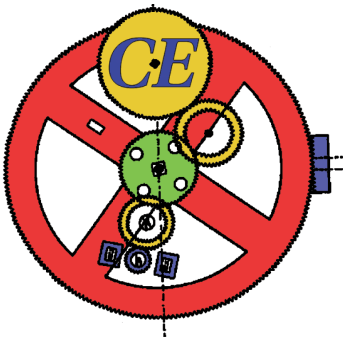


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2018

MSc THESIS

Performance Estimation of a Processor Module in the Real-Time Motion Control Platform of an ASML Lithostepper

Joachim Adriaan Voskes

CE-MS-2018-12

Abstract

At the core of state of the art microelectronic industry's drive for better technology, lies the continuing advancement in the development of Integrated Circuits using highly complex lithography machines, known as lithosteppers, which embed complex mechanical sub-systems performing intricate motions. These systems are controlled by means of custom real-time computing platforms containing off-the-shelf and specialized hardware components, and are optimized to keep pace with the continuing growing trend in performance requirements. At the ASML *Twinscan* lithostepper's heart resides the Control Architecture Reference Model (CARM) *motion control platform* which manages, among others, the *wafer-stage*, a multiple degree of freedom module, able to position a 15 kg heavy wafer-table with nanometer accuracy at extremely high acceleration and velocity. As the industry requirements for feature-size, overlay accuracy, and throughput keep increasing, the ASML lithosteppers, and the CARM platform in particular, should anticipate these demands by making early changes and upgrades with respect to computational performance and accuracy. Given that the current lithostepper configurations are not capable of sustaining the anticipated updates, which requires the increase of the control loop execution frequency from 20kHz to 40kHz, an early evaluation of potential CARM High Performance Process Controller (HPPC) successors has been performed. This indicated that the NXP-Freescale T4240 processor can potentially fulfill the expected requirements, however, the evaluation lacks accuracy as it was performed on a benchmark code not reflecting the actual CARM workload. To circumvent this problem, in this thesis, we introduce a more accurate evaluation methodology, which relies on the actual motion control application running on the HPPC and is able to capture aspects as scheduling, parallelism, and processor resource usage. To this end we develop a set of custom performance benchmarks able to emulate the CARM environment and evaluate the Freescale T4240 processor in this new context. Our results indicate that the T4240 is able to deliver enough computation power to fulfill the control loop execution frequency upscaling requirement from 20kHz to 40kHz. Additionally, we demonstrate that due to its clustered hardware architecture one T4240 can sustain 20kHz loop execution frequency for the workload of three current HPPCs, which suggest that its utilization in current lithosteppers can be beneficial.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Performance Estimation of a Processor Module in the Real-Time Motion Control Platform of an ASML Lithostepper

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Joachim Adriaan Voskes
born in Amsterdam, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Performance Estimation of a Processor Module in the Real-Time Motion Control Platform of an ASML Lithostepper

by Joachim Adriaan Voskes

Abstract

At the core of state of the art microelectronic industry's drive for better technology, lies the continuing advancement in the development of Integrated Circuits using highly complex lithography machines, known as lithosteppers, which embed complex mechanical sub-systems performing intricate motions. These systems are controlled by means of custom real-time computing platforms containing off-the-shelf and specialized hardware components, and are optimized to keep pace with the continuing growing trend in performance requirements. At the ASML *Twinscan* lithostepper's heart resides the Control Architecture Reference Model (CARM) *motion control platform* which manages, among others, the *wafer-stage*, a multiple degree of freedom module, able to position a 15 kg heavy wafer-table with nanometer accuracy at extremely high acceleration and velocity. As the industry requirements for feature-size, overlay accuracy, and throughput keep increasing, the ASML lithosteppers, and the CARM platform in particular, should anticipate these demands by making early changes and upgrades with respect to computational performance and accuracy. Given that the current lithostepper configurations are not capable of sustaining the anticipated updates, which requires the increase of the control loop execution frequency from 20kHz to 40kHz, an early evaluation of potential CARM High Performance Process Controller (HPPC) successors has been performed. This indicated that the NXP-Freescale T4240 processor can potentially fulfill the expected requirements, however, the evaluation lacks accuracy as it was performed on a benchmark code not reflecting the actual CARM workload. To circumvent this problem, in this thesis, we introduce a more accurate evaluation methodology, which relies on the actual motion control application running on the HPPC and is able to capture aspects as scheduling, parallelism, and processor resource usage. To this end we develop a set of custom performance benchmarks able to emulate the CARM environment and evaluate the Freescale T4240 processor in this new context. Our results indicate that the T4240 is able to deliver enough computation power to fulfill the control loop execution frequency upscaling requirement from 20kHz to 40kHz. Additionally, we demonstrate that due to its clustered hardware architecture one T4240 can sustain 20kHz loop execution frequency for the workload of three current HPPCs, which suggest that its utilization in current lithosteppers can be beneficial.

Laboratory : Computer Engineering
Codenumber : CE-MS-2018-12

Committee Members :

Advisor: Dr.ir. S.D. Cotofana, CE, TU Delft

Chairperson: Prof. Koen Bertels, CE, TU Delft

Member:	Dr. ir. T.G.R.M van Leuken, CAS, TU Delft
Member:	Ing. J van de Ven , Motion Control and Subsystem Facilities, ASML Netherlands

Dedicated to my family and friends

Contents

List of Figures viii

List of Tables ix

List of Acronyms xii

Acknowledgements xiii

1 Introduction 1
- 1.1 Nano Lithographic Challenges . . . 2
- 1.2 Problem Statement . . . 2
- 1.3 Thesis Contributions . . . 4
- 1.4 Thesis Organization . . . 5

2 Background and Preliminaries 7
- 2.1 ASML Lithostepper . . . 7
- 2.2 CARM Motion Control Platform . . . 9
  - 2.2.1 Software Platform and Controller Modeling . . . 10
  - 2.2.2 Hardware Platform . . . 12
- 2.3 CARM Hardware Limitations and Upgrade . . . 12
- 2.4 T4240 Processor . . . 14
- 2.5 Conclusion . . . 15

3 CARM Application Scheduling and Execution 17
- 3.1 Motion Control Application . . . 17
  - 3.1.1 WorkerBlocks . . . 18
  - 3.1.2 Sequences and Schedule . . . 19
  - 3.1.3 Synchronization and Communication . . . 20
- 3.2 Previous Work . . . 21
  - 3.2.1 Performance Estimation . . . 21
  - 3.2.2 Experiments . . . 22
  - 3.2.3 Results . . . 23
- 3.3 Conclusion . . . 25

4 CARM Custom Performance Estimation 27
- 4.1 Multi-core Performance Estimation and Benchmark Creation . . . 27
- 4.2 Round Trip Latency . . . 28
  - 4.2.1 Implementation . . . 29
- 4.3 Parallel Sequences . . . 30
  - 4.3.1 Implementation . . . 31

4.3.2 Experiments . . . 34
4.4 Experimental Evaluation Platform . . . 35
4.4.1 Baseline Validation . . . 36
4.5 Results . . . 37
4.5.1 Round Trip Latency . . . 37
4.5.2 Parallel Sequences . . . 39
4.6 Conclusion . . . 41

5 Conclusions and Future Work **43**
5.1 Summary . . . 43
5.2 Future Work . . . 44
5.3 Research Implications . . . 45
5.3.1 CARM Team and ASML . . . 45
5.3.2 Industry . . . 45

Bibliography **48**

A Appendix **49**

B Appendix **53**

List of Figures

1.1 Schematic representation of the photolithography process 1

2.1 An ASML Twinscan lithography machine. 8
2.2 Abstract representation of a Long-Stroke, Short-Stroke wafer-positioning stage in H-configuration, where the actuators act on the wafer-table in a lithostepper. 9
2.3 Abstract representation of a control loop 9
2.4 Organization of the CARM layers 11
2.5 CARM ATCA compute stack. 13
2.6 Abstract representation of the T4240 processor architecture. Figure taken from Freescale T4240 processor factsheet [1] 14
2.7 Abstract block diagram of the T4240's E6500-core pipeline architecture. Figure taken from the Freescale T4240 Reference Manual [2] 15

3.1 Blockdiagram representation of an example of the control loop for the SS-controller application [3]. 17
3.2 Abstract overview of the deployment of an application on the CARM hardware (Host and HPPCs). 18
3.3 Schematic representation of the critical and non critical part of the sample. 20
3.4 Visual representation of a schedule calculated for an application running on 7 cores of an HPPC. The red (left) and green (right) sections represent time critical and non-time critical parts respectively, while the arrows represent a communication or synchronization between WorkerBlocks running on separate cores. [4] 21

4.1 Schematic representation of the round trip latency experiment via the different cache levels. 29
4.2 Schematic representation of the round trip latency performance tests. . . 31
4.3 Abstract diagram of a block template used for the WorkerBlocks 32
4.4 Block diagram for the stand alone sequence test. 34
4.5 Block diagram for the start-finish synchronized sequence test. 34
4.6 Block diagram for the fully synchronized sequence test. 35
4.7 Schematic representation of the execution of the benchmark running parallel sequences. 35
4.8 Abstract block diagram of the hardware setup for T4240 performance testing and validation. 36
4.9 Round Trip Latency test results for the 1 to 1 tests in ns. 37
4.10 Round Trip Latency test result for the 1 to N tests in ns. 38

A.1 Abstract block diagram of the T4240 core-complex architecture. As can be seen, the clusters of 4 E6500 cores are share the L2 cache, and are connected to each-other via the CoreNet Coherency Fabric and L3 cache. Figure taken from Freescale T4240 Reference Manual[2] 50
A.2 Abstract block diagram of the T4240's E6500-core pipeline architecture. As can be seen, most of the pipeline has been duplicated to provide a dedicated functional unit or buffer per hardware thread. However, some of the critical (and more expenxive) resources are shared, like the Complex Unit and FPU. Figure taken from Freescale T4240 Reference Manual. [2] 51

List of Tables

2.1 Overview of shared critical functional units in the T4240's E6500 core pipeline. 14

3.1 An overview of the different WorkerBlocks running in the time-critical part in the SS-Controller scheduled on a P4080 HPPC. The critical part of the schedule contains 333 WorkerBlocks, in 7 sequences running parallel on the separate HPPC cores. 19

3.2 Overview of the computational block types used for the custom functional test [5]. 23

4.1 WorkerBlock details on memory and functional units utilization, and data-types (Integer or Float). 33

4.2 Comparison of the 1 to 1 Round trip latency results in ns for single and dual thread testing for both MBAR1 and SYNC operation. 38

4.3 Test results for the Parallel Sequence benchmarks, ran on both the T4240 and P4080. Results correspond to the average execution time of a thread running a single sequence workload. Both raw results in ns and normalized results are displayed, where the P4080 results have been set at 100. 39

4.4 Performance comparison between P4080 and T4240 SoC for the sequence tests, using different synchronization intervals. Values represent the performance increase of the T4240 over the P4080 when running the same workload on the processor. 40

Definitions and Acronyms

ATCA Advanced Telecommunications Computing Architecture

AMC Advanced Mezzanine Card

BSP Board Support Package

CARM Control Architecture Reference Model

DUT Device Under Testing

FPGA Field Programmable Gate Array

HPPC High Performance Process Controller

IC Integrated Circuit

I/O Input/Output

IPMI Intelligent Platform Management Interface

Ln cache Level n cache

nm Nano Meter

OS Operating System

RTS Real Time System

RPC Remote Procedure Call

RDB Reference Design Board

SCH Scanner Control Host

SoC System on Chip

SS Short Stroke

SRIO Serial Rapid IO

CARM facillity Implementation of the motion control platform using the Control Architecture Reference Model, mostly referred to as "CARM"

CARM stack An instance of the complete CARM facillity for a specific motion-controlled mechanical subsystem

Recipe Description of a procedure, containing the full set of relevant hardware, software and mechanical parameters and boundaries, specifically tuned by the user of the lithostepper.

Worker Single thread or process within a HPPC that can run the code of a sample. Multicore processors may contain a single worker per core or hardware-thread.

P4080 The Prodrive HPPC module containing the P4080 processor, used as HPPC within the CARM motion control platform.

T4240 The NXP Freescale T4240 PowerPC processor, used in the Vadatech AMC 702 module used as Device Under Test for this project.

Testbench Hardware and software environment emulating the (part of) ASML lithostepper and running the CARM motion control software for debug and benchmark purposes

Acknowledgements

Being part of a team is something that helps me personally to achieve tasks thought to be unachievable. Even though the project was my own, the team around me always gave me the possibility to discuss ideas and issues which helped me in proceeding with the project and achieving the goals set.
I would like to thank my colleagues of the CARM team at ASML for giving me the opportunity to perform research and development as part of my Embedded-Systems graduation project. I had a wonderful time working there, expanding my knowledge of the hard- and software development done within ASML. Furthermore i am looking forward to starting my career among these colleagues, giving me the opportunity to further develop my personal skills in the field, and being part of the continuing technological advancements made at ASML.

I would like to thank my supervisor Sorin Cotofana for the support during the long period of writing this thesis.

I would like to thank my girlfriend Jojanne for keeping me sane during the hard moments, and encouraging me to finish the project even when i did not feel i could.

Joachim Adriaan Voskes
DELFT, The Netherlands
May 14, 2018

1 Introduction

The technological advancements of the last couple of decades have played a large role in shaping the current socioeconomic environment we live in today. Most, if not all, of our daily activities have been impacted by the fast emerging changes in technology, ranging from transportation to communication techniques. The largest contribution to these changes is the rapid evolution of the electronic industry, especially in the microelectronics world.

At the core of each electronic device we utilize daily, lies at least a digital Integrated Circuit (IC), which mostly consists of nanoscale transistors, arranged to form complex computational structures and memory modules. Ever since the IC originated in the 1960s, size, speed, and capacity have increased significantly. In the last decades, the number of transistors on an IC roughly doubled each 18 months. This pattern follows Gordon Moore's predictions in 1974, such that current day IC's contain between just a few and billions of circuits, which allowed for high speed and low power dissipation at a reduced fabrication cost.

Fabrication of these ICs is performed by means of *photolithography*, imprinting nanoscale circuitry patterns onto a silicon *wafer*, much like a camera exposes an image on film. This process is executed using a set of complex *lithographic stepper* machines, the so called *lithosteppers*. In order for a lithostepper to print a circuit pattern onto a silicone wafer, UV-light is directed from a powerful light source, through a specific pattern (*reticle*) and a custom set of lenses onto the silicone wafer, as has been schematically depicted in Figure 1.1. To create the large amount of layers which make up the eventual chip, this process requires a large number of iterations.

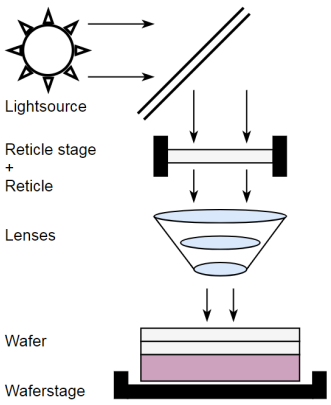


Figure 1.1: Schematic representation of the photolithography process

Furthermore, as a wafer typically holds more than one *die* (single IC), the patterning of the wafer is performed a large number of times in an as small timespan as possible to increase machine throughput. As moving the lithosteppers' light source would be a difficult and costly procedure, both the wafer and reticle are moved by their respective mechanical stages, in a synchronized fashion instead. As each iteration of exposed layers on the wafer has to be aligned with the underlying layer with as little *overlay* offset as possible, the 6 degree of freedom movable *wafer-stage* and *reticle-stage* are capable of moving with a nanometer accuracy, high acceleration, and velocity.

