Improving maintainability of time critical software by applying MATLAB Coder and architectural redesign: a case study in the PARIS domain

Citation for published version (APA):

Document status and date:
Published: 24/10/2018

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Improving Maintainability of Time Critical Software by Applying MATLAB Coder and Architectural Redesign

Laavanyaa Balasubramanian
Improving Maintainability of Time Critical Software by Applying MATLAB Coder and Architectural Redesign

A case study in the PARIS domain

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML Netherlands B.V.
Eindhoven University of Technology

Steering Group
Laavanya Balasubramanian
Wouter van Heijningen
Tom Verhoeff
Vanessa Rodriguez
Yanja Dajsuren

Date
October 2018

Document Status
Public

SAI report no.
2018/105

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.
Abstract
The final project is devoted to investigate and provide insights about the usage of MATLAB Coder for code generation of the algorithms for the image sensing production software within ASML. Thus, the deployment version of an algorithm can be automatically derived using MATLAB Coder to reduce the development lead time, the effort spent in testing, and synchronizing changes. An architectural redesign is proposed to minimize the external dependencies of the algorithm, i.e., the algorithm invocation and the algorithm parameter actions.

Keywords
PARIS domain, MATLAB Coder, Architecture redesign, ASML, TU/e, Software Technology, PDEng

Preferred reference
Foreword

Dear Laavanyaa,

A major challenge in development of lithography machines is the flawless translation of functional requirements into reliable and performing software. This project was initiated to prove that -at least for algorithms- this translation can be automated by applying code generation such that there is one single source of truth. Although the choice was made for an existing code generator from The MathWorks, significant challenges remained that we put on your plate. The most important is the application of this technology in a legacy code base that had incomplete documentation and did not have a prepared ‘landing spot’ for generated algorithm code.

You dove into this problem with a very structured approach, worked independently and responded well to feedback from all stakeholders. We were impressed when you came up with a detailed Gantt-chart planning but moreover you achieved your planned work packages well within the ten months timeframe.

The resulting proof of concept doesn't only show that we can apply code generation for time critical algorithms. It also shows that we can introduce this concept in legacy code without a big-bang redesign. You have proven that the architectural preconditions for generated algorithms can be introduced gradually. This enables ASML to innovate in the way we make extensions and changes on legacy code with nearly no additional project risks. Your results were recognized by several architects working on new solutions of Model Driven Engineering in ASML. Your work shows that their ideas work in practice: the example is there and well described in this report. You can be very proud of this result.

Thank you for your contribution!

Wouter van Heijningen, Project Mentor
September 20, 2018
Preface

This report is the summary of my PDEng (Professional Doctorate in Engineering) graduation project. The two-year Post-Master technological PDEng program is provided by the Eindhoven University of Technology under the banner of the 4TU.School for Technological Design, Stan Ackermans Institute – a joint initiative of the four universities of technology in the Netherlands. I executed the project “Improving Maintainability of Time Critical Software by Applying MATLAB Coder and Architectural Redesign”.

The project was executed within ten months at ASML Netherlands B.V. The project was devoted to evaluate code generation for algorithms in production software and to propose an architecture that is precondition for the application of code generation. This report contains the changes made in the MATLAB algorithm to enable code generation, the design solutions for architectural redesign, and the architectural solution evaluation. In addition to technical aspects, it also explains project management and retrospective.

The report also contains a generic flow to migrate MATLAB algorithm from manual C implementation to automatic C-code generation, and a generic flow to migrate the existing architecture to a more decoupled DCA-based architecture in smaller steps.

Readers who are interested in knowing more about MATLAB Coder, i.e., making m-code compatible, integrating the generated code, verifying the generated code, and regarding MATLAB Coder conclusions should read Chapters 6-8 (focused on 6.1, 7.1, and 8.1 sections) Readers who are interested in knowing more about the architectural redesign refer to Chapter 5. Readers with an interest in the implementation, verification, and conclusion of this architectural redesign should read Chapters 6 to 8 (focused on 6.2, 7.2, and 8.2 sections).

Laavanyaa Balasubramanian
September 2018
Acknowledgements

The success of the project required guidance and support from many people and I was fortunate to receive this support until the completion of the assignment. Therefore, I express my sincere appreciation to all of them.

I would like to express my deepest gratitude to my company supervisor, Wouter van Heijningen for his continuous support, constructive criticism and guidance during the entire project. His experience helped to get into the ASML domain very fast. Wouter invested his full effort in providing guidance throughout the project.

I would also like to thank my TU/e supervisor Tom Verhoeff who helped to coordinate my project. Tom’s feedback and questions helped to improve the quality of the project and documentation.

I would like to thank my company project manager Vanessa Rodriguez who shared her experience and provided guidance on project organization and presentation skills.

I would like to thank the stakeholders of the project: Diana Albu, Enno van den Brink, Wim Rumpff, Theo Baan, Mikhail Astafev, Alex Dinu, Alexander Danilin, and Vinoth Krishnan Elangovan, who were cooperating in my project, sharing their experience, and providing guidance despite their busy schedule.

I thank my fellow PDEng peers for the support and guidance throughout the program. I would like to thank Yanja Dajsuren, the director of the Software Technology (ST) program, the ST program trainers, and the management assistant for their support throughout the last two years.

Finally, I would like to thank my family and friends for their unconditional love, support and encouragement.

Laavanya Balasubramanian
September 2018
Executive Summary

ASML supplies semiconductor manufacturers with advanced lithographic machines. The short time to market is crucial for ASML’s business that also helps them in maintaining their position as market leaders. Additionally, ASML also strives to ensure the products’ quality. So, there are several pilot projects focused on code generation to replace manual effort and to create efficient products on time.

This graduation project aims to evaluate the code generation for algorithms in production software and to propose an architecture that is a precondition for the application of code generation. We conducted project investigations by choosing an algorithm from the image sensing domain within ASML. This project is a technical evaluation of the application of code generation for production software. The organizational preconditions and changes are out of scope for this project.

Within several ASML engineering departments, the functional teams develop MATLAB algorithms and the software teams translate the MATLAB algorithm into production software. The code generation will help ASML to maintain only the MATLAB algorithm and generate production code from the MATLAB algorithm without compromising the performance such as execution time. This will help ASML to decrease the overall time spent in development, decrease synchronization effort, decrease maintainability effort, and avoid translation errors. In our project, we evaluate MATLAB Coder that can generate C or C++ code from the MATLAB algorithm.

In the project’s preparation phase, we selected candidate algorithms with an average degree of complexity based on the stakeholders’ preferences. In the project’s design phase, we modified the MATLAB algorithm to make it MATLAB Coder compatible for code generation. Then, we replaced the existing algorithm implementation in production code with the generated code. In addition to this, we redesigned the existing architecture in the algorithm context to minimize the external dependencies of the algorithm, i.e., the algorithm invocation and the algorithm parameter actions. In the project’s verification and validation phase, we verified the functional equivalence of the Coder compatible MATLAB algorithm with the functional team. Additionally, we validated the generated C-code of the algorithm for functional and performance equivalence against the MATLAB algorithm and the existing C implementation respectively. We also explored the options to optimize the performance of the generated code. For the proposed architectural redesign, we evaluated the performance equivalence to the existing C implementation.

In conclusion, the algorithm MATLAB code was modified to enable code generation such that the core algorithm functionality still remains the same. The generated C-code of the algorithm was functionally equivalent to the MATLAB algorithm. The performance equivalence is not much deviating compared to the existing C-code for the algorithm. Thus, providing confidence for ASML to investigate more on using MATLAB Coder for code generation. MATLAB Coder supports C-code generation from MATLAB code that can be embedded in both time critical and non-critical software. The generated code may require additional optimization techniques to suit the needs of a time critical software. Thus, ASML can support code generation rather than manual implementation. The proposed architectural redesign introduces a two percent additional delay which is acceptable for the candidate algorithm. For future work, the code generation along with the proposed architectural redesign should be encouraged for the new MATLAB algorithms to be introduced in the future developments on ASML machines.
Table of Contents

Foreword ..................................................................................................................... v
Preface ..................................................................................................................... vi
Acknowledgements ............................................................................................... vii
Executive Summary ............................................................................................... viii
Table of Contents ................................................................................................... ix
List of Figures ......................................................................................................... 1
List of Tables ........................................................................................................... 2

1. Introduction ....................................................................................................... 3
   1.1 Context ........................................................................................................... 3
   1.2 Project goal and scope ................................................................................ 3
   1.3 PARIS Sensor .............................................................................................. 4
       1.3.1 Purpose of the PARIS sensor ......................................................... 4
       1.3.2 Functionality of the PARIS sensor ............................................. 6
   1.4 Outline .......................................................................................................... 6

2. Problem Analysis ............................................................................................... 7
   2.1 Problem Definition ..................................................................................... 7
       2.1.1 Analysis of current situation ....................................................... 7
       2.1.2 Analysis of prior investigation for MATLAB deployment ....... 7
       2.1.3 Problem defined from analysis .................................................... 8
   2.2 Project flow .................................................................................................. 13
       2.2.1 Study feasibility .............................................................................. 13
       2.2.2 Select candidate algorithm .......................................................... 13
       2.2.3 Make the algorithm Coder compatible ....................................... 13
       2.2.4 Integrate the generated code ........................................................ 14
       2.2.5 Evaluate generated C-code Performance ................................... 14
       2.2.6 Propose and implement the architectural redesign ..................... 14
       2.2.7 Evaluate redesign ......................................................................... 14
       2.2.8 Replace the manual code with generated code ......................... 14

3. Stakeholder analysis ......................................................................................... 15

4. Project Requirements ...................................................................................... 19
   4.1 Stateless Algorithm .................................................................................. 21
   4.2 Separation of concerns ............................................................................. 22
   4.3 Stateless error handling ........................................................................... 22

5. Software architecture ....................................................................................... 23
   5.1 PARIS software component ................................................................... 23
5.2 Production code current architecture ...........................................24
5.3 Architectural redesign for production code ..................................26
  5.3.1 Data abstraction component design alternatives ..........27
5.4 Comparison of the design alternatives .......................................30

6. Implementation ............................................................................32
  6.1 MATLAB Coder .........................................................................32
    6.1.1 Candidate algorithm selection ........................................32
    6.1.2 Phase unwrap algorithm ................................................32
    6.1.3 Coder compatibility .......................................................32
    6.1.4 Integrating the generated code ........................................34
    6.1.5 Design considerations before code generation .............36
  6.2 Architectural redesign ...............................................................39

7. Verification and validation ..........................................................42
  7.1 MATLAB Coder .........................................................................42
    7.1.1 Verification before code generation ................................42
    7.1.2 Performance of Generated code ......................................42
  7.2 Architectural redesign ...............................................................47
    7.2.1 Functional equivalence ....................................................48
    7.2.2 Performance equivalence ..............................................48

8. Conclusion and future work .........................................................50
  8.1 MATLAB Coder .........................................................................50
    8.1.1 Future work for MATLAB Coder ....................................50
  8.2 Architectural redesign ...............................................................51
    8.2.1 Future work for the architectural redesign .................51

9. Project management ........................................................................52
  9.1 Project planning ........................................................................52
  9.2 Risk Analysis ............................................................................52

10. Project retrospective .................................................................54
    10.1 Personal reflection ...............................................................54
    10.2 Design criteria .....................................................................55

Appendix ............................................................................................56
    MATLAB Coder user manual ................................................56
    Stakeholder Interest and Concerns ..........................................61
    Way of working ........................................................................63

Glossary .............................................................................................66

Bibliography .......................................................................................67
    References ................................................................. Error! Bookmark not defined.

About the Author ..............................................................................69
List of Figures

Figure 1 - The process of pattern printing on the wafer ........................................4
Figure 2 - An ideal lens producing a spherical wave front.........................................5
Figure 3 - A real lens with aberrations producing a deviated wave front.....................5
Figure 4 - The wafer life cycle ..................................................................................6
Figure 5 - Analysis of current situation ......................................................................7
Figure 6 - The desired solution expected ..................................................................9
Figure 7 - Work flow specific for this project addressing both MATLAB Coder and architectural redesign..................................................................................11
Figure 8 - PARIS fit sequence architecture ..................................................................23
Figure 9 - PARIS fit sequence architecture ..................................................................25
Figure 10 - Sequence diagram for the PARIS fit sequence focused on the Phase Unwrap algorithm ........................................................................................................26
Figure 11 - Separation of concerns for the PARIS fit sequence architecture .......27
Figure 12 - Algorithm specific implementation .........................................................28
Figure 13 - Scan specific implementation ..................................................................29
Figure 14 - Hybrid implementation ............................................................................29
Figure 15 - Interface definition ..................................................................................33
Figure 16 - Define the input types .............................................................................34
Figure 17 - Undefined variables handled in the core algorithm .................................34
Figure 18 - Integrate generated code in the production code ...................................35
Figure 19 - Generated C-code in production code.....................................................36
Figure 20 - Data Flow Diagram for the phase unwrap algorithm.............................39
Figure 21 - Data Flow Diagram of the proposed solution .........................................40
Figure 22 - Sequence diagram addressing separation of concerns for the PARIS fit sequence focused on the phase Unwrap algorithm ........................................40
Figure 23 - Architectural redesign considering the phase unwrap algorithm that reflects the actual implementation .................................................................41
Figure 24 - Test cases to evaluate the generated C-code for the phase unwrap algorithm ...........................................................................................................44
Figure 25 - Leak summary example ..........................................................................46
Figure 26 - Test case 4 used as a benchmark to evaluate architectural redesign ...47
Figure 27 - Test case 5 to evaluate the architectural redesign along with the generated C-code for the phase unwrap algorithm ............................................48
Figure 28 - MATLAB Coder App...............................................................................56
Figure 29 - Review code generation readiness .........................................................57
Figure 30 - Define the input types .............................................................................58
Figure 31 - Check for run-time errors .......................................................................58
Figure 32 - Verify code .............................................................................................59
Figure 33 - Generate code .......................................................................................60
Figure 34 - Finish workflow ....................................................................................60
Figure 35 - Steps involved in using MATLAB Coder for ASML environment ...64
Figure 36 - Steps involved in applying the proposed architectural redesign for ASML environment .................................................................65
List of Tables

Table 1 Project's requirement.................................................................19
Table 2 Design alternatives evaluation table...........................................30
Table 3 The impact and tradeoffs for the chosen options with respect to the phase unwrap algorithm ...........................................................................38
Table 4 Test cases used to evaluate the performance of the generated C-code ..........................................................43
Table 5 Execution time of test cases 1, 2, and 3 ........................................45
Table 6 Total heap usage for the test cases 1, 2, and 3 ..............................46
Table 7 Test cases for evaluating the proposed architectural redesign ..........47
Table 8 Total heap usage for the test cases 4 and 5 ...................................49
Table 9 V-model activities and the weeks spent .......................................52
Table 10 Project's risks ...........................................................................53
1. Introduction

This chapter provides a brief introduction to the project by discussing the context and goals. Moreover, this chapter also introduces the PARIS sensor’s functionality within ASML lithography machines. The outline section provides an overview of the report structure.

