Model-driven development methodology applied to real-time MEG signal pre-processing system design

Von der Fakultät für Ingenieurwissenschaften,

Abteilung Elektrotechnik und Informationstechnik

der Universität Duisburg-Essen

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

genehmigte Dissertation

von

Tao Chen

aus

Hebei, China

Gutachter: Prof. Dr.-Ing. Stefan van Waasen

Gutachter: Prof. Dr. N. J. Shah

Tag der mündlichen Prüfung: 16.11.2018

Acknowledgements

The presented research is supported by China Scholarship Council (CSC), in cooperation with the Central Institute of Engineering, Electronics and Analytics - Electronic Systems (ZEA-2) and the Institute of Neuroscience and Medicine (INM-4) at Forschungszentrum Jülich GmbH. The work and the thesis are completed with the help of many people. I would like to express my heartfelt gratitude to all of you who made my stay in Germany less lonely and more fruitful.

First of all, I would like to thank my doctoral advisor, a very nice, humorous and outstanding scientist, Prof. Dr.-Ing. Stefan van Waasen, for giving me the chance to write my PhD thesis in ZEA-2.

My special thankfulness goes to my supervisor, a serious looking but very nice scientist, Dr.-Ing. Sergey Suslov. Sergey, thank you very much for the great supervision in my work.

I further acknowledge my gratefulness to Dr. Michael Schiek who is always an optimistic, nice and knowledgeable leader. He is mighty and can solve all kinds of problems.

A special thank is dedicated to Dr. Jürgen Dammers for the opportunity to study in Germany and his kind help in my work.

I would also like to especially thank Prof. Dr. N. Jon Shah for giving me the chance to be a member of his institute.

Finally, I would like to thank my parents. You have always supported, encouraged and loved me for the many years. Thank you very much!

Abstract

The thesis is a multidisciplinary work that uses model-based systems engineering (MBSE) for developing real-time magnetoencephalography (MEG) signal processing.

In magnetoencephalography signal processing, biological artifacts in particular overtop the signal of interest by orders of magnitude and must be removed during signal processing from the measured signals to achieve a high quality reconstruction with minimized error contribution. This is a computational extremely demanding challenge as standard MEG systems include 248 and more channels in parallel. In this work, the automated real-time artifact rejection based on the recently presented method “ocular and cardiac artifact rejection for real-time analysis in MEG” (OCARTA) has been implemented by a MBSE approach and successfully verified on a Virtex-6 FPGA.

The requirement-driven, model-based development methodology (RDD & MBD) provides a high-level environment and efficiently handles the complexity of computation and control systems. The applied development methodology focuses on the use of Systems Modeling Language (SysML) to define high-level model-based design descriptions for later implementation in heterogeneous hardware/software systems. In order to demonstrate the capability of the proposed approach, it was applied and further developed to the implementation of a real-time artifact rejection unit in MEG signal processing.

The work of this thesis is embedded in the Jülich Research Center (Forschungszentrum Jülich GmbH, Germany) MEG-RT 2.0 project, which aims at developing an MEG real-time signal-processing device to be used as an add-on to existing MEG systems, thus enabling, for example, neuro-feedback applications. The system model of the real-time MEG signal processing chain developed here and the verified real-time artifact rejection implementation is a first and essential component of the Jülich Research Center MEG-RT 2.0 project.

Zusammenfassung

Diese Ausarbeitung adressiert als multidisziplinäre Arbeit die Anwendung von modellbasierter Systementwicklung (MBSE) für die Entwicklung von Echtzeit-Signalverarbeitung in der Magnetenzephalographie (MEG).

In der Magnetenzephalographie überragen biologische Artefakte die eigentlich interessierenden Signale um Größenordnungen und müssen daher zur Erreichung einer qualitativ hochwertigen Rekonstruktion mit minimierten Rekonstruktionsfehlern im Rahmen der Signalverarbeitung entfernt werden. Diese ist eine extrem rechenintensive Herausforderung, da übliche MEG System 248 und mehr Kanäle parallel aufzeichnen. In der vorliegenden Arbeit wurde eine automatisierte echtzeitige Artefaktunterdrückung, basierend auf einer vor einigen Jahren entwickelten Methode „ocular and cardiac artifact rejection for real-time analysis in MEG“ (OCARTA) auf einem Virtex-6 FPGA basierend auf einem MBSE Ansatz implementiert und erfolgreich verifiziert.

Die anforderungsgetriebene, modellbasierte Entwicklungsmethode (RDD & MBD) ist eine hocheffiziente Unterstützung für die Bearbeitung der Komplexität von Datenverarbeitungs- und Regelungssystemen. Die in dieser Arbeit angewandte Entwicklungsmethode fokussiert sich auf die Verwendung der Programmiersprache SysML (Systems Modeling Language) für die abstrakte, modellbasierte Design Beschreibung zur späteren Implementierung in heterogenen Hardware-/ Software-Systemen. Die besonderen Vorzüge dieser Methodik wurden beispielhaft an der Implementierung der echtzeitigen Artefaktunterdrückung in der MEG-Signalverarbeitung demonstriert und weiterentwickelt.

Diese Arbeit ist Teil des MEG-RT 2.0 Projektes im Forschungszentrum Jülich, welches die Entwicklung eines Echtzeit Systems zur MEG Signalverarbeitung als Zusatzkomponente zu bestehenden MEG Systemen zum Ziel hat, um z.B. Neuro-Feedback Anwendungen zu ermöglichen. Das innerhalb dieser Dissertation entwickelte Systemmodell der Echtzeit MEG Signalverarbeitung und die erfolgreich verifizierte Echtzeit Artefaktunterdrückung sind ein erster und essentieller Baustein für das MEG-RT 2.0 Projekt.

Table of Contents

1 Introduction

Along with the ever-increasing development of silicon IC technology over the last decades, increasingly progressive, complex and distributed systems have been enabled in which many different disciplines participate [1]. However, an interconnected **S**ystem-**o**f-**S**ystems (SoS) is far more than the sum of its subsystems [2]. The interconnectivity and communication among all the subsystems can be extremely complex and hard to control. **S**ystems **E**ngineering (SE) is an approach that effectively handles the complexity by raising the abstraction level in the design.

Systems engineering methods visualize a system and relevant parts at various levels of abstraction, offering deeper understanding and insight to a developer. The approach allows changing from technical to strategical development decisions as the cost due to principal mistakes increases with the complexity [3]. Over the years, systems engineering has continued to evolve and highly developed modeling tools and methods are generated to deal with the increasing complexity of today's heterogeneous systems. **M**odel-**B**ased **S**ystems **E**ngineering (MBSE) generates the advantages of low cost, time saving and accuracy compared with traditional methodologies in embedded system design.

We proposed a **R**equirement-**D**riven **M**odel-**B**ased **D**evelopment methodology (RDD & MBD) as a further development of comprehensive methodology for functional design in digital systems [3]. It has been elaborated in the Central Institute of Engineering, Electronics and Analytics (ZEA-2) to address the problems of heterogeneous computation and control system design using SysML [4]. The methodology is exploited in the design and implementation of a real-time signal pre-processing system for MEG. This work is part of the development of the MEG-2.0 Real-Time project (MEG-RT 2.0) at Research Center Jülich aiming on the development of a MEG real-time signal processing device as an add-on to the 248-channel whole-head magnetometer MEG system (MAGNES_3600WH) from 4D-Neuroimaging located at the Research Center in Jülich (Forschungszentrum Jülich GmbH), Germany. Applying system modeling to our design allowed a comprehensive and thorough evaluation of actual design requirements and forward planning of possible technical solutions to find an optimum solution in advance.

MEG is a non-invasive neuroimaging technique that investigates brain activities with high temporal resolution [5]. In modern MEG systems, an array of hundreds of low temperature coefficient (low-TC) SQUID sensors is used to record components of the magnetic field produced by post-synaptic neuronal activities. The magnitude of the magnetic field components is in the range of a few tens to hundreds of femto Tesla while some noise and biological artifacts, such as ocular and cardiac activity, can be ten to hundred times larger [6]. These artifacts must be removed from the measured signals as they can lead to source reconstruction errors [6]. However, artifact rejection is computationally extremely demanding [7].

In MEG studies, real-time MEG data analysis has become a topic of high interest recently and requires automatized real-time artifact removal [7],[8],[9]. One of the first real-time MEG source analyses was

demonstrated in [10], but only including cardiac artifact rejection and capable for real-time processing of a reduced number of channels. In other studies, real-time MEG analysis is limited to certain frequency bands, e.g., μ (9-12 Hz) [11], μ (9-15 Hz) and β (18- 30 Hz) [12] or α (8-13 Hz) [8].

In this thesis, the requirement-driven model-based development methodology was applied for developing a **S**ystem-**o**n-**C**hip (SoC) performing real-time artifact rejection based on OCARTA (ocular and cardiac artifact rejection for real-time analysis in MEG) [7]. A detailed description of the method can be found in [9]. Based on the performed system modeling and analysis, the target design will be developed and implemented on a single workstation platform combining field programmable gate array (FPGA) and graphics processing unit (GPU) co-processing boards containing a software framework for real-time MEG data analysis.

The thesis is organized as following: chapter 2 gives an introduction to MEG signal processing background and the basic concepts of the algorithms used in MEG artifact removal. Often recordings of MEG data contain not only brain activity, but also field components stemming from ocular movements, cardiac activities, muscle activity and environmental noise [9]. Biological artifacts in particular overtop the signal of interest by orders of magnitude and must be removed from the measured signals to avoid reconstruction errors. One approach for artifact rejection is OCARTA [7] which is the base method in this thesis.

Chapter 3 firstly introduces MBSE basics and existing MBSE methodologies. With the growth of system complexity, the design methodology becomes more and more important. Over the years systems engineering has continued to evolve and highly developed modeling tools and methods are generated to deal with the increasing complexity of today's heterogeneous systems. The chapter introduces the key aspects in MBSE and presents some existing MBSE methodologies that are widely used. Finally, the chapter discusses the technological trends in modern digital systems and gives an overview of the major modern computing technologies.

In chapter 4 the details of the RDD & MBD method are provided. It highlights the major points of the approach and outlines the corresponding workflow. Additionally, a small part of the chapter is devoted to a brief description of general FPGA design method.

Chapter 5 covers the demonstration of the proposed solution. The RDD & MBD methodology is exploited in the system design of a real-time signal pre-processing system for magnetoencephalography (MEG-RT 2.0), which is a detailed specification on how the employed language constructs can be adequately transferred into a procedural implementation. System modeling is laid out in compliance with the complete chain of top-down design and depicts gradual specification of the application at different levels of abstraction. General system requirements and constraints of stakeholders are first specified using package diagrams and requirement diagrams. System structure and behavioral features are modeled by SysML block and behavioral diagrams. Finally, performance is analyzed using parametric diagrams.

The realization of MEG artifact removal on a precise FPGA platform is presented in Chapter 6. It starts at the functional analysis of the method and concludes with the results of its implementation on the target FPGA platform. At the same time, Chapter 6 focuses solely on the latter steps of a top-down design process of SW development - the partitioning of an application to a target computing architecture. The peculiarities of the MEG artifact removal implementation on the given FPGA platform are depicted, which determine the implementation process. Although the system level studies of the MEG artifact removal are represented in the hardware-related chapter, the technology independent results of analysis at the higher levels of abstraction given in Chapter 5 are useful for evaluating the method's potentials for implementation on the FPGA platform.

Chapter 7 includes the experiment results and analysis. The findings acquired through the MEG artifact removal implementation are compared with previous studies. The comparison and analysis was carried out with respect to computing performance, scalability, implementation efforts, and functional modification capability. The chapter is concluded with a detailed reflection on the actual contributions of this work, an assessment related to the attainment of the previously formulated goals.

Chapter 8 presents a short compendium of the whole project and suggests some possible optimization methods for real-time MEG signal processing. Additionally, it discusses briefly some further interesting topics as possible subjects for future studies in this field.

2 Real-time MEG Signal Processing

Magnetoencephalography (MEG) is a non-invasive neuroimaging technique that directly measures the electrophysiological processes of the living human brain. During measurements an array of sensors is used to detect magnetic signals caused by neuronal activities. With MEG the spatial-temporal dynamics can be investigated at very high temporal and good to moderate spatial resolution depending on the depth of the active source [13]. MEG based real-time brain–computer interfaces (BCI) have been developed to benefit neuroscience research and clinical diagnosis. However, many existing brain–computer interface (BCI) studies overlook real-time artifact removal because of the demanding computational process [7]. In the following section there will be an overview of MEG background knowledge and state-of-the-art developments, followed by a detailed description of MEG signal pre-processing workflow.

2.1 MEG introduction

Many processes in the living human body, including neural activities, muscular movement, etc., are accompanied by weak variations of electrical current or potential, which is caused by changes of ion concentrations in intra- and extracellular compartments. The biological electrical activities produce electromagnetic fields that can be measured using highly sensitive devices placed outside the body [9]. For example, the small electric current flows in the human brain generate very weak magnetic fields, which can be measured outside the head. MEG deals with the detection and analysis of such tiny magnetic fields produced by the brain [9].

MEG is an important and advanced tool for the research of underlying oscillations in neuronal activity, with the potential for application across a range of brain disorders [14]. The aim of MEG is to study the dynamics of brain sources by localizing the weak magnetic fields generated by neuro-electrical brain activities and characterizing brain networks. As the magnetic fields of brain responses are extremely weak, the sensors used in MEG equipment must be exceptionally sensitive. The superconducting quantum interference device (SQUID) proposed in [15] is an exceedingly sensitive magnetometer for extremely subtle magnetic field detection. In order to reveal patterns of magnetic field distribution on the surface of the skull, a whole-head MEG system is required. The channel density of contemporary is very high with hundreds of data channels covering the whole-head [9]. It provides a great potential for both scientific research and clinical applications. MEG is now the optimal technique in magnetic source imaging (MSI) with a precise and high degree of localization. It is widely acknowledged by many neurologists in clinical practice. A detailed description can be found in [16],[17].

MEG has the properties of high temporal and spatial resolution, as shown in Fig. 2.1. Invasive EEG (iEEG) technology enjoys similar temporal and spatial resolution as MEG, but iEEG has the demerit of being invasive [18]. Other functional modalities are of either poor temporal or spatial resolution. In

the Institute of Neuroscience and Medicine at the Forschungszentrum Jülich GmbH, the 4D-Neuroimaging whole-head magnetometer system with 248-channels (MAGNESTM-3600WH MEG) is installed.

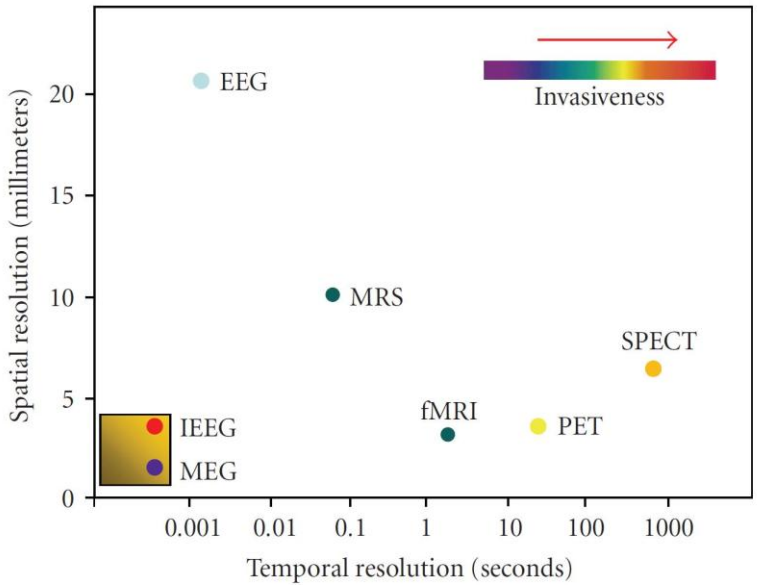

Figure 2.1: Comparison of spatial and temporal resolutions of neuroimaging techniques (from [18])

2.2 Previous real-time MEG signal processing studies

MEG can reconstruct activity of different brain regions even deep below the scalp with high temporal and spatial resolution. Therefore, it is the tool of choice for neuronal feedback experiments or brain–computer interfaces. For this, the challenging task is to derive the information in real-time which highly demands an automated artifact rejection as one signal pre-processing step. MEG based BCI have been developed recently to facilitate advanced and promising methods for neuroscience research. There have been studies [8],[11],[12],[19] using neuro-feedback received from MEG to develop BCI, which offers a new perspective to the task of artifact removal. Consequently, neuro-feedback is realized with tiny delays of around 44 (±17) ms between MEG signal measurement [8] and acquirng a feedback of the brain activities in sensor-space. However, in these studies real-time MEG analysis is limited to certain frequency bands, e.g., μ (9-12 Hz) [11], μ (9-15 Hz) and β (18- 30 Hz) [12] or α (8-13 Hz) [8].

Lately, it has been revealed in [20] that subjects can effectively be trained to adjust components of oscillatory neural activity within intended brain regions using MEG neuro-feedback stimulation. Accordingly, real-time source localization is realized, but limited to a region-of-interest (ROI). Comparable results are reported in [21], subjects are trained to improve coherence between two brain areas with the help of neuro-feedback stimulations. Note that MEG data acquisition and real-time filtering have already been studied in [8],[10],[20],[21].

As a summary, real-time MEG is useful and important in many fields. However, there exists some real-time problems to be tackled: data acquisition and filtering, artifact rejection, source localization and neuro-feedback. In the present work, the artifact suppression in real-time is the study focus.

2.3 MEG data pre-processing workflow

MEG data processing usually requires filtering, artefact rejection, source localization and connectivity analysis, as shown in Fig. 2.2. The analysis of recordings from modern multi-channel MEG system is computationally extremely demanding. Processing of such a high dense spatio-temporal data stream, is in standard systems usually in the range of a few days for a single MEG experiment. A special challenge is the artifact rejection in the magnetic field measurement data. The magnitude of the relevant magnetic field components is in the range of a few hundreds to tens to femto Tesla while some noise and biological artifacts, such as ocular or cardiac activity, can be tens to a hundred times larger [13]. These artifacts must be removed from the measured signals as they can lead to source reconstruction errors [7]. This study is focused on automated artifact rejection, which is neglected in many BCI approaches [7],[9]. OCARTA is chosen in our work, where the Infomax ICA algorithm in combination with cross-trial phase statistics is used for the development of automated real-time cardiac and ocular artifact rejection [7].