ASML, based in Veldhoven, The Netherlands, is one of the largest suppliers of lithog-

raphy machine for the semiconductor industry, and strives to manufacture and service these lithography machines for its customers while also continuously innovating the exposure process.

1.1 Nano Lithographic Challenges

For all of the IC-manufacturers in the semiconductor industry, the continuing trend of technological advancements in the lithographic process leads to an increased product quality, manufacturing productivity and improved profitability on the world market. As the semiconductor industry keeps pushing these requirements on the lithostepper machinery used in the lithographic process, ASML as developer of these machines has to evaluate the need and validity of software and hardware updates for current and future machines to accommodate these requirements. This means that ASML has to perform cutting edge research and development for these update and upgrades, ultimately driving the industries' technological advancements.

For a long time now, the most important parameter in the semiconductor industry is the so called IC *feature size*, describing the smallest possible geometrical shape being created by a lithostepper for an IC, and is measured in nanometer.

The shape and size of the features created on the silicone wafer, depend on the utilized UV-light wavelength, in combination with the reticle shape and the precise movements of the wafer- and reticle stage during exposure. As current average feature sizes lie at 7 nm diameter, and is expecting to shrink even more in the coming years, this means that the different aspects weighting in on the exposure process should accommodate for this change. This would mean that the wafer-stages and reticle-stages should increase their nanometer-accurate precision capabilities, to keep the current low error rate and high wafer yield. Furthermore for the semiconductor industry, the throughput (in wafers per hour) is an essential part of cost optimization. Moving to a higher velocity for the moving parts of the machine, would increase this much appreciated metric.

1.2 Problem Statement

A key element of the ASML lithosteppers is the Control Architecture Reference Model *(CARM)* motion control platform, which controls some of its most critical moving parts like reticle-stage and wafer-stage. To accurately control these precise mechanical sub-systems, it executes complex and highly time-critical control-loop applications, on a hybrid computing platform. This hardware platform consists of a set of *Field Programmable Gate Arrays* (FPGAs) and *High Performance Process Controllers* (HPPCs), integrated in a cabinet rack on the outside of the lithostepper machine. The HPPCs are off-the-shelf, general purpose processor modules, tasked with executing the complex real-time motion control calculations needed for moving the wafer-stage and reticle-stage mechanics at the required precision and velocity.

As previously discussed, ASML lithosteppers are updated and upgraded continuously, to accommodate semiconductor industry demands and ASML-internal performance goals. Some of the currently requested updates for the lithosteppers, impact the CARM

motion control platform and have been discussed between the system architects for the upcoming iteration of developments.

One of these requirements states **the increase of sample execution frequency** of the control loop applications running on the CARM motion control platform, from **20** kHz to **40** kHz. Increasing the sample loop frequency of the applications running on the HPPC modules, allows for an increase in calculation precision and sensor accuracy, thus enabling an increased mechanical velocity and precision for the wafer- and reticle-stage mechanical platforms. Furthermore, updating the motion control application in this fashion would contribute to the overall lithostepper performance increase, reducing errors and increasing production throughput.

The current HPPCs used in the CARM platform consist of NXP Freescale P4080 processors, containing an 8-core PowerPC processor, and running a custom Board Support Package (BSP) as Operating System (OS). On top of this custom BSP, the motion control applications are executed. Even though these processor modules have been tested and intensely optimized, they were ultimately found unable to handle the required increase in control-loop sample frequency to 40 kHz. Therefor the decision has been made to upgrade the computing platforms' hardware architecture, and replace the current HPPC processors with a more powerful processor.

Preliminary investigations[5] in the search of a P4080 replacement candidate identified the dual-threaded 12-core NXP Freescale T4240 as a viable candidate. Simulation results indicated that the processor cores of the T4240 module can sustain at least 3 times the execution rate reached by the P4080 processor cores. Furthermore, due to the architectural configuration of the T4240 containing 3 separate core-clusters of 4 dual threaded cores, the possibility to **reduce the *Cost of Goods*** of the CARM compute platform by replacing up to 3 P4080 HPPCs by a single T4240, has been theorized.

Analysis of the used methodology in previous work has however shown that the performance estimations for the T4240 are based on generic performance tests, not tailored to the CARM motion control application behaviour and design. The experiments were executed as single-core applications only, based on which the performance estimations are extrapolated to represent the full processor.
Analysis of the actual motion control application run on the CARM platform, has shown to contain a complex parallel execution schedule, with a large amount of synchronization and communication between the parallel running workloads, indicating that the performance estimations done in previous work are an unreliable basis for the current upgrade plans, and further research is required.

Given this statement, this thesis addresses the following question:
What type of tests are needed to validate the performance of a new processor against stated requirements, without the need to fully integrate a hardware-software implementation of a processor candidate in an ASML lithostepper prototype.

In order to qualify the NXP Freescale T4240 processor module as a certain alternative solution to the stated problem, this thesis investigates the following:

- What type of performance estimation experiments have been used in previous work, and what where their results and conclusions.
- What execution and scheduling profile is utilized by the CARM platform for running the motion control applications.

1.3 Thesis Contributions

In this thesis it is shown, by analyzing the actual embedded application run on the CARM platform and extracting key parameters and behavioural concepts, that previous work with respect to upgrading the CARM hardware has not been sufficient in determining realistic performance estimation for the processor candidates to replace the current HPPC processor modules. It is shown that the applications used do not amply represent the actual running application character with respect to its parallelism and synchronization. This thesis has had the goal to further analyze the current software running on the CARM hardware platform and develop a more accurate performance estimation procedure for the NXP Freescale T4240 processor candidate. The following tasks have been performed and contributions made during the thesis project:

- To be able to perform the necessary experiments and evaluations on the selected NXP Freescale T4240 processor as candidate replacement for the P4080, a test-environment was needed to observe and compare its behaviour. However, due to the fact that the an actual ASML lithostepper prototype is a too costly and difficult device to use as development environment, we built a custom *testbench* environment during this project, that is capable of emulating the needed hard- and software behaviour of an actual lithostepper motion control platform. The T4240 and P4080 processor modules have been installed into this test rack, and have been configured to represent the actual computing platform used in the CARM platform.
- In order to run any application level software on the T4240 processor module, we adapted and modified the custom Board Support Package for the current PowerPC based HPPCs, so it can be deployed and run on the T4240 processor architecture embedded in the CARM test environment.
- Furthermore, to avoid spending a majority of the limited project-time on reworking and adapting the actual CARM motion control software layer to the T4240 test environment, a detailed analysis of the complex Short-Stroke control-loop application running within CARM, has been performed. Unique characteristics, execution behaviour and bottlenecks that define the application are studied and isolated.
- Based on the isolated characteristics of the Short-Stroke control-loop application, we developed a set of custom performance benchmarks, that capture the real applications behaviour, while not requiring the full integration of the candidate processor into an actual CARM hardware platform of a lithostepper prototype.

- Experiments are conducted by us to evaluate the potential performance of the T4240 with respect to the current P4080 processor, using the developed benchmark applications running on the BSP.
- The results gathered from the experiments provided credible evidence that the T4240 is capable of sustaining the control-loop frequency increase from 20 to 40 kHz. Moreover it has been proven that due to the T4240 architectural design, it is cable of running up to three control-loop applications running at 20 kHz. However, it is shown that a combination of these two is not possible without severely compromising the real-time behaviour or computational accuracy of the system.

1.4 Thesis Organization

This report is organized as follows: Chapter 2 provides background information of the ASML lithostepper, and an in-dept analysis of the soft- and hardware components of the CARM motion control platform. Chapter 3 discusses the analysis of the motion control application running on the HPPC modules within the CARM platform, and briefly presents previous work performed with respect to upgrading the CARM hardware platform and discusses the hardware architecture of the processor candidate. In chapter 4 the experimental setup, implementation, and results are provided for the custom benchmark applications developed and run for this project. The results are analyzed and compared to previous work and expectations. Chapter 5 discusses the conclusions made based on the research and experiments, and future work is presented. Furthermore a detailed exposition of the possible implications based on the conclusions of this project is given.

2 Background and Preliminaries

In this chapter, a description of the ASML lithoscanners is given, and the implementation of the hardware- and software framework of the CARM motion control platform is discussed in detail. This information is meant to serve as foundation for key concepts and further research and development activities performed for this thesis.

2.1 ASML Lithostepper

As discussed in Chapter 1, during the chip fabrication process, the silicone wafer is patterned with nanometer sized structures using UV-light, in a process using various exposure and layering steps in multiple iterations. The lithostepper used to perform this process is build up of various components, which run in unison to perform the lithographic process. All the different components are controlled by separate systems, with their specifically tuned parameters provided in form of a *recipe*, created by the end-user of the lithostepper, to adhere to the specific needs and desires for the end-product.
Looking at the ASML TwinScan^{TM} lithostepper as seen in Figure 2.1, the machine can roughly be divided into different key components as follows.

The *Illuminator* light guidance system is used to guide the UV-light generated by the light source, via a set of mirrors and lenses, into the lithostepper where it is shaped into the right beam format and pattern.

The pattern of the IC circuit to be printed onto the silicone wafer is represented as various transparent and opaque areas of a quartz glass plate, called a *reticle* or photomask. The UV-light is guided through the reticle by a set of mirrors and lenses, forming a precise image in the shape of the reticle pattern. The reticle is housed in the *reticle stage*, which embeds a set of multiple degree of freedom actuators able to perform nanometer precise movements of the reticle.

The image created by passing the UV light through the reticle is focused and reduced in size by a large column of *lenses*, again shaping and forming the image into its final form, to be projected onto the surface of the wafer. The UV-light exposes the silicone wafer, creating the features that make up the eventual ICs.

As the wafer contains a large set of single ICs (dies), the wafer is moved by the *wafer-stage* containing 6-degree of freedom actuators, precisely positioning the wafer under the lens column for repeated image exposure. Furthermore, to accurately measure and position the wafer, a secondary wafer-stage is active in the machine, scanning and mapping the next wafer before its image exposure. This way, the machine is pipelining the lithograpic process by measuring the next wafer while exposing the current wafer. This simultaneous *step and scan process* reduces throughput latency, while increasing

the process accuracy.

To move the wafers from and to the wafer-stages from outside of the machine, the *waferhandler* is available, which is a robotic arm capable of carefully and precisely moving the wafers.

Controlling these different mechanical parts is done by a large amount of time-critical motion control platforms, within and outside of the lithosteppers. These motion control platforms play a large role in the sensitive and nanometer accurate fabrication process, and therefor are critical for the quality and throughput of the wafers going through the lithographic process.

Figure 2.1: An ASML Twinscan lithography machine.

For the wafer and reticle to be moved at the desired velocity with nanometer precision in a synchronous fashion, the wafer-stage and reticle-stage contain a multi-stroke positioning system, with accurate sensors and actuators for each axis. The multi-stroke stage system enables the achievement of the needed fine resolution. A schematic representation of a multi-stroke positioning system and its actuators can be observed in Figure 2.2. The multi-stroke mechanism contains the so called *Long-Stroke* for the course movements and positioning, while the *Short-Stroke* is used for the nanometer precise alignment and corrections of the component.

For these stages to be controlled with the desired movement pattern, velocity and precision, a large set of complex frameworks with *motion control applications* are running on various computing platforms, processing the nanometer accurate sensor data and controlling the actuators of the corresponding mechanical systems.

To move the wafer using the multi-stage actuators, the motion control platform runs a set of calculations and measurement algorithms as *controller*, that determine the desired accurate behaviour of the actuators based on the sensor input, feedback, and feedforward data from various sources within the machine and the lithographic *recipe* used by the machine. This type of controller is known as a *servo control loop* or *servoloop*, and can be found in many of the motion controlled parts of the ASML lithosteppers.

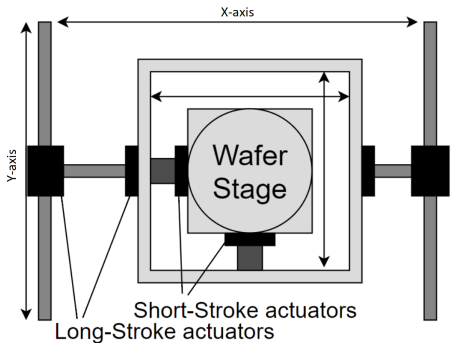


Figure 2.2: Abstract representation of a Long-Stroke, Short-Stroke wafer-positioning stage in H-configuration, where the actuators act on the wafer-table in a lithostepper.

An abstract and simplified schematic of a control loop algorithm is depicted in Figure 2.3. The desired signal that the actuator must react to is known as the *reference*. During operation, the current output of the actuator is accurately measured by a set of sensors and used as feedback, after which the difference between the feedback and the reference is calculated as the so called *error*. The controller translates this error value into a dedicated control signal for the actuator, with the goal to minimize the current error, and thus accurately control the actuator following the given reference.

The single loop of measuring the sensors, calculating the error and its corresponding control signal, and updating actuator control signals is called the *sample*. The control-loops algorithms used for the ASML lithosteppers are implemented with real-time behaviour, meaning that a single sample of the control loop has to be finished within its given time-budget.

To be able to abstract away from the large subset of complex servo control loops in the various parts of the lithostepper, the *CARM* motion control platform has been developed as framework platform to conceptualize and streamline the development for the different specific hardware platforms and layers.

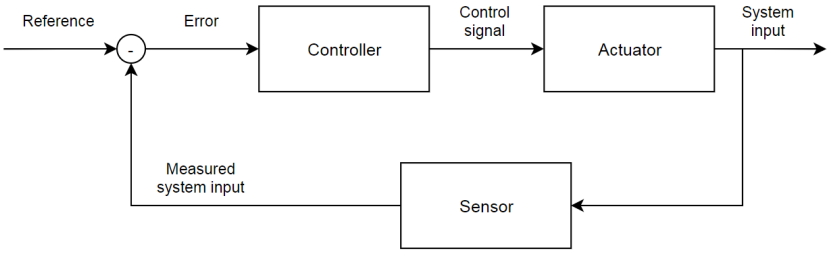


Figure 2.3: Abstract representation of a control loop

2.2 CARM Motion Control Platform

A large part of the different servoloop controllers used for motion control within the ASML lithosteppers, are modeled and developed using the *Control Architecture*

Reference Model (CARM) design. This design methodology has been developed to support clear design analysis and construction using a set of coherent and consistent concepts and domain layers. This methodology enables furthermore the multiple parallel design processes to abstract away from the large number of specific and complex configurations and motio control methods used in the different parts of the lithostepper.