1.1 Context

The project “Improving Maintainability of Time Critical Software by Applying MATLAB Coder and Architectural Redesign” was conducted as a final design project to pursue the Professional Doctorate in Engineering program (PDEng). The PDEng degree program in Software Technology provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology in the context of the 3TU.School for Technological Design, Stan Ackerman’s Institute.

ASML initiated this project, a company developing lithographic machines that help their customers to create effective geometric chip patterns on the wafers and hundreds of chips from a single wafer. They are market leaders in lithographic machines as their machines are designed to deliver high throughput, measured in wafers exposed per hour at nanometer precision.

Functional departments at ASML widely use MATLAB for solving complex numerical computations. In addition to this, ASML has software departments responsible for the C-code translation for the MATLAB code (m-code). ASML is aiming to avoid this error prone translation and to reduce the effort involved in development, version synchronization, maintenance, and to test for equivalence. ASML would like to maintain only m-code of the algorithm and automatically deriving the deployment version (C-code) from the m-code. This will significantly decrease the time spent in implementation, synchronization, maintenance, and verification. Thus, ASML shows interest in investigating a concept for deploying MATLAB algorithms in the existing Twinscan application. Ultimately, this could improve the development process within ASML.

For this investigation, we use MATLAB Coder. MATLAB Coder generates C-code from m-code for a variety of platforms, from desktop systems to embedded hardware. It supports most of the MATLAB language and a wide range of toolboxes. The generated C-code can be integrated into the development environment as source code, static libraries, or dynamic libraries.

1.2 Project goal and scope

The main goal of the project is encouraging the usage of MATLAB Coder within ASML. Both functional teams and software teams can be made aware of the advantages and guidelines in using MATLAB Coder. This pilot project will act as a technical Proof of Concept (POC) for embedding MATLAB algorithms using MATLAB Coder.

To investigate MATLAB Coder usage, the m-code developed for the PARIS domain within ASML is used. The PARIS domain is concerned with the sensors, which captures light source images and process them to calculate the corrections required to rectify lens aberrations. Correcting these lens aberrations result in more accurately exposed wafers.
This project is done with respect to the PARIS domain. The sub-goals that need to be achieved in this project are as follows:

- Generating C-code from algorithm m-code
- Integrating generated C-code into the existing Twinscan application
- Verifying and validating against the existing code’s functional and performance equivalence
- Optimizing possibilities to meet the performance equivalence
- Redesigning the existing PARIS software architecture to reduce the external dependencies related to the algorithm, i.e., the algorithm invocation and the algorithm parameter actions
- Reporting the project findings for future use in ASML
- Providing generic flows related to the way of working with MATLABCoder and architectural redesign

This investigation helps ASML in evaluating MATLAB Coder usage for their lithographic machines and designing a generic architectural solution for the existing system.

1.3 PARIS Sensor

1.3.1 Purpose of the PARIS sensor

The process of printing a pattern (circuit design) on the wafer using ASML lithographic machines is shown in Figure 1. First, beam of UV laser light is passed through a pattern that is to be printed on the wafer. Following this the light passes through a projection lens and then the lens projects the pattern on a wafer. Due to the high-resolution requirement and the low overlay budget, several sensors are essential to meet the accuracy requirements.

Figure 1 - The process of pattern printing on the wafer

Figure 2 shows how an ideal projection lens behaves when the light originating from a point source is passed through it, creating accurate patterns on the wafer. However, there are certain deviations introduced by the projections lens. These deviations lead to a phase shift of light rays through the lens and result in deviated wave front as shown in Figure 3.
The difference between the deviated wave front and the spherical wave front represents the lens aberrations or errors. The aberrations are due to manufacturing defects and production heating issues of the lens system. These aberrations need to be corrected to attain an accurate pattern on the wafer.

The PARIS sensor is located on the chuck that is used to carry the wafer as shown in Figure 2, represented by an orange oval shape on the wafer. The PARIS sensor captures the images of the light, which are used to measure the lens aberrations before the pattern is printed on the wafer. The PARIS sensor was developed in order to measure these aberrations per wafer, as the sensor used earlier was measuring these aberrations per lot. The PARIS software stack is responsible for controlling the measurement sequence and calculating the measurement results. The PARIS sensor client, the metrology, uses these results to correct the lens settings and reduce the effects of lens aberrations. The PARIS sensor measurement results are expressed in Zernike polynomials.
1.3.2 Functionality of the PARIS sensor

The life cycle of the wafer during lot production within the ASML lithographic machines is shown in Figure 4. In Measure stage, the wafer is checked for alignment, i.e., the wafer position, height, and tilt. During Expose stage, projection lens aberrations and the amount of light required are measured after the pattern alignment (pattern containing circuit design is aligned with the wafer). The wafer is then exposed to the pattern to be printed on it. Both stages are expected to be completed as quickly as possible due to the high throughput requirement.

The PARIS sensor measurement for aberrations is a part of the Expose stage. Metrology, a client of PARIS sensor system request PARIS to perform a scan and to return the Zernike polynomials that describe the lens aberrations. Metrology uses these values to fine-tune the projection lens settings such that the lens aberrations effect is reduced.

The PARIS sensor is used to increase the accuracy of the pattern printed on the wafer by measuring the lens aberrations per wafer and simultaneously increase the throughput by reducing the time taken for measurements. The ILIAS sensor (Older version of the sensor used to find aberrations) was used to measure lens aberrations per lot. Thus, the PARIS sensor measures the aberration per wafer while the ILIAS sensor measures the aberration per lot. The ILIAS sensor does not support per wafer measurement due to the time budget. For more information on the PARIS functionality refer to [5].

![Figure 4 - The wafer life cycle](image)

1.4 Outline

Chapter 2 provides the problem analysis and the roadmap to achieve the project goals. Chapter 3 introduces the stakeholders involved, their interests, and goals. The requirements gathered based on the stakeholder meetings are listed in Chapter 4. Chapter 5 describes the existing architecture, the proposed architectural redesign, and the design alternatives to implement the proposed architectural redesign. The implementation details related to MATLAB Coder and the proposed architectural redesign is discussed in Chapter 6. In Chapter 7, the verification and validation steps done to evaluate the m-code, the generated C-code, and the proposed architectural redesign is discussed in detail. The conclusion of this project is discussed in Chapter 8. Chapter 9 and Chapter 10 describes the project management and retrospective respectively.
2. Problem Analysis

This chapter describes the current situation of multiple teams within ASML, problem definition, and prior investigations. In addition to this, the project’s flow that describes the steps involved in our project are also explained.

2.1 Problem Definition

2.1.1 Analysis of current situation

Figure 5 shows the current situation of multiple domains within ASML. Most of these domains are composed of a functional team and a software team as shown in the left and the right side of Figure 5 respectively. The functional team consists of physicists, who develop m-code for algorithms that solve complex numerical computation. The software developers in the software team translates m-code into C-code manually. Thus, the functional and software teams maintain two versions of the same algorithm. Similar situation exists within the PARIS functional and software team.

Maintaining two versions of the same algorithm introduces a lot of drawbacks as shown in Figure 5. First, the development time spent in the manual translation of the m-code algorithm to C-code. Secondly, the teams check the m-code and C-code for functional equivalence. Finally, the teams also synchronize any changes in the m-code with the C-code version. Thus, the process of translating to C-code, testing for equivalence, and synchronizing changes is a repetitive, error prone, and time-consuming process.

2.1.2 Analysis of prior investigation for MATLAB deployment

To maintain only the MATLAB algorithm version (m-code) and derive automatically the deployment version (C-code) there are two approaches such as

- MATLAB runtime
- MATLAB Coder

MATLAB runtime is a collection of shared libraries and code that enables the execution of compiled and packaged MATLAB applications on systems without an installed version of MATLAB. Thus, the MATLAB runtime is used to run m-code directly on the TWINSCAN application via interpretation. As per prior investigation, MATLAB runtime usage is prohibited due to its low performance result. Because lot production cycle is very time critical and no further delays are allowed. Additionally,
the interpreter stability is a concern because in the prior investigation, it was noticed that the MATLAB runtime sometimes crashed while running for long period without being restarted. Thus, the MATLAB runtime is not a suitable candidate as the PARIS domain’s application is a part of the lot production cycle. For more information on this refer to [6].

MATLAB runtime is successfully used for performance non-critical applications like diagnostic tooling (used for diagnostic purposes), which are not time critical. These limitations result in the need for investigating code generation. In our project, we use MATLAB Coder for the code generation.

2.1.3 Problem defined from analysis

Based on the situation described above, the project focuses on generating C-code using MATLAB Coder and providing solutions to the following problems:

1. What is separation of concern? Why is separation of the concerns crucial for the PARIS domain? How can the separation of concerns be addressed within the PARIS domain?

2. How to integrate the generated C-code into the existing PARIS software? What is a stateless algorithm?

3. How can the MATLAB algorithm be made Coder compatible? How can the MATLAB implementation be aligned with respect to the software implementation? How can issues with MATLAB functions that are unsupported by MATLAB Coder be mitigated?

4. How can the performance of generated C-code against the existing C-code implementation be evaluated?

5. What are the options to achieve better performance in terms of execution time and memory management?

The answers to the above questions will help multiple domains within ASML in the following ways:

- Maintaining only m-code version for correction algorithms
- Reducing the time spent on implementing, updating, and testing
- Avoiding error prone translation
- Addressing separation of the concerns
- Validating and verifying functional and non-functional requirement
- Providing MATLAB Coder guidelines
Figure 6 shows the desired solution expected from this project. The functional team develops and maintains the m-code version as shown on the left side. The deployable c-code version as shown on the right side will be automatically derived from the m-code using MATLAB Coder. Based on the performance of the generated C-code, ASML could decide to use MATLAB Coder in lot production software stack and encourage more internal teams towards this transition. This project will serve as a proof of concept for this initiation.

Figure 6 - The desired solution expected
Figure 7 – Work flow specific for this project addressing both MATLAB Coder and architectural redesign
2.2 Project flow

To evaluate MATLAB Coder and integrate the generated code into the production code, we identified several steps. This section explains these steps. Figure 7 shows the steps involved in using MATLAB Coder in production code.

This workflow could act as a standard workflow for the ASML engineers who would like to use MATLAB Coder or the proposed architectural redesign in future. Certain steps in the workflow do not hold for every MATLAB Coder project or the architectural redesign as it is specific for the PDEng project or the PARIS domain. For example, the feasibility study, candidate algorithm selection, and architecture redesign are steps that are specific for this project. For more information on the way of working for the ASML engineer who would like to generate code for the algorithms or follow the proposed architecture refer to Appendix (Way of working).

2.2.1 Study feasibility

This phase is essential to understand the project’s context, get acquainted with the work environment, analyze the stakeholders’ concerns, and gather detailed requirements. This phase is focused on producing a prototype that intends to prove MATLAB Coder usability. This phase includes five major steps, such as

- Select an algorithm with limited complexity
- Make the algorithm Coder compatible
- Validate the generated code
- Replace the manual code with generated code
- Perform Equivalence test

This phase is aimed at providing a green light for further investigation over MATLAB Coder usage for this project. Thus, functional equivalence is checked, and performance equivalence was not of higher importance for this phase. This phase is very specific for the PDEng project, not necessarily a part of the general workflow. Additionally, the requirements specific to our project is elicited in this phase.

2.2.2 Select candidate algorithm

Within the PARIS domain, there are several MATLAB algorithms developed for tackling lens aberrations. Based on the stakeholders’ interest, several of these algorithms are chosen for investigation. This phase is very specific for the PDEng project, not necessarily a part of the general workflow.

