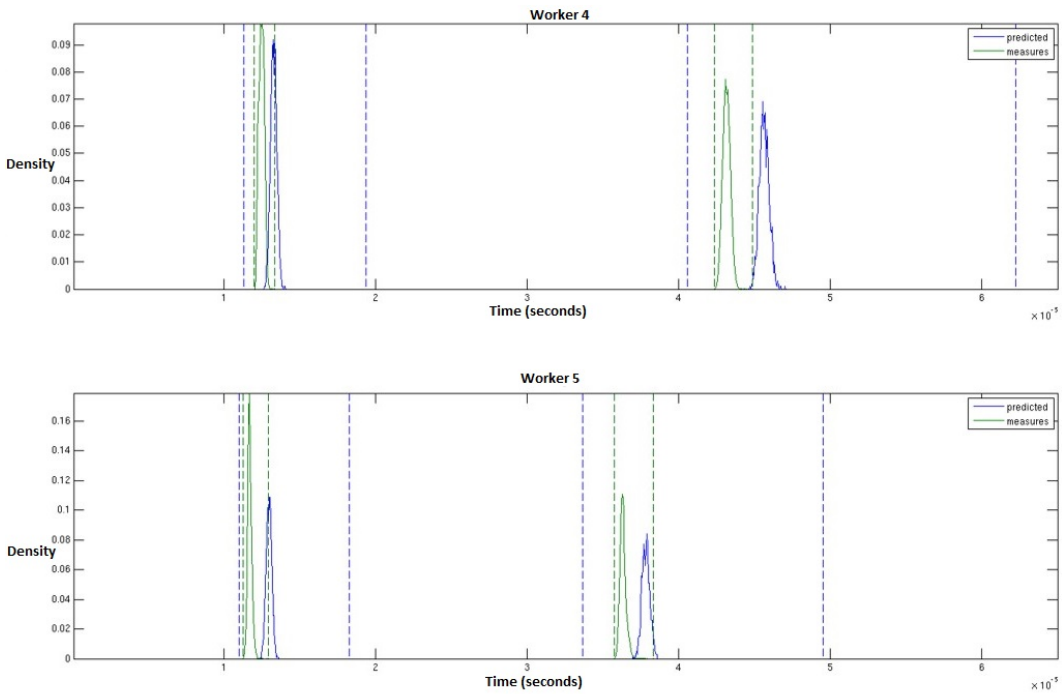


Figure 6.2: Single core results after calibration

Chapter 7

Conclusions

In this report, we have shown an approach to perform communication aware robustness analysis for high-end distributed servo control systems. We have shown how POOSL models are used to simulate control tasks that use shared packet-switched RapidIO communication interconnect. We have also shown how to compute best case and worst case completion time of tasks under communication contention and back-pressure in RIO networks. We have shown that this approach improves the predictability of deadline misses and hence improves the overall robustness analysis approach.

The simulation of the POOSL executable model allows to derive completion time histograms of the constituent tasks of a schedule. The analytical computation approach allows to derive the best-case and worst-case completion times of constituent tasks of a schedule under a shared communication resource by taking the waiting time due to both contention and back-pressure into account. This communication aware approach allows us to approximate a continuous probability distribution of the completion time and to find the robustness of all the tasks and the overall schedule.

In this project, the software tools needed to prototype the scientific assumption have been developed. The results of the application of the communication aware robustness analysis approach on one ASML wafer stage stack has been presented. Some of the results have been validated with actual measurements from execution of control tasks on a test wafer stage platform.

7.1 Future work

During the development of this thesis project some improvements to the communication aware robustness analysis approach have been identified. The wafer stage platform uses a number of multi-core processors, and tasks to model core-to-core synchronization have been added in ASML. This involves addition of core-to-core synchronization tasks in the schedule. The results of the simulation of schedules of many wafer stage stacks has shown that the core-to-core synchronization tasks constitute at least 40% of the total number of

tasks of the schedule of octo-core processors. But, currently the robustness analysis approach concentrates only on communication tasks. The resource contention due to core-to-core synchronization process is not taken into account. The approach must be extended such that it includes this feature.