2.2.1 Software Platform and Controller Modeling

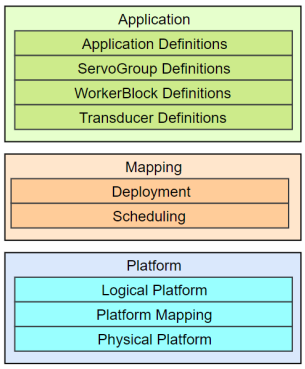
The main motion control platform based on the CARM architectural concept is *"CARM Facillities"*, in short also called CARM. Within the CARM platform, a motion control application is implemented as a network of servoloop controllers, the so called *ServoGroups*. A single ServoGroup contains a closed control loop algorithm build out of different control blocks (WorkerBlocks), and is responsible for the actuation of a specific subsystem within the motion control platform, the movements of the *wafer-stage* or *reticle-stage* for instance.
There are many ServoGroups present within the platform, working together to actuate on the physical motion sub-systems. A ServoGroup contains a large set of different types of interfaces linked together, where a typical representation contains *control blocks*, *measurement system interfaces* and *actuator interface blocks*. Furthermore, all components within the same ServoGroup run at the same sample- and execution-frequency, synced by an external synchronization controller.

The *WorkerBlocks*, which are instantiated control blocks executing the algorithms within the ServoGroup, model and implement the required functionality to perform calculations based on specific and time-critical inputs and parameters. These input parameters are either at runtime gathered from one or multiple other WorkBlocks in the control loop chain, or external parameter files loaded at run- or design-time as fixed properties. Examples of WorkerBlocks are for instance a *Nth order filter*, *M×N matrix multiplication* or *gain* functions, but also *status updates* and *signal tracing*.

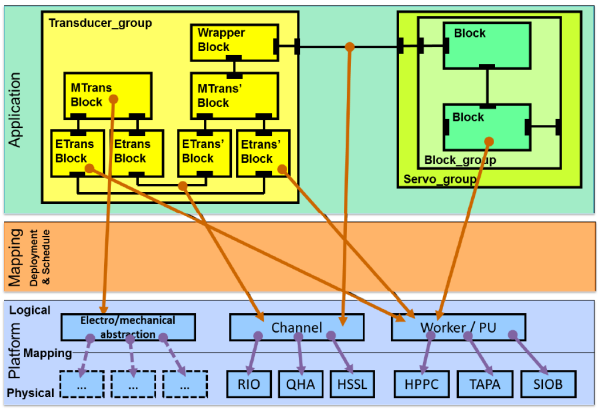
WorkerBlocks are sequenced together following a static schedule, which is based on critical timing requirements and known hardware specifications of the computing platform, to make sure that the lithostepper execution deadlines are kept. Because the WorkerBlock execution characteristics and specifications of the hardware architecture and I/O is known, the schedule has a static character and does not change at run-time of the control loop. This enables a deterministic real-time behaviour within the execution of the control platform, adhering to the assigned real-time budget per *sample*. Besides the order in which the WorkerBlock are executed, also the specific moment of synchronizations and data to be synchronized is implemented in the schedule. These scheduled sequences are then deployed on the different HPPC processor cores, and executed in a highly parallel and synchronized fashion.

The CARM motion control platform running the servo controllers as described previously, consists of three domain specific layers, each with their own software-hardware abstraction and implementation: The *Application layer*, *Platform layer* and *Mapping*

layer, as seen in Figure 2.4a with their containing architectural components.



(a) Abstract overview of the three layers within the CARM software platform.



(b) Example representation of application model, mapped onto logical and physical platform layer. Figure taken from [6].

Figure 2.4: Organization of the CARM layers

The **Application Layer** contains the control logic and its behavioural description. This layer implements the actual motion control application (PGAPP), the ServoGroup descriptions (PGSG), and their respective WorkerBlock (PGWB) components and interfacing. It furthermore models the behaviour for the sensors and actuators (Transducers) of the systems, which are observed and controlled by the application.

The **Platform Layer** provides the logical representation of the hardware and the actual interfaces to the hardware platform.
The *Logical* hardware representation contains all the necessary hardware configuration data available for the system, like location, processor types and interface types, while abstracting away from all the actual hardware configurations.
The *Physical* hardware contains the description of the actual hardware, and the physical connections present in the lithostepper.
The Platform Layer furthermore provides mapping from logical hardware onto available physical hardware by defining directed associations between the logical description and corresponding hardware available in the system.

The **Mapping Layer** implements the bridge between the Application Layer and the Physical Layer. This layer maps each element from the Application Layer onto a logical hardware element in platform layer based on known software and hardware parameters and requirements. An abstract representation of the mapping between the different layers can be seen in Figure 2.4b.

2.2.2 Hardware Platform

The hardware platform for the CARM motion control system is a hybrid control-platform, containing a multi-module setup within an *ATCA*[7] rack. An schematic overview, and photo are displayed in Figure 2.5a and 2.5b.

The **ATCA** rack contains a set of *blades* which house and interconnect the computation- or communication-modules, and uses a network of Ethernet and Serial Rapid IO (SRIO) [8] protocols. The slots within the blades are able to house different types of *Advanced Mezzanine Card*s (AMC modules) containing one or more processors, FPGAs, or IO-boards for a particular purpose. Within the CARM ATCA racks, there are two main components relevant for this exposition, namely the *Host* and the *High Performance Process Controller* (HPPC).

Hosts are single-core NXP Freescale MPC8548 [9] *PowerPC* processors, running a version of Linux OS. Hosts act as the controlling body for the CARM subsystems, and configure and manage the system-drivers, interconnect networks, HPPC workload deployment, and the inter-module synchronization signals.

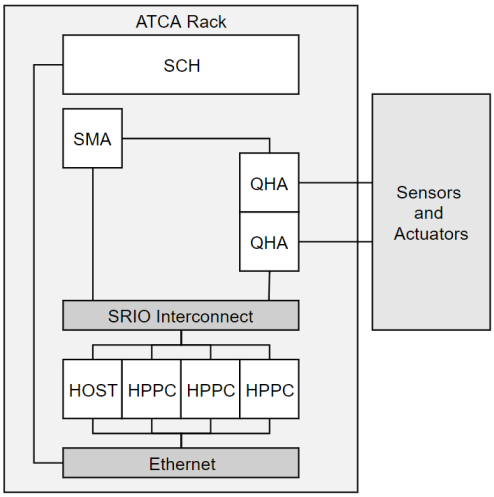
HPPCs are 8-core NXP Freescale P4080 [10] *PowerPC* processor modules, running a custom bare-metal Board Support Package (BSP). The BSP configures the HPPC, and initiates the strict real-time main application which is used to execute the scheduled control-loop applications. As the HPPCs are multi-core processors, generally one of the cores is reserved for background tasks, while the other cores run idle until a process is triggered for execution by a remote function call.

The connection to the actuators and sensors present in the machine is done via a set of so-called **QHA** interconnect modules, connecting the SRIO network of the system to the various types of actuator and sensor networks in the lithostepper and its sub-systems. The whole system is synchronized using the dedicated syncbus and Synchronization Manager (SMA) module, which provides external clock signals and interrupts to the other modules and the network.

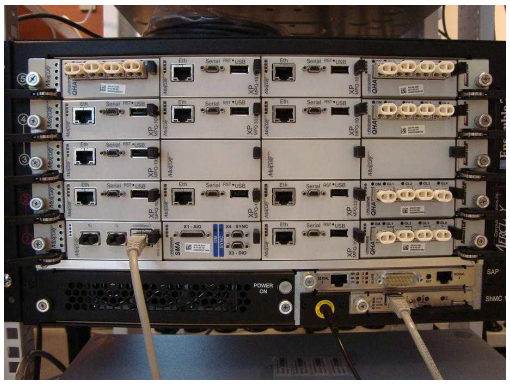
Furthermore, the system contains the **Scanner Control Host** (SCH). This is the main computer unit that governs the full software-hardware environment for the multiple compute racks of a lithostepper. The SCH embeds an Intel Xeon x86-64 processor with Linux OS, and is furthermore used to control the user-interface, managing software packages and schedule creation for the various control systems, among other things.

2.3 CARM Hardware Limitations and Upgrade

In light of the semi conductor industries' continuing trend to optimize ICs' cost and quality, while pushing the device boundaries and limitations, the critical parts of lithosteppers have to change accordingly. As the motion control platform is one of the most crucial parts of the machine, it stays in a constant state of being re-assessed and evaluated for software updates and upgrades to keep up with the continuing changes and refinements



(a) Simplified block diagram with relevant components and communication layers.



(b) Compute rack containing HPPC, QHA and Host modules.

Figure 2.5: CARM ATCA compute stack.

in the lithographic process. In view of this, requirements for the CARM motion control platform have been evaluated for the next generation of lithographic process optimizations with respect to speed and accuracy performance. The evaluations have led to the decision to increase the control-loop application execution frequency for the most critical controllers running on the HPPCs. Doubling the sample frequency from the current **20 kHz** to **40 kHz** would allow for higher order filter calculations and more accurate sensor and actuator control.

As stated in the hardware platform description of the CARM motion control platform, current HPPC processor modules contain the NXP Freescale P4080 octa-core processor. Initial tests and discussions within the CARM development team have indicated that the current P4080 HPPCs are not capable of fulfilling the 40 kHz requirement without compromising computational accuracy. The decision has therefor been made to replace the current HPPCs with a more powerful processor module, capable of delivering the necessary computational power.

Furthermore, due to the limited amount of space in the computing racks, a secondary requirement has been stated to *reduce the amount of processor modules in the racks*. This means that besides upgrading to more powerful processors for a 40 kHz execution, the new processor candidate should be able to handle a larger workload.

Previous work [5] has been performed to evaluate a set of possible processor module from different manufacturers, based on generic requirements like I/O support, multi-threading, raw processor performance, and possible long term support. Based on the initial performance and component tests, discussed in detail in Chapter 3, it was established that the NXP Freescale T4240 12-core PowerPC processor can be selected as the most likely successor. Previous work has concluded that this processor module is capable of sustaining a control-loop application execution at 40 KHz, and is due to

the clustered setup of the 12-cores and increased interconnect specifications capable of replacing up to 3 current HPPC modules.

2.4 T4240 Processor

NXP's Freescale QorIQ T4240 is a high-performance communications processor, containing 12 dual-threaded e6500 PowerPC cores running at 1.8 GHz. The clustered architecture of three times four cores allows for multiple configurations and utilization scenarios. The processor architecture is detailed in Figure 2.6, and in Appendix A Figure A.1. The shared L1 cache between two hardware-threads of an E6500 core, and the L2 cache between the four separate E6500 cores within the cluster, enable efficient data sharing between parallel running processes, while keeping latency low. The CoreNet Coherency Fabric and L3 cache connects the clusters, processor peripherals, and IO in a efficient manner, optimized for the multiple hardware configurations possible.

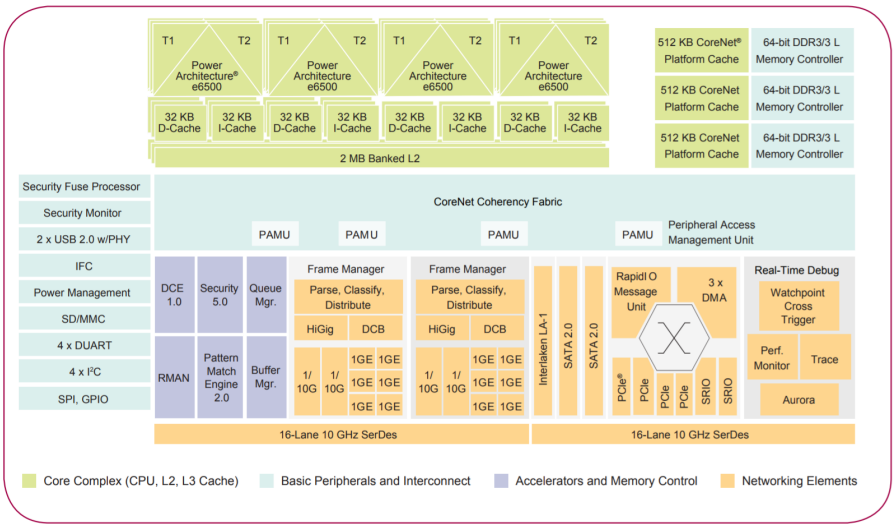


Figure 2.6: Abstract representation of the T4240 processor architecture. Figure taken from Freescale T4240 processor factsheet [1]

Table 2.1: Overview of shared critical functional units in the T4240's E6500 core pipeline.

Resource	Amount available	Distribution
Simple Unit	4	2 per thread
Complex Unit	1	Shared by 2 threads
Floating Point Unit	1	Shared by 2 threads
Load Store Unit	2	1 per thread
L2 Memory Bus	1	Shared by 2 threads

Looking at the E6500 processor core's pipeline in Figure 2.7, of which a larger figure can be found in Appendix A Figure A.2, and the summary of critical resource distribution

in Table 2.1, it can be observed that a large part of the functional units in the E6500 processors pipeline are double or quadruple implemented in the core, an thus available for a single hardware-thread's use. However, some of the critical units are shared between the two threads of the core.

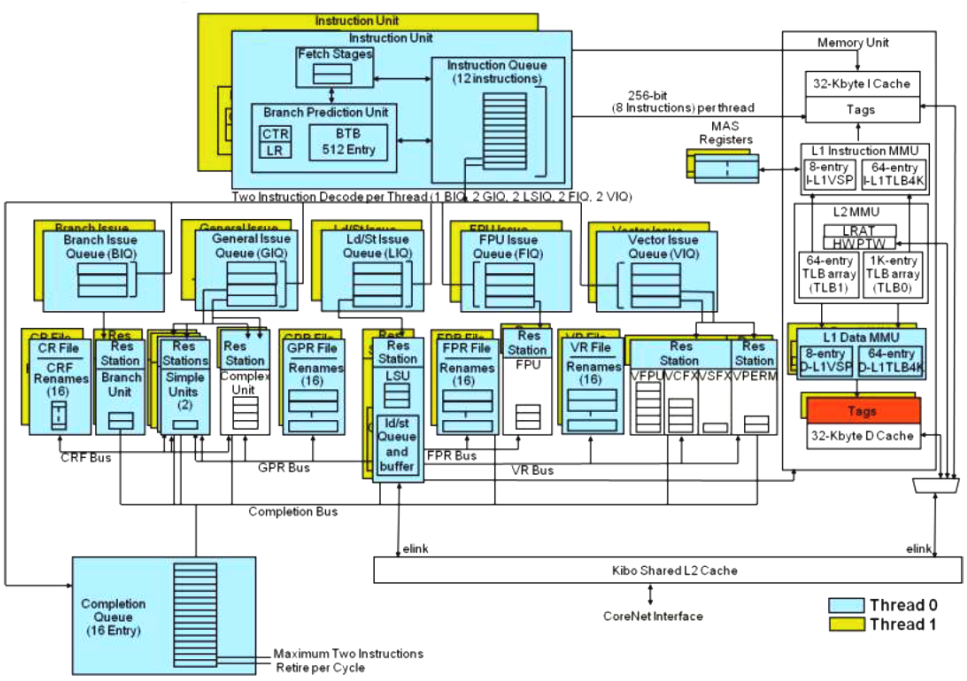


Figure 2.7: Abstract block diagram of the T4240's E6500-core pipeline architecture. Figure taken from the Freescale T4240 Reference Manual [2]

.