2.2.3 Make the algorithm Coder compatible

In this phase, the algorithm is made Coder compatible. The candidate algorithm is checked for the following:

- Custom and built-in functions
- Unsupported MATLAB built-in functions
- Limitations of MATLAB built-in functions

Based on this information, m-code might need some changes to make it MATLAB Coder compatible (MCC). Following this, C-code is generated for this MCC version. Then, the generated code is built and run, thus, fixing any run time errors and verifying that the generated C-code provides the same functionality as the m-code. For more information about this, refer to Section 6.1.3.
2.2.4 Integrate the generated code

In this phase, the manual C implementation of the algorithm in the existing Twinscan application is replaced by the generated C-code from MCC version. For more information about this, refer to Section 6.1.4. Additionally, the existing architecture is investigated for improvements in the context of algorithm deployment.

2.2.5 Evaluate generated C-code Performance

During Performance evaluation, the results of MATLAB code and generated C-code is checked for functional equivalence. Following this, the generated C-code is checked for performance equivalence against the existing C-code. The major performance aspects evaluated are the execution time, memory usage and memory leaks. Section 7.1.2, provides information on evaluation of the generated C-code. There are several optimization techniques to improve performance. Section 6.1.5, explains the optimization techniques evaluated in our project to improve performance.

2.2.6 Propose and implement the architectural redesign

In this phase, the existing architecture is explored. Following this, architectural redesign essential within the PARIS domain is proposed and implemented. Chapter 5 and Section 6.2, provides information about the proposed architectural redesign and its implementation.

2.2.7 Evaluate redesign

The implemented architectural redesign is verified and validated for performance equivalence as specified in Section 2.2.5 in the context of the logical action. This step is focused on finding any bottlenecks introduced by the proposed architectural redesign. Based on these evaluations, the proposal can be revisited, or the existing architecture could be used. Despite these results, the redesign proposal can be used as a reference architecture for the PARIS domain’s future projects. Section 7.2, explains how the proposed architectural redesign is evaluated.

2.2.8 Replace the manual code with generated code

In this phase, the results from Sections 2.2.5 and 2.2.7 are used to finalize the usage of MATLAB Coder for the PARIS domain. The possible outcomes of this phase are

- Replacing the manual code with the generated code
- Replacing the manual code with the generated code with architectural redesign
- Using generated C-code for time trivial context
3. Stakeholder analysis

This chapter identifies the main stakeholders of the project. Stakeholder analysis helps to gather their interest and concerns related to the project. These interests and concerns need to be translated or mapped to the project requirement. Based on the stakeholder meetings, the following aspects specific for the stakeholder are elaborated. For information regarding the stakeholders’ interest and concerns refer to the Appendix (Stakeholder Interest and Concerns).

- Goals: What is their interest/goal in the project?
- Roles: What is their role in this project?
- Acceptance criteria: What are their expectations from this project?
- Involvement: When are the stakeholders available? What is the communication channels used?

Wouter van Heijningen: Project supervisor from ASML

<table>
<thead>
<tr>
<th>Goals</th>
<th>Roles</th>
<th>Acceptance criteria</th>
<th>Involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Feasibility check to MATLAB Coder usability for the PARIS domain MATLAB algorithm.</td>
<td>• Provide guidance on the project such as directions, context information or contact information.</td>
<td>• Report weekly about the project’s process.</td>
<td>• Continuous communication by thrice weekly or ad-hoc based meetings and monthly PSG meeting with University supervisor.</td>
</tr>
<tr>
<td>• Address the architecture improvements for the algorithm deployment inside the production code of the Twinscan application SW client.</td>
<td>• Provide feedback and evaluate the project’s progress.</td>
<td>• Generic architecture solution for the PARIS MATLAB algorithm deployment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Evaluate the final deliverables.</td>
<td>• Proof of concept for Coder usability with one PARIS MATLAB algorithm covering performance evaluation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Review the final report.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Tom Verhoeff**: Project supervisor from TU/e

<table>
<thead>
<tr>
<th>Goals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Guide the trainee during the project.</td>
<td></td>
</tr>
<tr>
<td>• Deliver solution that is consistent with the defined goal of the project.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Provide guidance on the project such as directions and relevant contacts.</td>
<td></td>
</tr>
<tr>
<td>• Provide feedback and evaluate the project’s progress.</td>
<td></td>
</tr>
<tr>
<td>• Provide feedback and advice on the trainee’s approach and performance.</td>
<td></td>
</tr>
<tr>
<td>• Review the final report</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acceptance criteria</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Report regularly about the project’s process</td>
<td></td>
</tr>
<tr>
<td>• The project is designed, implemented and managed to meet the OOTI standards.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Involvement</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Continuous communication by monthly PSG meeting with ASML supervisors or ad-hoc meetings.</td>
<td></td>
</tr>
</tbody>
</table>

**Vanessa Rodriguez**: Managerial supervisor from ASML and MATLAB Embedding project lead

<table>
<thead>
<tr>
<th>Goals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Feasibility study which might help decrease the software development lead time.</td>
<td></td>
</tr>
<tr>
<td>• Experience gained with converting existing MATLAB algorithms to Coder compatible.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Provide managerial support to the trainee during the project.</td>
<td></td>
</tr>
<tr>
<td>• Provide guidance on the project such as directions and relevant contacts.</td>
<td></td>
</tr>
<tr>
<td>• Provide feedback and evaluate the project’s progress.</td>
<td></td>
</tr>
<tr>
<td>• Provide feedback and advices on the trainee’s approach and performance.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acceptance criteria</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Report regularly about the project’s progress.</td>
<td></td>
</tr>
<tr>
<td>• Keep the stakeholders well informed about the project’s approach and process</td>
<td></td>
</tr>
<tr>
<td>Involvement</td>
<td>Continuous communication by monthly PSG meeting, bi-weekly update meeting and ad-hoc meetings.</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

**Enno van den Brink**: Functional engineer

<table>
<thead>
<tr>
<th>Goals</th>
<th>One source code solution that reduces development work and extra effort spent on making more efficient algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Provide guidance on the project such as directions and relevant contacts.</td>
</tr>
<tr>
<td></td>
<td>Provide the trainee sufficient KT for the PARIS MATLAB algorithm code during the project.</td>
</tr>
<tr>
<td>Acceptance criteria</td>
<td>Provide a Coder guide to explain the steps involved in code generation.</td>
</tr>
<tr>
<td>Involvement</td>
<td>Weekly meeting or ad-hoc meetings.</td>
</tr>
</tbody>
</table>

**Diana Albu**: Software lead designer

<table>
<thead>
<tr>
<th>Goals</th>
<th>The MATLAB algorithm could remain as a black box to the SW developers. Thus, the software development team can concentrate on other improvements needed within PARIS domain.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Provide direction or relevant information regarding the present architecture.</td>
</tr>
<tr>
<td></td>
<td>Provide feedback on the proposed architectural design.</td>
</tr>
<tr>
<td></td>
<td>Provide feedback and advices on the trainee’s approach and performance.</td>
</tr>
<tr>
<td>Acceptance criteria</td>
<td>Report regularly about the design decision.</td>
</tr>
<tr>
<td></td>
<td>The architectural design should be generic to support all the correction algorithms.</td>
</tr>
<tr>
<td>Involvement</td>
<td>Ad-hoc meetings or regular meeting monthly.</td>
</tr>
</tbody>
</table>
**Wim Rumpf**: Software FC Architect

**Goals**
- The MATLAB algorithms could remain as a black box to the SW developers. Thus, more effort towards software architecture improvements.

**Roles**
- Provide direction or relevant information regarding the present architecture.
- Provide feedback on the proposed architectural design.
- Provide feedback and advice on the trainee’s approach and performance.

**Acceptance criteria**
- Report regularly about the design decision.
- The architectural design should be generic to support all the correction algorithms.

**Involvement**
- Ad-hoc based meetings or regular meeting monthly.

**MATLAB Embedding project**

**Goals**
- Based on the experience of using MATLAB Coder in production code (PARIS) some insights for the MATLAB Embedding team, such as providing feedback on the supporting tools created, evaluating the performance, and evaluating design decisions made for implementation.
- Encourage more teams to adapt to this transition.

**Roles**
- Provide managerial support to the trainee during the project.
- Provide guidance on the project such as directions and relevant contacts.
- Provide feedback and evaluate the project’s progress.
- Provide feedback and advices on the trainee’s approach and performance.

**Acceptance criteria**
- Report regularly about the project’s process.
- Provide more information about the Coder compatibility changes.

**Involvement**
- Ad-hoc based meeting or regular meeting monthly.
4. Project Requirements

Our project aims at validating the usage of MATLAB Coder for the PARIS domain and providing a generic architectural design that facilitates a smooth transition for this algorithm deployment. For this investigation, we selected an algorithm from the PARIS domain.

The stakeholders’ proposed several candidate algorithms for this investigation. We selected the phase unwrap algorithm based on the discussions with stakeholders. For the rationale behind this selection Section 6.1.1. The proposed architectural redesign must be evaluated for a logical action, i.e., a high-level check which includes a set of operations. We selected Fast Zernike (FZ) scan for investigation as it is a frequently requested by the client. For more information on FZ scan refer to Sections 5.1 and 5.2.

The projects requirements are defined by analyzing the problem, performing the feasibility analysis, and discussing with stakeholders. Table 1 shows the list of requirements that we derived by considering the stakeholder’s interests and priorities as well as the project’s time line.

The requirements are prioritized in the following categories.

- **Must:** Must be present in the final deliverable
- **Should:** It is recommended to have it in the project. However, how detailed these requirements were addressed depends on the project’s time line.

### Table 1: Project’s Requirement

<table>
<thead>
<tr>
<th>Id</th>
<th>Requirement</th>
<th>Rationale</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.1</td>
<td>All the mathematical algorithms used for lens aberration measurement must be in MATLAB to generate C-code.</td>
<td>To support the usage of the algorithm in production code by code generation.</td>
<td>Must</td>
</tr>
<tr>
<td>R.2</td>
<td>Coder compatible PARIS MATLAB algorithms shall have passed the check for runtime errors using the MEX files.</td>
<td>Before generating C-code, the MEX function could be tested to detect, and fix run time errors that are harder to diagnose in the generated code.</td>
<td>Must</td>
</tr>
<tr>
<td>R.3</td>
<td>Generated C-code for the PARIS MATLAB algorithms shall be stateless.</td>
<td>The algorithm is self-contained as explained in Section 4.1. So, easier to adopt to the separation of concerns pattern (Section 4.2).</td>
<td>Must</td>
</tr>
<tr>
<td>R.4</td>
<td>Generated C-code shall be integrated with the existing Twinscan SW client.</td>
<td>The generated C-code will replace the C-code, which is in the production code of the Twinscan application, developed by the software team.</td>
<td>Must</td>
</tr>
<tr>
<td>R.5</td>
<td>The results from the generated C-code shall be verified for functional equivalence against MATLAB results.</td>
<td>The generated C-code will use the existing test cases and the results should match with the MATLAB results.</td>
<td>Must</td>
</tr>
<tr>
<td>R.6</td>
<td>The execution time of the generated C-code for PARIS MATLAB algorithm shall not be very deviating from the execution time (seven millisecond).</td>
<td>The generated code should be able to meet the execution time requirement achieved by the existing code as a candidate for replacing it.</td>
<td>Should</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td>Must/Should</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>R.7</td>
<td>Generated C-code for PARIS MATLAB algorithm shall not introduce any memory leaks and the memory usage shall not be increased by more than ten percent in comparison with the existing C implementation (4Mb).</td>
<td>Memory leaks and high memory usage can lead to impaired performance or failure.</td>
<td></td>
</tr>
<tr>
<td>R.8</td>
<td>Generated C-code for PARIS does not require any manual implementation.</td>
<td>The generated C-code is not changed to meet the functional/non-functional requirements of the algorithm. Thus, generated code replaces the existing code in the Twinscan application.</td>
<td></td>
</tr>
<tr>
<td>R.9</td>
<td>A Coder guide shall be used to describe the steps involved in the deployment of MATLAB algorithms in production code.</td>
<td>A guide for a smooth transition, unsupported functions, and limitation using MATLAB Coder for PARIS domain that helps MATLAB Coder users for future works.</td>
<td></td>
</tr>
<tr>
<td>R.10</td>
<td>Generated C-code for the PARIS MATLAB algorithms shall have minimal interfaces.</td>
<td>The algorithm interface is provided with only essential data types.</td>
<td></td>
</tr>
<tr>
<td>R.11</td>
<td>Generated C-code for the PARIS MATLAB algorithms shall be isolated.</td>
<td>The algorithm shall depend only on the interface that it implements. Thus, making them easier to test/qualify, re-use, and deploy in different processes.</td>
<td></td>
</tr>
<tr>
<td>R.12</td>
<td>Algorithm specific data shall be available via a data abstraction module.</td>
<td>Separation of concerns for more information refer to Section 4.2</td>
<td></td>
</tr>
<tr>
<td>R.13</td>
<td>Data abstraction modules along with the generated C-code for the PARIS MATLAB algorithm shall not introduce unacceptable delays, i.e., within the time budget allocated for the PARIS domain (100ms).</td>
<td>The data abstraction module performance needs to be evaluated for future use. To validate this requirement R.6 can be used to check the additional delay introduced due to the data abstraction module.</td>
<td></td>
</tr>
<tr>
<td>R.14</td>
<td>Data abstraction modules along with the generated C-code for the PARIS MATLAB algorithm shall not increase the memory usage by more than ten percent in comparison with the existing C implementation.</td>
<td>The data abstraction module performance needs to be evaluated for future use. To validate this requirement R.7 can be used to check the increase memory usage introduced due to the data abstraction module.</td>
<td></td>
</tr>
<tr>
<td>R.15</td>
<td>An algorithm shall log an error in the event log of the TWIN-SCAN.</td>
<td>This requirement will help to trace back to the error in case of failure.</td>
<td></td>
</tr>
</tbody>
</table>
4.1 **Stateless Algorithm**

To understand the concept of stateless algorithm, let us first discuss both stateful and stateless concepts. A stateful application has one or more of these features:

- Modifies shared data structures
- Uses global variables
- Allows exception handling
- Is not self-contained (depends upon other functions)

Stateful functions are used in transactions that maintain prior history. For example, client specific applications that store client information, status, and history are developed using stateful concepts.