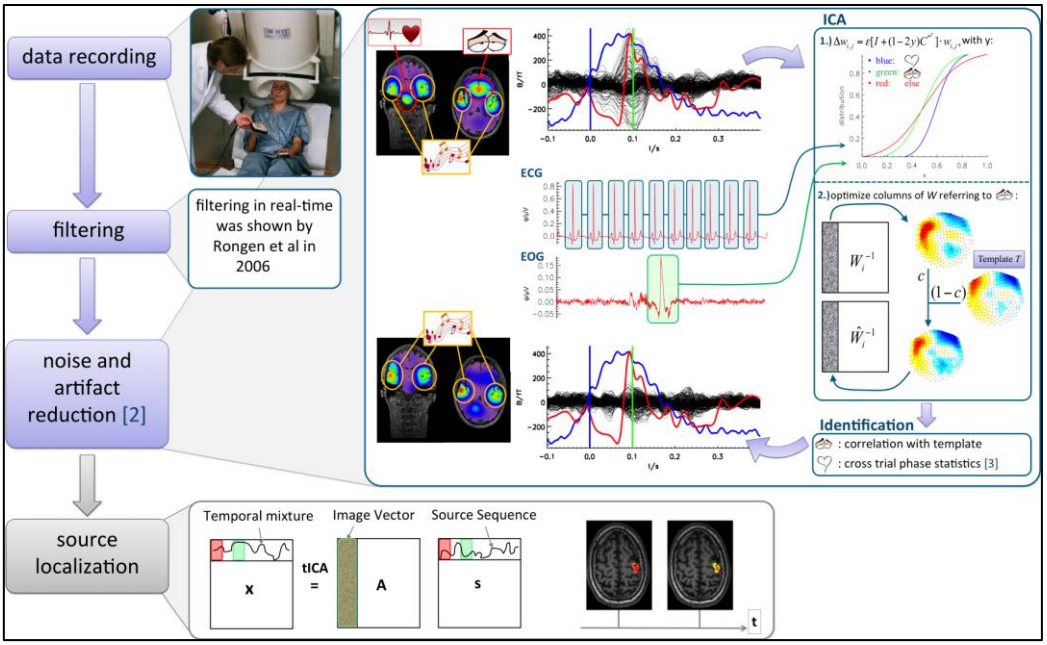

Figure 2.2: MEG data processing workflow ([7])

2.3.1 Data decomposition

A very challenging problem in MEG data analysis is that the measured signals contain a mixture of different biological sources (e.g., muscle movement, ocular or cardiac activity) as well as environmental noise and only as a very tiny part the wanted signal contributions from the brain

activities. These "unwanted" biological sources must be removed to reach high quality reconstruction data and minimize reconstruction errors [7]. Therefore, it is a mandatory step to decompose the measured signals into its underlying sources before further artifact rejection.

Independent component analysis (ICA) is a method for separating data into underlying informal components [9]. It is one of the techniques to solve blind source separation (BSS) problems [22],[23] in many fields including biomedical data processing [7],[24],[25], speech [26], or image [27]. In BSS, the measured signals are separated into its underlying sources without any prior knowledge about the source signals and the mixing system.

General linear BSS model

Suppose that the source signals consist of $\boldsymbol{m}$ independent components $\boldsymbol{s(t)} = [\boldsymbol{s_1(t)}, \dots, \boldsymbol{s_m(t)}]^{\boldsymbol{T}}$; the measured data has $\boldsymbol{n}$ mixtures of the source signals $\boldsymbol{x(t)} = [\boldsymbol{x_1(t)}, \dots, \boldsymbol{x_n(t)}]^{\boldsymbol{T}}$, where $\boldsymbol{n} \geq \boldsymbol{m}$. In the linear BSS, the mixtures are assumed to be linear, instantaneous, and noiseless, i.e,

$$\boldsymbol{x(t)} = \boldsymbol{As(t)}, \tag{2.1}$$

where $\boldsymbol{A}$ is the $\boldsymbol{n} \times \boldsymbol{m}$ mixing matrix. The goal of the classical ICA is to find an $\boldsymbol{m} \times \boldsymbol{n}$ demixing matrix $\boldsymbol{W}$ such that $\boldsymbol{m}$ output signals:

$$\boldsymbol{u(t)} = \boldsymbol{Wx(t)} = \boldsymbol{WAs(t)} = \boldsymbol{PDs(t)}, \tag{2.2}$$

where $\boldsymbol{P}$ is a permutation matrix and $\boldsymbol{D}$ a diagonal scaling matrix.

In the past decades, there has been many studies on solving the BSS problem. These techniques can be divided into three categories: higher-order statistics, minimum mutual information, and maximum entropy in their solutions (detailed in [28],[29],[30]). ICA belongs to the maximum entropy category. It has been shown that artifact rejection in MEG signals utilizing ICA does not harm the signal of interest if the decomposition and the selection of the artifact components are properly applied [6]. The measured signal mixtures are separated into underlying source by independence, which results in statistically independent components (often termed feature or basis vectors) [22].

A. ICA pre-processing steps

The typical toolchain for MEG consists of data recording, filtering, artifact rejection and source localization [7]. Principal component analysis (PCA) is applied in the context of artifact rejection to reduce the computational complexity of independent component analysis (ICA). PCA is one of the most commonly used techniques to reduce the dimensionality of a dataset while retaining most of the information by means of its variance [31]. PCA maps the original data into a new set of uncorrelated orthogonal basis vectors, where the components are stored in a decreasing order of variance [32]. The first axis (first principal component) constitute the greatest data variance of the original data set while the last one the least data variance. The first *k* principle components are chosen for data processing as they explain most of the data variance (in our case more than 95%). In the dimension reduction, computation of the eigenvalues and eigenvectors of large matrices is computationally complex [33].

A commonly used method for PCA is eigenvalue decomposition (EVD) of the data covariance matrix. It is comprised of the following computation steps:

1) The original data set $\boldsymbol{X}$ is centered by subtracting the mean vector $\overline{\boldsymbol{X}}$. The centered data are:

$$\widehat{\boldsymbol{X}} = \boldsymbol{X} - \overline{\boldsymbol{X}}\,, \tag{2.3}$$

2) The covariance matrix $\sum$ is computed:

$$\sum = \boldsymbol{cov}(\widehat{\boldsymbol{x_a}}, \widehat{\boldsymbol{x_b}}) = \frac{\sum_{i=1}^{n} \widehat{x_{ai}}\,\widehat{x_{bi}}}{\boldsymbol{n-1}}\,, \tag{2.4}$$

3) EVD is applied to the covariance matrix with the result:

$$\sum = \boldsymbol{K} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{K^T}\,, \tag{2.5}$$

with

$\boldsymbol{K} = [\boldsymbol{k_1}, \boldsymbol{k_2}, \dots, \boldsymbol{k_n}]$**:** square matrix with eigenvectors as columns,

$\Lambda^{-1/2} = \boldsymbol{diag}[\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \dots, \boldsymbol{\lambda_n}]$**:** diagonal matrix of the eigenvalues.

4) Sorting of eigenvalues in descending order.

5) Selection of the eigenvectors $\boldsymbol{K_m} = [\boldsymbol{k_1}, \boldsymbol{k_2}, \dots, \boldsymbol{k_m}]$ corresponding to the first $\boldsymbol{m}$ eigenvalues for data analysis:

$$\boldsymbol{Y} = \boldsymbol{K_m^T} \cdot \widehat{\boldsymbol{X}}\,, \tag{2.6}$$

Step 3 is here most computationally demanding of performing PCA

B. Principles of independent component analysis

This study is based on the recently presented method OCARTA [7], where Infomax ICA algorithm is adopted for data decomposition [9].

Infomax ICA

Bell and Sejnowski proposed an information maximization algorithm for realizing BSS with nonlinear artificial neural networks (ANN) [35]. The nonlinear function used is

$$\boldsymbol{g(x)} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e^{-x}}}, \tag{2.7}$$

where $\boldsymbol{g(x)}$ is the sigmoid function. The cumulative density function is supposed as sigmoidal function as many processes in nature underlie such a distribution [35].

The Infomax algorithm is to learn the demixing matrix $\boldsymbol{W}$ to give the outputs [35].

$$\boldsymbol{u} = \boldsymbol{Wx}. \tag{2.8}$$

When the entropy of the following signal $\boldsymbol{y}$, which is obtained by the sigmoid function, reaches its maximum, the source signals are separated.

$$\boldsymbol{y} = \boldsymbol{g(u)}. \tag{2.9}$$

Specifically, the learning rule of the Infomax algorithm is as follows:

$$\boldsymbol{\Delta W} = [\boldsymbol{W^T}]^{-\mathbf{1}} + (\mathbf{1} - \mathbf{2}\boldsymbol{y})\boldsymbol{x^T}. \tag{2.10}$$

For avoiding the computation of $[\boldsymbol{W^T}]^{-\mathbf{1}}$ in (2.6), the optimal demixing matrix $\boldsymbol{W}$ is computed iteratively using the natural-gradient version of Infomax ICA [36],[37]:

$$\widetilde{\boldsymbol{u}} = \boldsymbol{Wx} + \boldsymbol{W_0}, \text{ and}$$

$$\boldsymbol{\Delta W} = \varepsilon[\boldsymbol{I} + (1 - 2g(\widetilde{\boldsymbol{u}}))\widetilde{\boldsymbol{u}}^{\boldsymbol{T}}]\boldsymbol{W}, \tag{2.11}$$

where $\boldsymbol{I}$ denotes the identity matrix and ε the learning rate. $\tilde{\boldsymbol{u}}$ refers to the intermediate component matrix that more and more approximates $\boldsymbol{s}$ as the learning procedure goes on. $\boldsymbol{W_0}$ denotes a bias weight, which is initialized by $\boldsymbol{W_0} = \boldsymbol{I}$ and estimated by $\boldsymbol{\Delta W_0} = \boldsymbol{I} \cdot (1 - 2 \cdot g(\tilde{\boldsymbol{u}}))$. The bias weight centers the steepest part of the sigmoidal function on the peak of the probability density function of the input data $\boldsymbol{x}$ [9],[35].

The learning rate ε is reduced progressively over the learning course, which results in smaller changes in $\Delta \boldsymbol{W}$. These changes can be estimated by computing the Euclidean norm of $\Delta \boldsymbol{W}$. When the changes fall below a predefined threshold, the learning process converges [35].

2.3.2 Artifact identification

As already mentioned in MEG data analysis, ICA is used to separate brain signals from artifact components. After data decomposition, artifact components must be identified and afterwards excluded from measured signals.

Identification of cardiac artifacts

Components related to cardiac artifacts are automatically identified using cross-trial phase statistics (CTPS). CTPS is performed by testing the distribution of phase values against a uniform distribution across trials and for each point in time [6],[38]. For identification of cardiac components using CTPS, the phase values of all the ICs are firstly estimated using the Hilbert transform [39] and band pass filtered within the range of 10-20 Hz, which is the frequency band of the QRS complex[1] [6]. Then the filtered phase values are normalized and split into time windows (trials) of 600 ms around the R-peak of the QRS complex. The Kuiper's test is used to test the cross-trial phase distributions against a uniform distribution [40],[41]. The phase signals cluster near the time point of the R-peak when superposition of the phase trials from cardiac activity exists; in systems without any dependence on cardiac activity, the phase values will form a approximate uniform distribution whithout phase clustering. Details of the CTPS can be found in [6],[38].

Identification of ocular artifacts

Ocular artifacts are detected automatically by estimating the Pearson's linear correlation between each IC and the reference electrooculogram (EOG[2]) signal, which takes both the temporal and spatial dynamics of the independent components (IC) under investigation into account [7].

In terms of temporal correlation, the Pearson's linear correlation $\rho_{temp}(C_i, EOG)$ between each IC $C_i, i = 1, \dots, n$, and the reference EOG signal EOG is computed. Since columns of the demixing matrix $\boldsymbol{W_i^{-1}}$ contain the spatial information, it is necessary to estimate the spatial dynamics by computing the Pearson's linear correlation $\rho_{spatial}(w_i, T)$ between the template T and the field maps as extracted by ICA [7]. For improvement of robustness of ocular artifact identification, a single threshold combining both the temporal and spatial correlation is chosen.

[1] QRS complex represents the depolarization process of the ventricles in a heart [42]. On a typical electrocardiogram (ECG), QRS complex takes the form of three waves.

[2] EOG is a technique to record eyeball movements.

2.3.3 Data cleaning

In general, demixing matrices are estimated for segments separately, i.e., whenever the estimation of $\boldsymbol{W_i}$ is completed, the next demixing matrix $\boldsymbol{W_{i+1}}$ is estimated on basis of the last 12 s. Thereafter, estimating a demixing matrix $\boldsymbol{W_{i+1}}$ and identification of ICs referring to artifacts is performed in parallel to the data cleaning

The cleaning process is performed by zeroing columns of the demixing matrix $\boldsymbol{W^{-1}}$ that are corresponding to the artifact sources. Then the data are cleaned by back-transforming the independent source signals from the ICA space to the MEG space.

3 Model-based Systems Engineering and Computational Trends

With the great advancements of integrated circuit industry over the decades, we have reached a point where increasingly progressive, complex and distributed systems are possible with many different disciplines required [1]. An interconnected System-of-Systems (SoS) is far more than the sum of its subsystems [2]. Systems engineering is an approach primarily initiated in the aerospace and defense industry to provide system solutions to technologically challenging and mission-critical problems [43]. Over the years systems engineering has continued to evolve and highly developed modeling tools as well as methods are generated to deal with the increasing complexity of today's heterogeneous systems. Model-based Systems Engineering is about "*elevating models in the engineering process to a central and governing role in the specification, design, integration, validation, and/or operation of a system*" [47]. MBSE handles the ever rising complexity of those systems through an interrelated set of models, which adds traceability to development and enables comprehensive analyses. It generates the advantages of low cost, time saving and high accuracy compared with a traditional approach termed document-based systems engineering. The practice of MBSE bases on three pillars: a modeling language, a modeling tool, and a modeling method [44].

3.1 Systems Modeling Language- SysML

There are different modeling languages in different domains. Some of the languages are graphical computer languages (e.g., UML, UPDM, BPMN, MARTE, SoaML, IDEFx); others are text computer languages (e.g., SystemVerilog, Modelica) [44]. MBSE practitioners mainly use the Systems Modeling Language (SysML) to develop system solutions in response to complex and often technologically challenging problems [43],[44].

SysML is an extension of UML (Unified Modeling Language) created specifically for the systems engineering domain [45],[46]. It is a general-purpose graphical modeling language and inherits the advantages of both the traditional structured method and the object-oriented method. The language enables moving on different abstraction levels and effectively communicating the essential aspects of a large-scale and complex system in detail: structure, behavior, requirements, and parametrics (mathematical models) [44].

SysML defines nine kinds of diagrams to communicate all the system design information and supports the specification, analysis, design, verification, and validation of a broad range of complex systems [45]. Each diagram serves a specific purpose and conveys specific information about a specific aspect of the system [44]. The SysML diagrams are divided into 3 categories, which are plotted in Fig. 3.1 together with the relationships among them. Behavior diagrams include activity diagrams, sequence diagrams, state machine diagrams, and use case diagrams. Structure diagrams consist of block definition diagrams, internal block diagrams, and package diagrams. Parametric diagrams are a sub-type of internal block diagram. Finally, requirement diagram itself is a category.

A brief introduction of each kind of diagram is as follows.

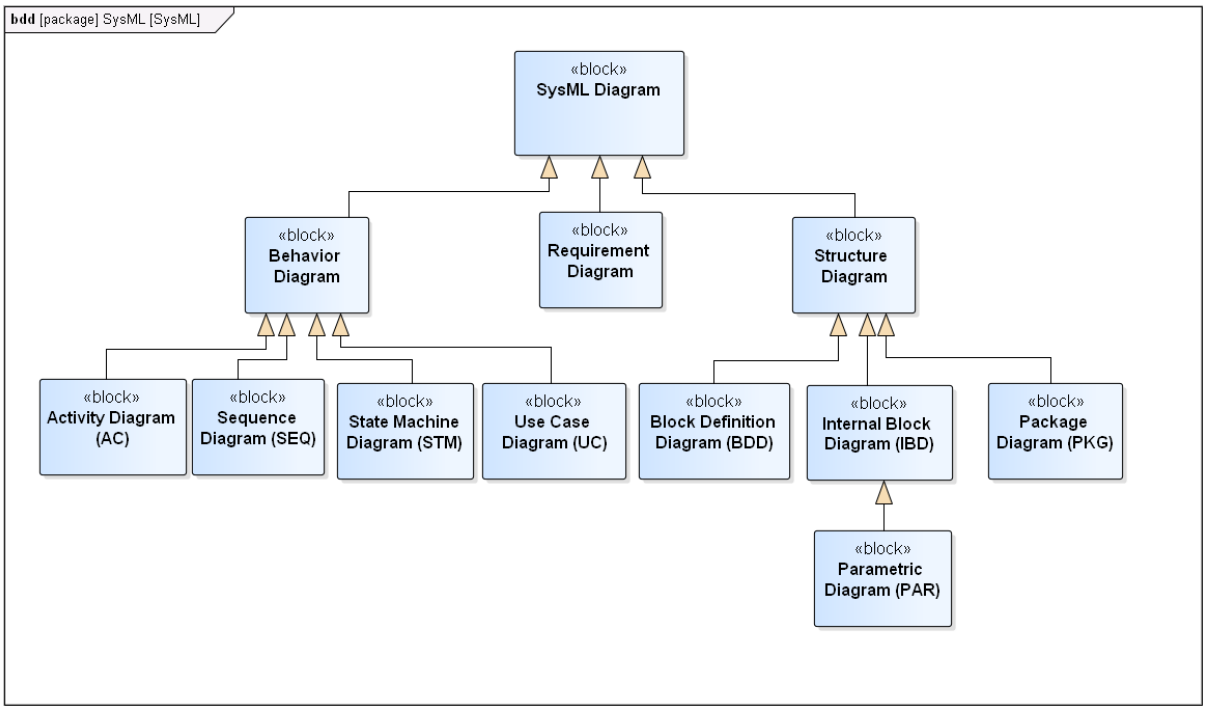

Figure 3.1: SysML diagram taxonomy ([45])

The **B**lock **D**efinition **D**iagram (BDD) is used to represent structural elements such as blocks and their relationships. It can also be used to display instances of blocks and relevant configurations and value properties. A BDD is also widely adopted to define system hierarchy trees and classification trees [44].

The **I**nternal **B**lock **D**iagram (IBD) specifies the inter-connections and interfaces between the parts of a single block [43].

The **U**se **C**ase Diagram (UC) conveys the interactions between a system and its external entities (e.g., actors). It shows how a system is invoked and used by external entities. A use case diagram is a black-box view of the system functionality.

The **Act**ivity Diagram (ACT) represents the system behaviors, focusing on how the actions transform the inputs into outputs and relevant control flows. ACTs are often used as analytical tools to understand and express the desired system behaviors [44].

The **Seq**uence Diagram (SEQ) shows the interactions between blocks or parts of a block via a sequence of message exchanges. A message can represent an invocation of a behavior or the sending of a signal [43],[44].

The **St**ate **M**achine Diagram (STM) is also a kind of behavior diagram. It specifies a set of states of a block and the possible state transitions triggered by event occurrences. STMs can be used in conjunction with other behaviors (e.g., constrain certain aspects of their behavior) [43].

The **Par**ametric Diagram (PAR) imposes mathematical relationships on the values of the design using constraint blocks to create prognostic models of implementation. Constraints are expressed as equations or inequalities, whose parameters are bound to the properties of the design. A parametric

diagram enables trade-off studies of different design configurations to find an optimal set of solutions. It can also be used for checking violation of the quantifiable constraints of a design, or for performing early functional prototyping (mathematical modeling).

The **P**ac**kag**e Diagram (PKG) shows the organization of a model in the form of package containment hierarchy. PKGs are also commonly used to convey the dependencies between packages and their contained model elements [44].

The **Req**uirements Diagram (REQ) is used to convey text-based requirements and their relationship with other requirements (containment, derive requirement, and copy), and the relationships between requirements and other model elements to establish requirements traceability.

3.2 System design methodologies

A modeling language is a set of fragmented modeling grammars. It never tells how and when to create a model. A modeling method is like a roadmap that helps put the modeling language into practice and create a system model. A modeling method determines the scope and order of modeling activities and the types of modeling artifacts to represent a system [43]. Several MBSE methods are documented in literature [47],[48]. Different methods may involve different modeling activities and produce different system representations [43]. MBSE practitioners can either adopt one of the existing modeling methodologies or create a custom modeling method.