2.5 Conclusion

In this chapter we discussed the organization of the hardware-software environment of the CARM motion control platform of an ASML litostepper. The current industry driven requirements for the next iteration of updates are evaluated in light of this system, and it is discussd that the current platform is unable to handle the performance requirements without compromise. To sustain the requirements of upscaling the control-loop execution frequency from the current 20 kHz to 40 kHz, the NXP Freescale T4240 has been identified as candidate processor to replace the NXP Freescale P4080 HPPC. Furthermore, the possibilities to reduce hardware cost for the system was discussed based on the unique hardware architecture of the T4240. Potentially the T4240 is able to replace up to 3 current HPPC processors.

In the next chapter the study of the CARM platform is extended further by analyzing a complex control-loop application running on the HPPCs with respect to its scheduling and execution.

CARM Application Scheduling and Execution

3

In the previous chapter it is argued that the hardware of the CARM motion control platform has to be upgraded to be able to handle the required increase in performance. This chapter analyses the motion control application, running on the hardware platform in more detail. Based on this analysis, the results and conclusions stated by previous work[5] are revisited and adjusted accordingly, as basis for further action on the hardware upgrade.

This analysis focuses on the execution, scheduling, and communication of a servo loop application running on a single HPPC. As previously mentioned, one of the most complex, resource demanding and time critical controllers within CARM is the Short-Stroke (*SS*) controller, providing the nanometer accuracy of the wafer positioning platform, at a very high acceleration and velocity. Based on this we make use of the SS-controller as discussion vehicle with respect to computational and communication complexity, to gain a deeper insight into the execution of the applications running on the HPPCs.
Looking at the schematic example of the SS-controller presented in Figure 3.1, the complexity of the control-loop can be seen with respect to the different feedback and feedforward loops from both within its own context and from the Long-Stroke Controller as influence.

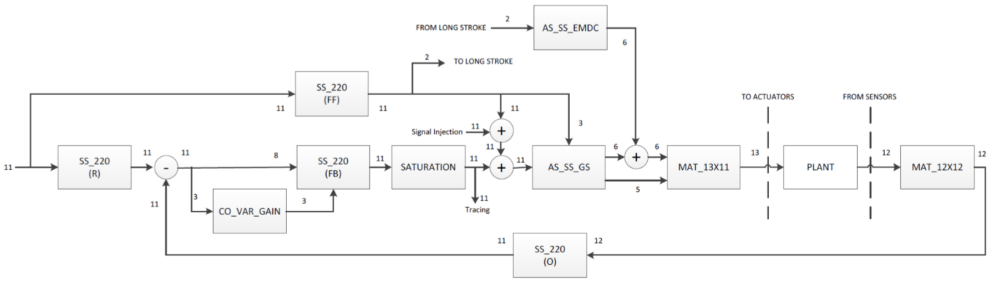


Figure 3.1: Blockdiagram representation of an example of the control loop for the SS-controller application [3].

3.1 Motion Control Application

The application layer of the CARM motion control platform has as goal to schedule and run the multiple parallel ServoGroups as optimized as possible, by deploying them on the different available hardware parts of the system. Each of the ServoGroups is deployed by the Host on a single HPPC module, and managed based on parameters and settings received from the applications recipe. Each HPPC runs a *Block Factory* and *Sequencer*

application initiated by the Host, instantiating the different WorkerBlocks with the correct parameters and interfacing, and schedules the blocks in separate sequences over the processor-cores according to a predefined schedule for optimal execution. An abstract representation of the deployment can be seen in Figure 3.2. The predefined schedule of parallel sequences is created to ensure that the servoloop application executes its *samples* of reading, computing and writing crucial data, according to the assigned time-budget, adhering to the strict real-time characteristics needed for the lithostepper to optimally function.

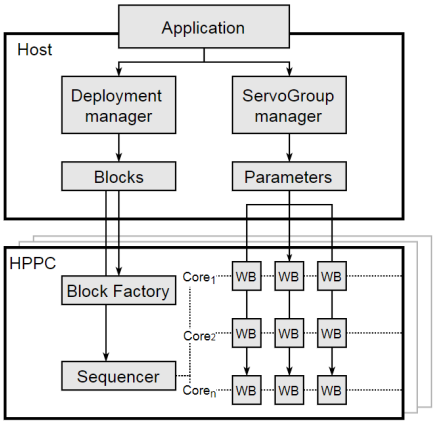


Figure 3.2: Abstract overview of the deployment of an application on the CARM hardware (Host and HPPCs).

3.1.1 WorkerBlocks

The WorkerBlock used by the servoloops are implemented based on a default interface structure, containing a set of functions and arguments to be used during execution. The *FullCalc* function calculates the full algorithm or operation of the WorkerBlock which in most cases consists of reading input arguments and performing a single algorithm or action before updating output arguments. To limit the amount of I/O-delay between the HPPC and the sensors or actuators in a servoloop, the FullCalc function is divided into two parts to distinguish between time critical and non time critical parts.
The *PreCalc* function is called in the non time-critical part of the sample, while the *PostCalc* function is called in the time-critical part of the sample.

Analyzing the time-critical part of the WorkerBlocks that make up the SS-controller, see Table 3.1, it is apparent that a large part of these WorkerBlocks are generic arithmetic functions or control and filter type functions. The remaining WorkerBlocks are mostly status and communication functions like remote procedure calls to sensors, actuators, and other HPPCs.

Table 3.1: An overview of the different WorkerBlocks running in the time-critical part in the SS-Controller scheduled on a P4080 HPPC. The critical part of the schedule contains 333 WorkerBlocks, in 7 sequences running parallel on the separate HPPC cores.

Block type	**Functions**	**Amount**	**Percentage of Total**
Generic arithmetic	SUMM	99	45%
	MULT	39	
	GAIN	12	
	MATRIX	3	
Control and Filters	FILTER	39	19%
	CONTROL	19	
	SWITCH	6	
Diagnostics and Communication	BLOCKGROUP ON_OFF	12	26%
	DIAGNOSTICS CHECKS	21	
	DELAY	20	
	MEASUREMENTS	11	
	PRODUCER CONSUMER	17	
	ACTUATOR	6	
Remaining		32	10%

3.1.2 Sequences and Schedule

As previously discussed, due to the multi-module setup of the CARM hardware platform, multiple alternatives exist for the motion controller ServoGroup deployment. To be able to run these ServoGroups most optimally with regard to time- and hardware resources, a static schedule is generated and analyzed at design-time at logical-platform level [11]. The schedule creates a set of specific sequences of WorkerBlocks for the active hardware configurations in the system. These sequences contain the function-calls to the *PreCalc*, *PostCalc* or *FullCalc* function of the WorkerBlocks, and the order in which they are called. The type of function call is based on the specific moment the WorkerBlock is being scheduled, either the time critical, or the non time critical part of the sample. Due to the strict real-time requirement for the servoloop, each schedule is generated with known hardware- and software-parameters like I/O- and function execution latency, to make sure that the sample is scheduled and executed within its given real-time budget. The SS-Controller, currently runs at a 20 kHz sample frequency, meaning a budget of 50 microseconds (μs) is available per sample.

Looking at Table 3.1 detailing a summary of the WorkerBlocks present in the schedule for the SS-controller, it can be seen that the time-critical part of the application has 333 WorkerBlocks. Scheduled over 7 parallel sequences an average of 47.5 WorkerBlocks have to be handled during the available time for the critical section of a sequence.

3.1.3 Synchronization and Communication

As the currently used HPPC contains a Freescale P4080 octa-core processor, a set of 7 WorkerBlock sequences is generated for the SS-controller, leaving a single core to be solely used for scheduling-, background- and BSP monitoring tasks.

Analyzing the set of 7 generated sequences for the SS-controller running on the HPPC, it is seen that each sequence of WorkerBlocks is split into two sections; *pre_CMD_sequence* and *post_CMD_sequence*. The pre_CMD_sequence is executed in the time-critical part of the sample, containing the *PostCalc* or *FullCalc* function of the WorkerBlock, while the post_CMD_sequence contains the non time-critical part of the sample.

The *CMD* is the *Command*, communicating the Critical-Data-Ready signals that synchronizes critical application data used by sensors and actuators. A schematic representation of the different parts of a sample can be seen in Figure 3.3.

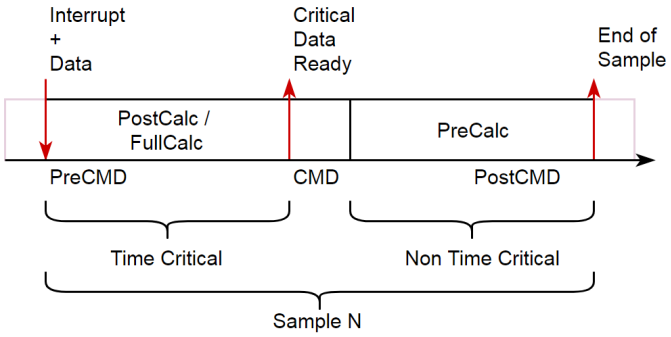


Figure 3.3: Schematic representation of the critical and non critical part of the sample.

Each of the sequences is scheduled to run on a different core of a single HPPC processor. To be able to correctly preserve WorkerBlock execution order, and to transfer data between the WorkerBlocks in the sequences, synchronization is performed between WorkerBlocks in parallel sequences.
The synchronization and communication between WorkerBlocks is implemented using guards, keeping track of *update* and *wait* statements in the generated schedule. The update statements ensures that the WorkerBlocks update an internal Integer value representing the state of the Block, which is publicly readable for other WorkerBlocks. The wait statement triggers a WorkerBlock to wait and make sure that one or more WorkerBlocks in other sequences have completed their tasks up to a certain update state, before proceeding with their execution or reading and writing parameters in memory. This guarantees that critical data are available at the right inputs and outputs when needed by other WorkerBlocks, and no race-condition or unexpected data dependency is created.
An example of a generated schedule of sequences can be seen in Figure 3.4, and as can be seen, a large amount of synchronization moments exist in the schedule. Furthermore, analyzing the SS-Controllers schedule, it can be deduced that the sequences are scheduled to synchronize between 23 and 33 times during a single execution, with an average

of 27 synchronizations over a sequence of 47.5 WorkerBlocks. Furthermore it is observed that the amount of threads involved in a single synchronization event ranges between 1 and all running sequences.

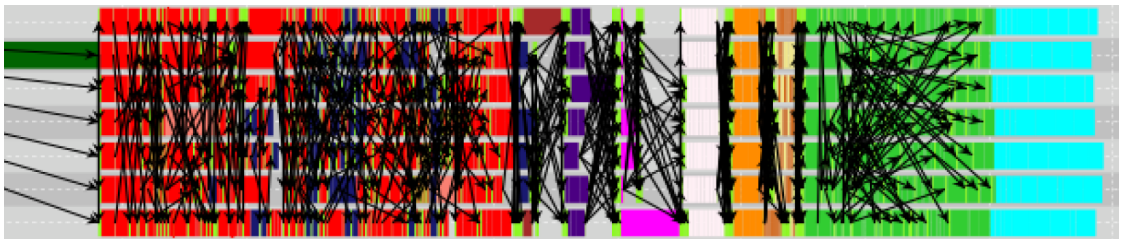


Figure 3.4: Visual representation of a schedule calculated for an application running on 7 cores of an HPPC. The red (left) and green (right) sections represent time critical and non-time critical parts respectively, while the arrows represent a communication or synchronization between WorkerBlocks running on separate cores. [4]

3.2 Previous Work

As discussed in Chapter 2, the current HPPCs of the CARM motion control platform are scheduled to be upgraded due to increased performance demands. In the preliminary research performed [5], a set of specific SoC modules have been taken under consideration to replace the currently utilized NXP Freescale P4080 processors in the CARM platform. The processors under consideration had been chosen based on the basic constraints of performance specifications, instruction set architecture, I/O compatibility, release date, and processor family. The processors under considerations are as follows:

- Intel *Broadwell D* hexa-core x86-64 processor [12].
- NXP Freescale *T4240* 12-core dual-threaded PowerPC processor[2].
- Texas Instruments *TI66AK* quad-core ARM Cortex A15 processor [13].

As can be seen, the processors candidates for replacing the current Freescale P4080 processor are from different manufacturers and have different architectures, thus, to be able to estimate performance and compare them, certain tests should be performed.

3.2.1 Performance Estimation

Chip manufacturers publish detail of processor parameters and expected performance estimations in their product reference manuals, for developers to quickly estimate and make comparisons. However, these published details are mostly theoretical expectations based on the architectural features of the device and generic workloads. Performance evaluation of the device can be performed by the broadly used method of benchmarking, enabling the precise estimation of processor performance. This enables software and hardware-architects to compare and make trade-offs in the design-space exploration of a system.

For the different processor types available on the market, various benchmark suites are accepted by the community to accurately enable processor performance estimation by identifying characteristics and bottlenecks using standardized applications. For the General Purpose Processors (GPPs), used in daily household devices like desktop computers and multi-purpose server racks, there is the SPEC benchmark suite [14]. Looking at the Embedded Systems industry, a set of different benchmarks is accepted as a reliable source of performance estimation, including the LMBench and NBench benchmark suites.

However, as has been stated previously, the processor modules used in the CARM platform are General Purpose Processors used in a highly complex embedded environment. The community accepted benchmarks used in both cases stated, are not tailored to the custom usage of the HPPC modules within CARM, and thus are unable to provide the necessary required information to accurately estimate the performance. Therefor, the claim is made that specific custom benchmarks are needed, which are tailored to the processor usage within te system.

3.2.2 Experiments

In previous work, the experiments performed to determine the most suited processor to replace the current HPPCs can be categorized into three different classes with respect to performance estimation techniques used; *Component Tests*, *Generic Functional Tests*, and *Custom Functional Tests*. These tests have been performed on stand-alone development modules distributed by their manufacturer, and running a generic Linux Operating System. The main goal for these experiments have been to determine which of the processor candidates is best suited for the short-term hardware upgrade, scheduled for the current HPPC modules in the CARM platform.

Component Testing

The component tests performed in previous work, so called *micro benchmarks*, are executed using the LMbench [15] test suite. Using the suits various benchmark applications, memory latency and bandwidth have been tested for *read*, *write* and *copy* operations on a large spectrum of data types and stride sizes. Furthermore, testing is performed using both sequential- and random- read and write operations for testing cache line performance. The tests are executed in a single-core single thread-fashion, using a variable sized linked list, running an arbitrary large test size of 1.000.000 loads to reduce jitter. Memory bandwidth and latency is measured for the different processors under consideration, and compared with the current P4080 processor.

Generic Functional Testing

A set of generic functional tests, so called *macro benchmarks*, is performed using the N-Bench [16] test suite, and can be divided into two classes; *Integer intensive* and *Floating-Point intensive*. These tests have been run in single-core single-thread fashion for both 32- and 64-bit data types.

The Integer-intensive tests executed consisted of the *Floating-Point Emulation*, *Huffman Compression* and *IDEA Cryptography*, while the Floating-Point intensive tests executed consisted of the *Fourier transform*, *Neural network simulation* and *LU decomposition* algorithms. Furthermore, limited tests have been performed in comparing the single and dual threaded capabilities of the processors, and their impact on execution duration.