An algorithm is stateless if the above-mentioned features are avoided, i.e.,

- No shared data structures
- No global variable usage
- No exception handling
- No external function calls needed (self-contained)

In a nutshell, stateless algorithms provide the same response to the same request or function call every time. Thus, no memory of the past needs to be maintained and every transaction is performed as if it is encountered for the first time. So stateless programming is ideal for implementing a mathematical notion of function or mathematical computations. There are several advantages for defining a stateless algorithm such as:

- Simplified debugging
- Parallelized computations (not sharing any data structures)
- Simplified development

Stateless functions are easier to debug as they are pure functions which are straightforward and do not modify the state in other parts of the application. In addition to this, since there is no state manipulation, special libraries, unintended interactions, or global libraries certain deadlocks or bugs could be avoided.

A stateless algorithm less effort is spent towards understanding the function it turns into a black box that provides outputs for the given inputs. A Stateless algorithm might also result in improved performance since there is no state behavior, unnecessary checks or memory allocation associated.

Stateless functions are at disadvantage for the systems that require maintaining information about prior executions. Within the PARIS domain, the correction algorithms are responsible for some mathematical calculations and return the measured or corrected values. So, the algorithms within the PARIS domain can be made stateless, which makes it more suitable to adapt to the DCA pattern (Section 4.2)
4.2 Separation of concerns

Separation of concerns is an increasing software design pattern which is implemented to minimize the dependencies. The term concerns in the context of separation refers to the different aspects of the software functionality. For example, the business logic of software could be one concern and the interface which is used to access the logic is another concern. Separation of concerns reduces the dependencies between the business logic and interface, i.e., any changes in the interface do not require changes in business logic and vice versa. Having the concerns separately brings in advantages such as

- Ensure maintainability
- Facilitate reusability
- Increase code quality
- Isolate developers’ work
- Increase readability or understandability

ASML lithography scanners are designed to provide high throughput with nanometer precision. To achieve this performance, a huge amount of software is needed to implement and apply many corrections and control loops, although several of this software communicate by passing values directly. The problem arises due to the lack of common datatypes definitions that take care of the conversion needed to facilitate the communication across applications.

To solve this problem, a group of architects at ASML came up with a new architectural pattern called DCA, which separates control from data and actions.

The term DCA is described as follows:

- **Data** represents the parameters needed to make logical decisions
- **Control** represents the fixed/state behavior sequence
- **Algorithm** represents actions with a long execution time

Adapting the existing architecture to align with the DCA might help the PARIS domain to attain all the specified advantages. For more information on the DCA architecture pattern refer to [4].

4.3 Stateless error handling

As discussed earlier, exception handling is not allowed within the concept of statelessness. The reason is that handling exceptions by calling control actions might introduce state behavior to the algorithm, thus turning them into a stateful algorithm. R.13 specified in Table 1 is related to Error logging. Error logging involves tracing the errors, but no action is taken to handle these errors, since error handling introduces stateful behavior. Error handling is decided by the client. For the PARIS domain, the client metrology or the control part of the PARIS application decide upon how to handle errors. ■
5. Software architecture

This chapter explains the architecture of the PARIS calculation process, the existing architecture related to the project’s scope, the architecture redesign required to embed the generated C-code, and the impact of the redesign architecture.

5.1 PARIS software component

Figure 8 shows the overview of the PARIS sensor functionality and the fit sequence. After the client’s request for a scan, the PARIS sensor captures the light images to retrieve a raw frame. As soon as these frames are available, the PARIS domain applies a set of algorithms which are applied per frame. This is represented as “Process full frame” in Figure 8. In this stage, the whole frame is retrieved, and corrections are made on the whole frame. After the frame processing stage, the frame is split into seven sub-images for each detector channel and each detector channel is processed separately. These seven parallel processes are referred to as “PARIS fit sequence” in the remainder of this report. Each PARIS fit sequence consists of a set of algorithms. The Zernike polynomials are calculated from each of these fit sequences. Metrology, the client of PARIS sensor software stack uses these Zernike polynomials to correct the lens settings and reduce the impact of aberrations. Section 5.2 describes how the PARIS fit sequence is implemented within the PARIS software stack.

Figure 8 - PARIS sensor functional overview and fit sequence
5.2 Production code current architecture

The PARIS domain provides several scan types to its Twinscan-internal client: Metrology. In this project, we considered the Fast Zernike (FZ) scan for investigation. Figure 9 shows the abstract view of this FZ existing architecture.

An FZ scan consists of a sequence of actions to collect multiple camera images (in FZ scan one camera image is considered as one phase step). A Zernike calculation algorithm processes those camera images into a set of Zernike polynomials per detector channel. Additionally, this scan type is optimized for performance critical circumstances. We chose FZ scan for investigation as it is a frequently requested scan type and the architectural redesign should be verified and validated to check if it introduces bottlenecks to the FZ scan’s current time budget and memory footprint.

Following the client’s request for the FZ scan, the next steps performed are: initializing the essential data, allocating memory essential for the FZ scan and creating the PARIS fit sequences for the FZ scan. A PARIS fit sequence consists of a set of algorithms that are executed sequentially to find the lens aberrations. These algorithms either apply corrections to the measurements or perform calculations based on the measurements. There are several phase steps and each of these phase steps is split into seven sub-images for each detector channel. For each of these detectors, a PARIS fit sequence needs to be created. This is the preparation stage for the FZ scan.

Following the creation of the PARIS fit sequence per phase step, the current phase step is set. Then, the execution of the algorithms in fit sequence per detector takes place, which triggers the algorithms consecutively in the PARIS fit sequence. This is the measurement stage for the FZ scan.

In Figure 9, the modules enclosed in the orange box implements the preparation stage and sequence trigger. The modules enclosed in the green box consist of the algorithms executed to calculate Zernike coefficients. Additionally, the supporting functions such as memory allocation for the algorithm are made available in their respective modules. The phase unwrap algorithm, which is a part of the PARIS fit sequence, is selected for investigation. For more explanation on the algorithm refer to Section 6.1.2.
Figure 10 shows the sequence diagram of the PARIS fit sequence focused on the phase unwrap algorithm. The sequences in the orange box represent the preparation stage for the FZ scan explained earlier. The sequences in the green box represent the measurement stage where the fit sequence algorithms are executed consecutively, which also includes the phase unwrap. The phase unwrap algorithm is implemented in the Calculate phase module, as shown in Figure 10.

Each algorithm to be executed is triggered with the parameters required via void pointers, represented by the blue boxes in Figure 10. Before the execution of the algorithm, the supporting functions such as memory allocation are also required. Thus, there is a tight coupling between the control and data, i.e., the sequence of the algorithms to the executed and the parameters needed are passed via the pointers. In addition to this, the control and algorithm are also tightly coupled, i.e., the algorithm requires supporting functions (control). These tight coupled modules decrease the code readability, maintainability, scalability, and understandability. In Section 5.3, the proposed architectural redesign, which helps to minimize the drawbacks created by the tight coupling of the data, control, and algorithm is explained.
To make the software more scalable, maintainable, readable, and understandable the tight coupling between the data, control, and algorithm in the existing architecture (Section 5.2) should either be minimized or removed fully. To minimize the coupling, separation of concerns is proposed as a solution. Refer to Section 4.2 for information on separation of concerns. The proposed architectural redesign is shown in Figure 11. The tightly coupled data and control can be addressed by introducing a data abstraction component, Fast Zernike Data (FZD), enclosed in the blue box in Figure 11. The data abstraction component is responsible for managing the parameters essential for the algorithm and provide them via interface functions. The modules enclosed in the orange box of Figure 11 show the control. The controller uses the data abstraction component’s interface functions to get and set data. The modules enclosed in the green box represent the algorithms. For this investigation, the C-code is generated only for the phase unwrap algorithm, but it is possible to generate C-code for the m-code for all the algorithms.

The tight coupling between algorithm and control can be addressed by the generated C-code using MATLAB Coder. The generated C-code is a stateless function which takes care of all the supporting actions internally. The module represented in dark green color contains the generated C-code for the phase unwrap algorithm. For more information on the stateless concept refer to Section 4.1.

The design alternatives to implement the data abstraction component are discussed in Section 5.3.1. For more information on how the data abstraction component is implemented in the production code, refer to Section 6.2.
5.3.1 Data abstraction component design alternatives

In our project, to implement the FZD component, several alternatives were investigated. The alternatives are discussed in this section.

Algorithm specific implementation

Algorithm specific implementation for the data abstraction component is represented in Figure 12. This component consists of several modules. Each of these modules represents the data abstraction module for the algorithm that is responsible for computational intensive actions. The interface functions are made available via which the parameters required for the execution of the algorithm are managed. In case of the PARIS domain, the data abstraction modules for the algorithms in the PARIS fit sequence are used to measure lens aberrations. The data module of phase unwrap manages the algorithm data and exposes them via the interface functions as shown in Figure 12. The set function is used to update the parameters needed before the execution of the algorithm and the result parameters after the execution of the algorithm. The get function is used to retrieve the parameters needed for the execution of the algorithm.
In general, the control is a fixed set of sequences as specified in Section 4.2. So, a data abstraction component which manages all the parameters needed for this fixed set of sequences can be created. Within the PARIS domain, there are different types of scans that are made available to the client, the Metrology. Each of these scan types executes a set of different algorithms. Scan specific data abstraction component for the PARIS domain is represented in Figure 13. The data modules are created for the different scan types. The scan specific data implementation abstracts an individual algorithm via a scan specific interface. The scan specific module uses the data module of the individual algorithm required for that scan type. This improves the reusability of the SW within the data component and prevents code duplication. The scan specific parameters are accessed via set and get functions provided by a scan specific interface. For example in Figure 13, the interface of scan type 1 abstracts the internal algorithms (Algorithm 1 – Algorithm N) that will be executed in scan type 1. Similarly, the different scan types abstract a different set of algorithm specific data modules.
Another alternative to implement the data abstraction component is going for a hybrid implementation as shown in Figure 14. This design alternative is specific to the PARIS domain. The various scan types have a different set of algorithms. These Scan types differ either in the execution order of the algorithms or use totally different algorithms in its execution. Thus, a hybrid data abstraction component implementation has a data abstraction that manages the data of the algorithm that is commonly used across multiple scan specific data modules for algorithms. Depending upon the invoked scan type the corresponding interfaces will be used.
Though the hybrid data abstraction component is a potential design alternative, it is not considered in this investigation as it might decrease cohesion by maintaining unrelated items within a module. In addition to this, a hybrid implementation decreases the understandability and readability for the developers. Thus, hybrid implementation is not considered for further investigation in Section 5.4.

### 5.4 Comparison of the design alternatives

The pros and cons of algorithm specific and scan specific implementation is discussed in this section.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Algorithm specific implementation</th>
<th>Scan specific implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testability</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Reusability</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Easy refactoring</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Number of interfaces</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Memory usage</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 2 shows on which criteria the design alternatives are investigated.

**Testing:** A data abstraction component’s main functionality is to maintain data integrity, abstract the underlying complexity of the data and provide it to a client when requested via interface functions. Such a component can be tested by qualifying the data abstraction interface. The algorithm specific data abstraction component can be used to test the algorithms, i.e., setting the parameters, retrieving the parameters, executing the algorithm and verifying the result. Thus, the Data and Algorithm explained in Section 4.2 are tested. The scan specific data abstraction component enables to qualify scan type flow i.e., checking the sequence of algorithms in a scan type without any exceptions; in addition to data and algorithm. Thus, the Data, Control, and Algorithm explained in Section 4.2 are tested. Both algorithm specific implementation and scan specific implementation provides good testability but differ in the scope of the test. For the studied PARIS fit sequence qualifying individual scan type is preferred since scan specific implementation also include testing the algorithms used within the scan.

**Reusability:** The algorithm specific data component provides interfaces that can be used by clients for handling data for individual algorithms. This makes it highly reusable for different scan types. The scan specific algorithm is tailored to a specific use case and hence it has limited reusability. From design perspective the scan specific implementation is proposed as an extension of algorithm specific implementation; the internal software design of the data abstraction component can take advantage of the reusability of algorithm specific implementation for different use-cases. Hence the scan specific data component becomes modular without too much code duplication.

**Refactoring:** Any changes in the algorithm data i.e., addition/deletion of the algorithm parameters will be localized in the corresponding algorithm data abstraction module in both these implementations. The scan specific implementation being an extension of algorithm specific implementation holds this advantage as well.