Numerous lifecycle development models have been proposed and widely used to deal with the increasing design complexities in the fields of industry and research. However, most of the lifecycle development models are based on one of three meta models: Waterfall Model [49], Spiral Model [50], and V-model (or "Vee" Model) [52],[53]. The software development process is divided into serial steps in the waterfall model. The development stages are accommodated in rectangles going downwards in roughly one direction like a waterfall through the steps of requirements analysis, design, coding, testing and maintenance. The Spiral Model orders multiple instances of the developments phases in a spiral shape as the development process is often iterative. Variations of the waterfall and spiral models are targeted at designs in the software domain, which facilitates well-organized, iterative or progressive development. In contrast, the V-model and corresponding alterations are widely used in systems engineering. Thus, only a brief description of the V-model is provided here, as plotted in Fig. 3.2.

The V-model was originally proposed by Paul Rook [51] in 1986 as an extension to the waterfall model. It has been widely used in software development until now. The V-Model specifies the associations between each step of the development life cycle and its associated phase of testing [54]. As apposed to steering down in a straightforward direction, the development streamline goes upwards after the coding phase is done, shaping a V structure. The V-Model has been becoming flexible and agile with the evolution over the years. It is being utilized by more and more industries, including medical device, etc.

A coherent and organized workflow is utilized in the V-model where every step can be realized based on the detailed documentation of the former step. Within V-model, testing is launched at the initial phase of the project prior to coding; and this significantly accelerates the project development. The V model is of great efficiency and effectiveness for software development and specifies the relationships between test activities and development activities [54].

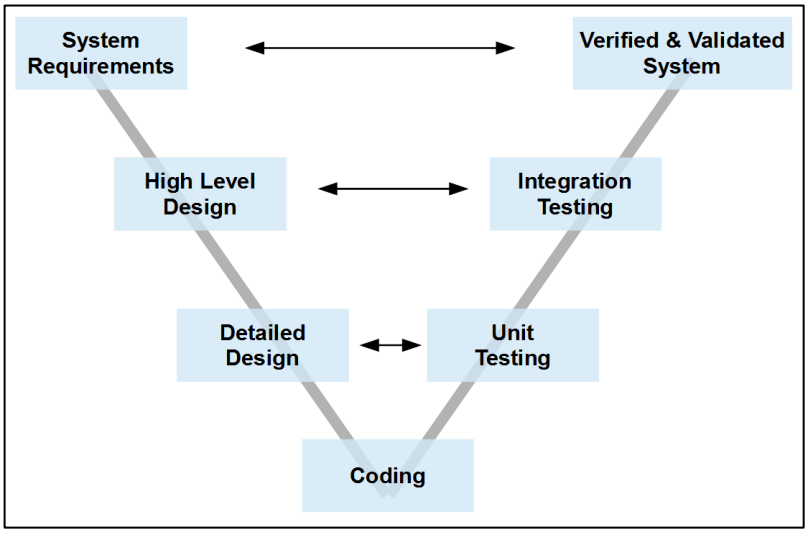

Figure 3.2: V-Model ([53])

Some well-known MBSE methodologies are introduced briefly as follows, which leverage the V-model lifecycle development model.

3.2.1 INCOSE Object-Oriented Systems Engineering Method

Object-Oriented Systems Engineering Method (OOSEM) is a top-down, scenario-driven methodology that leverages SysML for the analysis, specification, design, and verification of systems [43]. It uses object-oriented concepts and other modeling techniques to construct more flexible and extensible systems, which are intended to ease the accommodation of evolving technology and changing requirements [47]. OOSEM also facilitates the integration with object-oriented software/hardware development, and test processes [43]. Fig. 3.3 shows a high-level summary of the OOSEM activities in the system development process.

OOSEM consists of the following development activities:

The *Analyze Stakeholder Needs* activity characterizes the as-is system, users, and enterprise to understand the stakeholder problems to be solved and specifies the goals the system is intended to support [43].

The *Define System Requirements* activity defines the system requirements in a black box view with respect to its input and output behaviors and other externally observable characteristics.

The *Define Logical Architecture* activity decomposes the system into logical components that interact with each other to satisfy system requirements. The logical architecture works as an intermediate abstraction level between the black box system requirements and the physical architecture [43].

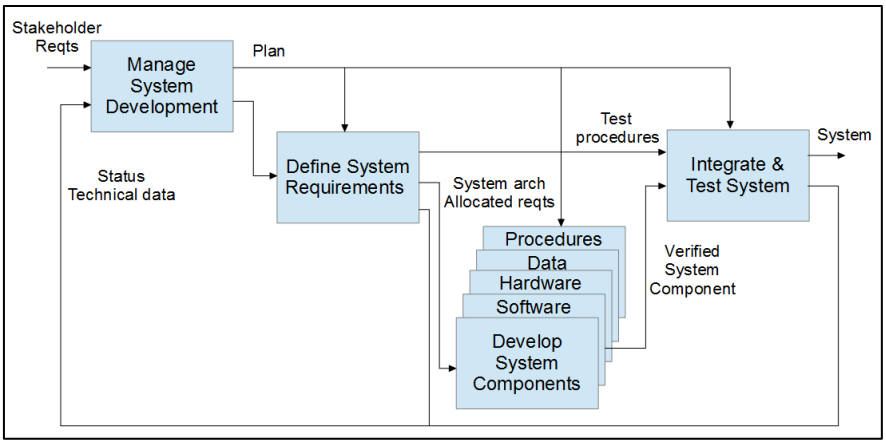

Figure 3.3: OOSEM activities in the system development process (following [47])

The *Synthesize Candidate Physical Architectures* activity allocates the logical components to physical components including hardware, software, persistent data, and operational procedures [43].

The *Optimize and Evaluate Alternatives* activity is performed by other OOSEM activities to do engineering analysis and trade studies.

The *Validate and Verify System* activity verifies that the system design satisfies its requirements and validates that the requirements meet the stakeholder needs [47].

3.2.2 Embedded systems development using SysML

Model-based systems engineering (MBSE) provides a high-level environment and can efficiently handle the ever rising complexity of embedded systems. A development roadmap for embedded systems using SysML is introduced in [55], as shown in Fig. 3.4.

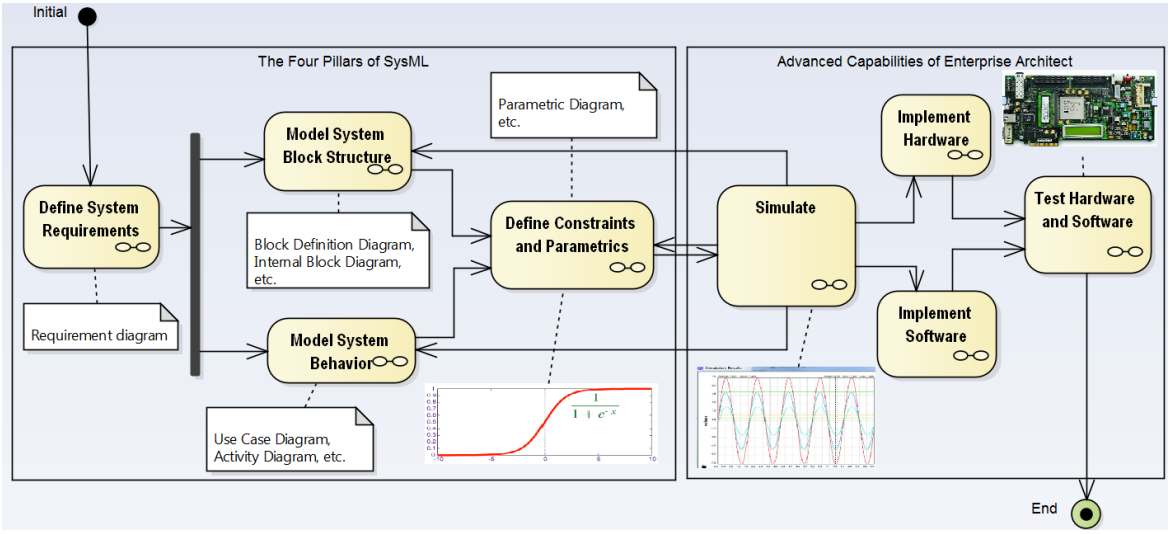

Figure 3.4: Embedded systems development methodology with SysML

The roadmap starts off by specifying general system requirements and constraints of stakeholders, providing an initial outline of the system. Requirements are organized into hierarchies using requirement diagrams. Requirements can be allocated to other modeling elements to establish traceability within the relevant toolkit.

Then system requirements definition is followed by modeling system dynamic behaviors and block structure in parallel. Structurally, block definition diagrams define the overall system structure. Internal block diagrams detail the internal structure. Additionally, package diagrams are utilized to organize the models.

In terms of behavior modeling, SysML provides four kinds of behavioral diagrams to show different views of system behaviors from different perspectives. They are use case diagrams, activity diagrams, sequence diagrams and state machine diagrams. The toolkit used in [55] has the ability to generate both hardware and software executable functional code from state machines.

After that, we define constraints and parametrics to establish mathematical relationship whose parameters are bound to the properties of a system. First, constraint blocks and parametric diagrams are specified. Then the simulations are performed based on the executable SysML Parametric diagrams supported by the corresponding toolkit.

The final step is to integrate and test the design in your favorite IDE (Integrated Development Environment) based on the MDG (Model Driven Generation) Integration technology developed by Sparx Systems.

3.3 System design tools

Just as the saying goes, what is a workman without his tools? A proficient and powerful modeling tool is indispensable to MBSE.

The modeling language specification SysML is vendor neutral and there are a lot of options of modeling tools available, including commercial-grade modeling tools such as Enterprise Architect (vendor: Sparx Systems), Rhapsody (vendor: IBM Rational), Agilian (vendor: Visual Paradigm), etc., as well as free modeling tools Modelio (creator: Modeliosoft), Papyrus (creator: Atos Origin), etc. [44]. Based on the project requirements and cost constraints, we selected Enterprise Architect in our project. Enterprise Architect contains a number of unique features for systems and software engineers working on embedded systems, including executable SysML Parametric Diagrams and hardware/software code generation from UML/SysML models. It also bundles licenses for various integration products to provide powerful model-driven construction tools to tightly bind Eclipse or Visual Studio development with the UML/SysML model [55].

3.4 Computation trends

With the rapid development of silicon IC technology over the past decades, more and more sophisticated and calculation intensive computing methods are widely used and have brought great benefits to scientific research and industry. However, around the turn of the century, the computation

power of digital systems has come to saturation due to single-processor performance growth constraints [56]. One trend for better computing performance is parallel computing. We have witnessed the rise of parallel processing in many calculation intensive applications, workstations and in handheld devices [57].

The technology platforms for parallel computation addressed in this study are the configurable logic of Field- Programmable Gate Arrays (FPGA) and the stream processing architecture of Graphical Processing Units (GPU). Both parallel processing solutions are widely used in the scientific research and industry because of low development costs and high computation power. Two high performance computing (HPC) trends are System-on-a-Programmable-Chip (SoPC) and accelerated processing units (APU).

3.4.1 System on a programmable Chip (SoPC)

System-on-a-Programmable-Chip (SoPC) reconfigurable computing is becoming a new paradigm for high-performance computing and heterogeneous system design with the advantages of re-configurability, flexibility, and a short development cycle [58]. It enables designers to integrate an entire system on a single device by utilizing a large FPGA that combines memory, logic elements and intellectual property (IP) cores [59].

Altera first proposed SoPC technology and launched Nios soft core in the year 2000 [60]. Soon afterwards, Xilinx integrated PowerPC embedded processor to some of its FPGAs and released MicroBlaze soft core.

In 2011, Xilinx launched ZYNQ-7000 series SoPC chips [61]. For the first time high performance RISC ARM hard core is integrated with programmable logics. On ZYNQ SoPC platform, programmable logic devices can be viewed as peripherals enjoying programmable characteristics. For example, programmable logics can be extended as serial ports or Ethernet interface, high-speed serial interface, video interface, etc. On the other hand, programmable logic devices also work as another host. It initiatively accomplishes data interaction with external chips, including pre-processing of network, algorithm, video, etc. With the integration of ARM and programmable logic devices, the system has the advantages of processors with respect to complex control algorithm and operating systems, but also it utilizes the characteristics of FPGA in terms of algorithm acceleration and reconfigurability to implement more complex designs, which adds high flexibility to the system [62].

As shown in Fig. 3.5, the upper part of ZYNQ is ARM CortexA9 hard core as processing system and the lower part is Xilinx FPGA as programmable logic.

3.4.2 Accelerated processing units (APU)

An APU is produced by the integration of a CPU and a GPU on the same chip, which improves data transfer rates between these components previously confined by the PCIe (Peripheral Component Interconnect Express) bus [63],[64]. In parallel applications, GPUs are connected via PCIe bus as a discrete device. The APU removes the bus between the CPU and GPU, which pushes parallel

computing one step further and enables better performance than a separate GPU or multi-core CPU processors [65].

Here an overview of the architecture of the first generation APU is presented, i.e. AMD Fusion E-Series APU.

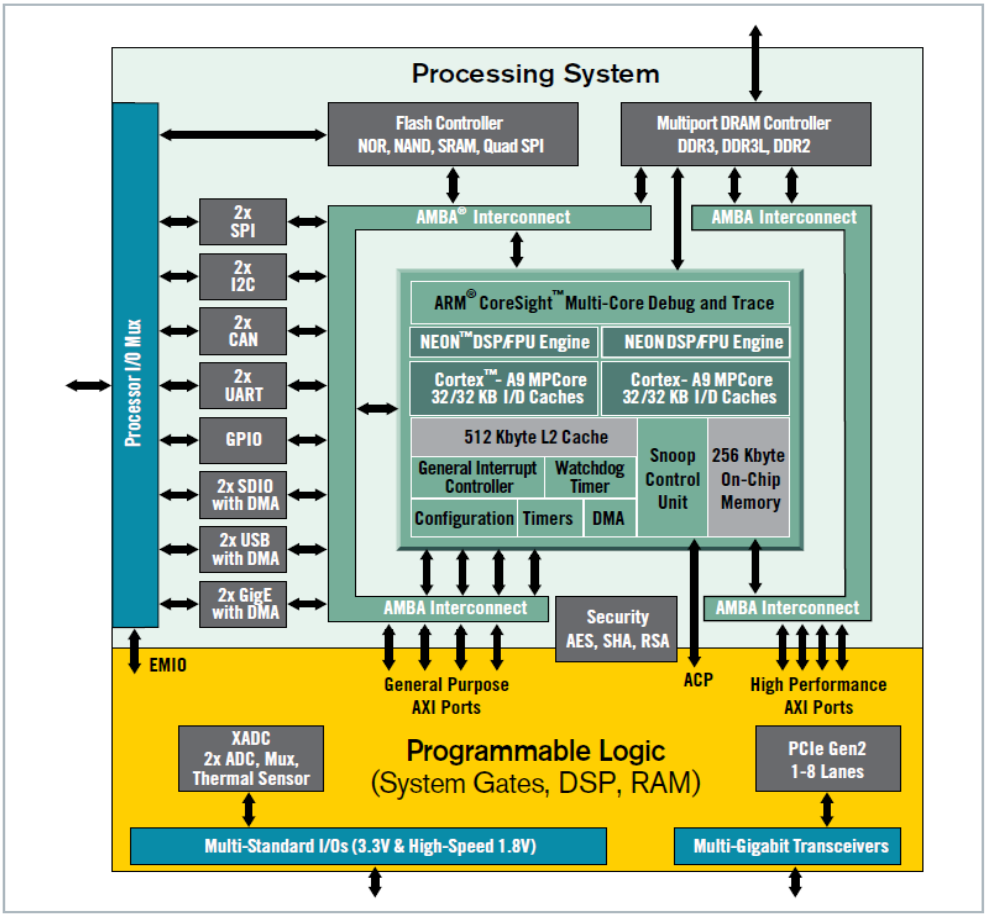

Figure 3.5: Internal structure of ZYNQ (from [61])

At the most basic level, the Fusion architecture integrates general-purpose X86 CPU cores and programmable vector processing engines onto a single silicon die, which results in a heterogeneous computing processor [65]. The combination raises the advantages of both (precise scalar computation of the X86 cores and large-scale parallel vector workloads) to higher level, which enables designers to adopt the most appropriate method and develop new application with great flexibility.

Fig. 3.6 plots the block diagram of AMD Fusion architecture. On the same silicon chip, the x86 CPU cores and the vector (SIMD) engines are connected to the system memory via the same high speed bus and memory controller [65], enabling both devices to directly access the memory with high speed. Apart from the processing cores, the Fusion architecture also consists of other system elements, including memory controller, I/O controller, video decoder, display output, and bus interfaces.

Even though the x86 cores and SIMD engines access the system memory via the same high performance bus on Fusion, the system memory is divided into two parts. One part is managed by the operating system of the X86 cores; the other part is operated by the software of the SIMD engines. The data transfers from one part of memory to the other are done via memory copy, which is much faster than data transfers through PCIe.

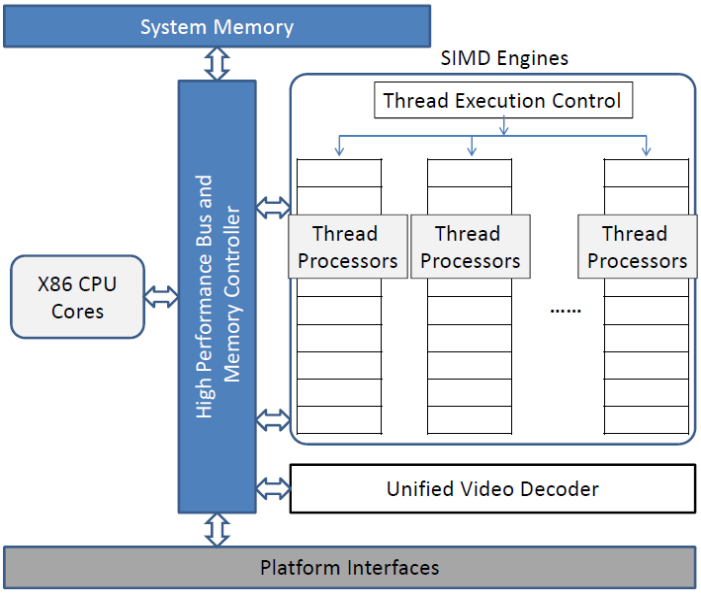

Figure 3.6: Block diagram of the AMD Fusion architecture (from [65])

Programming on Fusion uses the OpenCL (Open Computing Language) standard. Therefore, existing designs realized with OpenCL for the discrete CPU and GPU combination can be transplanted to the integrated CPU+GPU of Fusion architecture without modification [65].

4 Methodology

Chapter 4 is dedicated to the model-driven system design methodology. The real-time MEG signal pre-processing system is implemented following the methodology proposed in this chapter. The chapter elaborates on the theoretical knowledge of the model-driven system design methodology workflow, which maps the computation task to the target implementation platforms. The methodology introduced in this chapter relies on the state of the art techniques of systems engineering.

4.1 Requirement-driven development

The goal of systems engineering is to find a set of optimal solutions fitting for operational parameters (requirements and constraints) and project conditions. System-level requirements are important both for principle decision making in the development process, as well as building a validation plan for feature checking, and in a certain amount for verification process - an implementation at any abstraction level must match the system-level requirements. As shown in Fig. 4.1, the number of dimensions in solution space depends on operation conditions, which is usually larger than 3 and includes power dissipation, weight, size, etc. The requirements should be studied thoroughly as lack of certainty destroys optimization function. Therefore, requirements and constraints should be treated unambiguously, which indicates that they should be formalized with a good as well as clear specification.