Custom Functional Testing

A Custom performance test has furthermore been executed, focused on a more CARM application like execution of functional tests. A set of representative WorkerBlocks are synthesized and sequenced together in manner that is reminiscent of a single sequence running on a HPPC core during sample execution. The WorkerBlocks used are the following: Mult, Div, Mult-Max and 4x4 Matrix, containing the operations seen in Table 3.2. The performance test is performed in a single-thread single-core fashion, and is run a large number of iterations to reduce jitter and cold-start influences.

Table 3.2: Overview of the computational block types used for the custom functional test [5].

Block	**Function**	**Computation**	**Memory**
Mul	Multiplication	4 Multiplications	12 Loads 4 Stores
Div	Division	4 Division	12 Loads 4 Stores
Matrix	Matrix Multiplication	16 Multiplications	24 Loads 4 Stores
Mul-Max	Multiplication and Max-operation	4 Multiplications 4 Comparisons 4 Branches	12 Loads 4 Stores

3.2.3 Results

The **micro benchmarks** used for testing memory behaviour of the processors, show that for all processors the 32kB L1 cache behaves more or less similar. However, when looking at L2 and L3 cache results, large differences are observed between the different architectures. As both the T4240 and ARM Cortex A15 have shared L2 cache between 4 cores, their latency is slightly higher than expected, and appears to be slower than the P4080's L2 for small datatypes. However, as the L2 cache sizes for the T4240 and the ARM are much larger than that on the P4080, the overall performance of the processor cache is expected to be higher when working with large data types as no L3 has to be used in these cases. It can furthermore be seen that the bandwidth reached for all processors is approximately the same as reported in their respective documentation.

The **macro benchmarks**, testing functional performance of the processor cores show that for both Integer and Floating-Point tests, the Intel processor architecture is clearly much more powerful than the other processors. However, running the same tests on different cores have shown that the Intel processor has fluctuating performances depending on core choice. This fluctuating behaviour is not seen for the Cortex A15 and T4240, which show no performance difference between their cores. Furthermore, due to the architectural design of the Intel processor, it exhibits unexpected performance degradation and fluctuations when multithreading. These performance drops create *nondeterministic* behaviour at runtime, making performance predictions in time-critical applications problematic. When looking furthermore at the processor core performance of the T4240's E6500 core, it is seen that the performance of its hardware threads slightly decrease when multithreading with respect to single-threading, due to the shared resources in the pipeline. Furthermore, the cores don't differ from each other in performance, ensuring the needed deterministic behaviour regardless of chosen processor core.

The experiments with the **custom benchmarking** applications, running the single chain of WorkerBlocks, show that the processor cores of the Intel device perform up to **8x** faster than the P4080's E500mc cores, while the T4240's E6500 PowerPC cores perform up to **3x** faster. Performance of the Cortex A15 has however shown to be only marginally better than the P4080 and in some cases even worse, concluding that the Cortex A15 is not suited as replacement based on these experiments.

Based on the results of the performance analysis in [5] on the set of processors under consideration, the Freescale T4240 processor has been chosen as the device that is the most suited candidate to replace the current HPPC modules in the CARM platform. The experiments suggest that the E6500 PowerPC cores are capable of delivering up to 3x the performance of the currently used P4080 e500mc cores, thus having the potential to enable the performance increase to a control-loop sample frequency of 40 kHz.

Even though the Intel processor is clearly much faster, it adds a large uncertainty with respect to reliable deterministic performance when multi-threading, making it less suited for the time-critical application scheduling of the CARM platform.

Furthermore, the T4240 processor has the unique hardware architecture containing 3 independent core clusters, capable of running fully independent from each other. This feature creates the possibility to run multiple applications on the same processor in parallel, without interference from each other. This enables te ability to reduce the amount of processors in the motion control platform, without compromising in processor power.

Besides, as the T4240 is based on a next iteration of the PowerPC platform used in the P4080, adapting the current motion control applications would be less time and effort consuming with respect to the other architectures, given the already available infrastructure.

3.3 Conclusion

By analyzing the SS-controller, it can be observed that the SS-application provides a high level of task parallelism, while requiring a substantial amount of inter-core communication and synchronization. However, looking at the preliminary performance tests that have been performed, it is seen to focus mostly on single-core processing, while furthermore only exploring the multi-threading capabilities in generic terms. The parallelism and the communication that is extensively present in the motion control applications running on the different cores of the HPPC, is not at all used in the performance tests used in previous work.

It can be concluded that the method of testing in previous work has been adequate in determining the best suited processor to replace the current generation of HPPCs, and based on the results it is seen that the Freescale T4240 clearly seems to be the best option given the considered processors. However, it is clear that the performance estimation of the T4240 in previous work has not been based on techniques that emulating the actual contextual use of the processor within the CARM platform.
This means that the estimations given in previous work are not a reliable basis to perform the hardware upgrades scheduled. This indicates that further testing is needed, using more custom performance benchmarks that emulate CARM specific characteristics.

CARM Custom Performance Estimation

4

Reflecting on the software analysis performed on the CARM platform and the performance analysis experiments executed in previous work [5], the conclusion is made that the benchmarking application used to estimate the performance of the NXP Freescale T4240 processor does not adequately represent the motion control application running on the CARM hardware platform. As experiments performed in previous work have had the primary goal to distinguish between the different candidates as replacement for the current HPPC modules, it did not implement a context-realistic behavioural representation of the CARM motion control application. Although previous work did provide preliminary estimations with respect to the performance increase and cost of goods requirements discussed in Chapter 1, the test results and method of testing have not provided enough information for definitive judgment on the matter.

This chapter discusses the development of a custom performance analysis benchmark, based on the conclusions made from the previous work and the analysis of the motion control application running the SS-controller on the HPPCs. Using the custom benchmarking software, the requested performance increase of 40 kHz control loop frequency, and the cost of goods replacement possibilities are tested. The gathered results form a more accurate basis for estimation and validation of the performance expectations of the T4240 module, as it is based on key behavioural characteristics of a CARM motion control application execution.

4.1 Multi-core Performance Estimation and Benchmark Creation

Based on the analysis of the CARM SS-controller, a custom multi-core benchmarking program is sought for the Freescale T4240 Device Under Testing (DUT) performance estimation and validation, in a manner that reflects actual behaviour of the CARM platform in context of the motion control application.

As discussed in the analysis in Chapter 3, the control loops running on the HPPCs are scheduled and executed in a complex parallel fashion, with a large amount of inter-core synchronization and communication between instruction sequences (threads) running on different processor cores. A key performance metric for such a multi-threaded workload relates to the way the execution time scales when the amount of parallel threads composing the workload is increasing. Ideally, to fully utilize the available application and platform parallelism, performance improvements should scale linearly with the amount of threads. However, as the software analysis of the SS-controller has shown, there are certain characteristics that limit the performance increase when

parallelizing the application.

The two most apparent bottlenecks identified from the analysis, which are determining the performance of the processor running the parallel control loop sequences, are the ***synchronization*** between the processor cores, and the ***contention*** on shared resources when executing parallelized applications. To be able to measure the impact of these phenomena, a set of two distinct performance benchmarks is created and executed to compare T4240 performance to the current P4080 HPPC within a CARM test environment (*testbench*). These two tests are the ***Round Trip Latency*** and ***Parallel Sequence Execution***.

These custom developed benchmarks are implemented on top of the adapted P4080 BSP for the T4240, and based and previously performed experiments by [5]. The software package containing both the T4240 BSP and the benchmarking application is compiled into a PowerPC E6500 compatible image, and flashed onto the T4240 using the Vadatech provided *UBoot* bootloader. After booting and initialization sequences are complete, a single master thread initiates the main function of the custom benchmark suite, while the other hardware threads (cores) wait for activation by remote function call.

4.2 Round Trip Latency

The analysis in Chapter 3 shows that the synchronization between the running sequences on the processor cores is a very important, and takes quite an extensive part of the parallel execution of the control loop algorithm. Due to the large amount of data dependencies within the algorithm, and timing requirements of the control-loop, the synchronization latency is a large potential bottleneck on performance. The fact that hardware memory latency does not scale with the same pace as processor speed and functional unit latency when moving to a processor with increased processor frequency, makes this performance aspect hard to be estimated without proper testing.
Furthermore, when looking at the processor core architecture [2], it can be seen that the critical path of communication between processor cores has changed significantly with respect to the currently used HPPCs. On the P4080, there exists only a single layer of L2 cache between the separate cores, while the T4240 has multiple levels of processor cache ranging from L1 to L3, depending on which cores are scheduled to communicate. These architectural differences between the current HPPC and the one based on the T4240 processor, change the deterministic behaviour of the generated sequence schedule for the application, and thus further research is needed to estimate what impact it has on the potential processor performance.

The communication between the sequenced WorkerBlocks running on different processor cores, is performed by read- and write operations on shared memory and synchronization using Integer flag *wait* and *updates*.
Performance testing for this type of communication is performed by emulating the communication and synchronization methodology used in the CARM application, and mea-

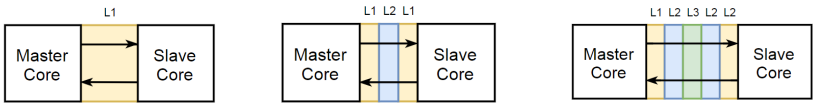
suring the *round trip latency* through the various levels of processor cache between two or more running threads. Round trip latency is defined in this benchmark scenario as the time (in nanoseconds) it takes for a certain processor core to *set* a value in shared memory, and read the value to be *reset* by an other core.
By measuring the latency for the different levels of cache during different types of producer-consumer model execution, a clear set of estimations can be made with respect to the performance impact of workload spreading and multi-workload usage of a HPPC.

4.2.1 Implementation

Due to the utilization of the extended PowerPC instruction-set and the T4240 hardware architecture, the different levels of cache connecting the processor cores induce varying latencies for different inter-core communication scenarios. To test the synchronization latency differences between the various levels of cache, three different testing scenarios are contemplated, based on which results can be extrapolated and conclusions made:

- Intra-core (thread to thread) communication via L1, within a single E6500 processor core.
- Inter-core communication via L2, between two E6500 processor cores within the same core-cluster.
- Inter-core communication via L2 and L3 (CoreNet Fabric), between two E6500 processor cores placed in different core-clusters.

A schematic representation of the three different inter-core communication scenarios is seen in Figures 4.1a, 4.1b, and 4.1c.



(a) L1 intra-core. (b) L2 inter-core. (c) L3 inter-core.

Figure 4.1: Schematic representation of the round trip latency experiment via the different cache levels.

As synchronization activity rate between parallel running sequences varies for the different applications running on the HPPC. Multiple scenarios regarding the amount of inter-core synchronizations need to be evaluated. This enables detailed insight in the impact on synchronization latency for the different levels of cache and varying amounts of synchronized parallelism. Because the T4240 processor' PowerPC E6500 cores are dual-threaded, some critical functional pipeline resources are shared. To capture the effect of this sharing on core performance, the difference between single threaded and dual threaded execution is examined as well in this test, as one of the shared resources is the memory bus connected to the Load-Store units. Contention on this bus is expected to increase the memory latency for dual threading with respect to single threading

execution.

Furthermore, the PowerPC instruction-set has a range of varying synchronization operations[17], each with different behaviour with respect to operation execution. These include the *MBAR0*, *MBAR1*, *SYNC*, and *MISO* operations. The two most viable synchronization operations *SYNC* and *MBAR1* are tested for each of the execution models, as these operations best represent the desired synchronization behaviour to be used in the CARM application with respect to pipeline behaviour and operation order determinism.

The *SYNC* operation provides a heavyweight memory barrier that ensure the order of memory accesses. The operation stalls all other operations and makes sure that the data memory access becomes visible to the entire memory hierarchy by publishing it to the CoreNet L3 cache before any other operations are executed.

The *MBAR1* provides a more lightweight memory barrier that flows through the pipeline and queues, preventing reordering with respect to the operations on memory hierarchy.

Two basic algorithms are developed for testing the round trip latency between multiple cores, as depicted in Figure 4.2. The first test is a single 1-to-1 synchronization, in which a single master core synchronizes with a single slave core. The second test is a 1-to-N synchronization, in which a single master core synchronizes with a varying amount of slave cores.

The basic algorithm of the round Trip Latency test is executed in 4 steps as follows.

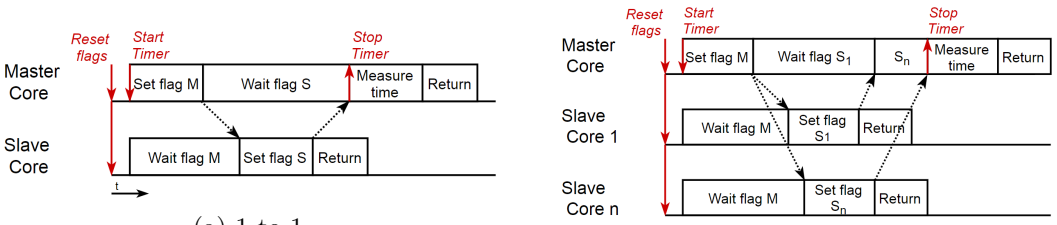
1. A master-thread allocates addresses in memory, for master- and slave-flag integers, and makes them known to the involved cores.
2. The master-thread assigns a value to an integer flag in memory.
3. A slave-thread reads the master assigned flag from memory, and assigns that value to another integer flag in memory.
4. The master-thread reads the slave assigned flag and calculates the time passed.

To reduce jitter in the measurements, the tests are run an arbitrary large number of iterations, after which the results are averaged. Furthermore, after rebooting or resetting the device, the results from the first iteration of the benchmark are evicted to reduce the cold-start memory overhead influences on the results.

The Code examples B.1 and B.2 in Appendix B, present the pseudo-code for the implementations of the main functions of both master and slave threads for the Round-Trip-Latency benchmark application.

4.3 Parallel Sequences

The software analysis in Chapter 3 indicates that the software running on the HPPC is highly parallel and heavily synchronized at specific moments during the execution



(a) 1 to 1

(b) 1 to n

Figure 4.2: Schematic representation of the round trip latency performance tests.

of the pre-calculated scheduled application. To accurately test the parallel functional performance of the processor cores, the developed performance benchmark, emulate the behaviour of the parallel running sequences found in the motion control applications. This benchmark uses predefined WorkerBlocks with a function workload reminiscent of the execution complexity found in the sequences of the motion control application.

The CARM sequence generator that is part of the CARM motion control application, is emulated in a simplified manner and produces a set of predefined function blocks and sequences them in the same way the sequence generator of the CARM servo loop does. The sequences are synchronized on specific moments through different levels of shared memory, emulating the inter-core synchronization methods utilized in the application running on the HPPCs.

To asses the performance of the processor and compare it with one of the currently used P4080 HPPCs, a workload containing a set of fixed size sequences is executed on both the T4240 and P4080 modules, and timed in ns. Furthermore, the available processor performance counters, accessible via the PowerPC Instructionset, are used to measure other effects like functional unit contention and cache misses.

4.3.1 Implementation

The workload for the test is based on a set of parallel *sample* sequences running on the different cores of the the processor. These sequences are all identical to ensure the maximum level of resource contention possible, ultimately emulating the worst case behaviour of the processor.