**Number of Interfaces:** In the studied PARIS fit sequence, there are 21 algorithms and 7 scantypes. The number of interfaces that needs to be exposed for choosing the algorithm specific implementation or scan specific implementation is 21 and 7 respectively. The fanouts of an algorithm specific data component is higher than a scan...
specific data component which could bring down the software quality metrics for algorithm specific implementation. If an algorithm specific interface is modified, all scan types using the algorithm should be qualified. In a scan specific implementation, only the specific scan type is impacted. Since the number of interfaces in an algorithm specific implementation is higher than the scan specific implementation, the test effort to qualify a functional change is most likely to be high due to the impact on the clients that use the interface.

**Memory usage:** In PARIS fit sequence, there are certain algorithm parameters which are generic for a certain set of algorithms in the different scantypes. For the algorithm specific implementation, a separate data module for generic shared parameters can be introduced. This data module avoids the duplication of memory for the generic parameters across the algorithms. Similarly, the scan specific implementation could also avoid memory duplication by maintaining the generic parameters across the whole scan process separately such that all the algorithm specific implementations internally use the parameters.

Both the algorithm and scan specific implementation helps to achieve decouple the data and the control. The scan specific implementation is preferred for the studied PARIS fit sequence since it provides a highly cohesive and modular data abstraction component for the scan types. Also, from the testing perspective, the scan specific implementation covers the data, control and algorithm part of different scan types within the studied PARIS fit sequence. This recommendation is very specific for the studied PARIS fit sequence considering its architecture and functionality. ■
6. Implementation

This chapter explains the two separate implementations performed in our project with respect to MATLAB Coder and the proposed architectural redesign. In Section 6.1, the candidate algorithm selection, the selected candidate algorithm’s introduction, the guidelines to make the m-code Coder compatible for code generation, and the generated C-code integration with the production code are discussed. In Section 6.2, the implementation of the proposed architectural redesign for the studied application: the PARIS fit sequence is also explained.

6.1 MATLAB Coder

In order to experiment with MATLAB Coder usage, the MATLAB scripts developed by the functional team is used. This section describes the steps involved in making the script Coder compatible for generation and integrating the generated code.

6.1.1 Candidate algorithm selection

In the PARIS domain, there are 21 algorithms that apply corrections to the measurements or perform calculations based on the measurements to handle lens aberrations. The phase unwrap algorithm was chosen for investigation after discussions with the stakeholders. The rationale behind this selection is as follows

- High complexity of the algorithm
- A time-consuming algorithm in the PARIS domain
- An undesired difference between the existing C implementation and Matlab implementation

We selected a complex algorithm such that if it is feasible to generate code for this algorithm, it acts as a representative and is expected to be feasible for most of the other 20 algorithms as well. Considering a time-consuming algorithm helps to validate whether the generated C-code matches or exceeds the performance requirements. Additionally, there is a difference between the MATLAB and the C implementation, i.e., the Current C-code is not a precise translation of the m-code. So, the results of the m-code and C-code are not equal, and the PARIS domain allows a tolerance threshold for the mismatching results. These aspects make the phase unwrap algorithm a suitable candidate for investigation. The following section explains phase unwrap algorithm.

6.1.2 Phase unwrap algorithm

The phase unwrap algorithm is used in this investigation. In interferometry, the phase is calculated by an arctangent expression that returns the values between -π and +π. Hence the result is modulo 2π and discontinuities with values near -π and +π are found in the phase values. In the phase unwrap algorithm, these discontinuities are detected and resolved to achieve result with the desired continuous phase value. Unwrapping is done by adding or subtracting 2π to the phase value to remove the phase shifts. The algorithm used in the PARIS domain is known as quality guided phase unwrapping. For a more detailed explanation on the algorithm implementation refer to [3].

6.1.3 Coder compatibility

In this section, the changes made to the phase unwrap m-code to make it Coder compatible to generate the C or C++ code for the phase unwrap algorithm are explained. The major changes performed in the phase unwrap m-code are as follows.
**Interface definition**

The interface is defined for the entry function and the subsequent functions, so the generated C-code functions have interfaces, which are clearly defined. The parameters needed for the phase unwrap algorithm are provided in the functions as shown in Figure 15. We provide the parameters needed for the entry and the subsequent functions.

![Diagram showing function calls](image)

**Input types definition**

In addition to this, MATLAB is a dynamically interpreted language, it determines the variable properties at runtime. In contrast, C and C++ are statically interpreted languages, at compile time. In order to generate C-code, MATLAB Coder should determine the properties of all variables in the m-code. MATLAB Coder should infer the properties of entry-point function inputs. There are several options to define the inputs for MATLAB Coder.

In our project, we defined the inputs programmatically in the m-code. The MATLAB assert function was used to define the properties of the entry function inputs, i.e., the name of the input types, the data type of the input, and the maximum size of the input. Figure 16 shows how the input variables are defined programmatically to facilitate code generation.
In the core algorithm, certain variables are not declared earlier as the MATLAB supports dynamically growing variables, but MATLAB Coder does not support this feature, so it is declared prior as shown in Figure 17.

In addition to this, the nested functions are not supported by MATLAB Coder, so this feature is altered as it does not affect the core functionality. In this project, we made the m-code coder compatible and we generated C-code after addressing the above-mentioned aspects. For more information about the steps involved to make the m-code Coder compatible and to generate the C-code refer to Section ‘MATLAB Coder user manual’ in Appendix.

6.1.4 Integrating the generated code

From the Coder compatible m-code, we generate the C-code. When an application uses a generated C/C++ code, the engineer needs to provide a C/C++ function which calls the generated code. By default, MATLAB Coder generates an example main C/C++ function as explained in Section ‘Generate C/C++ code’ in Appendix.

The generated C/C++ main functions declare and initialize data, including dynamically allocated data. It invokes the entry-point functions and just returns the values. The main C/C++ functions are not a part of the package created in Section ‘Finish workflow’ in Appendix. So, these functions need to be copied outside the build folder for reference. The package of the generated C-code should be uploaded to the ASML production code development environment.

Thus, the generated source and header files should be made available in the development environment for integration. Refer to [1] for using the internal tool to install the generated code into the ASML development environment. In this project, the
existing C implementation of the phase unwrap algorithm was replaced by the main C functions, which call the entry-point function generated by MATLAB Coder. Then, the results from the generated phase unwrap algorithm are returned to the application for further steps.

The code snippet shown in Figure 18 is an example of how the generated code is integrated into the development environment in our project. The manual C implementation of the phase unwrap algorithm is replaced by the code snippet shown in Figure 18. The input variables available in the manual C implementation are passed to the generated C-code. The input arguments needed for the generated function are defined and initialized. The codes to define and initialize the input variables are also available in the main C file. Then, the arguments are passed with the values and then the generated entry-point function is called. After the execution of the generated C-code for the phase unwrap algorithm, the updated results are sent back to the manual C implementation for further steps. The pictorial representation of this integration is shown in Figure 19. Figure 19 shows that the generated C-code is deployed into production code and the existing C implementation of the phase unwrap algorithm is altered such that it calls the generated code for the phase unwrap algorithm. The execution time of the generated C-code for the phase unwrap algorithm was verified against the existing handcrafted implementation. The generated code was around 42 percent slower in comparison for the existing implementation, which needs to be optimized. Section 6.1.5 explains the design considerations and performance optimization techniques that help to generate efficient C-code.

```c
Phase unwrap algorithm (const ROI_Struct ROI, sheared_wf_struct data){
    //define the input argument for the generated C code
    static struct0_T sheared_wf_data;
    struct2_T r0;
    //initialize the input arguments for the generated C code
    arginit struct0_T(&sheared_wf_data1);
    r0 = arginit struct2_T();
    //set the ROI and data1 values into the input arguments r0 and sheared_wf_data
    r0.size.r = ROI.size.r;
    r0.size.c = ROI.size.c;
    for (idx0 = 0; idx0 < i; idx0++) {
        sheared_wf_data1.sheared_wf.phase.data[idx0] = data1.phase[idx0];
        
    }
    //call the entry-point function
    wrapper_F07_PhaseUnwrap(&sheared_wf_data, &r0);
    //set the updated results in sheared_wf_data into the data
    for (idx0 = 0; idx0 < i; idx0++) {
        data1.phase[idx0] = sheared_wf_data1.sheared_wf.phase.data[idx0];
        
    }
}
```

Figure 18 - Integrate generated code in the production code
6.1.5 Design considerations before code generation

The engineer who wants to convert MATLAB code into efficient, standalone C or C++ code should consider the following aspects:

- **Data types**
  C and C++ are statically interpreted. So, the variables needed for the entry-point function should be declared and defined. To reduce the number of variables created during code generated and avoid the redundant copies of the variables, the trick is providing the output variables as IN/OUT variables in the MATLAB functions and reuse the variable properties.

  Using this technique for the phase unwrap m-code contributed in generating a C-code with a fewer number of variables, but the execution time did not improve. This practice might result in lesser execution time for huge MATLAB model.

- **Array sizing and Memory**
  Arrays and matrices are supported for code generation. Additionally, the variable-size array is also allowed. The developer can choose to generate code that uses static or dynamic memory allocation for these arrays. Dynamic memory allocation uses less memory at the expense of time taken to manage the memory. With static memory, the user can achieve lower execution time but the memory usage increases.

  For the phase unwrap algorithm, the array size is fixed as it is the size of the camera frame, thus static and dynamic memory allocation is possible for the phase unwrap algorithm. Dynamic memory allocation resulted in a higher execution time, whereas static memory allocation resulted in a lower execution time.

- **Execution time**
  The code must be fast enough for embedded applications as it is direct overhead, so its execution time must be minimized. To optimize the speed of
generated code, certain optimization techniques are possible. The techniques experimented in our project are as follows:

1. Static memory allocation
2. A suitable C compiler specific for your application
3. Disabling support for integer overflow and non-finites
4. Application specific code replacement libraries
5. Inline functions
6. Provide the output variables as IN/OUT variables

These techniques were selected for investigation from MATLAB Coder user manual [2] and based stakeholder’s inputs who are experienced with MATLAB Coder.

Technique 1 applies in situations where static memory allocation is possible. For example, within the PARIS domain, the camera frames are the most typical and largest sets of data. The array size is the size of the camera frames which is constant throughout the PARIS fit sequence. So, Technique 1 is suitable for the PARIS domain.

Technique 2 applies in situations where the default compiler supplied by MathWorks must be replaced. The default compiler is for Windows 32-bit platforms is not good for performance. The code will be compiled by the default Twinscan SW compiler in Linux, so this technique is not applicable in our project.

Technique 3 applies for situations if the application does not introduce an integer operation overflow or does not require supporting code for non-finite values.

Technique 4 applies for situations that requires to change the code that the generator produces for functions and operators to meet the application code requirements. For example, compliance with a standard, eliminate calls to memcpy or memset, processor specific, or eliminate math.h. Intel IPP/SSE for x86-64 Linux was tried as this technology performs matrix operations efficiently [7].

Technique 5 applies for situations that requires to replace a function call with the contents of the function to eliminate the function call overhead.

Technique 6 is a trick, which helps in generating fewer variables and avoids redundant copies. In some cases, this might also improve the performance.

Table 3 below shows the techniques which were evaluated in our project, the impact and tradeoffs with respect to the phase unwrap algorithm. For the phase unwrap algorithm, these techniques were used, and the code was generated. Then, the generated code was integrated into the production code to evaluate its performance.
Table 3 The impact and tradeoffs for the chosen options with respect to the phase unwrap algorithm

<table>
<thead>
<tr>
<th>Technique</th>
<th>Impact</th>
<th>tradeoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lower execution time compared to the dynamic generated C-code (nearly 28 percent faster in comparison with dynamic)</td>
<td>Might cause memory management issues depending upon the application</td>
</tr>
<tr>
<td>2</td>
<td>No evident improvements in the performance</td>
<td>Does not introduce any evident tradeoffs needs more investigation</td>
</tr>
<tr>
<td>3</td>
<td>Reduces the size of the generated code and no evident improvements in the performance</td>
<td>If the integer overflow or non-finites values occurs the generated code might not match the behavior of the original m-code</td>
</tr>
<tr>
<td>4</td>
<td>No evident improvements in the performance</td>
<td>Does not introduce any evident tradeoffs needs more investigation</td>
</tr>
<tr>
<td>5</td>
<td>No evident improvements in the performance.</td>
<td>Inlining decreases the code readability</td>
</tr>
<tr>
<td>6</td>
<td>Reduces the number of redundant variables and no evident improvements in the performance</td>
<td>Increases code readability</td>
</tr>
</tbody>
</table>

For the phase unwrap algorithm, the technique 1 that does static memory allocation was chosen since its execution time is the least deviating in comparison with the existing production code. This analysis is specific for the phase unwrap algorithm, these techniques might create a different impact when considering a different MATLAB model.
6.2 Architectural redesign

This section discusses how the proposed architectural redesign described in Section 5.3 is implemented. This redesign is generic since it can be followed for all the algorithms within the PARIS fit sequence. In our project, we implemented the redesign in the context of the phase unwrap algorithm. Figure 20 shows the flow of data for the phase unwrap algorithm in the existing architecture and the parameters required for the execution of the phase unwrap. These parameters are made available in the Fast Zernike Data (FZD) component. Figure 21 shows how the modified data flow diagram to incorporate the FZD component in the existing architecture.