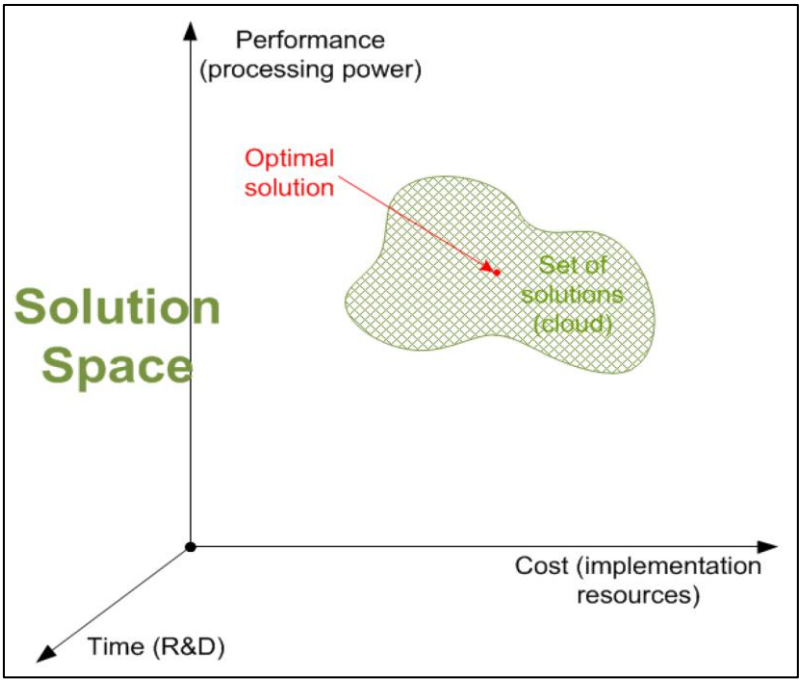

Figure 4.1: Number of dimensions in solution space (from [4])

4.2 Model refinement

One of the major challenges of system design is system complexity. Recent technological progress in digital circuits has enabled the ability to implement increasingly complex systems. The complexity of embedded systems doubled every 10 months in the last decades [66]. The advances produce unprecedented computational power, but it also makes the systems extremely difficult to develop and verify. One approach to address the challenge is to start the model-driven development at higher levels of abstraction[67]. MBSE has been adopted to improve the work efficiency of computational systems developers by specifying an object with models where some low-level details are disregarded [68]. The key to system design is refining the system model gradually. The refinement process reveals the relationship between specification and implementation. The gradual refinement enables a modeler to compare the specifications at various abstraction levels by inserting more intermediate models and examine the possible alternatives of one component [69]. The higher the stage a mistake is detected, the lower the costs for fixing are; later change of specification costs more.

Fig. 4.2 represents the design process. The requirement-driven model-based approach starts from creating a model at the highest abstraction level. The model is analyzed and synthetized with **C**omputer-**A**ided **D**esign (CAD) tools at the next lower abstraction level [70]. Each model is verified against the previous one. Model description should be formal and complete to be mapped to the target platform by CAD. But CAD does not know what to do with ambiguities and CAD is not fault-free by default. The higher number of implementation artefacts, the more distinct is a human factor.

For ensurement of predictable results, it has to document all decisions during the continuous refinement through all abstraction levels and link these decisions to the requirements and between the different abstraction levels. At the end and already during the development it always has a link from the specific requirements to the technical decisions which has to be done and vice versa.

4.3 Model abstraction levels

For the modeling and refinement process, it is important to specify the dominating model features and the corresponding resolution levels [3]. The Virtual Socket Interface Alliance (VSIA) carried out a concrete work on the standardization of the electronic system design, which contains an extended definition of a precision scale for taxonomy and model classification system for digital system models [71],[72], as plotted in Fig. 4.3.

Since the digital system taxonomy is persistently being refined and utilized in other scenarios (e.g., standardization for electronic system-level, ESL, design was performed in [73]), four major precision scales are addressed in this study for the accuracy specification of hardware logic design models, which is a little different from the VSIA specification [3].

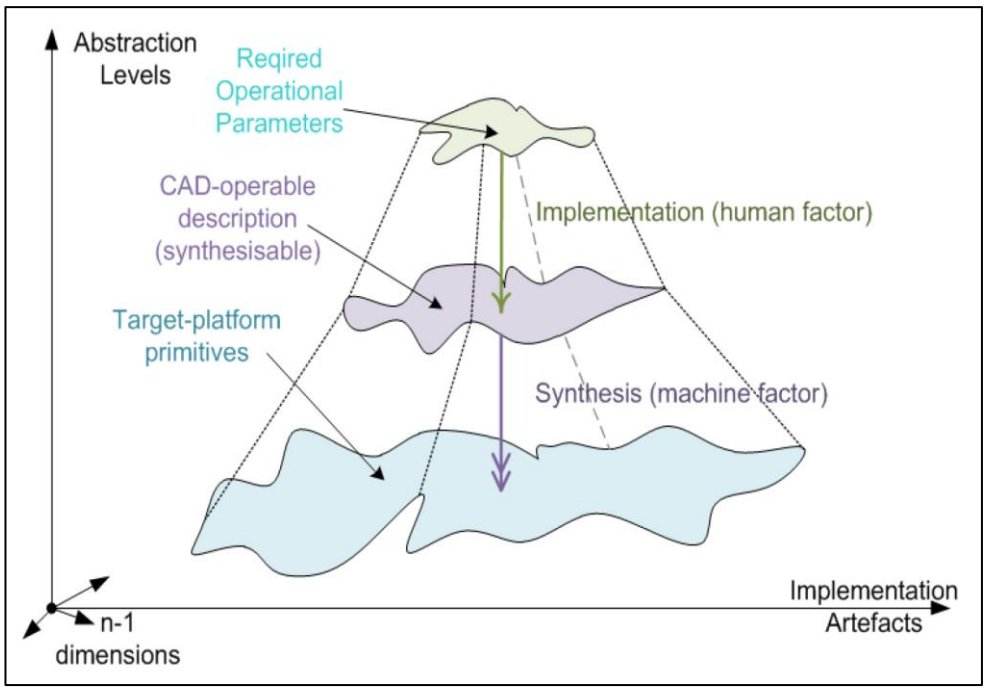

Figure 4.2: Model-based design process (from [4])

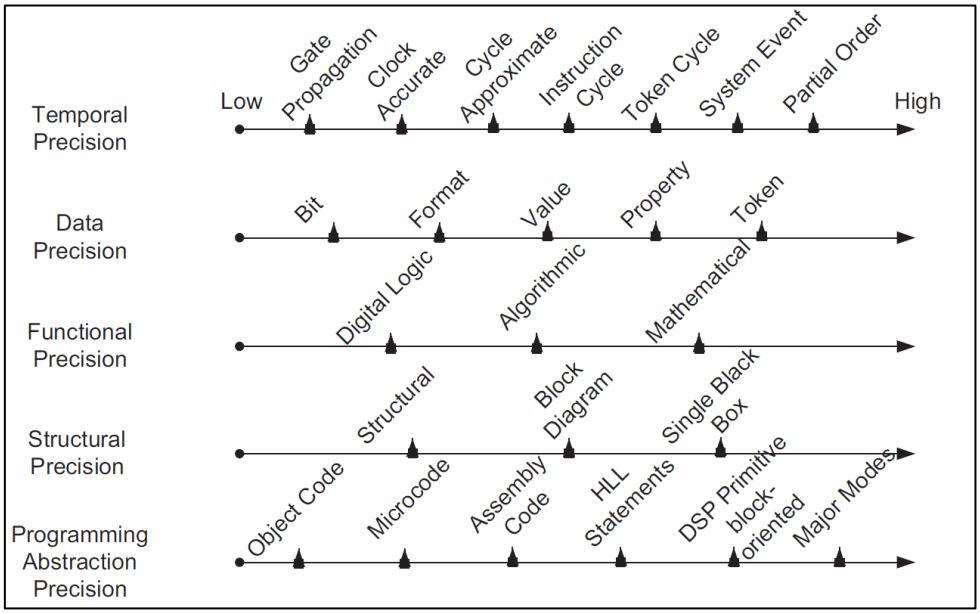

Figure 4.3: VSIA digital system taxonomy (from [3])

Four major constitutes of the VSIA taxonomy are elaborated in this thesis:

Time accuracy specifies the level of event ordering in a model; models may transform from irregular and roughly precise models to the models imposed by physicaldata transmission delays;

Data abstraction defines the signal load conveyed by the data objects; models can transform from those running with characterless signal elements to refined models at the signal level;

Computation accuracy specifies how accurate a data processing approach is depicted; the specification can transform from abstract computation specification, which only describes the result to be obtained while the implementation method (e.g., functional languages) is not detailed, to an accurate realization of the chosen approach in terms of technological elements working as the execution constraints;

Topology precision represents the elemental abstraction level of the system, which specifies the duty of the system elements and details the interconnections between the factors and the characteristics of the connectivity; models can transform from obscure interface models to placed and routed models at the gate-level.

The more detailed and precise a model is, the more efforts and resources are demanded for implementation. With respect to global processes of a design, it is wise to perform the modelling work at a higher level of abstraction. In other words, the impact of a mistake caused by making an incorrect major judgment at the design stage is more perceptible; that is because the higher the decision is made, the lower the mistake is localized. Furthermore, to acquire more global system information using models of higher abstraction levels, one has to invest more on collecting higher precision data and the acquisition characteristics at the global level [3].

4.4 Model refinement process

A successful system development depends on the progressive refinement of the system models. The system design process initiates with the elicitation of system requirements and constraints at the global level, drawing an preliminary system-level sketch [3]. The system development approach refines the system model following some polishment steps down to a lower abstraction level ready for automatic hardware implementation with CAD tools. There are several development stages within each abstraction level, which includes a specification, an implementation, a verification and an analysis, forming a development cycle [3]. The design flow of these cycles in the refinement process makes up a top-down spiral like shape as shown in Fig. 4.4.

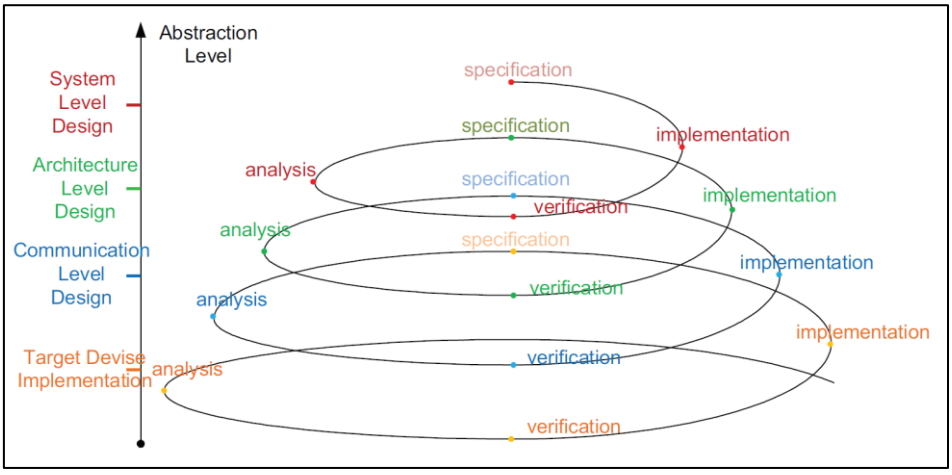

Figure 4.4: Model-based systems engineering for real-time systems (MBSE-RT) workflow (from [3])

In the system design cycle, general system requirements and constraints are firstly defined as requirements are the basis of system design. Then all potential keys to tackle the computation problem and the existing techniques for system implementation are explored. The ideas are cast upon one model or several models, making up the preliminary implementation stage. In the implementation step, the system is defined at an abstraction level based on the relevant model specification. After the implementation process, the implemented design models are verified to make sure they satisfy the the system specifications and establish a suitable replacement when inconsistency exists. The final step of the development cycle is analysis, which leads the design process to a more accurate decision direction for the following design process iteration.

The proposed top-down development approach is capable of uncovering possible bottlenecks and problems of a system implementation early in the higher level implementation stages. It also works as a golden rule for a developer.

The refinement process of the models is not necessary to be in parallel. The design technology nowadays enables different parts of a system to be specified at various abstraction levels simultaneously in the same model. The progressive refinement process of system modeling is aimed at strong consistency of a model, which eliminates critical fractures in model development. The asynchrony ensures great elasticity in the project administration and according methods. It empowers developers during the development process to be devoted to the most crucial parts of a system and engaged in different parts synchronously and autonomously.

4.5 System-level development path

We have introduced an RDD & MBD approach for real-time computation systems based on vendor neutral specifications. The methodology, as the further development of a comprehensive methodology for functional design in digital systems presented in [4], has been elaborated in the Central Institute of Engineering, Electronics and Analytics - Electronic Systems (ZEA-2) by Dr. Sergey Suslov to address the problems of heterogeneous computation and control system design using SysML [3] and first verified in this work. The workflow is plotted in Fig. 4.5.

It starts off by specifying general system requirements and constraints of stakeholders and formalizing them into three categories, i.e. *Qualitative*, *Quantifiable* and *Major Operations*, which are further refined into sub-categories. The qualitative category mainly includes immeasurable requirements such as safety, continuous observability, ease of operation, etc. *Quantifiable* category captures measurable parameters and constraints. With the refinement of system models, some qualitative requirements may become quantifiable. The *Major Operations* category depicts the functionality of the system, including maintenance modes and functional targets of an application.

After the system's technical requirements are specified, the second step of the methodology- functional and structural analysis- is performed. The corresponding system dynamic (behavior) and block structure are modeled in parallel, including *Data modeling*, i.e. data structure, data dependency, data exchange flows, and *Methods and algorithm modeling*. *Quantifiable* requirements are afterwards

imposed in the form of mathematical constraints to the relevant elements of structural models to properly allocate them to relevant computation structures with predictable implementation parameters such as performance, resource budget and power consumption. Thus, one partitions the functions and allocates them to corresponding computation hardware. This process is iterative, which indicates the procedure may be repeated for each sub-system performing refinement of the system on different abstraction levels [3],[4].

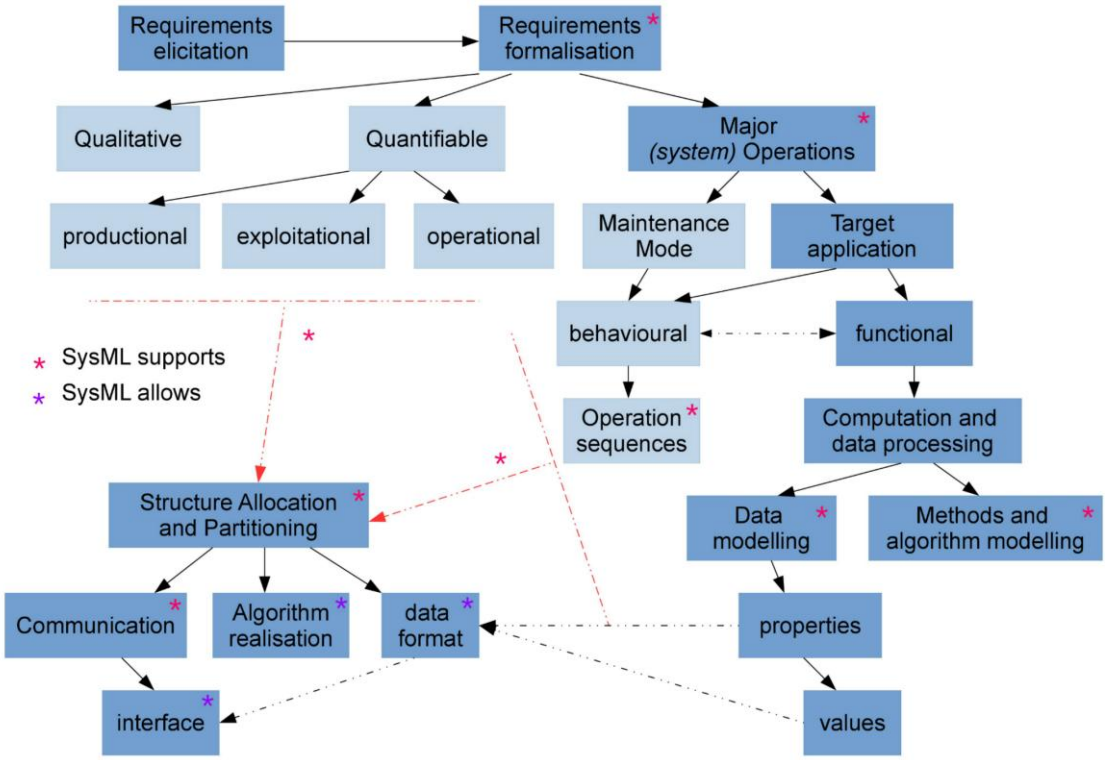

Figure 4.5: Model-based systems engineering for real-time systems (MBSE-RT) workflow ([4])

4.6 Verification

Verification is a group of techniques analyzing the established model at post-implementation stage. It aims at examining the compliance between the achieved and desired qualities of a design and the determining of the faults or gaps causing discrepancy [3].

Functional verification

Logic design is generally dominated by checking the accuracy and precision from the perspective of functionality. Functional verification is an intricate process and costs the majority of time and effort for most digital designs, making it a methodologically developed and formalized part of the verification process.

Many implementation errors keep bursting out from each step of model refinement. The number of the amassed bugs is large enough to make the verification of a model as a whole impossible, because it is extremely difficult to locate the errors. Therefore, the divide-and-conquer strategy is adopted in

functional verification in accordance with the system hierarchy [3]. In this context, component-based verification techniques significantly reduce the overheads by verifying one component at a time while taking component interdependencies into account. The component-based verification is compulsorily followed by integration tests that reveals unexpected inconsistencies in the components interaction. With the ever-rising complexity of designs and the demand of time-to-market acceleration, it is crucial to verify the design in a short time frame through automation [74].

The significance of verification improves with the increasing complexity of digital systems and the broad reusing of existing components (Intellectual Property blocks), where thorough verification coverage of the glue logic between the components and the global system functionality is necessary. The broad range of verification is carried out because of unforeseen errors in various component configurations, which did not show in the initial design [3].

Conceptually functional verification can be broadly classified into two major verification models, *black-box* verification and *white-box* verification [75]. White-box verification models are built based on an internal perspective of the system and can test the internal structures. By comparison, the black-box models are constructed with no concern with the internal mechanisms of a component or a system and focus solely on the inputs and outputs of the design interfaces.

These two models are opposed but complement to each other. Black-box verification is reusable when testing the functionality of an application. White-box verification is widely utilized in determining and localizing implementation errors [3],[76]. In practice, it is necessary to combine both the two approaches for design verification.

Timing verification

Modern digital systems have undergone unprecedented growth and become increasingly complex. The system circuits operate under tight timing constraints and signal transmission delays among components critically influence the design functionality. Therefore, it is of great importance to check the correctness of system temporal behaviors.

The timing verification is performed at the gate-level during a back-annotation process using **E**lectronic **D**esign **A**utomation (EDA) tools to assign delays to each related component of the design. The assigned delays are specified in a separate file (e.g., in the Standard Delay Format [77]), thus there is no extra change in the netlist file. The timing models can be precise enough to specify all the crucial timing features of various electronic elements, which include sensible signal pulse width, trigger setup time, signal hold time etc. [3].

Two strategies of timing verification are broadly in use: static timing verification and dynamic timing verification [78],[79]. In dynamic timing analysis process, the system functionality may be verified by using the same test cases for functional verification, which discloses the influence of gate delays and parasitic impact on functionality [78]. In static timing verification, path delays are compared against a set of timing constraints in back-annotation to locate the real sources of errors.