The utilized base sample sequence is created by repeating a small *array* of 5 generated functional WorkerBlocks. These short WorkerBlock arrays are padded with different *update*- and *wait*-blocks, according to the desired inter-core synchronization. The WorkerBlocks in the short array are linked to each-other using the same linked-list technique found in the CARM sequencer. This means that, at design-time, a static schedule is created describing the WorkerBlock in- and outputs and the connection between them using pointers to their memory address.

The total base sample sequence is then created by repeating the short WorkerBlock array a fixed number of times. It has been chosen to repeat them a total number of 16 times, creating a large sample sequence of 80 compute blocks regardless of the synchronization blocks in between. The length of this sequence has been choses based

on the need for an arbitrary large sequence, which at least represented the average length of a CARM application sequence, and made synchronization impact clearly visible when comparing different scenarios.

The generated set of WorkerBlocks utilized for the workload, mirror the computational complexity of the different WorkerBlocks found in the SS-controller application, and are furthermore based on the WorkerBlocks used in previous work and approved by the CARM system architects. These WorkerBlocks contain a fixed number of inputs and outputs, dummy input parameter values, *Status* value and a single *FullCalc* function. A schematic representation can be seen in Figure 4.3.

Based on these criteria, the following WorkerBlocks have been selected: *IN, MUL, DIV, MATRIX, MULMAX, PID, UPDATE, WAIT, OUT*, with the functional and memory operations described in Table 4.1.
The IN and OUT blocks function as source and sink of the sample sequence, providing dummy input data at the start of the sequence, and collecting output data at the end of the sequence. As the data used in the calculations are not actual functional data, the results are irrelevant and thus not kept. The WAIT and UPDATE blocks are used as synchronization points in the sequence. The WAIT block stalls the WorkerBlock function and waits for a specific UPDATE WorkerBlock, on the same or a different core, to increments its state to a certain value before continuing execution. These WAIT and UPDATE functions are controlled using a set of *GUARDS* created at design time and incorporated into the static schedule, keeping track of core-id and status value for synchronization actions between sequences.

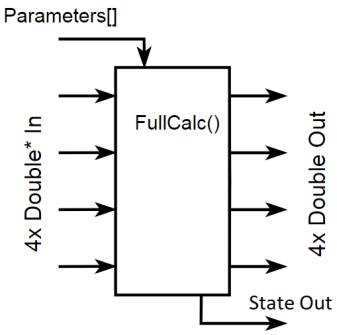


Figure 4.3: Abstract diagram of a block template used for the WorkerBlocks

Due to the unique multi-threaded and clustered nature of the processor-cores, and the utilized memory architecture [2] in the T4240 processor, it is essential to vary a set of different application characteristics during workload execution. This way their impact on eventual performance behaviour of the processor can be compared with performance delivered by the P4080 processor under the same conditions. These key characteristics are as follows:

- Workload spreading over cores and clusters.
- Multi-threading.
- Synchronization intensity.

Workload Spreading

The possibility to aggregate the processor cores of the T4240 in separate clusters, is one of the most attractive hardware architectural features of the device, and theoretically allows for three almost complete separate clusters of cores with a low amount of impact on each others behaviour and performance.

Table 4.1: WorkerBlock details on memory and functional units utilization, and data-types (Integer or Float).

Block Type	Function	Computations	Memory Usage
IN	Data source	1 Add I	3 Load 5 Store
MUL	Multiplication	4 Multiplication F 1 Addition I	13 Load 5 Store
DIV	Division	4 Division F 1 Addition I	13 Load 5 Store
MATRIX	Matrix multiplication	4 Multiply F 12 Multiply-Addition F 1 Addition I	25 Load 5 Store
MUL-MAX	Multiplication and Max operation	4 Multiply F 4 Compare F 4 OR 4 Branch Equal 1 Addition I	13 Load 5 Store
PID	PID controller function	12 Multiply-Addition F 4 Subtract F 4 Addition I	28 Load 17 Store
UPDATE	Sync	1 Addition I	1 Load 1 Store 1 Sync
WAIT	Wait for update	1 Compare I 1 Branch Greater/Equal 1 Addition I	3 Load in loop 1 Load 1 Store 1 Sync
OUT	Data sink	1 Addition I	9 Load 5 Store

Varying the workload spread over different cores, and the total number of workloads running side by side on different clusters, is used to determine the impact of the core clustering on processor performance. Furthermore, executing up to 3 benchmark workloads simultaneously on different core-clusters, can provide evidence for the viability of running multiple motion control applications on a single T4240 processor.

Threading

Looking at the pipeline architecture of the E6500 processor core, one can observe that certain functional units and memory buses are shared between the two hardware threads of the E6500 cores of the T4240. Just as has been done with the synchronization tests, measuring the performance difference between single and dual threaded execution of the parallel sequences, will determine the impact of dual threading on T4240 processor performance.

Synchronization Intensity

As the communication intensity between different running threads on a HPPC is not uniform, different levels of synchronization scenarios are considered in the experiments in order to measure the impact of synchronization overhead and resource contention on the overall performance. This is done by having three different synchronization intensities induced in the workload of the test: *stand alone*, *start-finish*, and *full* synchronization.

The **stand alone** scenario executes all running sequences in parallel without inter-core synchronization. This means that the processor cores are free to execute their workload without having the constraints of waiting or updating other cores, as can be seen in Figure 4.4.

The **start-finish** test with 2 synchronization points, has a wait and update block at start and finish of the sequence as can be seen in Figure 4.5. This means that all sequences wait on each-other after the initial executed block. The same concept holds for the last block, where again all blocks wait for each other before the last block is executed and the sequence ends. This induces the situation where all sequences start and stop executing at the same point in time, but the processor pipelines have some form of freedom of scheduling the use of their functional units.

The **full synchronized** test with 5 synchronization points, see Figure 4.6, includes a wait and update block after each of the blocks. This implies that after each executed function block, a synchronization is performed to guarantee functionality is executed before continuing. Furthermore this scenario induces the maximum amount of overhead and contention on shared resources, resulting in the worst case scenario for the execution.

By testing with these three experimental scenarios, an adequate overview can be created and estimations can be extrapolated with respect to other cases of synchronization and communication in a workload.

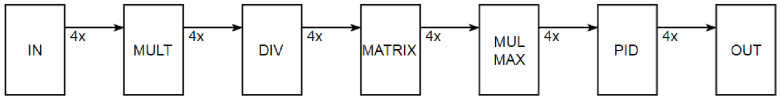


Figure 4.4: Block diagram for the stand alone sequence test.

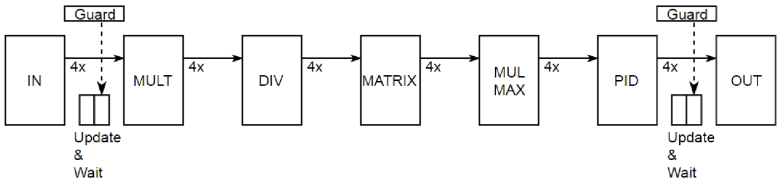


Figure 4.5: Block diagram for the start-finish synchronized sequence test.

4.3.2 Experiments

To execute the performance tests, a single master-core starts a number of threads on specific other cores according to the test scenario, and waits for all threads to finish

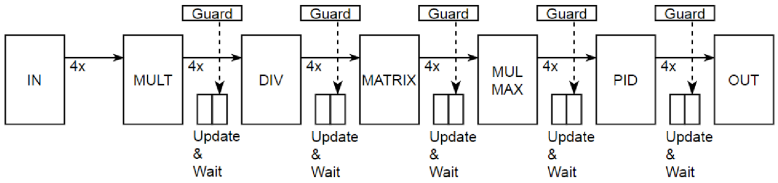


Figure 4.6: Block diagram for the fully synchronized sequence test.

initializing their sequence. When all threads have signaled finishing their initialization to the master thread, all threads are given the signal to start running their workload. After all threads have finished executing their workloads and signaled this to the master, the timer and performance counters are read to obtain the performance data. A schematic representation of this execution pattern can be seen in Figure 4.7. To reduce memory associated jitter and cold-start, the tests is run an arbitrary total number of times ranging from 1000 to 1000.000 times, after which results are averaged.
A pseudo-code example of the implementation of master and slave functions are presented in Appendix B, Code example B.3 and B.4.

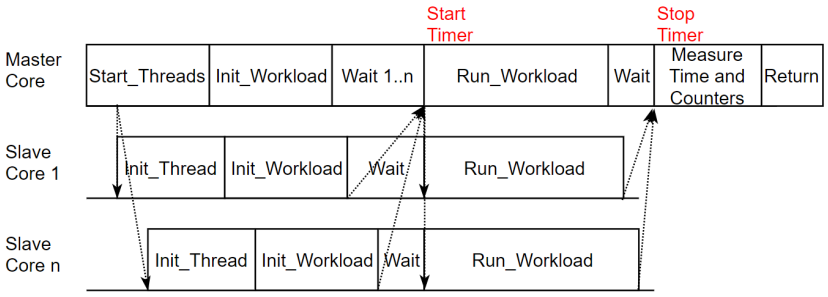


Figure 4.7: Schematic representation of the execution of the benchmark running parallel sequences.

4.4 Experimental Evaluation Platform

In the previous work, the tested NXP Freescale T4240 SoC module was a stand alone development board, augmented with a generic off-the-shelf Linux operating system. In this project, to accurately determine the performance of the processor module within the context of the CARM platform, a VadaTech T4240 AMC module [18] has been integrated into a working CARM testbench hardware environment.

The current NXP Freescale P4080 HPPCs are running a custom (BSP) Operating System, optimized for managing te multi-core hard-real time CARM motion control applications. To be able to execute the CARM software on the T4240 within the CARM hardware platform, the BSP software package has been adapted to run on the T4240 processor. As the T4240 is a natural successor of the P4080, both running on an extended PowerPC RISC instruction set, the P4080 BSP has been used as the basis for the new BSP. The BSP adaptation required the changing of critical functions and instructions to

account for the difference in hardware configuration and instruction-set variation.

However, due to the nature and time frame of this project, the full range of specific optimizations and accelerators featured by the T4240 processor have not been utilized, as only the default core components and memory functionality were necessary.

For testing and comparing performance of the T4240 and P4080, a hardware environment is assembled and installed in a CARM testbench ATCA rack, emulating an ASML *stages rack* used in the Twinscan lithosteppers. This testbench provides the needed interfacing and software components for configuration and running the benchmark application on the HPPCs. The test setup is build up out of the following components, where external communication with the T4240 processor module is done by a remote serial connection over Ethernet to the SCH.

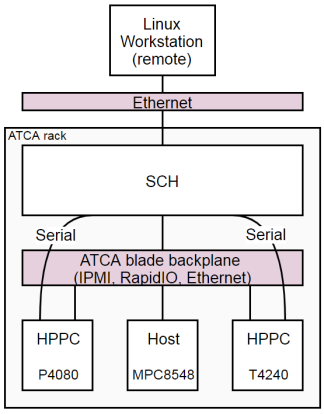


Figure 4.8: Abstract block diagram of the hardware setup for T4240 performance testing and validation.

- SCH with Intel Xeon x86 processor running Linux.
- ATCA rack
- ATCA Blades
- AMC modules
  - HOST Freescale MPC8548 (Prodrive).
  - HPPC Freescale P4080 (Prodrive).
  - HPPC Freescale T4240 (Vadatech AMC702).
  - PGEA I/O module.

4.4.1 Baseline Validation

Due to the fact that the T4240 processor used in previous work was build into a stand-alone Reference Design Board (RDB), this device is unsuited to be directly integrated into the dedicated testbench. To be able to integrate a T4240 processor into the testbench environment and the actual CARM hardware platform, it demands to be an AMC module compatible with the ATCA rack infrastructure. For this project, the used *Vadatech 702 AMC* module with T4240 processor has this compatibility. Because the NXP Freescale T4240 RDB and the AMC-module are manufactured by different vendors, hardware- and configuration differences do exist between them. To be able to use the results gathered from previous work as baseline, the tests performed in previous work have to be validated by recreating them on the AMC module.

The Linux based performance benchmarks used in previous work, are executed in a simplified manner on the Vadatech T4240 module integrated into the CARM testbench. It is observed that the results from both the component test and the function tests, are comparable to the results gotten in previous work, and don't differ much besides the expected jitter due to iteration and cold-start influences.

4.5 Results

The performance results gathered from the custom performance benchmarks can be measured in both processor cycles and nanosecond duration. In this project it is chosen to measure the experiment results in nanometers, as this creates the easier comparability with tests from previous work and known results from CARM design documents, and furthermore is processor clock-frequency independent, enabling quick comparison between the two processor platforms. These execution times in ns are measured by recording the start and finish time of the benchmark iterations, by calling the *HPtime()* function provided by the custom BSP.

FPU contention and cache misses are furthermore recorded using the available performance counters, by making use of the extended T4240 PowerPC instruction set However, it has been seen that the the performance counters don't shown much relevant results besides the quantification of expected FPU contention when using dual-threading.

4.5.1 Round Trip Latency

The round trip latency tests are executed for both the *SYNC* and the *MBAR1* synchronization operation from the T4240 PowerPC instruction set, for both the dual and single threaded operation mode. Furthermore, to evaluate the impact of forced synchronization on performance, a execution scenario is tested with no synchronization instruction used to trigger synchronization at the fixed communication moments in the test scenario. This however creates the non-realistic scenario where the synchronization is left to be performed by the processor pipeline, impacting the deterministic behaviour of the processor from an application point of view, and is thus only used as context knowledge.

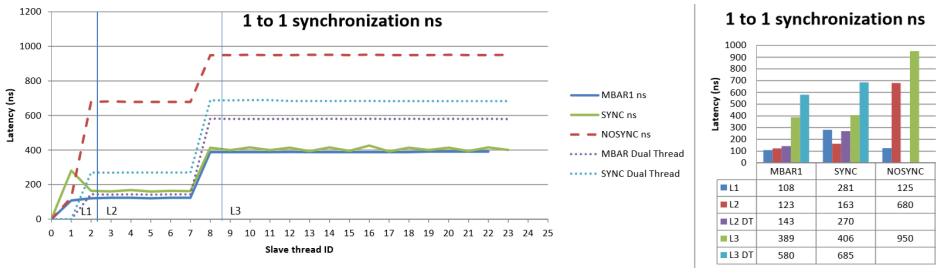


Figure 4.9: Round Trip Latency test results for the 1 to 1 tests in ns.

As can be seen in the *1 to 1* test results depicted in Figure 4.9, the different levels of cache are clearly distinguishable by the amount of latency. Looking at the single threaded execution of the test, it can be seen that the MBAR1 operation is respectively 2.6x, 1.32x, and 1.05x faster than the SYNC operation for L1, L2, and L3 cache.

When dual-threading is activated, it can be observed that the difference between MBAR and SYNC is 1.88x and 1.18x for L2 and L3 respectively.