![Figure 20 - Data Flow Diagram for the phase unwrap algorithm](image)

The parameters needed for the phase unwrap are available across both the preparation and measurement stage as explained in Section 5.2. For example, in Figure 21, STEP 1 sets a parameter ROI and STEP 2 sets the parameters Phase, DC, and contrast values. The parameter ROI is available during the preparation stage before the algorithm 1 execution and the parameters Phase, DC, and contrast values are available during the measurement stage after the algorithm 7 execution. When the phase unwrap algorithm in the PARIS fit sequence is invoked, the corresponding parameters are retrieved from the FZD component, represented as STEP 3 in Figure 21. After the execution of the phase unwrap algorithm, the result is set back to the FZD component. Thus, the results of phase unwrap algorithm are available for the next set of algorithms to be executed in the PARIS fit sequence.
Figure 21 - Data Flow Diagram of the proposed solution

Figure 22 shows the sequence diagram of the proposed architecture which is implemented. The module enclosed in the blue box is the FZD component, i.e., the data abstraction component introduced to achieve separation of concerns explained in Section 5.3. The modules enclosed in the green box consist of the implementation of the existing algorithms and the generated C-code for the phase unwrap algorithm. The modules enclosed in the orange box act as the controller which sets or gets the parameters needed for the FZ scan. The first two sequences highlighted in the yellow boxes show the essential parameters required for the phase unwrap algorithm is set to the FZD component. The sequence highlighted in grey show the data retrieval from the FZD component before the phase unwrap algorithm is executed. The last yellow sequence shows how the updated parameters are set back to the FZD component for the next set of algorithms to be executed in the PARIS fit sequence.

Figure 22 - Sequence diagram addressing separation of concerns for the PARIS fit sequence focused on the phase Unwrap algorithm
Figure 23 shows the actual implementation done in our project. The data abstraction component with a data module for the phase unwrap algorithm was implemented. The control component was modified such that it makes use of the data abstraction component to get and set the parameters of the phase unwrap algorithm. The hand-crafted implementation of the phase unwrap algorithm was replaced by the generated C-code of the phase unwrap algorithm.

Figure 21, Figure 22 and Figure 23 are only focused on the phase unwrap algorithm due to the project’s scope and timeline. As a future work, the separation of concerns can be applied to all the other algorithms in the PARIS fit sequence.
7. Verification and validation

In this Chapter, the techniques used to verify and validate the two separate implementations specified in Chapter 6 are discussed. In Section 7.1, we discuss the techniques used in our project to verify the m-code before code generation and the generated code from MATLAB Coder. In Section 7.2, we discuss the verification and validation techniques used in our project for the proposed architectural redesign. In our project, the verification and validation are focused on the functional equivalence and performance equivalence. The performance equivalence checks for the execution time, memory usage and memory leaks. The existing C implementation of the phase unwrap algorithm and existing architecture acts as a benchmark for verifying the generated C-code for the phase unwrap algorithm and the architectural redesign respectively. These verification and validation techniques are represented by the orange blocks in Figure 7, Figure 35, and Figure 36.

7.1 MATLAB Coder

7.1.1 Verification before code generation

The m-code is made Coder compatible as explained in Section 6.1.3. Before generating C or C++ code, it is a best practice to detect and fix run-time errors that are harder to diagnose in the generated code. The run time checks are related to array bound check, dimension check and memory integrity violations. A MEX file is needed for this verification.

A MEX file is a type of computer file that provides an interface between MATLAB and functions written in C or C++. It stands for "MATLAB executable". When compiled, MEX files are dynamically loaded and allow external functions to be invoked from MATLAB like built-in functions. The MEX file can be generated using MATLAB Coder.

In order to perform these run time checks, a test script is created. This test file calls the entry-point function with the parameters required for execution. In order to check for run time issues, the test file call to the entry-point function is replaced with a call to the MEX function. If MATLAB Coder App finds any issues during the MEX function execution, it shows corresponding warnings or error messages.

In addition to this, the MEX function can also be verified if it provides the same functionality as the original MATLAB entry-point function. This is performed by running the test using MATLAB code which calls the original MATLAB function and running the test using the generated Code which calls the MEX function. These results are compared for equivalence. For more information on this verification refer to Section ‘Verify MEX function’ in Appendix.

Following the verification of the MEX function, the C or C++ code can be generated using MATLAB Coder. The generated C-code is verified and validated against the existing C-code as explained in Section 7.1.2. The Coder compatible version of the m-code was verified with the functional team stakeholder. The functional stakeholder was able to read and maintain the m-code.

7.1.2 Performance of Generated code

The m-code of the phase unwrap algorithm is made Coder compatible and C-code is generated as described in Appendix (Generate C/C++ code). Then, the generated code needs to be integrated with the development environment as described in Section 6.1.4 The integrated C-code was checked for functional equivalence, i.e., check whether the generated C-code results obtained is equivalent with the m-code results.
Following this, the non-functional requirements were checked i.e., the execution time, memory leaks and memory usage.

The generated C-code for the phase unwrap algorithm is of two variations such as

- Dynamic memory allocation (dynamic)
- Static memory allocation (static)

The dynamic memory allocation threshold limit is decreased or increased in MATLAB Coder to achieve dynamic or static memory allocation respectively. Refer to Section 6.1.5 for more information on the alternative variations that were experimented.

In dynamic, the memory is allocated at runtime. The memory is allocated whenever the program, application, or variable demands with the required number of bytes. The important mechanism about this allocation once utilized allocated memory is no longer required for program, application or variable, which can be again available to other purposes. The memory manager could not reuse that memory without deallocating the no longer required memory.

In static, the memory is defined at compile time, but allocated at the start of the program and not changed in size anymore. All the memory which is required for programming, applications or variable in its lifecycle is allocated at beginning of execution. Static is suitable for applications where the size needs not to be flexible.

In PARIS algorithms, the camera frames are the most typical and largest sets of data that is used. The array size represents the size of the camera frames, which is constant throughout the PARIS fit sequence. The dynamic and static generated C-code is suitable for this investigation. The architectural redesign proposal discussed in Section 6.2 is not a part of this verification and validation. Table 4 shows the test cases used in Section 7.1.2.

**Table 4 Test cases used to evaluate the performance of the generated C-code**

<table>
<thead>
<tr>
<th>Test case</th>
<th>Existing C-code for the phase unwrap algorithm without architectural redesign is verified for execution time and memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case 2</td>
<td>Dynamic generated C-code for the phase unwrap algorithm without architectural redesign verified for execution time and memory</td>
</tr>
<tr>
<td>Test case 3</td>
<td>Static generated C-code for the phase unwrap algorithm without architectural redesign is verified for execution time and memory</td>
</tr>
</tbody>
</table>

Figure 24 shows the pictorial representation of these test cases. In these test cases, the modules enclosed in the violet boxes represent the generated C-code for the phase unwrap that is integrated into the Twinscan application. These test cases act like a unit test to evaluate the performance of the generated C-code against the manual C implementation of the phase unwrap algorithm excluding the architectural redesign.
Figure 24 - Test cases to evaluate the generated C-code for the phase unwrap algorithm

Functional equivalence

In the PARIS domain, the functional equivalence between the m-code and the existing C-code is validated, i.e., to check whether the existing C-code results are equivalent to the m-code results obtained by running in MATLAB. To perform this functional equivalence, the python test script available in the development environment was used. This test script calls the m-code of the phase unwrap algorithm created by the functional team and the C-code of the phase unwrap algorithm by the software team and the results are compared for equivalence.

As the implementation of the m-code and the existing C-code for the phase unwrap algorithm are different. The results are not compared for direct equivalence. Instead, the results are checked with a tolerance level for the mismatch.

In our project, the C-code is generated for the phase unwrap algorithm using MATLAB Coder as described in Section 6.1.3. The existing C-code is modified such that it calls the generated C-code as discussed in Section 6.1.4. The test script is modified such that it checks for functional equivalence, i.e., the results from the m-code and the generated C-code match.

For the test cases 2 and 3, few results from the generated C-code were not equivalent to the standard m-code results. These mismatching results were of two types such as

- Good accuracy – A minimal difference from the m-code results
- Poor accuracy – A $2\pi$ difference from the m-code results

These differences are due to the data type mismatch. In the generated C-code expects double inputs and the existing C-code the inputs datatype is float. To check whether the mismatch is due to the difference in data type, the existing C-code’s input data type was altered to double to check for functional equivalence. Then, the results from m-code and the generated C-code were equivalent to five significant digits. Both the test cases 2 and 3 were satisfying the functional equivalence.

For the reasons behind such differences refer to ‘Differences in behavior after compilation compiling MATLAB code’ in [2]. For further investigation, the float datatype is considered for fair comparison as the existing C implementation uses float datatype.
Performance equivalence

The non-functional requirements that are crucial for the Twinscan application are the execution time and the memory.

Execution time

Within ASML, the execution time is very crucial due to high throughput requirement, i.e., number of wafers exposed per hour. In the PARIS domain, the execution time limit for completing the PARIS fit sequence is 100ms.

The phase unwarp algorithm is part of the PARIS fit sequence and the execution time of this algorithm is recorded. The execution time is the time interval since the entry and the exit of the phase unwarp algorithm, which is recorded using the real-time tracing function available on the Twinscan application. The execution time for the existing phase unwarp algorithm implementation is considered as the desired execution time limit.

After integration of the test case 2 and 3 as explained in Section 6.1.4, the corresponding execution time is also recorded using the available real-time tracing function. Table 5 shows the execution time of the phase unwrap algorithm using the existing C-code, the dynamic generated C-code and the static generated C-code.

<table>
<thead>
<tr>
<th>Test cases</th>
<th>Execution time(milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case 1</td>
<td>7 (6.5 to 7)</td>
</tr>
<tr>
<td>Test case 2</td>
<td>10 (9.5 to 10.5)</td>
</tr>
<tr>
<td>Test case 3</td>
<td>8 (7 to 8)</td>
</tr>
</tbody>
</table>

Based on the results obtained, it is evident that the test case 2 and test case 3 is three milliseconds and one millisecond slower than the execution time of the existing code, respectively.

Memory

The memory is checked by enabling the valgrind, i.e., the tool used to check for memory leaks and memory usage. Valgrind is an instrumentation framework for building dynamic analysis tools. The valgrind memcheck was enabled to check for memory leaks and usage information. Memory leaks occur when a block of dynamically allocated memory is never deallocated when it is no longer used.

The valgrind detects the use of uninitialized bits, false or missing frees, and overlapping memory reads and writes. The valgrind logs were obtained by enabling the options “leak-check=full” and “track-origin=yes”, which helps in getting the details on definitely/possibly lost blocks and the origins of any uninitialized values respectively. The python test script used to test functional equivalence is executed for which the valgrind information is logged.

Based on the valgrind logs recorded, it was evident that there were no memory leaks introduced for the test cases 1, 2, and 3. Refer to Figure 25 for the sample leaks summary. From the leak summary obtained for the three test cases, there were no blocks which are definitely, indirectly or possibly lost, these are cases that indicate that the program is leaking memory. The still reachable and suppressed are ignored as they are trivial errors.
In addition to the leak summary, the total heap usage for the three variations was also logged and the corresponding values are shown in Table 6. These values, in general, indicate that pointers are not discarded without first freeing corresponding allocated memory. When the program exits, any allocated memory is freed automatically.

Within the PARIS domain, there is no specific requirement for the memory usage. Though, a major increase in memory usage needs to be prevented to maintain sufficient memory for other processes on the Twinscan application. The major concern is the risk of memory leakage. Both the test cases 2 and 3 have not introduced any memory leaks. Test case 3’s memory usage is around 12 percent less in comparison with the existing C-code, whereas for the test case 2 the allocated memory is around 14Mb (10Mb more than the existing code).

Table 6 Total heap usage for the test cases 1, 2, and 3

<table>
<thead>
<tr>
<th>Test case</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case 1</td>
<td>4852 allocs, 1562 frees, 4,735,786 bytes allocated</td>
</tr>
<tr>
<td>Test case 2</td>
<td>5059 allocs, 1769 frees, 14,852,538 bytes allocated</td>
</tr>
<tr>
<td>Test case 3</td>
<td>4868 allocs, 1578 frees, 4,161,382 bytes allocated</td>
</tr>
</tbody>
</table>

Based on these results, the test case 3 that locates memory statically is suitable as it fourteen percent more and twelve percent less in comparison with the existing code in terms of execution time and memory usage respectively.

Considering the results of Section 7.1.2, the static generated C-code is not very deviated from the existing C-code. So, it is feasible to replace the existing phase unwrap algorithm as further optimization techniques can be used. As a future work, more investigation on MATLAB Coder optimization techniques might help to achieve the existing C-code’s performance.
7.2 Architectural redesign

The proposed architectural redesign is implementation as explained in Section 6.2 and this solution is verified whether it introduces any bottleneck to the existing architecture. Though the proposed architectural redesign addresses the separation of concerns these improvements might come with the tradeoff between code modularity and execution time or memory, so this section helps to evaluate the architectural redesign performance. Table 7 shows the test cases involved in this evaluation. In order to perform this verification, the python script which calls the Fast Zernike (FZ) scan type that process the PARIS fit sequence.