On the one hand, dynamic timing analysis has the advantage of higher accuracy and is widely adopted in both synchronous and asynchronous designs. On the other hand, despite the requirement of additional effort for constraints specification, static timing models have the merits of high verification coverage and high level of observability with sufficient accuracy, usually playing a dominating role in FPGA designs [3]. Typically, both static and dynamic approaches are used in practical applications.

5 Systems Engineering

In this chapter, the RDD & MBD approach was applied to the development of the Research Center Jülich MEG-RT 2.0 project. Requirements of MEG-RT 2.0 were analyzed and formalized. For the further model levels, real-time artifact rejection was chosen as the study focus. The SysML diagrams were created to detail the dynamics of both structural and behavioral aspects of the design. Afterwards, resource consumption and computation speed of the design were predicted based the performance analysis using parametric models.

5.1 Requirement analysis

5.1.1 State the problem

Magnetoencephalography (MEG) studies spatio-temporal dynamics of the underlying electrophysiological interaction in the brain. During measurements an array of hundreds of low-TC SQUID sensors is used to record components of the magnetic field produced by the post-synaptic activity. With MEG the spatio-temporal dynamics can be investigated at very high temporal and relatively high spatial resolution depending on several factors (e.g., the strength and depth of the active sources) [13].

To push MEG more forward towards a clinical diagnosis tool (e.g., pre-surgery planning, early diagnosis, localization of epileptogenic zones, rehabilitation of stroke) results must be available during or soon after the investigation. The dynamics of the brain are extremely rich and it is possible to see major effects by applying stimuli with respect to the state of the neuronal dynamic system [80], which is termed neuronal feedback. The ability of controlling in real-time provides new insights into brain function of both normal and impaired brain processes. As it has been indicated in [81], a time delay of > 200 ms between brain activity like e.g., movement intention and the feedback output by the device is clearly noticeable. In real-time computing, processing steps, including data acquisition, filtering, artifact rejection, etc., are designed in a pipelined structure. The latency is caused by the step with the largest delays. Therefore, signal processing systems with an overall constant delay of < 200 ms is considered to be real-time for neuroprosthetic control [81]. Brain–computer interfaces (BCI) utilizing neuro-feedback stimulation offer a great potential for neuroscience and therapy in neurology, but require adequate real-time analysis of on-going brain responses. Therefore, the MEG-RT 2.0 project was proposed through a scientific collaboration with research groups from the Forschungszentrum Jülich: the Institute of Neuroscience and Medicine (INM-4) and the Central Institute of Engineering, Electronics and Analytics (ZEA-2 - Electronic Systems) to address advanced MEG signal processing, including current source reconstruction, which will enable real-time neuro-feedback applications.

5.1.2 System requirements definition

The most important part of system modeling is defining requirements, because requirements are the basis of system design and the aim of the system design is to satisfy the requirements [55].

System requirements are closely related to the environment that the system will interact with. The block definition diagram (BDD) *OperationContext* shown in Fig. 5.1 was created to define operation contexts to which MEG-RT 2.0 is subjected. In order to improve the feasibility of MEG as a clinical diagnostic tool, the aim is to process MEG signals in real-time and thus enabling e.g., neuro-feedback applications. The Fig. 5.1 shows specific relationships of major environment elements. The *MEG-RT 2.0* design is an add-on to an existing data acquisition system (DAS). The MEG measurements are filtered and processed by the *MEG-RT 2.0* block. The processing results of *MEG-RT 2.0* block can be used in a hospital or research group for neuro-feedback applications and clinical diagnosis. The operation contexts drive downstream requirements, e.g., artifacts such as ocular or cardiac activity lead to reconstruction errors in source localization. The design was expected to perform real-time and thus automated artifact rejection, source localization and neurofeedback in real-time.

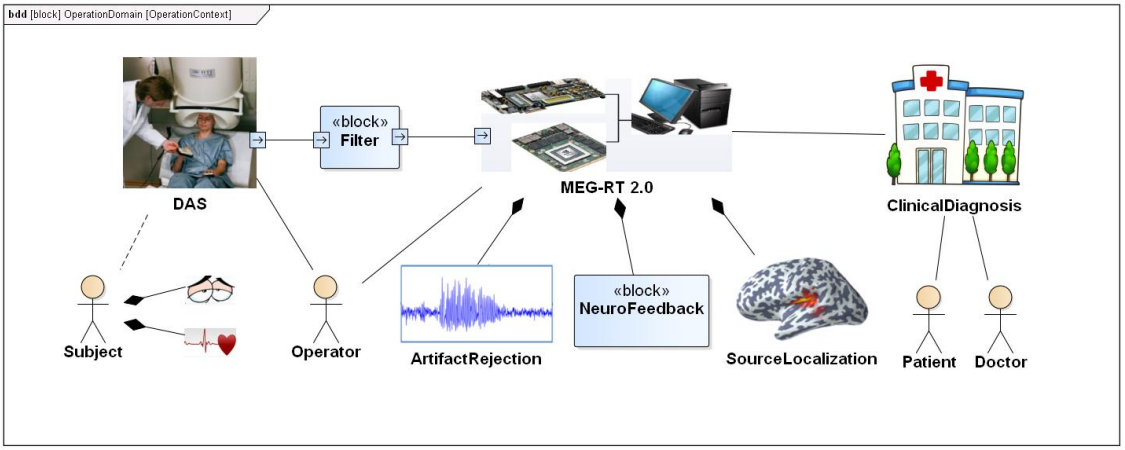

Figure 5.1: Contexts of MEG-RT 2.0 system. The solid line connecting two elements is the symbol of reference association, which indicates that they interact and associate with each other. The black filled-in diamond attached with a solid line represents a composition relationship.

5.1.3 Top-level requirements

Requirements form the elementary basis in system development and determine the capability, function or performance that the system has to achieve [1],[45]. Requirements for MEG signal processing need to be captured and traced in the system model. Section 5.1.1 provided a set of mission statements that result in more specific requirements. First, we try to identify all the MEG equipment stakeholders and collect as many requirements as possible. Then we formalize the requirements in 4 categories: *Functional*, *Qualitative*, *Quantifiable*, and *Unqualified*. Fig. 5.2 shows the top-level package structure that accommodates these categories of requirements. Functional requirements represent specific capabilities and major operations in the design of MEG-RT 2.0, including maintenance mode and

major mode. We focus on major mode including data decomposition, artifacts rejection, etc. Qualitative requirements refer to unmeasurable stakeholder needs including operator interface in Python, real-time operation, etc. Quantifiable requirements address properties, which can be measured and quantified, including the number of channels, the sampling rate. Other requirements are classified as unqualified requirements. A more detailed description of the package diagram Fig. 9.4 is given in Appendix 9.1.1.

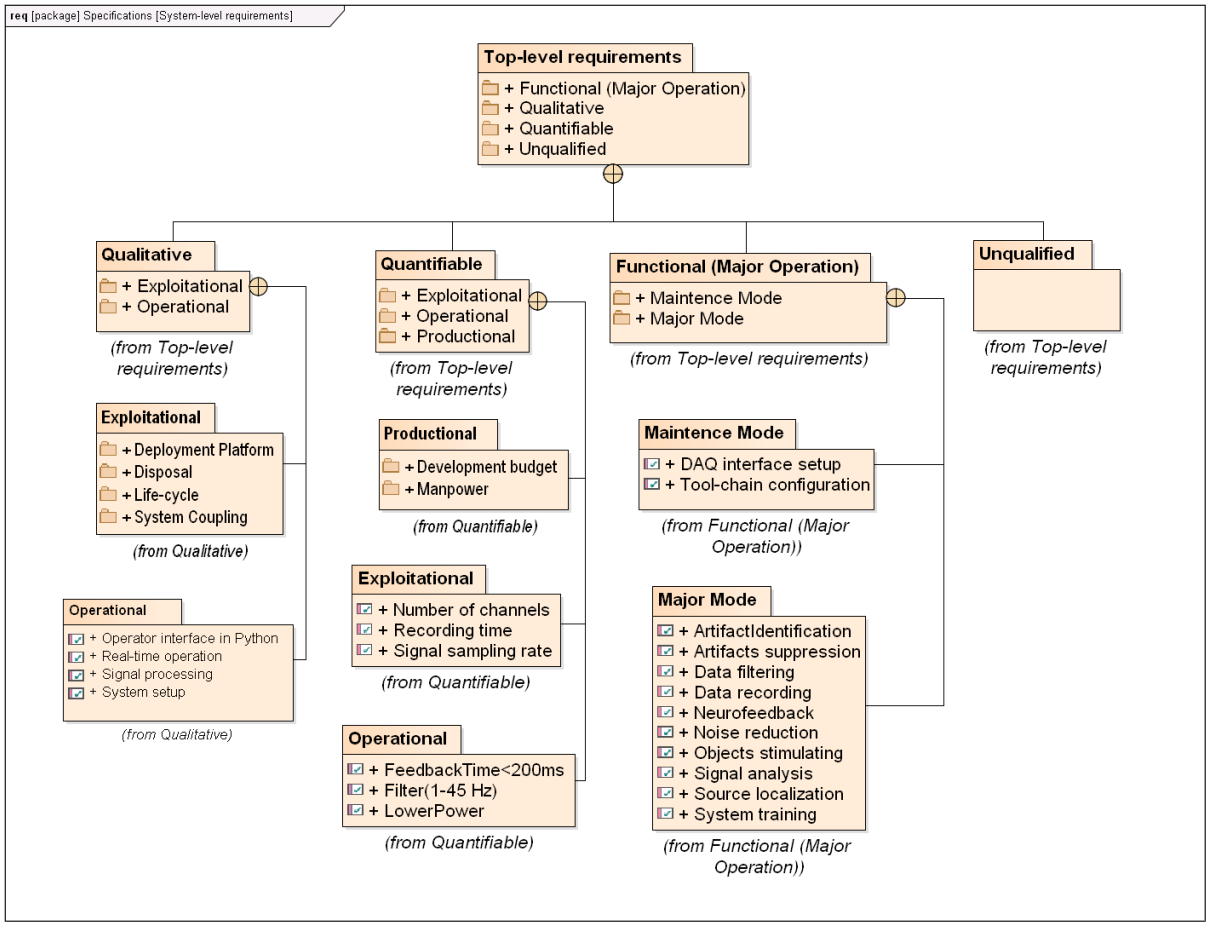

Figure 5.2: Top-level requirement diagram of MEG-RT 2.0. The package diagram shows the general organization of the model. The folder symbols represent packages that accommodate different categories of requirements. The "crosshair" notation that appears as a solid line with a circled plus sign conveys containment relationship to organize requirements into hierarchy.

5.1.4 Top-level requirements refinement

The top-level requirement diagram is elaborated further by the breakdown of stakeholder needs and constraints.

The top-level system model for MEG-RT 2.0 was developed in this study. As the approach steers down to the modeling and design of lower levels, this study only concentrates on the design of real-time artifact rejection in the MEG-RT 2.0 workflow. That is because artifact rejection was either

neglected or simplified in previous studies [8],[11],[12],[19]. The method used for noise and artifact removal in this work is termed OCARTA [7].

Fig. 5.3 shows the data measurement requirements in MEG signal pre-processing. The data specifications are shown in detail in Fig. 5.8 in section 5.2.2. In addition to relationships among requirements, traceability from data requirements to *Value Type* blocks is also established.

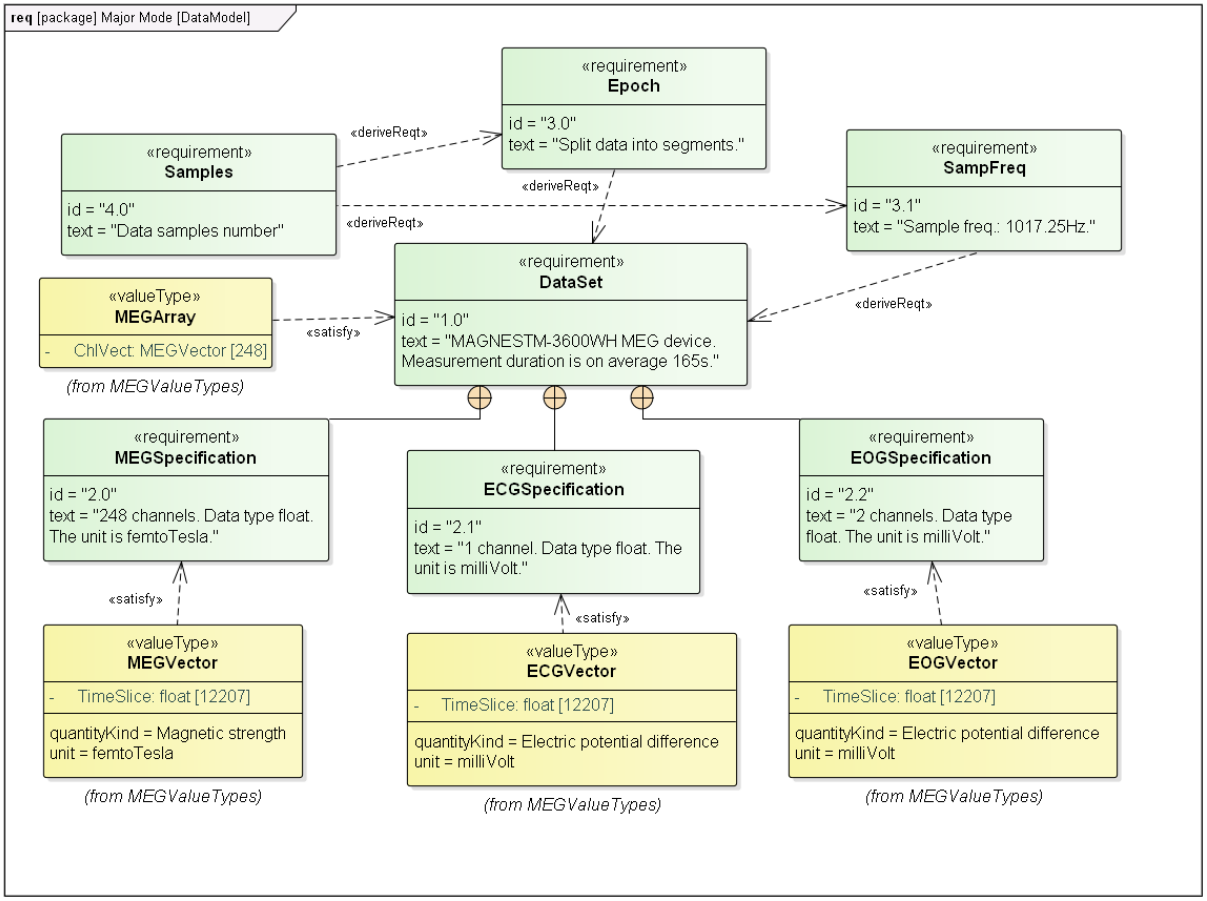

Figure 5.3: Data type requirement diagram. A requirement contains two properties: *id* and *text*. They are user defined as SysML does not dictate what they can be. The dashed lines with an arrowhead represent relationships to establish traceability among requirements or from requirements to structural and behavioral elements.

This section describes the breakdown of the mission statements into a cohesive set of requirements [43]. All essential requirements for MEG signal processing are clearly described so that they can be satisfied and traced to the system design. Fig. 5.4 depicts requirement decomposition and the initial derivation of requirements in MEG signal pre-processing. The compound requirement *MEGPreprocessing* is separated into simpler requirements. The requirements of data processing methods are derived from an analysis of each functional requirement of the system. In addition to relationships among requirements, traceability from system requirements to behavioral and structural elements is also established. A more detailed description of the requirement diagram and traceability is given in Appendix 9.1.3 and 9.1.4.

5.2 System structure modeling

System static structure models the blocks of which the system is composed. As shown in Fig. 5.5, structural modeling includes four steps: *Model the Problem Domain*, *Define Blocks*, *Allocate Requirements to Blocks* and *Define Ports* [55]. A more detailed description of the structural diagrams is given in Appendix 9.1.1.

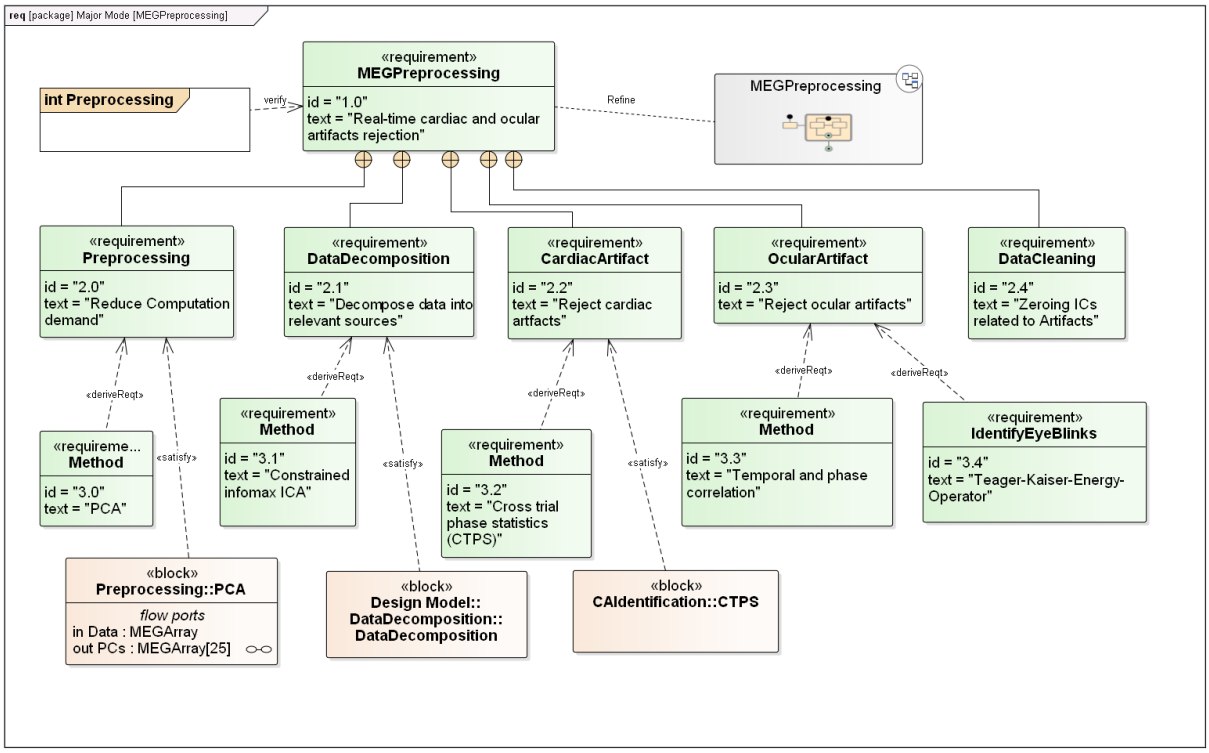

Figure 5.4: Requirement diagram of MEG signal pre-processing. The *derive* requirement relationship is represented by a dashed line with an arrowhead and attached with *«deriveReqt»* stereotype. It indicates that the requirement at the tail end is derived from the one at the arrowhead side. The refine relationship conveys that the requirement at the tail side is more concrete than the one at the arrowhead end. The satisfy relationship indicates the block at the tail end fulfills the requirement at the arrowhead end.