The POOSL models have been constructed such that it can deal with priorities of communication data packets. However, the current ASML data flow applications do not deal with priorities for communication data packets and therefore the analytical computation approach doesn't deal with priorities of data packets yet. The analytical computation approach must be extended if the ASML data flow application supports the priorities of data packets in the future. The POOSL model also supports buffer occupancy. The analytical computation approach doesn't take buffer occupancies into account. The algorithm can be extended with buffer sizes at ports and can be made to compute communication time delay. However, both these extensions may make the approach slower and the scalability of the tool-chain may also be affected. Hence, prior to developing these extensions to the approach it must be verified whether this makes it usable under industrial context.

The communication between transducers and processors constitute transfer of large data packet transfer. This can influence the robustness of the schedule. Currently, the detailed analysis approach only deals with worker-to-worker communication. This must be extended to support IOWorker-worker communication when the DSLs are updated accordingly. Also, the POOSL language has been updated to a new version and it is currently in beta testing phase. The model transformation algorithm must be changed to the new POOSL language after investigating if the usability of the new language is needed for the organization.

Currently, there is active research going on in the organization for using FPGA platforms as an accelerator to a single-core CPU instead of using multi-core general purpose platforms. It has also been proven that their usage can improve the sample frequency of the schedules [14]. Using FPGAs as an accelerator platform requires transfer of large datasets through the communication interconnect. This can severely increase the network load. The robustness analysis approach must be investigated whether it needs to be adapted for such an application specific platform and it can help in verifying whether using the platform improves the sample frequency under network load.

Appendix A

Software Prototyping

The complete software tool-chain that is developed to prototype the communication aware robustness approach is shown as a flow chart in Figure A.1. The DSL models in CARM are transformed to schedule languages represented by *ds_graph* and *ds_schedule* file extensions as explained in Section 1.5. This schedule model is transformed using a transformation algorithm implemented in the Java programming language to a POOSL model which incorporates detailed communication timing behavior as explained in Section 4.2. Two types of POOSL models are produced. These POOSL models are simulated using the rotalumis simulator and the time variations of all tasks in the application ($\forall t \in T$) are obtained along with Gantt chart visualization for an average case schedule sample run. In addition to that, the task overruns due to deadline misses can also be visualized. The POOSL models represented by *.p4r* file extensions and are executed by using Rotalumis. Rotalumis is a high speed execution engine for POOSL.

The worst case and the best case completion times are computed by taking the delays due to communication into account as explained in Section 3.4. The analytical computation algorithm is also implemented in the Java programming language. These completion time variations (both from POOSL model simulation and computation) are then used to derive a continuous distribution of the variation of completion times of all the tasks in the schedule ($\forall t \in T : C(t)$) by using a PERT curve fitting technique. The results from the analytical computation algorithm are in the form of $*.txt$ file extension. The curve fitting algorithm is also implemented in the Java programming language. The results from the curve fitting algorithm are the PERT parameters of all the tasks stored in a $*.txt$ file extension. The corresponding distributions are used to find the deadline miss probabilities of every task in the schedule ($\forall t \in T : dl_t$). Using all the deadline miss probabilities, robustness of the schedule is computed. The final results are stored in $*.csv$ file extensions. This whole process has been automated such that after a schedule DSL model is generated, without any user input a designer can analyze the schedules in detail using this tool.