Table 7 Test cases for evaluating the proposed architectural redesign

| Test case 4 | Dynamic generated C-code without the architectural redesign verified for execution time and memory |
| Test case 5 | Dynamic generated C-code with the architectural redesign is verified for execution time and memory |

The pictorial representation of the test cases 4 and 5 are shown in Figure 26 and Figure 27, respectively. These test cases are an integrated test that checks the logical action that is the Fast Zernike (FZ) scan, i.e., executes all the algorithms in PARIS fit sequence. In these test cases, the modules enclosed in the violet boxes represent the generated C-code for the phase unwrap that is integrated into the Twinscan application. In test case 5, the module enclosed in the yellow box represents the architectural redesign that is implemented. The scan type control 1 was altered such that it makes use of the phase unwrap data module.
7.2.1 Functional equivalence

In the PARIS domain, the functional equivalence was performed using the python test script available in the development environment. This test script executes the Fast Zernike (FZ) Scan, i.e., execute all the algorithms including the phase unwrap algorithm and verified for result equivalence. This test script was used to verify and validate the functional equivalence of the architectural redesign. The architectural redesign performed in the context of the phase unwrap algorithm passed for functional equivalence.

7.2.2 Performance equivalence

Execution time

In order to find the bottleneck introduced by the proposed architectural redesign, the tracing function available with the PARIS domain was used. The architectural redesign is focused on the phase unwrap algorithm, i.e., As soon as the phase unwrap algorithm is invoked it gets the parameters stored in the data component, calls the phase unwrap algorithm implementation, and sets back the updated parameters to the data component.

To evaluate the proposed architectural redesign, the trace function and the test script, which calls the logical action available in the PARIS domain is used. For this experiment, the test case 4 is considered as a baseline to find the additional bottleneck introduced by the architectural redesign. The test case 5 introduced an additional delay around 0.150 - 0.200 milliseconds to the test case 4, which is around 2 percentage increase in the execution time.

Memory

The architectural redesign implementation needs to be verified if it introduces any memory leaks. The valgrind memcheck is used as explained in Section 7.1.2. From the valgrind logs, it was evident that no memory leak was detected and Table 8 below shows the total heap usage recorded in the logs.
Table 8 Total heap usage for the test cases 4 and 5

<table>
<thead>
<tr>
<th>Test cases</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case 4</td>
<td>5472 allocs, 2107 frees, 44,027,917 bytes allocated</td>
</tr>
<tr>
<td>Test case 5</td>
<td>5627 allocs, 2262 frees, 45,487,431 bytes allocated</td>
</tr>
</tbody>
</table>

Within the PARIS domain, there is no specific requirement for the memory usage. Though, a major increase in memory usage needs to be prevented to maintain sufficient memory for other processes on the Twinscan application. The major concern is the risk of memory leakage. The test case 5 has not introduced any memory leaks and the memory usage is three percent more in comparison with the test case 4.

Considering the results of Section 7.2, the proposed architectural redesign introduces a two percent time delay and the memory usage is three percent more than the test case 4. So, for the phase unwrap algorithm this solution can be adopted to attain the separation of the concerns’ advantages explained in Section 4.2. As a future work, a more extensive study should be performed on the proposed architectural redesign to verify and validate the performance and memory usage for the new algorithms that are to be added to the PARIS domain.
8. Conclusion and future work

In this chapter, the concluding remarks about this project is given in the following section. Section 8.1 gives the conclusion and the future work regarding MATLAB Coder. Section 8.2 gives the conclusion and future work related to the architectural redesign.

8.1 MATLAB Coder

In our project, the sub-goals related to MATLAB Coder are selecting a candidate MATLAB algorithm for investigation, providing insights on the adaptations required in the MATLAB algorithm to enable code generation using MATLAB Coder, and evaluating the performance of the generated C-code. I conducted this study using the phase unwrap algorithm within the PARIS domain.

- **Candidate selection:** Among the several PARIS MATLAB algorithms available for investigation. The phase unwrap algorithm was selected based on certain criteria such as complexity, durative nature and imparity between the MATLAB code (m-code) and the existing C-code. Thus, this selection is expected to be representative for most of the other algorithms within the PARIS domain.

- **Coder compatibility:** The phase unwrap algorithm m-code was made Cod- er compatible to enable the code generation using MATLAB Coder. The phase unwrap algorithm did not have many unsupported functions or features which need changes. These changes were mostly related to defining the input properties and initializing undefined variables. The core functionality of the phase unwrap algorithm was modified such that the functional team stakeholder is still able to read and maintain the m-code. Several candidate algorithms were selected as a risk mitigation plan and also checked for Coder compatibility. This investigation showed that the PARIS domain is suitable for code generation because these algorithms did not possess any unsupported functions or features that are unmitigable.

- **Performance evaluation:** The generated C-code was evaluated as these are time critical being part of lot production. There were no memory leaks introduced by the generated C-code. Regarding memory usage, there was no remarkable deviation that need to be addressed. When considering the execution time, the generated C-code for the phase unwrap algorithm with static memory allocation (eight milliseconds) was almost equivalent to the existing C-implementation of the same algorithm (seven milliseconds). The optimization techniques were experimented to attain this execution time. These results provide confidence to encourage MATLAB Coder usage for the PARIS domain. MATLAB Coder supports C-code generation from m-code that can be embedded in both time critical and less critical software. The generated code may require additional optimization techniques to suit the needs of time critical software.

8.1.1 Future work for MATLAB Coder

Based on our project’s results, MATLAB Coder has gained confidence to be a suitable candidate to generate code for time critical ASML machines. As a future work, the code generation can also be encouraged for the new MATLAB algorithms that will be introduced in the PARIS domain rather than manual implementation. In addition to this, the MATLAB Coder usage can be encouraged for production code algorithms in time critical and less time critical software within various domains. In case of time critical software performance optimization should be addressed.
8.2 Architectural redesign

In our project, the sub-goals related to the architectural redesign are providing insights on the adaptations needed to the existing architecture to minimize the dependencies and evaluating the performance of the generated C-code along with the proposed architectural redesign. I conducted this study using the Fast Zernike (FZ) scan within the PARIS domain.

- **Architectural redesign**: In the existing architecture, the algorithm is tightly coupled with the parameters essential for the algorithm (data) and the trigger for the algorithm (control). This dependency was minimized by introducing a data abstraction component that maintains the parameters required. The control module uses this data abstraction component to get the data before the algorithm execution (generated C-code for the algorithm) and to set the data after the algorithm execution. This pattern increases code readability, maintainability, and cohesion. The FZ scan was used for investigation as it was the most frequently requested scan with the performance constraint within the PARIS domain. Thus, the recurring use case was verified and validated.

- **Performance evaluation**: The proposed architectural redesign was evaluated as these are time critical being part of lot production. There were no memory leaks introduced by the proposed architectural redesign. Regarding memory usage, there were no remarkable deviations that need to be addressed. The proposed architectural redesign introduced an additional two percent time delay due to the “get and set functions” used for accessing the algorithm parameters. For phase unwrap, this two percent delay is acceptable in consideration with the advantages.

8.2.1 Future work for the architectural redesign

The proposed architectural redesign can be investigated further as future work within ASML. The proposed architectural redesign should be evaluated for different algorithms whether the introduced delay is acceptable. This delay might be proportional to the data size. An approach to evaluate this design is to apply the proposed architecture for the new MATLAB algorithms to be introduced within the PARIS domain. Thus, the proposed architectural redesign is evaluated for data with different properties. ■
9. Project management

In this chapter, the project management process that was followed during the project is explained.

9.1 Project planning

The project plan was created during the start of the project that roughly gives an overview of the tasks involved. During a particular phase, the project plan was refined with the detailed information specific to that phase. Also, the plan was flexible for adaptations based on the various priorities. The project plan followed a V-model and the activities, and the planned week numbers are shown in Table 9.

Table 9 V-model activities and the weeks spent

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
<th>Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquaintance phase</td>
<td>To know the domain and the source of information</td>
<td>1-3</td>
</tr>
<tr>
<td>Domain and feasibility analysis</td>
<td>Analyze the ASML domain and evaluate usage of MATLAB Coder</td>
<td>3-6</td>
</tr>
<tr>
<td>User requirements and system requirements</td>
<td>Gather the user requirements from various meetings and translate them to system requirements</td>
<td>6-12</td>
</tr>
<tr>
<td>Implementation phase</td>
<td>Generate C-code for the selected candidate algorithm and implement proposed architectural redesign</td>
<td>12-24</td>
</tr>
<tr>
<td>Verification and validation phase</td>
<td>Verify and validate the implementation against system requirements</td>
<td>24-29</td>
</tr>
<tr>
<td>Performance optimization phase</td>
<td>The performance optimization techniques for the code generation were experimented</td>
<td>29-33</td>
</tr>
</tbody>
</table>

In order to check, control and regulate the project there were few tracking mechanisms carried on such as

- **Project Steering Group (PSG) meeting:** A monthly meeting that involves the PDEng trainee, the company supervisor, the company mentor and the TU/e supervisor was held. This meeting was used to report the progress, to indicate foreseen risks, and to ensure all the stakeholders’ expectations were aligned.

- **Performance Evaluation:** Every three months the performance of the trainee was evaluated on various categories by the supervisors and feedbacks were provided. These feedbacks helped the trainee to steer the project in a better direction.

- **Weekly meeting:** The weekly meetings with the supervisor was used to discuss the projects progress, the design decision, and the forthcoming risks. The supervisor and trainee made use of these meetings to monitor and align the project with the organization’s goal.

9.2 Risk Analysis

This section indicates the major risks foreseen in this project. Table 10 describes the foreseen risks, the impact caused, and the respective mitigation plan. The risks were identified throughout the project and tackled with these mitigation plans.
Table 10 Project's risks

<table>
<thead>
<tr>
<th>Risk ID</th>
<th>Description</th>
<th>Impact</th>
<th>Mitigation plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>R01</td>
<td>The scope of the project is too broad, such as investigating many algorithms.</td>
<td>Might cause a delay in identifying the project’s requirements.</td>
<td>Discuss with stakeholders to prioritize needs and reduce the scope.</td>
</tr>
<tr>
<td>R02</td>
<td>Absence of a stakeholder</td>
<td>Might impact the requirement gathering and reviewing phase</td>
<td>Planning based on their schedule or planning alternative tasks with does not require guidance or alternative stakeholder with the domain knowledge</td>
</tr>
<tr>
<td>R03</td>
<td>Selected algorithm candidate is not suitable for code generation</td>
<td>Might cause delays in implementation phase</td>
<td>Decide upon several alternative algorithms candidates during the requirement phase.</td>
</tr>
<tr>
<td>R04</td>
<td>Generated code does not meet the performance requirement</td>
<td>Might cause delays in the implementation phase</td>
<td>Communicate about the delay with the project manager and schedule meeting with available MATLAB Coder experts.</td>
</tr>
<tr>
<td>R05</td>
<td>Conflict of interest between functional and software team</td>
<td>Might result in larger project scope</td>
<td>Address the issue to project mentor and ask for his priority or try to reach consensus between the teams.</td>
</tr>
<tr>
<td>R06</td>
<td>Architecture improvements was not achieved by June</td>
<td>Might impact the performance evaluation phase.</td>
<td>Prioritize the most required architectural changes and address them. Scope down the performance evaluation phase to address major concerns.</td>
</tr>
<tr>
<td>R07</td>
<td>Selected algorithm candidate requires adaptation to make it Coder compatible</td>
<td>It might take time to adapt m-code compatible with MATLAB Coder</td>
<td>Communicate about the delay with the project manager and schedule meeting with available MATLAB Coder experts.</td>
</tr>
<tr>
<td>R08</td>
<td>Biased towards MATLAB Coder investigation</td>
<td>Project might only include MATLAB Coder related issues and failed to capture architecture improvements needed</td>
<td>Communicate with project manager and terminate the investigation on MATLAB Coder, thus, there is sufficient time for architecture improvements.</td>
</tr>
<tr>
<td>R09</td>
<td>Not all the requirements specified are met in the project</td>
<td>Might affect the project’s and stakeholder’s expectations</td>
<td>Prioritize the requirements with the main stakeholder and update the stakeholder with the progress. Based on these, get agreement over the unmet requirements.</td>
</tr>
<tr>
<td>R10</td>
<td>Stakeholder leaves the position</td>
<td>Might affect the project’s progress</td>
<td>Arrange knowledge transfer session as early as possible and check for an alternative employee with similar experience.</td>
</tr>
</tbody>
</table>
10. Project retrospective

In this chapter, the personal reflection on the project and the design opportunities chosen are discussed.

10.1 Personal reflection

During this project, there were several challenges encountered. One of the challenges was using MATLAB Coder for this investigation. I had experience with MATLAB during masters that came in handy for this project. However, MATLAB Coder was a new tool that was learned during this project. In our project, the deployment code for the algorithms within the PARIS domain was generated using MATLAB Coder. Another challenge was to propose an architectural solution to reduce the external dependencies of the algorithm, i.e., the algorithm invocation, and the algorithm parameter actions. Followed by the challenge of evaluating the functional and performance equivalence for generated C-code and the proposed architectural redesign. Also, several performance optimization techniques were explored.

During the initial phase of the project, I selected an algorithm with limited complexity to gain acquaintance to MATLAB Coder, and I altered it to make it compatible for code generation. This helped in speeding the learning curve related to MATLAB Coder. In addition to this, I integrated and tested the generated C-code into the Twin-scan application. This helped to get familiar with the ASML tool-set that has a very steep learning curve. These actions also acted as a feasibility analysis to check whether MATLAB Coder is suitable for this investigation. Feasibility analysis helped to accelerate the process of generating and integrating the C-code for the selected candidate algorithm.

Though there was some documentation to explain the PARIS domain and the algorithms used in the PARIS domain, there were no documents available to explain the architecture of the existing production code. The challenge was to understand the existing architecture from the code base. Following this, the architectural redesign was proposed, and the design alternatives were discussed.