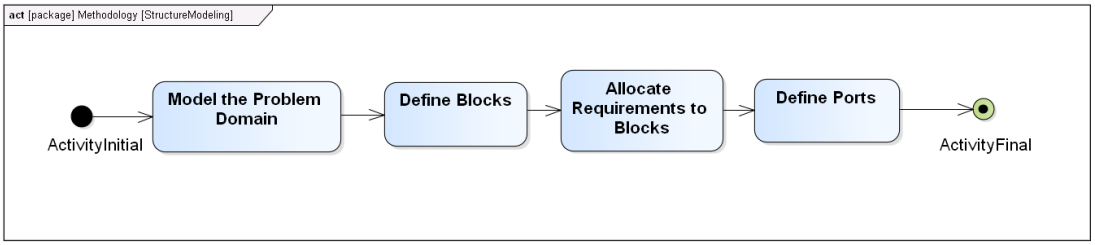

Figure 5.5: Workflow of structure modeling roadmap. The roadmap starts with domain modeling, which defines a set of abstractions based on things in the real world. Then the system is modeled using block diagrams. BDDs describe the composition of a block by relating nested blocks to each other using the composition relationship. IBDs specify in detail how each of the block 'parts' are connected together to form the subsystem. Afterwards, block elements are allocated to relevant requirements to establish traceability. Finally, ports are defined to show data flow.

5.2.1 Domain modeling

Domain modeling describes real world (problem domain) entities and the relationships between them, which models the system in the context of its environment [82]. Fig. 5.6 shows a domain model for MEG signal processing. The operation domain model defines a set of abstractions based on real-world elements external to the system, including environment noise, clinical diagnosis, neuro-feedback, etc. [55].

The problem domain model is intended to depict the "system" under which *MEG-RT 2.0* will operate. Fig. 5.6 shows the systems that interact together with *MEG-RT 2.0* [55]. The *OperationDomain* is defined as a system containing other subsystems, including the humans (*Operator*, *Subject*, etc.) who directly or indirectly interact with the system, external systems (*DAS*, *Filter*, etc.) that communicate with the system, and environmental elements (*EnvironmentNoise*, etc.) that could impact the system. The operational domain model efficiently provides an overview of the system and how it will operate [82]. Details of the *OperationDomain* system are further elaborated in the *OperationDomain*'s BDD shown in Fig. 5.1.

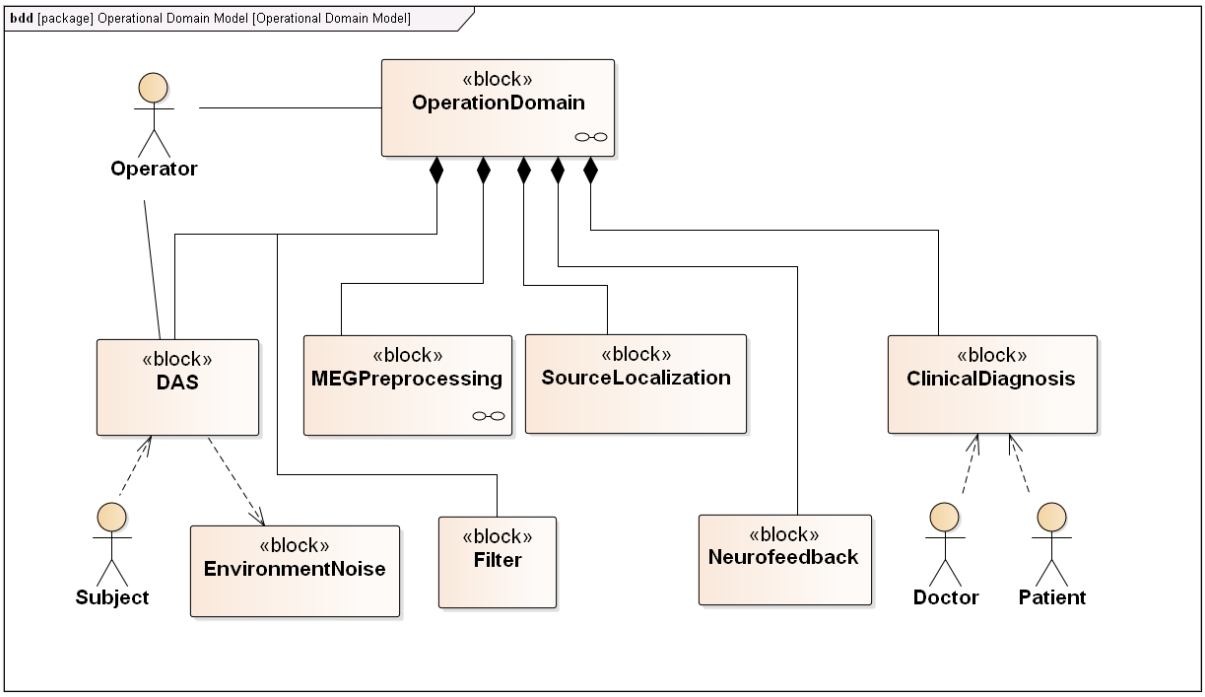

Figure 5.6: Domain model of MEG signal processing. The dashed line with an arrowhead conveys that the element at the tail end depends on the element at the arrowhead end. The black filled-in diamond attached with a solid line represents a composition relationship, indicating that *OperationDomain* block is made up of *DAS* block, *MEGPreprocessing* block, *SourceLocalization* block, *Neurofeedback* block and *ClinicalDiagnosis* block. The solid line between two blocks is the notation of reference association relationship, which indicates that *Operator* interacts with *DAS* block and is associated with *OperationDomain* block.

5.2.2 System architecture of MEG signal pre-processing

A block is the fundamental modular unit of structure in SysML. Blocks contain different categories of features, including structural features, behavioral features and constraints to model any type of entity within the system of interest or in the system's external environment [43],[44]. Block diagramming in SysML models system structures with respect to hierarchy and interconnection. There are two types of block diagrams: block definition diagram (BDD) and internal block diagram (IBD).

The BDD is used to display various kinds of model elements (blocks, actors, value types, interfaces, etc.) and their structural relationships (associations, generalizations and dependencies) to express information about a system's structure [44]. Fig. 5.7 shows the child BDD of *MEGPreprocessing* block in Fig. 5.6 in order to highlight the details. The composite association for the *MEG-RT 2.0* block is shown: the *MEGSignalPreprocessing* block is composed of four main sub-blocks: *InitialTraining*, *DataDecomposition*, *ArtifactIdentification*, and *DataCleaning*. Additionally, value properties (e.g., *meg_raw: MEGArray*) are listed in the second compartment of a block, which hold parameter values in MEG signal processing. The flow ports are listed in the third compartment of the *MEGSignalPreprocessing* block. The flow ports model the data that flow in and out of the *MEGSignalPreprocessing* block, which are detailed in the corresponding IBD. In the third compartment of part properties are operations of the relevant block, which represent behaviors the blocks perform.

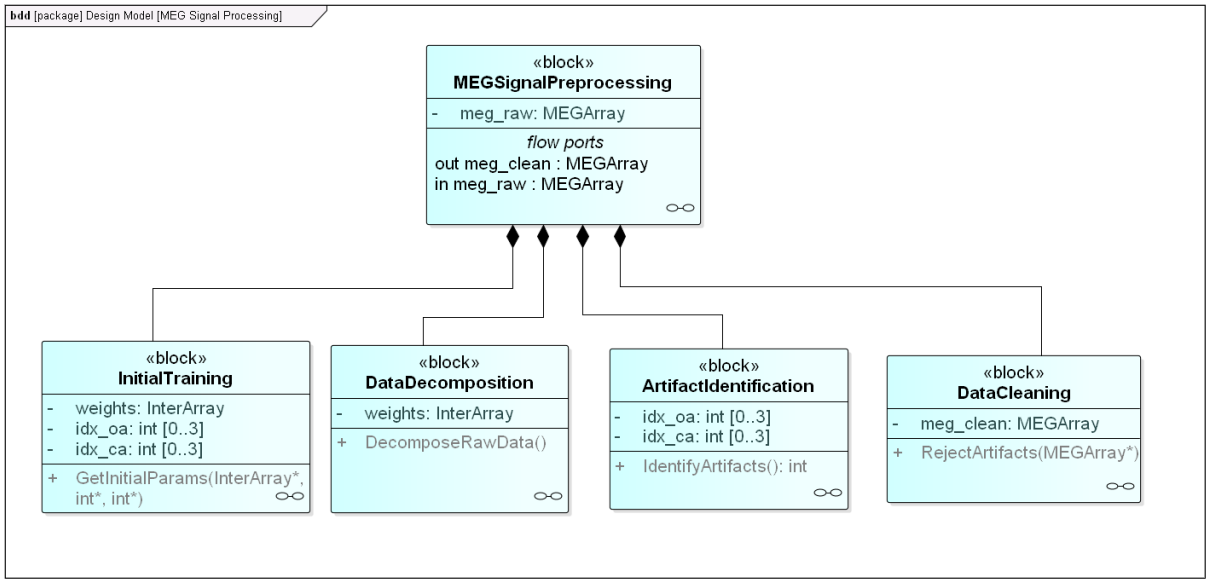

Figure 5.7: Top-level BDD. A block is represented by a rectangle applied with the stereotype «block». In the second compartment of a block value properties are listed. The "eyeglass" icons indicate that detailed child diagrams (BDDs and IBDs) exist.

It is of essential importance to analyze the information or data handled by the system. SysML provides "DataTypes" to define the data of MEG-RT 2.0 and its subsystems. The MEG data recordings tested for real-time analysis were measured using a 248-channel whole-head MEG device (MAGNESTM-

3600WH MEG). The MEG experimental setup is detailed in [9]. Major information of the MEG recordings of the experimental data to be used for the verification of the real-time artifact rejection is listed in Tab. 5.1.

Table 5.1: Test MEG data details ([9])

sampling rate	1017.25Hz
bandwidth	0.1-400Hz
number of presented stimuli	120
stimulus frequency	1000 Hz
stimulus duration	50 ms
inter stimulus interval	2.0 ±0.5 s
number of data samples (mean)	168098
experiment duration (mean)	165 s

The MEG recordings are split into segments of 12s for real-time processing, which attributes to 12,206 data points in each segment. Processing of one segment is based on the prior calculations of the last segment as the data segments overlap within a range. Fig. 5.8 maps the MEG signal description in Tab. 5.1 into a BDD of value types. A value type specifies the data structure of a quantity, which is used to type the value properties of a block.

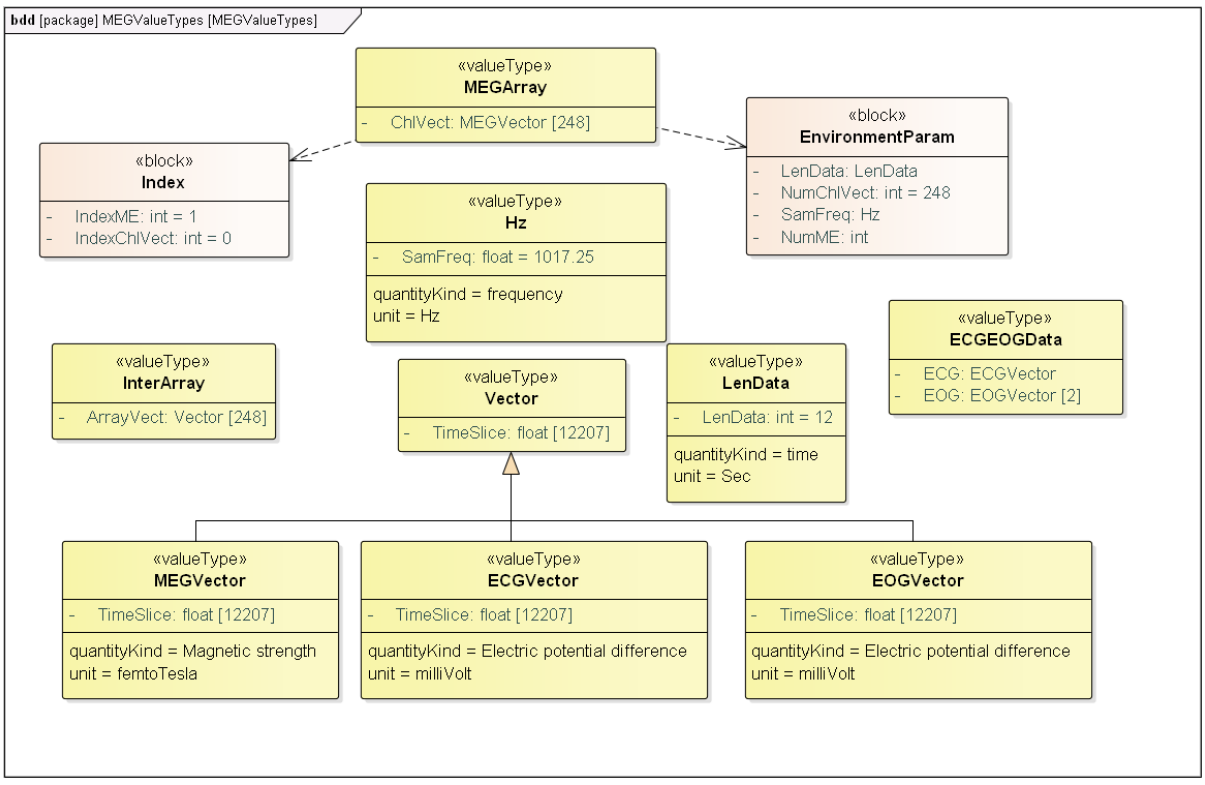

Figure 5.8: Data types of MEG signal pre-processing. User-defined value types are supported. The symbol for a structured value type is a rectangle applied with the stereotype *«valueType»*.

5.2.3 System internal structure

The IBD is a complementary view to the BDD. An IBD specifies the internal structure of a block with respect to how its parts are interconnected [43],[44]. Fig. 5.9 shows the IBD of *MEGSignalPreprocessing* block in Fig. 5.7. It conveys the part properties of the *MEGSignalPreprocessing* block and the related connectors. Moreover, it is allowed to specify multiple levels of nested parts in a single view by showing part symbols within part symbols, e.g., the *it:InitialTraining* part property [43],[45],[46].

5.3 System behavior modeling

Behavior models visualize, specify, construct, and document the dynamic behaviors of the system during the interactions with users and the environment [55],[83]. In SysML, the behavioral diagrams include use case diagrams, sequence diagrams, state machine diagrams, and activity diagrams as plotted in Fig. 3.1.

Use-case diagrams describe the high-level functionality of the system and model interactions between users and the system. Sequence diagrams detail use cases and model interactions among parts of a block. State machine diagrams describe event-driven behaviors that are not user-centric Activity diagrams convey system behaviors in terms of the flow of inputs, outputs, and control.

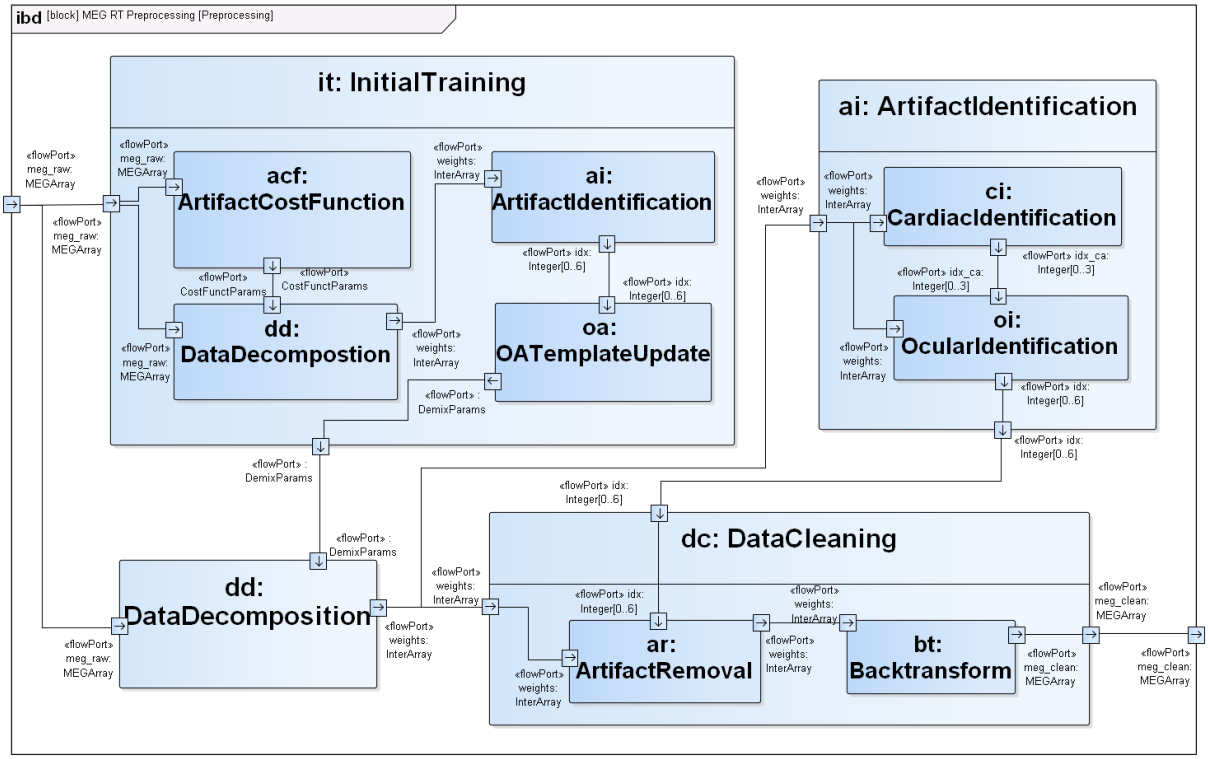

Figure 5.9: Top-level IBD. The notation of a part property is a rectangle with a solid border and a connector a solid line. The small squares straddling the borders of parts are ports defined to illustrate data flow. The arrow within a port conveys the direction of data flow [46].

Behavior modeling is further divided in two parallel paths, one path focuses on use cases and interactions, the other on state machines and activities, as shown in Fig. 5.10 [55]. In the Enterprise Architect toolkit, traceability from requirements to behaviors in the system model is created by allocating requirements to behavioral elements using a simple drag-and-drop [55], which ensures the consistency of the design. A more detailed description of the behavioral diagrams and the relationships among the elements is given in Appendix 9.1.2.

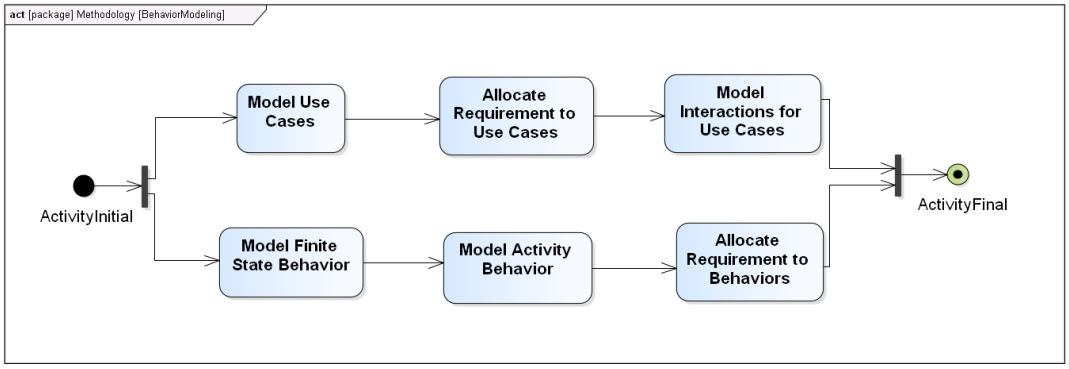

Figure 5.10: Behavior modeling roadmap ([55]). The roadmap has two parallel branches, one for use cases and the other for state machines. Each branch includes allocation of requirements to model elements (use cases or states). Use cases are described in natural language at the high level, and are detailed on interaction (sequence) diagrams.