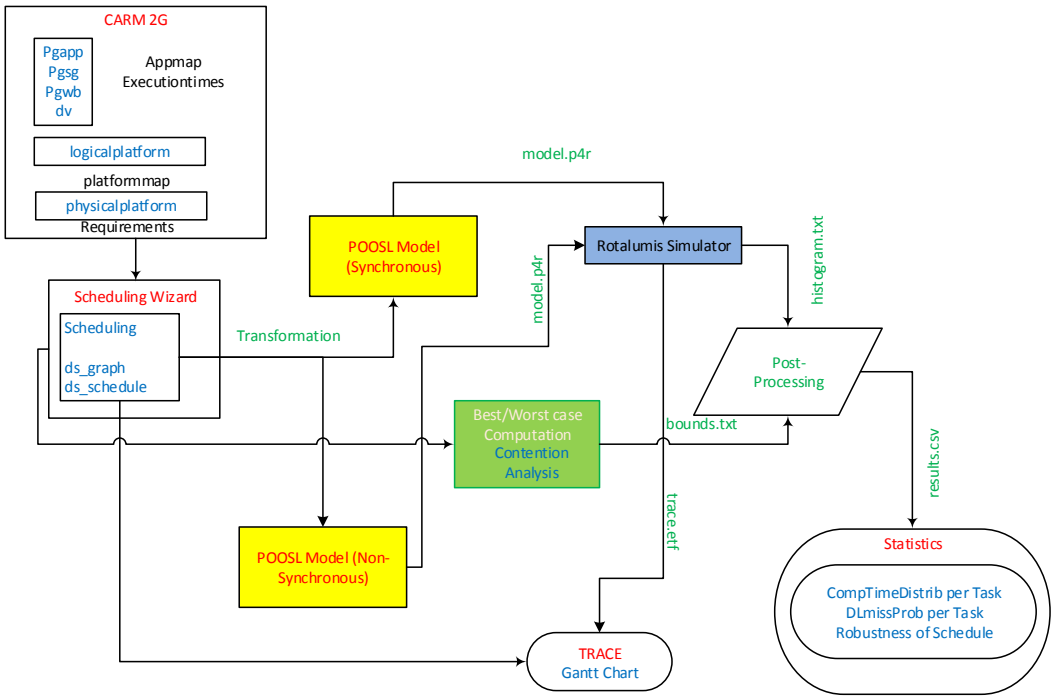


Figure A.1: Complete Robustness Analysis Toolchain

Appendix B

Visualization tools

The schedule DSL models can be visualized as a Gantt chart using the ESI trace viewer [15]. This is automated in the CARM tool IDE. In this project the capability of visualizing the POOSL model simulation results of average case sample schedule runs as a Gantt chart has been added. The visualization of a average case sample run help in validation of the POOSL models at each update in POOSL model transformation algorithm. Also, the construction of model with non-synchronized workers enables visualization of task over-run schedules. This can give an insight on the effect of deadline misses of a task on consequent tasks in the POOSL model with non-synchronized workers. Also, the ability to visualize the communication tasks has been added. Figure B.1 shows a screen-shot of an example Gantt chart that was derived for an ASML stack from the communication-aware robustness analysis approach. The rectangle boxes in yellow show the communication tasks. The rectangular boxes of colors other than yellow represent the control tasks. The x-axis represents time and the y-axis represents the resource on which the task is mapped. The arrows represent the data dependencies between the tasks.

Also, a software tool to visualize the PERT completion time distributions (Figure B.3) has been added as part of this project. Additionally it shows the deadline miss probability, the robustness and the PERT distribution parameters of the task. Also, another software tool that can visualize robustness of a worker is added. Figure B.2 shows an example robustness curve. The x-axis represents completion time. Each data-point represents the deadline miss probability of a task that completes at that time and tshe y-axis represents the deadline miss probability. For example, we can see in this example that there are many tasks missing their deadline close to the end of the sample. Both these tools were written in *Python* programming language using *Matplotlib*[16] graphics library.