During the verification and validation phase, the project’s progress went a bit down due to certain infrastructure issues. This risk was not predicted earlier but I mitigated this risk by finding alternative tasks. As the performance of the generated code was not in line with the existing implementation further time was dedicated on performance optimization techniques.

Reviewing the MATLAB changes and the architecture proposal was a great experience in managing people expectations. Apart from the technical work related to the project, I also acted as a consultant between the functional and the software team on various scenarios in this project.

This project was a valuable experience in process management. Apart from being the project designer, I was also expected to be a project manager. This project has improved both my technical and organizational skills. Organizational skills such as planning the project, communicating with stakeholders, clarifying information, and taking initiatives were exercised.

To summarize, the project was a great experience and opportunity to implement and extend the technical and organizational skills. The learnings from this project are helpful for my professional life.
10.2 Design criteria

Structuring the existing code to support the embedding of the generated C-code was the most important design opportunity in this project. Separation of concerns was identified as the most important requirement for the architecture and design. The external dependencies related to the generated algorithm, i.e., the algorithm invocation and algorithm parameters needed were separated by introducing a data abstraction component. The data abstraction component is responsible for maintaining the parameters essential for the algorithm. This architectural redesign helps to increase code readability, maintainability, and cohesion. This was also verified by a stakeholder via an informal meeting during the validation phase of the project.
Appendix

**MATLAB Coder user manual**

MATLAB Coder is used to generate C/C++ code from m-code and it supports most of the MATLAB language and a wide range of toolboxes. However, there are certain unsupported functions and limitations which should be considered while using MATLAB Coder. In this investigation, MATLAB version R2016b and R2018a were used. Initially, the m-code is made MATLAB Coder compatible to generate C-code from them. The several steps to make an m-code compatible and generate C-code using MATLAB Coder are discussed below.

![MATLAB Coder App](image)

**Open MATLAB Coder App**

First, the user clicks on MATLAB Coder App icon on the MATLAB Toolstrip ‘Apps’ under code generation. MATLAB Coder App opens as shown in Figure 28.

**Specify source files**

Next, the user opens the existing m-code in MATLAB Coder App that requires code generation as shown in Figure 28. After the selection of the entry-point functions, MATLAB Coder App analyzes these functions for code generation readiness and coding issues. If the App identifies issues, the Review code generation readiness page, as shown in Figure 29, opens automatically.
Code readiness and define input types

In this step, the m-code is checked for usage of any unsupported MATLAB features and functions that hinder code generation. Additionally, it also checks for code violations at design time to minimize compilation errors. The code is reviewed continuously as the user types, by reporting problems and recommending modifications. An indicator is provided in the top right of MATLAB Coder App as shown in Figure 29. This indicator is either green, red or orange, which represents the code’s readiness state as follows:

- **Green** - No code generation issues detected
- **Red** - Errors are detected
- **Orange** - Warmings are detected

When the indicator is red or orange, similar colored markers appear in the scroll bar where the error occurs. The user can view the error information by placing the pointer over these markers as shown in Figure 29. For detailed explanation or suggested mitigations, the user can click the details tab.

The errors shown are also related to specifying the input properties of all variables in the MATLAB files. As MATLAB is a dynamically interpreted language, it determines the variable properties in runtime. In contrast, C and C++ are statically interpreted languages, at compile time, MATLAB Coder should determine the properties of all variables in the m-code. MATLAB Coder should infer the properties of entry-point function inputs. There are several options to define the inputs for MATLAB Coder.

In this project, the inputs are defined programmatically in the m-code. The MATLAB assert function was used to define the properties of the entry function inputs. Figure 30 shows how the input variables are defined programmatically to facilitate code generation. After defining the input types, the user can click and select ‘Determine input types from code preconditions’ which instructs MATLAB Coder to make use of the assert statements.
After all the issues related to Coder compatibility and code violations are handled, the indicator’s color changes to green. Then, MATLAB Coder App opens the check for run-time issues as shown in Figure 31.

Check for run-time issues
A MEX function for the MATLAB function is created in this step. A MEX file is a type of computer file that provides an interface between MATLAB and functions written in C or C++. It stands for "MATLAB executable". When compiled, MEX files are dynamically loaded and allow external functions to be invoked from MATLAB like built-in functions.

In order to perform the run-time issue check, MATLAB Coder App generates a MEX function for the entry-point function and runs the MEX function with the test file as shown in Figure 31. The test file calls the entry-point function with the parameters required for execution. In order to check for run time issues, the test file call to the entry-point function is replaced with call to the MEX function. This checks for issues related to arrays bound, dimension, and detect memory integrity violations in the generated code. If MATLAB Coder App finds any issues during the MEX function generation or execution, it shows corresponding warnings or 1messages. Then,
MATLAB Coder app moves on to the code generation phase if all the issues are addressed.

**Verify MEX function**

Before generating C/C++ code for the MATLAB code, the MEX function can be verified if it provides the same functionality as the original MATLAB entry-point function. The test file which calls the original entry function with the parameters is provided for this verification. The steps involved in MEX verification are as follows. Refer to Figure 32 in the following steps

- Choose build type to MEX in Generate for this verification step.
- Click verify code and select the test file for verification.
- Select run test using MATLAB code which calls the original MATLAB function.
- Select run test using generated code which calls the MEX function.
- Compare results from testing on the MATLAB function and the MEX function.

![Figure 32 - Verify code](image)

**Generate C/C++ code**

After the verification of the MEX function, the C and C++ code can be generated from MATLAB Coder. The user can open the generate dialog box by clicking the shown in Figure 32. The user can select the build type as source code and language to C (specific for the project) as shown in Figure 33. Then the user can click generate, which by default also generates an example C main function. This function is a template that helps to integrate generated C-code into user’s development environment. These optimizations can be performed making used of the “more settings” in Figure 33. The different settings related to speed, memory, customizing code, code replacement library can be explored.
Finish workflow

The finish workflow page indicates the code generation succeeded as shown in Figure 34. The user can click package to save the generated C-code as a zip file in the same folder as the MATLAB. This package zip file is used for relocating to another development environment in this case ASML Twinscan application.
## Stakeholder Interest and Concerns

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Interest</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TU/e mentor</td>
<td>Successful completion of OOTI project.</td>
<td>OOTI project matches the specification of completion.</td>
</tr>
<tr>
<td>Technical mentor</td>
<td>Exploring MATLAB Coder usability for PARIS domain and address the architecture improvement for the algorithm deployment.</td>
<td>OOTI project matches the specification of completion and the results from the project act as a baseline/reference for future improvements in PARIS domain.</td>
</tr>
<tr>
<td>Management mentor/ MATLAB Embedding PL</td>
<td>Feasibility study which might help decrease the software development lead time. Experience gained with converting existing MATLAB algorithms to Coder compatible.</td>
<td>The study on the practicality is exported to the MATES team.</td>
</tr>
<tr>
<td>Functional engineer</td>
<td>One source code solution which reduces development work and extra effort spent on making more efficient algorithm.</td>
<td>MATLAB Coder restrictions might have impact on the way of working for the functional team. Converting MATLAB algorithm to Coder compatible might require additional task. Functional team might require alternative version of an algorithm for future improvements. Dealing with these varying algorithm versions efficiently.</td>
</tr>
<tr>
<td>Software lead engineer</td>
<td>PARIS MATLAB calculation algorithm could remain as a black box to the SW developers. Thus, the software development team can concentrate on other crucial improvements needed within PARIS domain.</td>
<td>Generated C-code for PARIS MATLAB algorithm meets the current performance requirement.</td>
</tr>
<tr>
<td>Software FC architect</td>
<td>PARIS MATLAB calculation algorithm could remain as a black box to the SW developers. Thus, more effort towards software architecture improvements.</td>
<td>Generated C-code for PARIS MATLAB algorithm meets the current performance requirement.</td>
</tr>
<tr>
<td>MATLAB Embedding team</td>
<td>OOTI project is focused on using MATLAB Coder in production code (PARIS). Thus, able to provide insights to MATLAB Embedding team. Such as feedback on the supporting tools created, address the performance issues and design decisions made for implementation.</td>
<td>Generated C-code for PARIS MATLAB algorithm meets the current performance requirement. So, encourage more teams for this transition.</td>
</tr>
</tbody>
</table>
**Way of working**

Based on the experience gained from the project, Figure 35 and Figure 36 were developed which depicts the way of working. Figure 35 shows the steps to be taken by an ASML engineer to generate the C-code from a MATLAB algorithm. After the management decides upon the algorithm, the engineer checks for feasibility and starts with the migration if feasible. Then, the algorithm is adapted in MATLAB Coder and checked for run-time issues. When there is no run time issues C-code can be generated for the algorithm. The generated C-code is integrated into the Twinscan application and then checked for functional equivalence against the MATLAB model or the existing C implementation of the algorithm. Following this, the performance of the generated code is evaluated against the existing C implementation and if the results are sufficient (to be verified and validated with stakeholders). Then, the logical action is verified, i.e., a high-level check that includes the generated C-code as a part of the sequence. Any functionality mismatch or additional delays introduces additional work for the engineer in terms of debugging and adapting m-code for performance optimization. The engineer can proceed to replace the existing C implementation with the generated C-code after its performance is acceptable and approved by the management. The activities in red indicate that these steps require manual effort related to implementation for more information refer to Section 6.1. The activities in orange indicate the checks performed to verify and validate the modified m-code and generated C-code. For more information refer to Section 7.1.

Figure 36 shows the steps to be taken by the ASML software engineer to follow the proposed architectural redesign. Following the management’s decision to migrate to the new architecture, either all the algorithms or part of the algorithms or one algorithm can be chosen for migration. Then, the data needs to be separated from the control and data module needs to be created for the number of algorithms undergoing the architectural redesign. After the functional equivalence verification against the existing C implementation the logical action is verified, i.e., a high-level check that includes the algorithms selected for architectural redesign as a part of the sequence. If the delay is acceptable and the functionality match is achieved, then architectural redesign can be practiced after management’s approval. In contrast, if the delay is unacceptable or functionality mismatch the management should be informed about this impractical situation. The activities in red indicate that these steps require manual effort related to implementation for more information refer to Section 6.2. The activities in orange indicate the checks performed to verify and validate the proposed architectural redesign. For more information refer to Section 7.2.
Figure 35 - Steps involved in using MATLAB Coder for ASML environment
Management decision to migrate to new architecture

Separate Data from control

Algorithm 1-NDData module

Functional equivalence (unit test for the specific algorithm)

Against the existing C implementation

Logical action check (high level check to evaluate performance)

Replace the existing architecture with proposed architecture after approval

Inform about the impractical situation

Figure 36 - Steps involved in applying the proposed architectural redesign for ASML environment by software engineer
## Glossary

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-code</td>
<td>C is a computer programming language. C-code means that you can use C to create lists of instructions for a computer to follow</td>
</tr>
<tr>
<td>Coder compati-ble</td>
<td>The m-code is changed such that it can generate C-code using MATLAB Coder</td>
</tr>
<tr>
<td>DCA</td>
<td>DCA is a separation of concerns for Data, Control and Durative actions (management term: Algorithms)</td>
</tr>
<tr>
<td>FZ</td>
<td>Fast Zernike</td>
</tr>
<tr>
<td>FZD</td>
<td>Fast Zernike Data component</td>
</tr>
<tr>
<td>Lot</td>
<td>A set of wafers which is processed together in a lithographic machine from start to finish</td>
</tr>
<tr>
<td>MATLAB Coder</td>
<td>MATLAB Coder generates C and C++ code from MATLAB code for a variety of hardware platforms, from desktop systems to embedded hardware</td>
</tr>
<tr>
<td>m-code</td>
<td>The MATLAB application that is built around the MATLAB scripting language (.m file)</td>
</tr>
<tr>
<td>MEX</td>
<td>File type of MATLAB executable</td>
</tr>
<tr>
<td>PARIS</td>
<td>Parallel ILIAS</td>
</tr>
<tr>
<td>PARIS fit se-quence</td>
<td>A PARIS fit sequence consists of a set of algorithms that are executed sequentially to find the lens aberrations</td>
</tr>
<tr>
<td>PDEng</td>
<td>Professional Doctorate in Engineering</td>
</tr>
<tr>
<td>ROI</td>
<td>Region of Interest</td>
</tr>
<tr>
<td>Throughput</td>
<td>The number of wafers successfully printed with the pattern in an hour</td>
</tr>
<tr>
<td>TU/e</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>Twinscan</td>
<td>The photolithography machine that ASML is producing</td>
</tr>
<tr>
<td>Wafer stage</td>
<td>The sub-system in the photolithography machine that is responsible for positioning wafer during measure and expose stage</td>
</tr>
</tbody>
</table>
Bibliography


[2] MATLAB Coder user’s guide R2016b


About the Author

Laavanyaa Balasubramanian received her bachelor’s diploma in Electrical and Electronics Engineering (2009) from Anna University (India). She then worked for one and half years as a Program Analyst for Cognizant Technology Solutions. After reaching Netherlands she decided to pursue master’s degree in business information systems (2016) from Technical University Eindhoven. Her thesis project entitled “Hospital Workflow Data Mining for Workflow Analysis” was carried out with Philips research (Eindhoven) was related to data mining and analysis. A generic solution which could be used to verify whether the frequently visited spots or correlated spots are organized efficiently was developed. She is interested in software architecture and design.