5.3.1 System interactions of MEG signal pre-processing

Use case diagrams are used to model the interactions between users and the system. It shows how a system is invoked and used by external entities. A use case diagram is a black-box view of the system functionality.

Fig. 5.11 shows the use cases for the MEG signal processing. The central use case of the *MEG-RT 2.0 Complex*, termed *Perform MEG Study*, has two subtype use cases: *Perform RT MEG Preprocessing* and *Perform Offline MEG Study*. Actors may interact directly or indirectly with the system. Four actors are shown in Fig. 5.11: an *Operator* (a stick figure sign) who operates the system, an *MEG Database* (a rectangle with the keyword «actor»), an *MEG Acquisition System (DAS)*, and a *Subject (Patient)* who interacts indirectly with the system through the *MEG Acquisition System (DAS)*.

Additionally, use cases can be related to one another by such relationships as *inclusion*, *extension*, and *classification* [43]. The *inclusion* relationship conveys that, for example, when the use case *Perform RT MEG Preprocessing* is invoked, the included use cases (*Record Online Data*, *Pattern Stimuli Application*, and *Train the System for Particular Subject*) are also executed. “An extending use case is any use case that is the source—the element at the tail end—of an extend relationship” [44]. The *extending* use case *Online Measurement Abort* describes some exceptional behaviors in the interaction, such as abnormal subject behavior, defined in extension points.

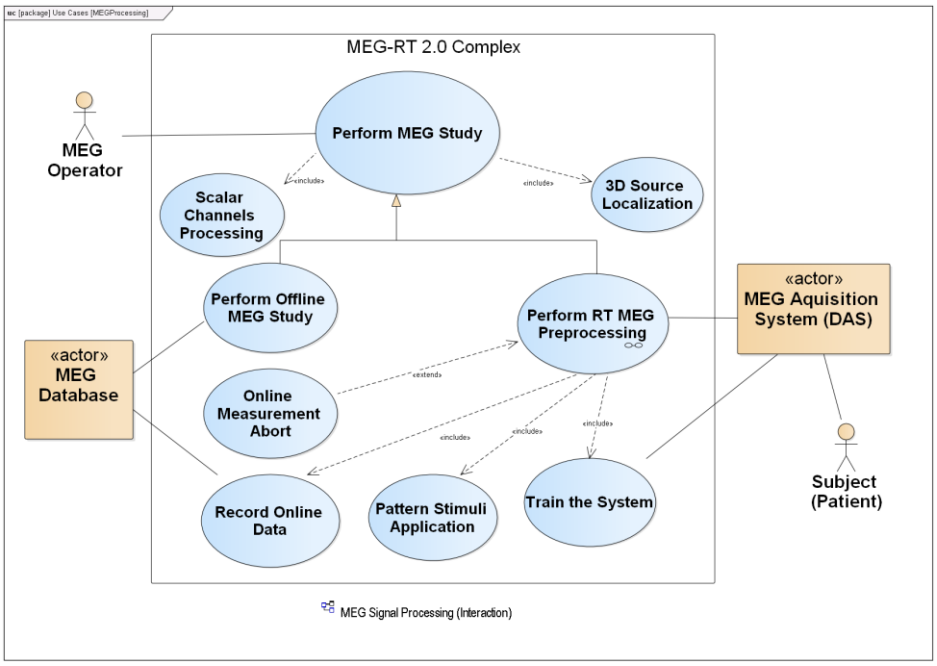

Figure 5.11: Top-level use case diagram of MEG signal pre-processing. The symbol for a use case is an ellipse. The dashed line with an open arrowhead and the stereotype «extend» represents an extend relationship. The extend relationship indicates that when the use case at the arrowhead side gets invoked, the extending use case may be optionally executed.

The eyeglass icon on the use case *Perform RT MEG Preprocessing* indicates that it is a composite element, which indicates it is a pointer to a child diagram [55],[82]. The child diagram shown in Fig. 5.12 presents details on data decomposition, which is a most important step in the use case *Perform RT MEG Preprocessing*.

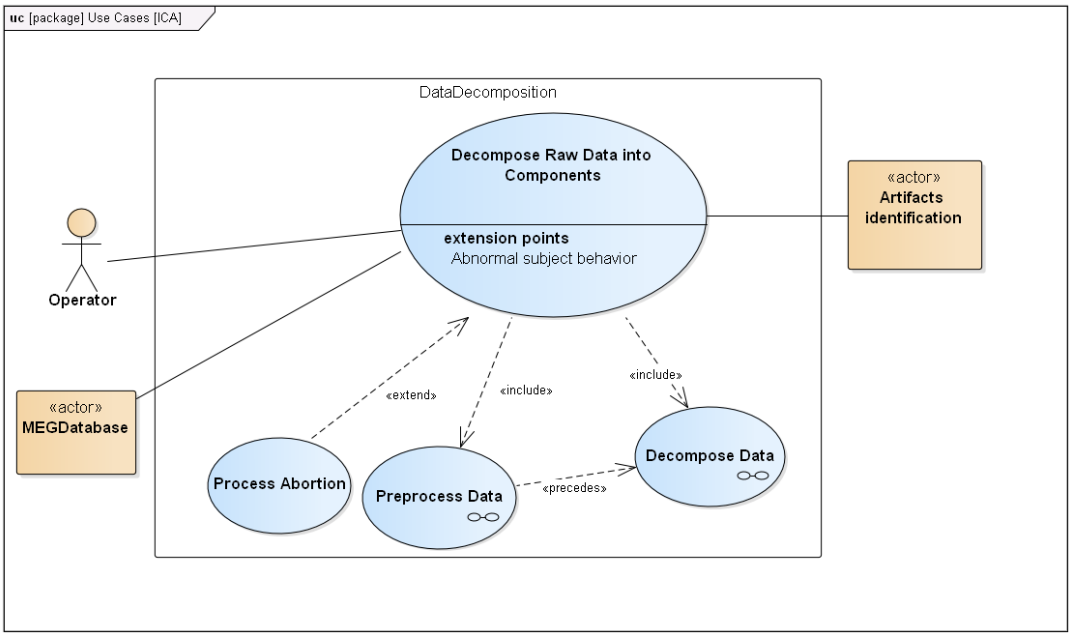

Figure 5.12: Use case diagram of data decomposition

Enterprise Architect supports hyperlink elements to point to other diagrams, which establishes traceability among the diagrams. The hyperlink element *MEG Signal Processing (Interaction)* in Fig. 5.11 links to the sequence diagram in Fig. 5.13, which shows how sub-modules of MEG pre-processing interact by exchanging messages. Additionally, *MEG-RT 2.0 Complex (Use Case)* in Fig. 5.13 is a link back to the use case view in Fig. 5.11.

Sequence diagrams depict the interaction between system components as a sequence of message exchanges [43]. As plotted in Fig. 5.13, the structural components of MEG pre-processing are represented by lifelines (a rectangle attached with a dashed line), *InitialTraining*, *DataDecomposition*, *ArtifactIdentification*, and *DataCleaning*. The dashed line represents the relevant lifetime of the structural component. Lifelines are participants involved in an interaction which interact with each other by exchanging messages. A message is represented by a line with an arrowhead between two lifelines with the message name floating over the line. In *InitialTraining*, the first 12 s of recording of each measurement are used as training data set to produce initial parameters, including *CalculateCostFunction*, *DecomposetheData*, *IdentifyArtifacts*, and *UpdateArtifacts* [9]. In real-time MEG signal pre-processing, each measurement is split into segments (e.g., 12s each) and the segments are processed separately [7],[9].

As mentioned earlier, the processing of one segment is based on the prior calculations of the last segment as the data segments overlap within a range. Therefore, based on the prior results obtained from *InitialTraining*, the design forms a 2-stage pipeline structure, with *DataDecomposition* and *ArtifactIdentification* being the first stage and *DataCleaning* the second stage. In addition, it is also permissible for a lifeline to send messages to itself, as the case *DataDecomposition* shows. A constrained independent component analysis (cICA) approach is used to decompose MEG data into statistically maximal independent components, including updating data with prior results (*UpdateRawData*), transforming the data with the cost function (*FitSigmoid*) and then computing the demixing matrix iteratively (*ICALearning*) [9]. SysML supports a mechanism termed combined fragments to model complex control logic of interactions. The control logic type is defined by a string (an interaction operator) that appears in a pentagon in the upper-left corner of the combined fragment rectangle [44]. The many computation iterations of *DataDecomposition* are modeled with a loop operator loop (0, 200), as shown in Fig. 5.13. The loop stops when the demixing matrix converges. When raw data areF successfully decomposed into independent components, *ArtifactIdentification* identifies cardiac components based on CTPS (*IdentifyCardiacArtifact (CAIdx)*) and ocular components based on a spatial and temporal correlation (*IdentifyOcularArtifact (OAIdx)*) [6],[9],[44]. Finally, *DataCleaning* rejects artifacts components (*RemoveArtifacts (Weights_clean)*) and back transforms data to MEG space (*Backtransform(Meg_clean)*). A *par* operator is used to indicate that *DataDecomposition* and *ArtifactIdentification* are done in parallel to *DataCleaning*.

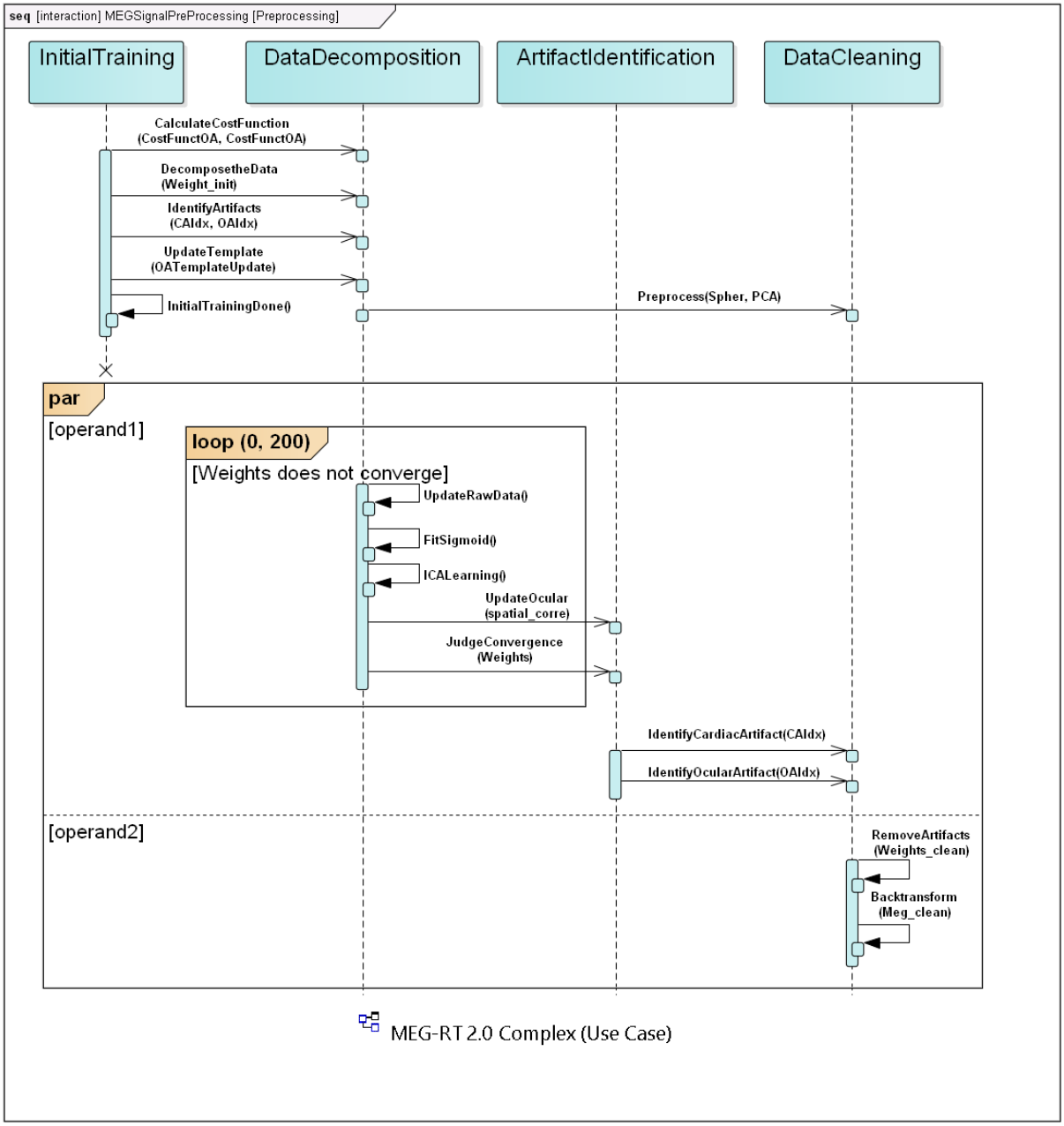

Figure 5.13: Top-level sequence diagram of MEG signal pre-processing. A rectangle with a dashed line attached to it represents a lifeline (Participants of an interaction). A solid line with an arrowhead between two lifelines specifies data exchange. The rectangle placed over one or more lifelines and encapsulates one or more messages is termed combined fragment, which allows adding control logics including loop, parallel operations, etc.

5.3.2 System behaviors of MEG signal pre-processing

During the pre-processing process of MEG data, data decomposition and artifact identification are performed in parallel to the data cleaning. Hence there are two data streams in the pre-processing workflow. One data stream is used to update the demixing matrix and the other is used to reject the artifacts in real-time. Enterprise Architect supports Hardware Description Language (HDL) generation

based on state machine diagrams. State machine diagrams describe system behaviors with respect to operation states, triggers and system actions [55].

Fig. 5.14 shows the detailed state machine diagram for MEG signal pre-processing. It starts with an initial pseudostate, a small, filled-in circle named *GlobalInitial*. Then a transition (a solid line with an arrowhead) leads to *Idle* state. The notation of a state is a round-cornered rectangle, which displays state name and other information. When the guard *selection = process* evaluates to be true, it transits to the entry point (a small, hollow circle) of *MEGSignalPreprocessing* state machine. Again, the state machine starts in an initial pseudostate, then transits to *ReadData* state to read raw data from memory and Filter state to filter data. The strings that follow *entry/*, *exit/*, or *do/* are internal behaviors to be executed. After that, the transition reaches *InitialTraining* state machine. Starting from an initial pseudostate, the transition goes to a fork node (a line segment that indicates concurrency) and then enters *CalculateCostFunction_CA* state, *CalculateCostFunction_OA* state and *CalculateSpatialTemplate* state concurrently.

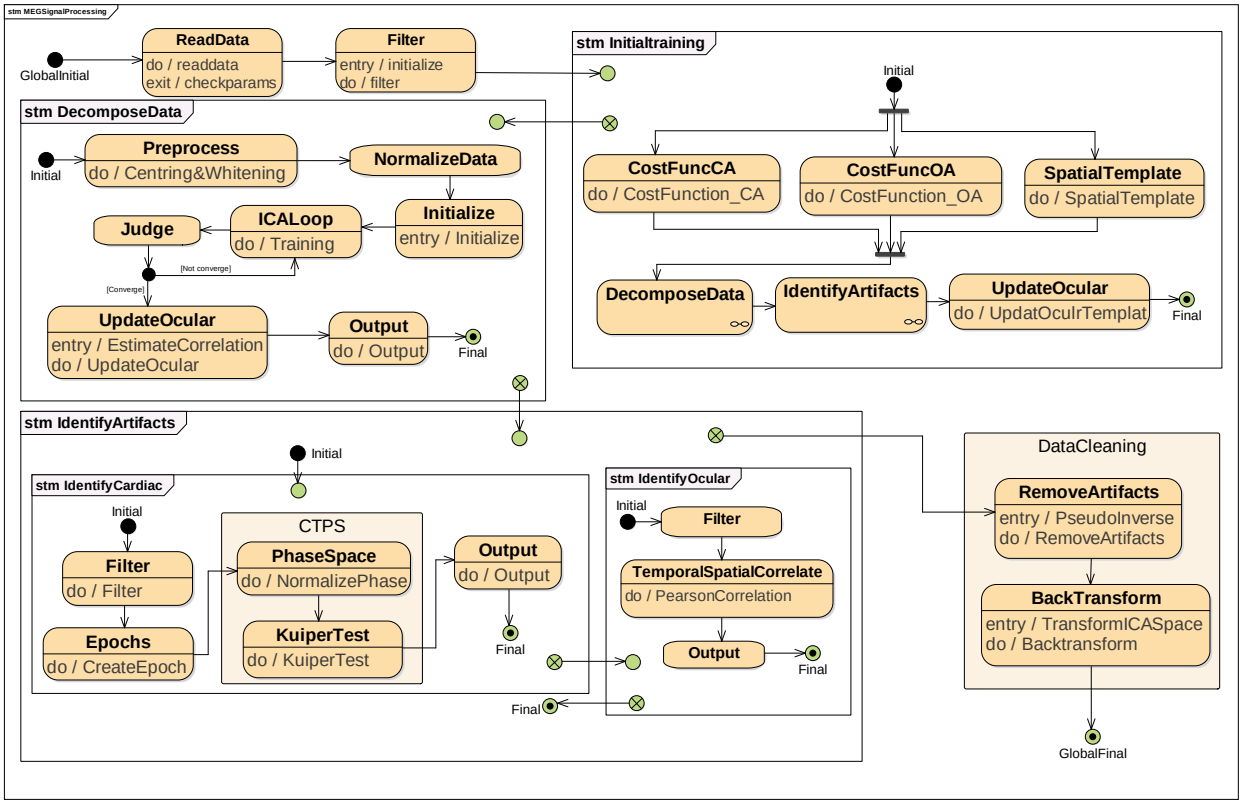

Figure 5.14: Top-level state machine diagram. State machine diagrams allow nesting of sub-states on a single diagram. The “eyeglass” icons on the state rectangles indicate that child diagrams exist [55].

Afterwards, three transitions reach a join node (a line segment that marks the end of concurrency) and then one outgoing transition from the join node goes through the following states and ends in a final state (a small, filled-in circle surrounded by a larger circle [44]). Next, the transition goes from an exit point (a small circle containing an X) and enters the entry point of *DecomposeData* state machine.

Similarly, the model transits through *DataDecomposition* state machine, *IdentifyArtifacts* state machine *RemoveArtifacts* state and *BackTransform* state until *GlobalFinal* final state. Later on, the transition returns from an exit point to Idle state on condition that the guard *selection = idle* evaluates to be true.

MEG signal pre-processing mainly involves matrix–matrix multiplications or matrix–vector multiplications, for which parallel computation is optimal [9]. Enterprise Architect contains numerous features to help transform behavior models (activity diagrams, state machine diagrams) into executable code for parallel platforms including FPGA, etc. [55]. Additionally, MDG Integration technology integrates the benefits and rich modeling power of Enterprise Architect and UML with IDEs such as Visual Studio and Eclipse [82].