It can furthermore be observed that the communication via L3 is up to 3.6x slower than communication via L1, and 3.14x slower than via L2, which indicates

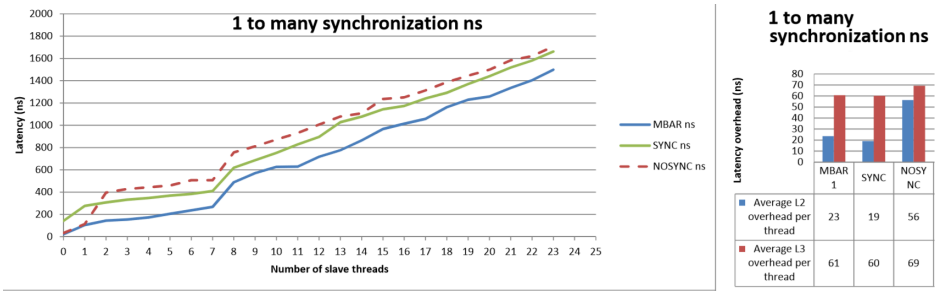


Figure 4.10: Round Trip Latency test result for the 1 to N tests in ns.

that synchronizing across different clusters via L3 cache has a substantial detrimental performance impact.

Looking at the results from the *1 to N* synchronization tests in Figure 4.2b, it can be concluded that there exists a linear relation between the experienced latency with respect to the amount of slave cores that the master core is waiting for to synchronize. The results suggest that for both MBAR1 and SYNC, for every extra core within the same cluster (synchronized via L2) an average of 21 ns overhead is added. For every core outside of the cluster (synchronized L3), an extra 60 ns is added. Both overheads add on top of the respective initial L1 latency. Running the same benchmark application on the P4080 processor, the results can be compared to the previously gathered results of the T4240, as seen in in Table 4.2.

Table 4.2: Comparison of the 1 to 1 Round trip latency results in ns for single and dual thread testing for both MBAR1 and SYNC operation.

		T4240			**P4080**
Operation	**Threading**	**L1**	**L2**	**L3**	**L2**
SYNC	*Single Thread*	281	163	406	227
	Dual Thread	NA	270	685	
MBAR1	*Single Thread*	108	123	389	227
	Dual Thread	NA	143	580	

Using the MBAR1 synchronization operation, communication between cores on the T4240 via the L1 or L2 cache is much faster than on the P4080 (52% to 37% respectively) for both single as dual threaded execution. However, it is seen that when communicating between cores within another cluster, via the L3-CoreNet, the latency increases quite rapidly with up to 71% and 156% for single- and dual-threaded respectively, compared to the L2 communication on the P4080 processor. This means that the synchronization performance outside of the core-clusters induces a heavy penalty on performance during the synchronization parts of the sample.

When looking at the results from experimenting with the SYNC operation, the results are in line with what has been seen previously. Communication within L1 has a

latency of 281 ns, which compared to the 227 ns of the P4080 is a performance decrease of 23%. Single core L2 communication is slightly better with 162 ns latency, but when dual threaded, the latency increases to 270 ns which is a performance decrease of 19% with respect to the P4080.
Communication with cores outside of the same cluster via L3, further increases latency compared to the P4080 L2 communication, to a 406 and 685 ns single and dual threaded respectively.

4.5.2 Parallel Sequences

As revealed by the *round trip latency* test, the MBAR1 synchronization operation provides a significant better performance than the SYNC operation with respect to communication latency through the different levels of processor cache. Furthermore the heavyweight nature of the SYNC operation is unnecessary in most of the CARM motion controller applications. Therefor the performance test results discussed in the next sections assume only the MBAR1 synchronization operation. The measurements corresponding the Parallel Sequence benchmark execution for the 3 scenarios of synchronization, are presented in Table 4.3.

Table 4.3: Test results for the Parallel Sequence benchmarks, ran on both the T4240 and P4080. Results correspond to the average execution time of a thread running a single sequence workload. Both raw results in ns and normalized results are displayed, where the P4080 results have been set at 100.

			actual ns	***normalized***
Standalone				
	P4080	8 cores	13155	100
	T4240	12 cores single thread	6439	49
		8 threads 1 cluster	8831	67
		16 threads 2 clusters	8861	67
		24 threads 3 clusters	8870	67
2 Flag Synchronized				
	P4080	8 cores	14769	100
	T4240	12 cores single thread	7808	53
		8 threads 1 cluster	11029	75
		16 threads 2 cluster	10968	74
		24 threads 3 cluster	11706	79
5 Flag Synchronized				
	P4080	8 cores	20796	100
	T4240	12 cores single thread	12896	62
		8 threads 1 cluster	13307	64
		16 threads 2 cluster	15553	74
		24 threads 3 cluster	21798	105

Looking at the results for the **stand alone** testing scenario, it is clear that the T4240 shows no performance difference between running a single workload on a cluster, or running multiple workloads in the same time at different clusters. It proves that the core functional pipelines have no effect on performance outside of their core-cluster. Furthermore, the experiments indicate that when dual-threading, the parallel sequences running both on a processor-core experience an approximate 37% performance degradation with respect to the single threading execution time.

The results for the experiments with both synchronization scenarios on the T4240, show that the added synchronization induces a clear strain on processor core performance. When comparing the execution times of the 2-flag synchronized test scenario, the execution duration of the application increases with 30%. Increasing the amount of synchronization even more using the 5-flag test scenario, it is clear that the amount of synchronization heavily adds to the execution duration, increasing the execution time with up to 146% when using all 24 cores simultaneously.

It can be observed that the extreme results for the 5-flag synchronized test scenario using all 24 cores simultaneously, lie outside of the line of expectancy looking at the other results. However, as this phenomenon has been present in all the experiments using this scenario it cannot be igored and it is assumed to be a side-effect of the processor architecture design. The performance counters recording the amount of FPU contention show furthermore no differences with the other scenarios, which makes us to assume that this phenomena is related to the large amount of contention on memory buses.

Table 4.4: Performance comparison between P4080 and T4240 SoC for the sequence tests, using different synchronization intervals. Values represent the performance increase of the T4240 over the P4080 when running the same workload on the processor.

			1 Workload (Non Clustered)	***3 Workloads (Clustered)***
Standalone				
	Single Thread	*12 cores*	3x	1.02x
	Dual Thread	*24 threads*	4.5x	1.5x
2 Flag Synchronized				
	Single Thread	*12 cores*	2.83	0.94x
	Dual Thread	*8 threads*	1.33x	1.33x
		16 threads	2.7x	1.35x
		24 threads	3.8x	1.27x
5 Flag Synchronized				
	Single Thread	*12 cores*	2.42x	0.81x
	Dual Thread	*8 threads*	1.56x	1.56x
		16 threads	2.7x	1.35x
		24 threads	2.86x	0.95x

Table 4.4 presents the relative T4240 performance results with respect to the P4080 ones. Comparing these results with the estimations made in previous work, it can be concluded that a T4240 is indeed capable of performing at least 3x as fast as the P4080, however this estimation is observed to only be correct for a small subset of the executed scenarios. Our results confirm that the estimations from previous work are accurate for the non-synchronized scenario, running a single workload spread over the processor cores of the T4280. The results indicate a performance increase of 3x and 4.5x that of the P4080 for the single- and dual-threaded execution respectively. Furthermore, when executing a workload on each of the 3 core-clusters of the T4280, each cluster can be seen to perform up to 1.5x faster than a single P4080 HPPC. However, when taking into account that the CARM software runs according a heavily parallel and synchronized execution scenario on multiple processor cores, the results we obtained from the custom performance estimation show a different story.

The results for a single workload execution in Table 4.4, show that the performance gain of the T4240 over the P4080 degrades significantly when increasing the amount of synchronization between the cores. This is most clearly seen in the T4240 single-threaded scenario results, where increasing synchronization reduces the performance gain with up to 20%. Observing the synchronized execution scenarios while dual-threading is active, it can be seen that the T4240 performance scales quite consistently with the amount of active clusters sharing the workload, running up to 3.8x the performance of a P4080. However the performance drop when executing on all 24 cores in parallel shows a decrease in estimated performance up to 35%, with respect to the non-synchronized scenario.
Furthermore, our measurements on the clustered workload execution suggest that regardless of the utilized synchronization scenarios, with exception of the discussed performance outlier, the T4240 is capable of running up to 3x the workload of a P4080 in the same time at an approximate 1.3x the processor performance of a P4080.

4.6 Conclusion

Based on the results gathered via the proposed performance benchmarking approach, we can conclude without doubts that the considered synchronization methods and memory barriers have significant impact on the T4240 core-to-core communication and memory behaviour. In particular, when the MBAR1 lightweight synchronization is utilized the thread to thread communication via shared L1 has a 50% smaller latency than the P4080 corresponding counterpart. For the L2 based communication between T4240 cores in the same cluster the latency adds ups to 65% of the P4080 corresponding counterpart. When the more heavyweight SYNC operation is utilized however, the performance improvement is diminishing, and even performance loss is observed in comparison with the P4080 execution.

Furthermore, it is seen that synchronization outside of a single core-cluster, induces a large performance penalty. This brings us to the conclusion that executing a heavily synchronized workload spread over multiple cores in different clusters should be discouraged in the real CARM motion control applications, as it would reduce the T4240 performance increase over the P4080 quite significantly.

Looking at the performance of the parallel sequences, it can be seen that the single threaded performance of the T4240's E6500 processor cores lie at approximately 2x that of the P4080's E500MC processor. However when dual-threading, the performance increase of the E6500 cores' hardware threads diminish to approximately 1.5x that of the P4080 due to critical component contention in the pipeline. These results are in disagreement with the ones produced by previous investigations which has indicated that the T4240 cores are always 3x faster than the P4080.

However, previous estimations do hold true when the performance of the processor modules as a whole is compared, ie a T4240 running the same workload as a P4080. In that case, a T4240 is capable of executing the application between 2.4x and 4.5x faster than a P4080, depending on synchronization intensity. This allows us to draw the conclusion that the T4240 is definitely capable to deliver the performance increase required to sustain the 20 kHz to 40 kHz upsampling necessary for the ASML lithosteppers upgrade.

Furthermore, the experiments demonstrate that it is possible to execute up to 3 workloads simultaneously on the separate core-clusters of the T4240, while running at the current application frequency. However, the amount of synchronization does induce performance variations, indicating that the hardware architecture might not be as powerfull and versatile as it has been suggested in previous work and processor documentation.

5 Conclusions and Future Work

Based on the results from the analyses and experiments performed in previous Chapters, conclusions are drawn for the discussed requirements on the CARM hardware platform. The future work ahead is discussed and the implication the results might have on the current status of affairs within ASML and the semiconductor industry.

5.1 Summary

In this thesis we theorized that the off-the-shelf benchmarks, used for initial performance estimating the processor candidates for the CARM platform, have not been sufficient in representing the motion control application running on the hardware platform, thus providing inaccurate estimation data.
In this work, we performed an analysis into the scheduling and execution of the application performing the controller-function of the Short-Stroke mechanical platform. This analysis indicated that the application executes as a highly synchronized set of parallel sequences of functions, deployed on different cores of the HPPC processor module. We established that the performance bottlenecks, characteristic to this execution method, mainly exist in the synchronization between the cores via shared processor cache and the contention of shared resources due to the parallel functional unit scheduling in the processor. By thorough analysis of the benchmarks executed in previous work, we concluded that they do not expose the previously mentioned CARM specific bottlenecks, as they mainly focused on the raw single-core performance of the processor modules under consideration. Therefor we can conclude that the off-the-shelf performance benchmarks available, are not effective enough in estimating the performance for the custom use of the multi-core processor in the CARM platform, and custom and specific performance benchmarks are needed that focus on the known hardware bottlenecks of the motion control applications executed on an ASML lithostepper.

Using the analysis of the CARM Short-Stroke application, we developed a set of custom performance benchmarks emulating the synchronization and parallelism behaviour found in the application executed on the HPPC processors. This enables us to make performance estimations of the new processor in a more realistic manner, while not needing to implement a full CARM software and hardware setup.
In order to execute these benchmark applications, we installed and configured a custom testbench hardware environment emulating the CARM platform found in the ASML lithosteppers, in which the Freescale T4240 processor module is integrated and controlled by means of the adapted Board Support Package operating system we developed specifically for this project. By executing the custom performance benchmarks on the testbench, and analyzing the gathered results, we can conclude that the industry driven

performance requirement of upgrading from a 20 kHz to 40 kHz sample execution budget is definitely possible when using the NXP Freescale T4240 processor module. The experiments indicated that the T4240 performs between 2.86x and 4.5x as fast as the currently used NXP Freescale P4080 module, in the scenario where the singular 24-core T4240 module is used to replace a singular 8-core P4080 module.

Evaluating the results with respect to the desired cost-of-goods hardware upgrade, where three P4080 modules are replaced with a single T4240 module, the performance gain of the T4240's E6500 cores is seen to diminish significantly. Although the measurements have indicated that the T4240 is indeed capable of executing 3x a workload of a P4080 in the same time-period using the separate core-cluster architecture, contrary to the theorized situation in previous work [5], this is only viable when the application workload is ran at the current 20 kHz execution frequency. The results show that each of the 3 core-clusters of the T4240 is capable of reaching up to 1.5x the performance of the P4080. It is furthermore shown that fluctuations in the performance may occur when varying between the different synchronization scenarios. It is seen that a performance decrease of up to 19% with respect to the P4080 can occur in the worst case, signaling the need to thoroughly research the available architecture optimizations available for the device to predict and prevent this non-deterministic behaviour.

5.2 Future Work

Based on the work performed and experiments executed, the following recommendations are made for future work on the system;
Scheduling. For the currently used hardware- software platform, the motion control applications have been scheduled to run in 7 sequences for the P4080's processor architecture. The implications of distributing the same application over 12 to 24 cores on the T4240 will have to be researched.. Due to the fact that the latency between the different cores changes based on core location in the processor, the impact on the determinism of the created schedule should be investigated.
Rebuilding the BSP. For this project, the Board Support Package of the currently used P4080 processor has been adapted to run on the T4240 processor. However, as the T4240 has a quit different architecture as the P4080 with respect to use of accelerators and core-distribution, it is advised to revisit the BSP and make an effort to rebuild it with the new hardware architecture in mind, making optimally use of the available resources.
Full TestBench Integration. For this project, the Vadatech T4240 processor module has been built-in into a CARM testbench. However, although physically integrated into the system, it is not fully functional and usable as actual HPPC due to configuration settings not implemented. To test and use the processor fully, it should be configured using the correct system settings for full integration into the CARM software and hardware.
Investigation into long term solutions. During this project, it has become known that the current line of PowerPC processors from NXP Freescale is coming to an end,

due to the fact that the demand for this processor type is diminishing fast. Therefor it is advised to investigate a different processor architecture besides the PowerPC line, that would be more future proof. This way, it is possible to move more easily to a new processor platform in case a new round of fundamental hardware upgrades is needed.

5.3 Research Implications

Based on the results and conclusions gathered from the executed tests as discussed in this report, the implications can be deduced for different involved parties.

5.3.1 CARM Team and ASML

The performance benchmark results have indicated that replacing up to three currently used P4080 HPPCs in the CARM motion control platforms, by a single T4080 processor module is possible for the current application execution frequency of 20 kHz. When this replacement step is implemented, it would reduce the total amount of HPPCs in the system with up to 2/3th, and free up space in the ATCA racks making it possible to integrate other modules into the system, or combine racks into one.