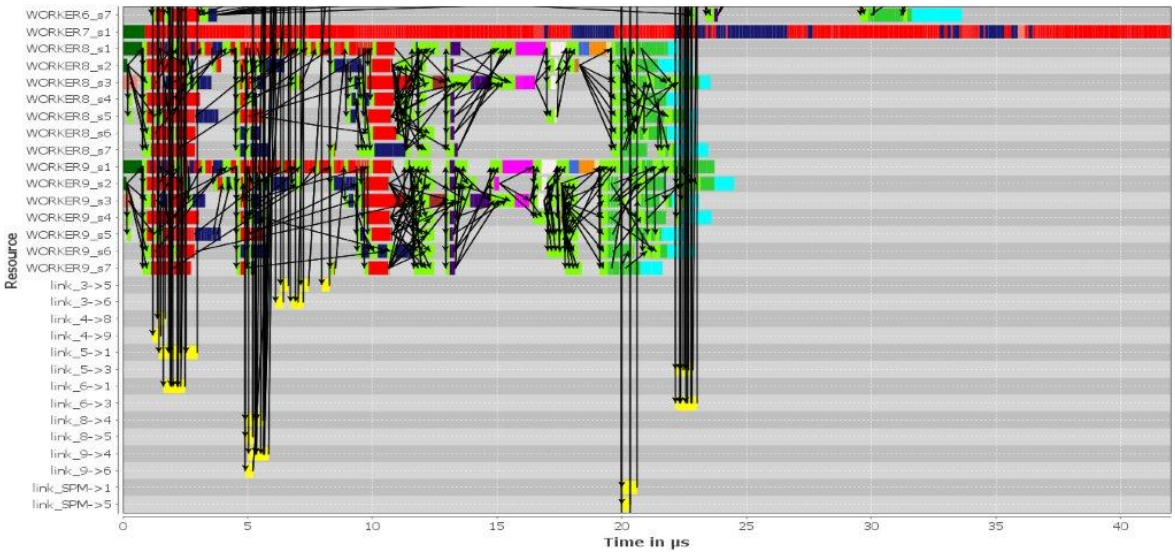


Figure B.1: Gantt chart showing communication tasks

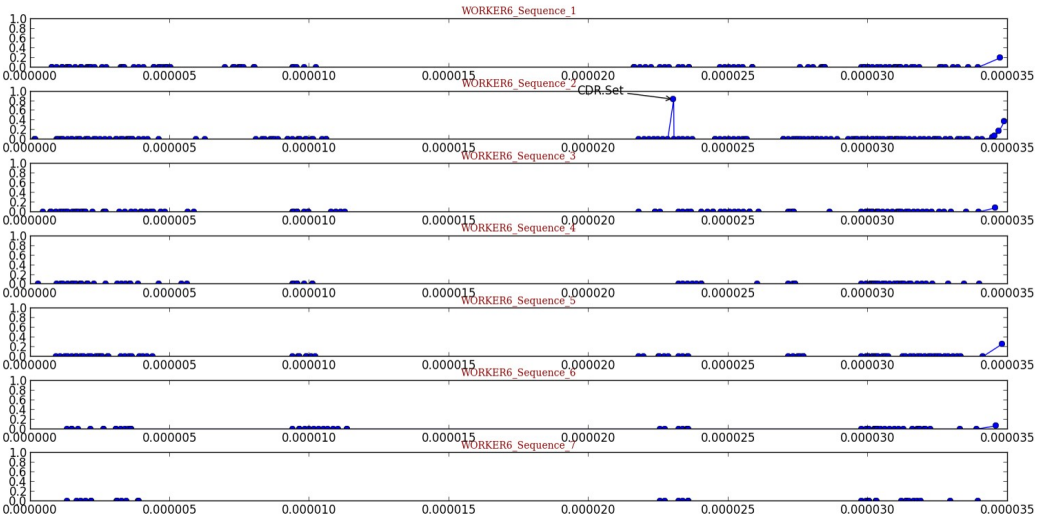


Figure B.2: Robustness Curve tool

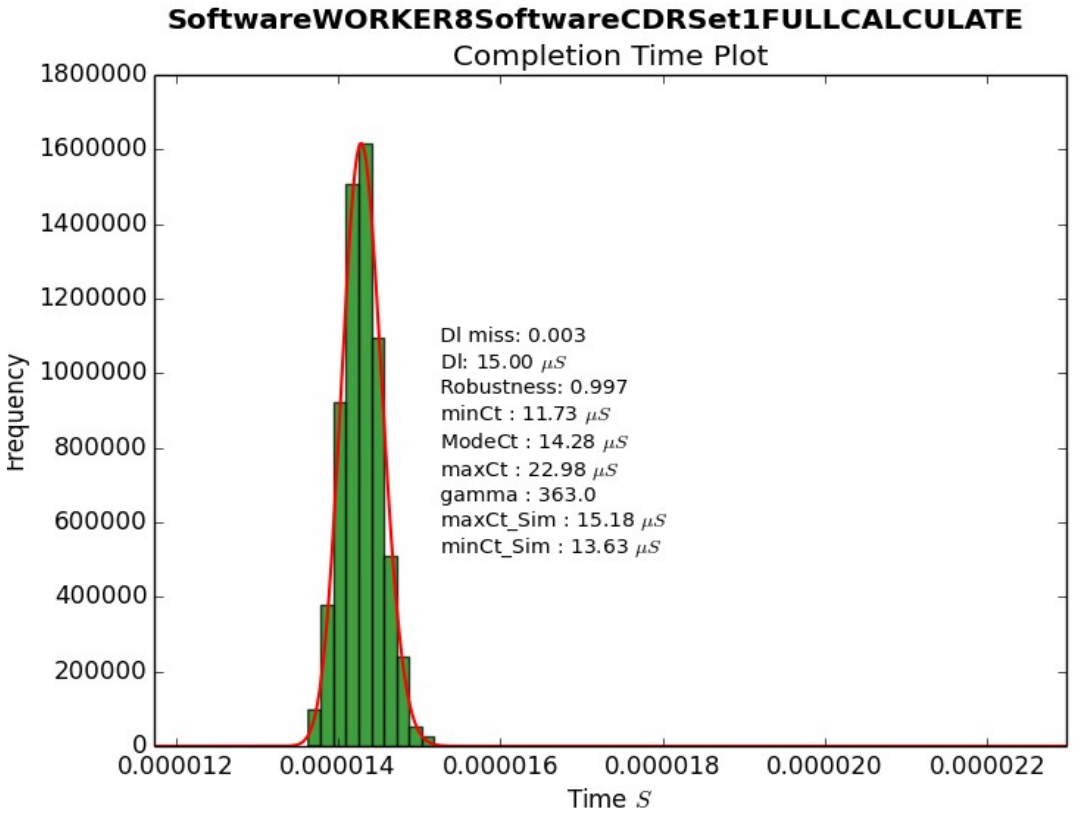


Figure B.3: PERT distribution visualization tool

Appendix C

Typical ASML platform

Figure C.1 shows a conceptual view of a typical ASML execution platform. It has 5 AMCR[1] blades. Each blade contains 4 slots. In a slot, a worker or an interface to an IOWorker is placed. As we can see from the figure the slots have two RapidIO endpoints. Each blade contains 2 SRIO switches. Currently, only one end point in each of the slots are used. The other end point is reserved for future use. The numbers 0,2,4 to 14 refers to the 16 switch ports (8 ingress and 8 egress). For example, the connection from port-0 to EP.1 shows that EP.1 is connected to one ingress port and one egress port in the switch. Four ports of one switch in a blade are connected to the four endpoints of the slots. Further there are two connections between the two switches of a blade. Every blade is connected to every other 4 racks using one unique connection link.

[1]See Appendix D

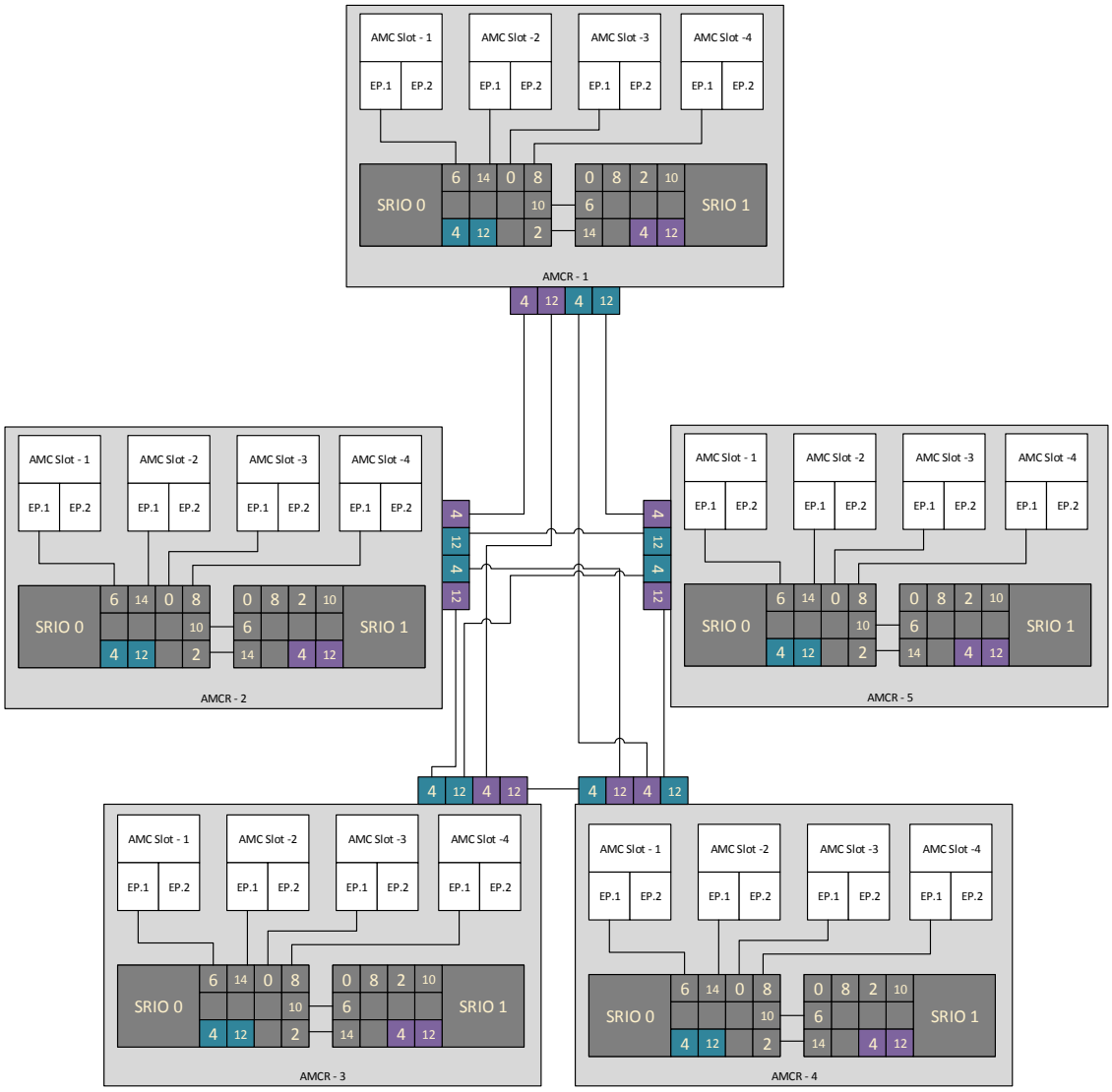


Figure C.1: Typical ASML platform

Appendix D

Terminology

AMCR	Advanced Motion Control Rack (instance of ATCA)
ATCA	Advanced Telecom Computing Architecture (defines a rack, which is used by CARM). It is an industry standard defining a set of electrical and mechanical interfaces intended for creation of electronics racks for computing, control and communications functions.
Block	Part of a Servo Group that transforms values received via input terminals into values that are put on its output terminals
Block-Group	Subset of a Servo Group
CARM	The Control Architecture Reference Model is a description to create a well defined layered model of a system where each layer has responsibilities at a specific abstraction level. It is meant to be used in a multidisciplinary environment covering the software, electrical and mechanical disciplines.
DAG	Directed Acyclic Graph
DSL	Domain Specific Language
FCFS	First Come First Serve
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor

HPPC	The High Performance Process Controller is the hardware module onto which the tasks are mapped.
IDE	Integrated Development Environment
IOBoard	Input-output board
IOW2W	IOWorker to Worker
IOWorker	It is an entity that abstracts from the type of IOBoard
ISF	Internal Switching Fabric
OSPRM	Overlapping switch port based resource mapping approach
PGAPP	The Process Control Application is the DSL that defines the networks of servo and transducer groups
PGSG	The Process Control Servo Group is the DSL that defines the servo groups
PGWB	The Process Control Worker Block is the DSL that defines the worker control blocks
PU	Processing Unit : it is an entity abstracting from a processor/core
RIO	Rapid IO
Servo-Group	Mesh of interconnected blocks where each block performs a mathematical operation.
SW	Switch
T1	Task 1
W2W	Worker to Worker
Worker	It is an entity that can perform computations (i.e abstracts from a HPPC)

Table D.1: Terminology

Bibliography

[1] Jeroen Voeten, T Hendriks, B Theelen, J Schuddemat, W Tabingh Suermondt, J Gemei, K Kotterink, and C van Huët. Predicting timing performance of advanced mechatronics control systems. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 206–210. IEEE, 2011.

[2] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183, 2004.

[3] J. Voeten R. Schiffelers, W. Alberts. Model-based specification, analysis and synthesis of servo controllers for lithoscanners. In *Proceedings of 6th International Workshop on Multi-Paradigm Modeling (MPM'12)*, 2012.

[4] Bart D Theelen, Oana Florescu, MCW Geilen, Jinfeng Huang, PHA van der Putten, and Jeroen PM Voeten. Software/hardware engineering with the parallel object-oriented specification language. In *Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pages 139–148. IEEE Computer Society, 2007.

[5] Shreya Adyanthaya, Marc Geilen, Twan Basten, Ramon Schiffelers, Bart Theelen, and Jeroen Voeten. Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 979–988. IEEE, 2013.

[6] AE Emre Gozek, JPM Jeroen Voeten, RMW Raymond Frijns, and N Nikola Gidalov. Task execution time prediction for motion control applications. 2013.

[7] Sarvesh Bhardwaj, Sarma BK Vrudhula, and David Blaauw. Au: Timing analysis under uncertainty. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 615. IEEE Computer Society, 2003.

[8] Shreya Adyanthaya Zhihui Zhang and Jeroen PM Voeten. Simulation based robustness analysis for asml multi-core servo control. 2013.

[9] Sam Fuller. *RapidIO: The embedded system interconnect*. John Wiley & Sons, 2005.

[10] Tundra Semiconductor Corporation. Tsi568a rapidio switch user manual. `http://www.tundra.com`, 2013. [Online; accessed January-2014].

[11] S.Stuijk J.P.M.Voeten M.C.W.Geilen R.R.H.Schiffelersz R.M.W.Frijns, S.Adyanthaya and H.Corporaal. Timing analysis of first-come first-served scheduled interval-timed directed acyclic graphs. 2013.

[12] Daniel Thiele and Rolf Ernst. Optimizing performance analysis for synchronous dataflow graphs with shared resources. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 635–640. EDA Consortium, 2012.

[13] Martijn Hendriks and Marcel Verhoef. Timed automata based analysis of embedded system architectures. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE, 2006.

[14] RMW Frijns, ALJ Kamp, Sander Stuijk, JPM Voeten, M Bontekoe, KJA Gemei, and Henk Corporaal. Dataflow-based multi-asip platform approach for digital control applications. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 811–814. IEEE, 2013.

[15] Embedded Systems Institute. Trace. `http://trace.esi.nl`, 2013. [Online; accessed January-2014].

[16] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.