Furthermore, this replacement action would reduce the amount of I/O latency between the different workers running the control loops, due to the fact that a large part of the communication between the different HPPCs would then be performed over the CoreNet fabric on the processor module itself, instead of over external I/O (SRIO or Ethernet). This would improve the real-time properties of the system, and therefor improve scheduling properties of the applications. Reducing the amount of hardware modules in the platform will furthermore reduce the total cost of the CARM motion control hardware platform, thus making it possible to invest these funds in other parts of the CARM platform.

By replacing the current P4080 HPPCs with T4240 processor modules, the control loop applications can be executed at a higher execution frequency, and thus be scheduled with a decreased execution budget at the highly demanded 40 kHz. This would mean the possibility of increasing the accuracy of the control-loop by up-scaling the number of measurement samples, and increasing the control-accuracy for the mechanical systems being driven by the control-loops. Furthermore, using the faster processor module will increase the execution budged for background tasks like monitoring. This enables the possibility to improve development and issue prevention, increasing the efficiency of the CARM development team during the development stage of new system features, and decreases time when solving client specific issues.

5.3.2 Industry

Replacing up to three HPPCs with a single HPPC in the CARM system, reduces hardware platform real-estate per motion control platform. This creates the possibility to pack multiple CARM platforms into a single machine cabinet, reducing the footprint of the hardware platforms at the customer sites.

Furthermore, when upgrading the current applications to a 40 kHz execution frequency, the CARM motion control platform would also increase the sensor-actuator read frequency and thus enable the possibility to compute with higher order algorithms, resulting in more accuracy in the servoloop calculations. This accuracy improvement would as result enable the possibility to increase velocity of the mechanical movements at increasing physical precision for alignment and placement of wafer and reticle. In the semiconductor industry, these improvements would lead to a higher wafer-throughput, smaller feature size accuracy, and lead to higher die-yield per wafer, thus resulting in a stronger marker position within the semi-conductor industry.

Bibliography

[1] *QorlQ T4240, T4160 and T4080 Multicore Processors*, Freescale Semiconductors, 2016, rev. 7.

[2] *T4240 Product Brief*, Freescale Semiconductors, 10 2014, rev. 1.

[3] D. Hernandez, *A Design-Space exploration for high-performance motion control*, ASML, Eindhoven, The Netherlands, 2012.

[4] T. Engels, *SIA CARM support for 20KHz NXT2000ci*, ASML, Eindhoven, The Netherlands, 2017.

[5] M. Pieters, *Qualification Results Document of CPU Benchmark 2015/2016*, Prodrive Technologies, Eindhoven, The Netherlands, 2016, internal documentation ASML.

[6] R. R. H. Schiffelers, W. Alberts, and J. P. M. Voeten, "Model-based specification, analysis and synthesis of servo controllers for lithoscanners," in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, ser. MPM '12. New York, NY, USA: ACM, 2012, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2508443.2508453

[7] PICMG. (2017, 02) Advancedtca overview. [Online]. Available: https://www.picmg.org/openstandards/advancedtca/

[8] RapidIO.org. (2017, 02) About us. [Online]. Available: http://www.rapidio.org/about-us/

[9] *MPC8548E PowerQUICC III Integrated Processor Hardware Specifications*, Freescale Semiconductors, 06 2014, rev. 10.

[10] *QorlQ^{TM}P4080 Communications Processor Product Brief*, Freescale Semiconductors, 09 2008, rev. 1.

[11] S. Adyanthaya, M. Geilen, T. Basten, J. Voeten, and R. Schiffelers, "Communication aware multiprocessor binding for shared memory systems," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[12] Intel. (2014, 06) Intel xeon processor e5-2620 (15m cache, 2.00 ghz, 7.20 gt/s intel qpi) specifications. Rev. 10. [Online]. Available: https://ark.intel.com/products/64594/Intel-Xeon-Processor-E5-2620-15M-Cache-2_00-GHz-7_20-GTs-Intel-QPI

[13] *Multicore DSP+ARM KeyStone II System-on-Chip (SoC), 66AK2H14/12/06 Features and Description*, Texas Instruments, 11 2012, revised November 2013.

[14] (2018, 02) Spec cpu2017. SPEC - Standard Performance Evaluation Corporation. [Online]. Available: https://www.spec.org/benchmarks.html#cpu

[15] U. F. Mayer. (1998, 06) Lmbench. Bitmover. [Online]. Available: http://www.bitmover.com/lmbench/

[16] (1996) Nbench. BYTE magazine. [Online]. Available: https://www.tux.org/~mayer/linux/bmark.html

[17] *EREF: A Programmers Reference Manual for Freescale Power Architecture Processors*, Freescale, 06 2015, rev. 1 (EIS 2.1).

[18] *VadaTech AMC702 Hardware Reference manual*, VadaTech, 10 2016, version 1.0.0.

Appendix A

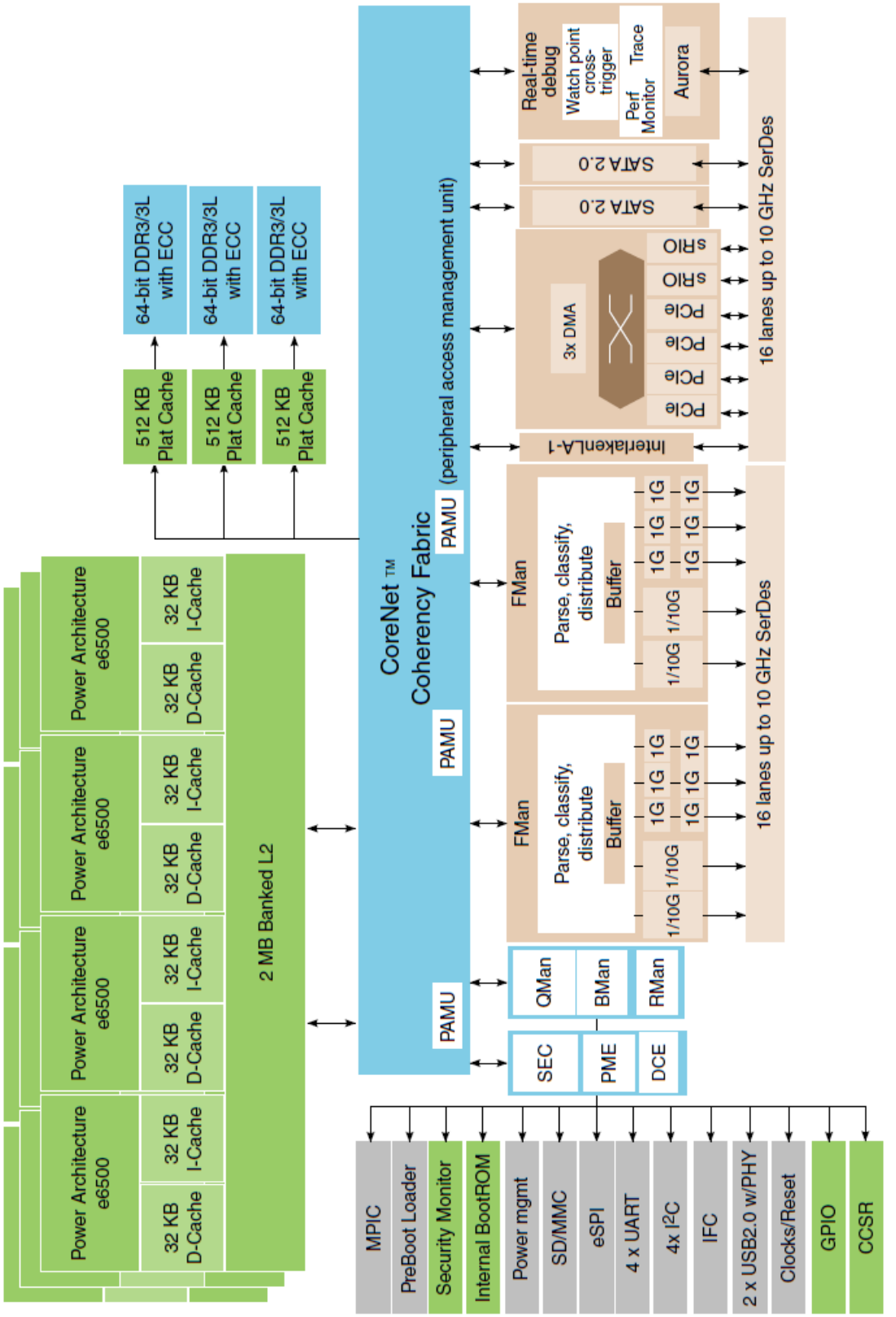


Figure A.1: Abstract block diagram of the T4240 core-complex architecture. As can be seen, the clusters of 4 E6500 cores are share the L2 cache, and are connected to each-other via the CoreNet Coherency Fabric and L3 cache. Figure taken from Freescale T4240 Reference Manual[2]

.

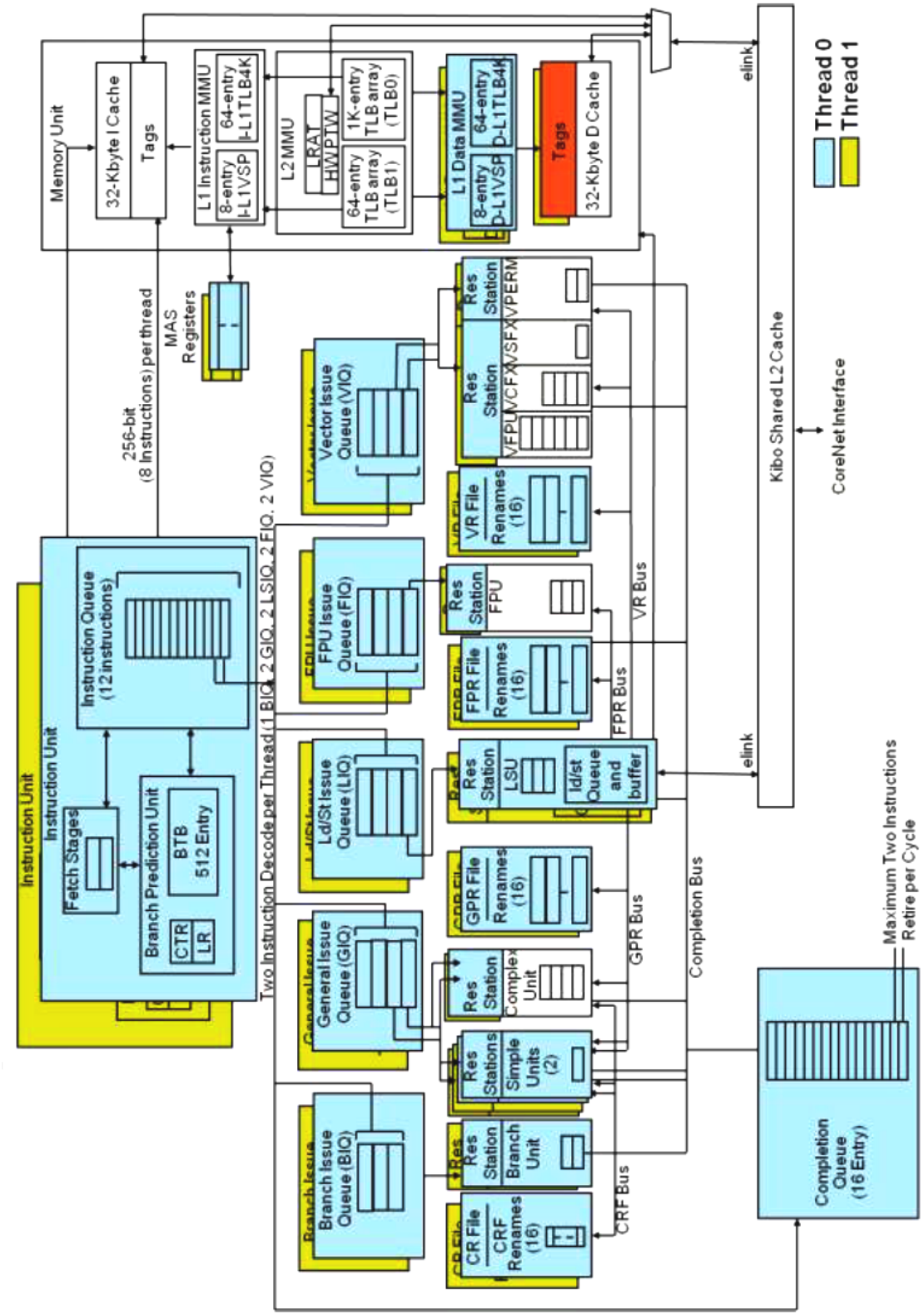


Figure A.2: Abstract block diagram of the T4240's E6500-core pipeline architecture. As can be seen, most of the pipeline has been duplicated to provide a dedicated functional unit or buffer per hardware thread. However, some of the critical (and more expenxive) resources are shared, like the Complex Unit and FPU. Figure taken from Freescale T4240 Reference Manual. [2]

.

Appendix B

Code example B.1: Pseudo code implementation of the master thread running round trip latency application.

```
void c2c_master(){
 init_flags();

 flags.m = malloc(sizeof(int));
 flags.s = malloc(sizeof(int));

 //iterate over cores
 for(NR_SLAVES){
  *(flags.m) = 0;
  *(flags.s) = 0;

  //start slave thread
  start_thread(
    c2c_slave, &flags);

  start_perf_ctrs();
  start_timer();

  //execute iterations
  for(NR_ITER){
   *(flags.m)++;
   SYNC;
   while(
     *(flags.m)!=*(flags.s));
  }
  stop_timer();
  stop_perf_ctrs();
}}
```

Code example B.2: Pseudo code implementation for slave thread running round trip latency benchmark application.

```
void hppc_c2c_slave(*flags){
 for(NR_ITER){
  while(*flags.m == *flags.s);
  *flags.s ++;
  SYNC;
}}
```

Code example B.3: Pseudo code of the implementation of the Parallel Sequence Benchmark for both the master- and slave-main functions.

```
void hppc_master(void *arg){
 testdata =malloc(
        sizeof(
         testdata_struct));
 SYNC;
 for(CORES_ACTIVE){
  thread_start(
        slave_main,
         &testdata);
 }
 while(!cores_init(&testdata));
 SYNC;
 synchronized_start(&testdata);
 SYNC;
 while(!cores_done(&testdata));
 SYNC;
 measure_duration(&testdata);
 SYNC;
 return;
}

void slave_main(void *arg){
 testdata_struct *testdata
        = arg;
 init_blocks();
 SYNC;
 set_init_done(&testdata)
 SYNC;
 while(!synchronized_start(
        &testdata);
 SYNC;
 hppc_samples(&testdata);
 set_core_done(&testdata);
 SYNC;
 return;
}
```

Code example B.4: Pseudo code for the functions called by the slave main thread when running the Parallel Sequence benchmark.

```
void hppc_samples(void *arg){
 testdata_struct *testdata
        = arg;
 while(< NR_ITER){
  start_timer();
  start_perf_ctrs();
  calc_blocks();
  stop_timer();
  stop_perf_ctrs();
  SYNC;
  reinit_blockstates();
 }
 SYNC;
 return;
}

void calc_blocks(){
 for(MAX_SAMPLE_SIZE){
  for(MAX_SEQUENCE_SIZE) {
   execute(BLOCK,FUNCTION);
}}}
```