Activity diagrams specify a controlled sequence of actions that transform inputs (matter, energy, or data) to outputs [43],[44]. Fig. 5.15 shows the top-level activity diagram for MEG signal pre-processing. The initial node (a small, filled-in circle) *ActivityInitial* marks the starting point of *MEG Signal Preprocessing* activity. Then the control token on the initial node concurrently triggers the execution of *ReadData* activity (a round-cornered rectangle) and *DemixParams* action via an outgoing control flow (a line with an arrowhead) and a fork node. That is because *DataDecomposition* and *ArtifactIdentification* are done in parallel to *DataCleaning*. Furthermore, an object node models the input or output parameters of an action or activity. The notation for an object node is a small rectangle attached to an action or activity with the object node name and data type floating beside it, e.g., *ReadData* activity has one input parameter *Meg_raw* and one output parameter *Meg_raw* of data type *MEGArray*. When *ReadData* activity is executed, the output parameter *Meg_raw* is transmitted to *Filter* activity by an object flow (a solid line with an arrowhead). *InitialTraining* activity, *DecomposeData* activity and *IdentifyArtifacts* activity are enabled in a similar manner. When the *IdentifyArtifacts* activity is executed, the output parameters flow to *dp: DemixParams* send signal action (a convex pentagon shaped like a signpost), a specialized action that asynchronously sends matter, energy, or data [44]. *DemixParams* accept event action (a rectangle with a triangular section missing on one side [43]) waits for the parameters of *dp: DemixParams* send signal action to arrive asynchronously. A *send signal action* and an *accept event action* work together to model communication between two concurrent work flows. Then the following activities are executed. Finally, the activity final node *ActivityFinal* indicates the termination of the entire activity.

“A diagram of the model is never the model itself; it is merely one view of the model” [44]. Different diagrams show different model views of the system from different perspectives. As many behavioral views as necessary were created, including use case diagrams, sequence diagrams, state machine diagrams, and activity diagrams, to detail the dynamic behaviors of the system [55]. Moreover, traceability is established by allocating requirements to the model elements.

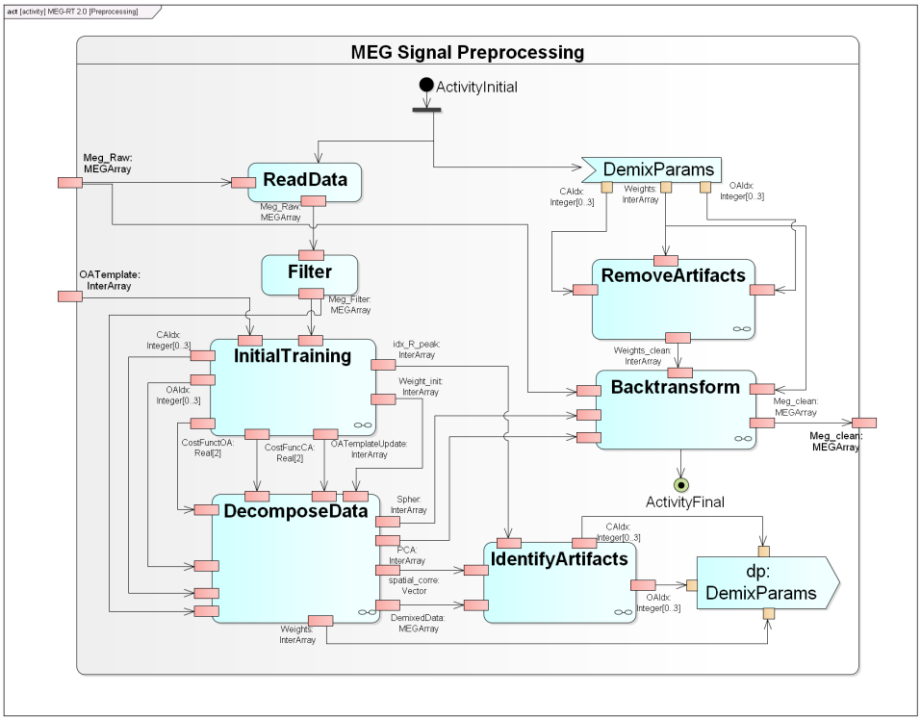

Figure 5.15: Top-level activity diagram. The notation for a basic action is a round-cornered rectangle. The notation for a send signal action is a convex pentagon shaped like a signpost, which sends a signal instance to a target. The notation for an accept event action is a concave pentagon that looks like a rectangle with a triangular notch cut out on one side. An accept event action is the partner of the send signal action and conveys that the activity must wait for an asynchronous event occurrence.

5.4 Performance analysis

5.4.1 Constraints and parametrics definition

SysML parametric models include constraint blocks and parametric diagrams, which together impose mathematical relationships on the values of the design.

Constraints are expressed as equations or inequalities whose parameters are bound to the properties of the design [43]. A constraint block is a special kind of block used to encapsulate constraint expressions [44]. Constraint blocks have two main features: constraint parameters and a constraint expression to impose equations or inequalities. The BDD in Fig. 5.16 conveys that the *MEGSignalPreprocessing* constraint block is composed of the four simpler blocks to build a more complex computation. In the constraint blocks defined by Enterprise Architect, a constraint parameter has the notation of a small rectangle floating inside the block. A constraint expression is defined by adding JavaScript, Jscript or VBScript scripts to a block, which provides the underlying mathematical foundation for further analysis or simulation.

MEGSignalPreprocessing uses these computations mainly to decompose MEG raw data into independent components, for subsequent identification and rejection of artifacts.

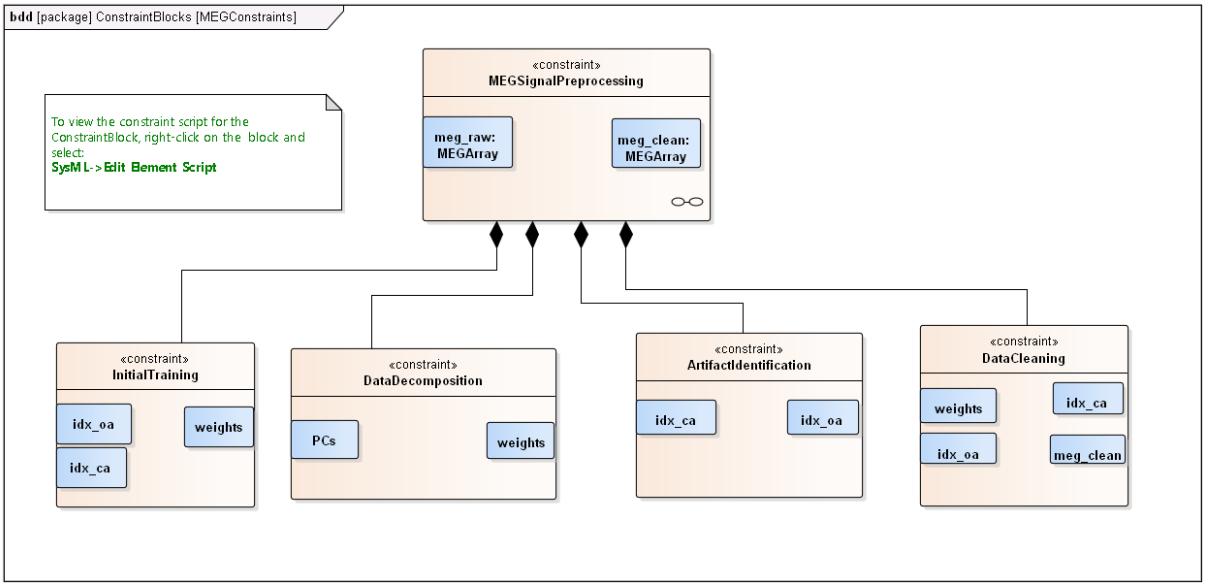

Figure 5.16: Constraint blocks of MEG signal preprocessing

In Figure 5.16, constraint blocks contain properties to specify the input and output parameters. Additionally, scripts are embedded in the constraint blocks to depict the executable components. However, Fig. 5.16 does not specify how the constraint blocks and expressions are connected to each other to accomplish a complex computation; this is done by a parametric diagram [44],[46], which is a restricted type of IBD.

5.4.2 Parametric diagrams and performance analysis

The mathematical relationships in SysML parametric models can specify the physical properties (e.g., equations, inequalities, or parameter bindings) or the non-functional properties (e.g., performance, speed, power dissipation, or reliability) of a system. Fig. 5.17 shows a parametric diagram for the constraint block *DataDecomposition* from Fig. 5.16, where the *DataDecomposition* is depicted as the frame of the parametric diagram. The constraint properties in the diagram are usages of the constraint blocks nested within *DataDecomposition*. The parameters of the constraint properties are bound to each other and to the parameters of *DataDecomposition*. Together, they show the computation flow of the data decomposition process in MEG signal pre-processing. Specifically, Fig. 5.17 defines precisely how the sub-modules of MEG data decomposition are related. These relationships model the engineering physics and behaviors of the data propagation between the ports of the components within the ICA subsystem. Parametric constraints represent equations addressing the inputs and outputs. Constants are given in the form of value properties.

Based on the established models and analysis, the design is first partitioned to a certain technological basis to best suit system requirements and constraints. The development of MEG-RT 2.0 will be based on a single workstation platform combining FPGA and GPU co-processing boards, including a software framework for real-time MEG data analysis. The project will cover developments for data

pre-processing steps utilizing a Xilinx Virtex-6 FPGA board as well as the integration of hardware modules connecting the software framework of the workstation. The GPU is connected to the FPGA for exchanging intermediate data. Further 3D reconstruction and multi-modal visualization (i.e. MEG and magnetic resonance imaging [MRI]) takes place in the GPU. As the work focuses on real-time artifact rejection, performance analysis is carried out on the MEG signal preprocessing unit.

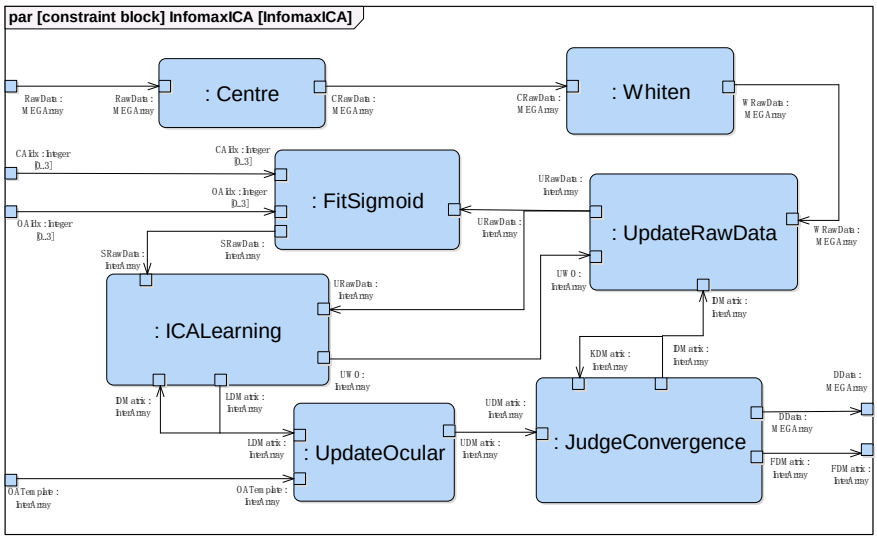

Figure 5.17: Parametric diagram of MEG signal processing

Given the parametric analysis and the deduced data-stream models, the MEG signal pre-processing can be quantitatively estimated for performance. In the depicted systems engineering models, the design performance depends on the amount of data exchanged through the modules. Thus, the performance is assessed by calculating the data traffic from the top module to lower level modules.

The present study uses a 16-bit fixed-point numeric. One bit is used for the sign, one bit for the integer part, and 14 bits for the fractional part. Even though the double precision floating point is of higher precision, fixed-point number representation is more practical for hardware designs. The disadvantages of realizing floating-point units on FPGA are high resource and high clock frequency demands. Although previous studies have implemented floating-point computation architectures on FPGAs, very few practical applications exist [84]. On the other hand, modern FPGAs are equipped with numerous hard-wired fixed-point DSPs suitable for fixed-point arithmetic realization, which potentially improves performance and power efficiency. Convergence has been reported in [84],[85],[86],[87] for the FPGA implementations of ICA and deep neural network training using 16-bit fixed-point computation units. The rough precision of the 16-bit fixed-point numeric can reach 0.000030518. Before the computation, normalization is performed on the MEG data to avoid overflow. According to the data specification in Tab. 5.1, the volume of MEG data is exceedingly large for FPGA on-chip BRAM (block random access memory). Thus DDR3-SDRAM external memory is used

to store MEG data and intermediate results of large volume. The Xilinx DDR3 IP core is utilized to instantiate and control the DDR3 RAM with the read/write width of 256 bits and clock frequency of 400 MHz.

The hardware resource model is based on counting of clocked memory (in bits) required for allocation of local data aggregates of the FPGA. Although it is not intended to yield a precise flip-flop budget for each processing unit of the FPGA implementation, it can predict each design module's order of complexity with a high degree of certainty. Tab. 5.2, Tab. 5.3, Tab. 5.4 and Tab. 5.5 presents the prognostic resource model computed for different modules of the MEG signal pre-processing system.

Tab. 5.2 shows the data traffic of different sub-modules in PCA. It mirrors the application performances in different sub-modules depending on the data flow in the processing units. Tab. 5.2 also specifies the processing time (data in brackets estimate the processing time with an operating frequency of 100 MHz).

The PCA computation is separated into four steps. The data traffic bottleneck is the estimation of eigenvalues, where the calculation requires a number of iterations. The intermediate results of the *EigenAnalysis* and *PCMatrix* modules are small-volume and stored in the block RAM on FPGA for speeding up the computation by the short memory access time. It is important to note that *Centre* and *COV* modules involve processing very large dimension matrices. The input and output streams of the two modules have external SDDR3 as the memory bank, which adds to the processing time. Therefore, the upward and downward data streams of the two modules are split into three channels operating in parallel.

Table 5.2: Data traffic of PCA for performance estimation and total percentage of Virtex-6 flip-flop memory (in memory words)

Module	Centre	COV	EVD	PC	Percentage
Data traffic	3,027,336 (60.54 ms)	3,027,336 (375.36 ms)	61,504 (152.52 ms)	3,033,536 (75.68 ms)	41%

Tab. 5.3 specifies the traffic volume of different sub-modules in ICA and the application performances in different sub-modules are estimated based on the data flow in the processing units. The table 5.3 also calculates the processing time (data in brackets estimate the processing time with an operating frequency of 100 MHz).

The ICA computation is separated into three modules: data update, judge, and learning. The data traffic bottleneck is the learning module, where exponential computation and multiple matrix calculations are required. The intermediate results have small data volume and are stored in the block RAM on FPGA for speeding up the computation by the short memory access time. Prior knowledge from the previous data segment is incorporated during the computation. The measured signal mixtures are separated into underlying sources within one iteration, which greatly speeds up the decomposition.

Table 5.3: Data traffic of ICA for performance estimation and total percentage of Virtex-6 flip-flop memory (in memory words)

Module	Data update	Judge	Learning	Percentage
Data traffic	305,775 (8 ms)	625 (0.006 ms)	305,775 (158.68 ms)	68%

Tab. 5.4 shows the data traffic of different sub-modules in CTPS. Application performances in different sub-modules are presented based on the data flow in the processing units. Meanwhile, the processing time is analyzed (data in brackets specify the processing time under the operating frequency of 100 MHz).

The CTPS computation is divided into three modules, which are the Hilbert transform, the CDF estimation and the Kuiper test. The data traffic bottleneck is the Kuiper test, where exponential computation is involved. The intermediate results are of small data volume and stored in the block RAM on FPGA for speeding up the computation by the short memory access time.

Table 5.4: Data traffic of CTPS for performance estimation and total percentage of Virtex-6 flip-flop memory (in memory words)

Module	Hilbert transform	CDF estimation	Kuiper test	Percentage
Data traffic	509 (0.37 ms)	509 (0.15 ms)	509 (0.43 ms)	11%

The data traffic density of temporal and spatial correlation design is much lower as the computation is simple and only involves addition, vector–vector multiplication, and division. Thus, memory access takes up most of the computation time.

Table 5.5: Data traffic of temporal and spatial correlation for performance estimation and total percentage of Virtex-6 flip-flop memory (in memory words)

Module	Covariance	Variance	Division	Percentage
Data traffic	12,206 (0.21 ms)	12,206 (0.18 ms)	16 (0.08 ms)	4%

6 Implementation of MEG signal pre-processing

Based on the behavioral models, HDL code can be generated automatically, which is often sufficient for direct implementation in hardware. The algorithms covered in artifact rejection were implemented by modifying the generated Verilog HDL code. Chapter 6 is devoted to the discussion of these algorithms and their implementation. Each algorithm is described briefly, and then the implementation process is detailed..

6.1 Code generation

One of the biggest differences between SysML and UML is the ability to simulate portions of a SysML model, based on mathematical and physical laws that describe key aspects of the system [46],[55]. An advantage of Enterprise Architect over other SysML modeling tools is that Enterprise Architect is able to do the simulation within the modeling tool, as opposed to simply interfacing to external simulators, speeding up the convergence towards an engineering solution that meets the requirements. Model simulation can be applied to behavioral models and parametric models in Enterprise Architect. During the simulation process Enterprise Architect supports setting break-points, firing triggers, examination of variables, viewing the call stack and visually tracing the active nodes as the simulation proceeds [82].

Enterprise Architect supports code generation and reverse engineering very well in both hardware (VHDL, Verilog and SystemC) and software (C, C#, C++, Java, Python, etc.).

For the purpose of acceleration, reconfigurable hardware solutions such as FPGA are currently valuable approaches for tackling the computation complexity. We generate Hardware Description Language (HDL) code from MEG-RT 2.0 state machine diagrams. For the generation of code from behavioral models, all behavioral constructs should be contained within a class diagram, as plotted in Fig. 6.1. The design of the MEG-RT 2.0 functionality is contained in a multi-level state machine that is nested within the *MEGSignalProcessing* class. Then two triggers (*reset* and *clk*) are added to and associated with the top level state machine diagram. Meanwhile, the triggers are associated with the component's ports as shown in Fig. 6.1. After the preliminaries above, the desired HDL code is generated.

As with HDL code generation, the activity diagram should be nested within a class diagram to generate C++ code. The design of MEG-RT 2.0 functionality is modeled with a multi-level activity diagram, shown in Fig. 5.15. The VHDL and C++ code generated with Enterprise Architect are extremely detailed and robust, as shown in Fig. 6.2.

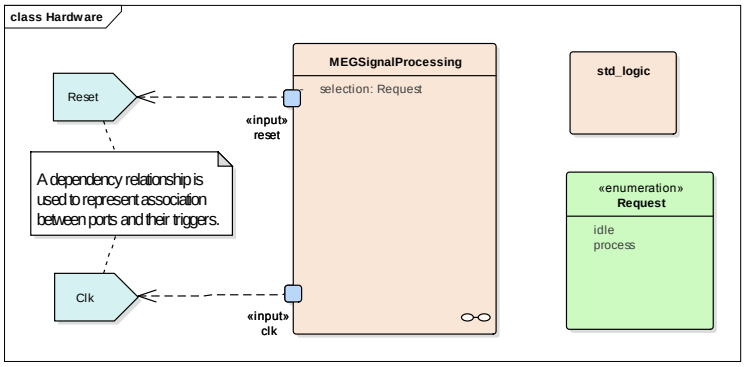

Figure 6.1: Class diagram for VHDL code generation. The convex pentagons represent triggers that drive the state machines; the small squares straddling the borders of parts are ports of data flow.

```
75      CLOCK_DIV4 : process(clockDiv2 )
76      begin
77          if(rising_edge(clockDiv2)) then
78              clockDiv4 <= not clockDiv4;
79          end if;
80      end process CLOCK_DIV4;
81
82      TRANSITION_LOGIC: process(currState )
83      variable bFlag : boolean;   begin
84          currTransition <= TT_NOTRANSITION;
85          case currState is
86                  when ProcessData_ICA =>
87                  bFlag := false;
88                  if((state=jconverg)) then
89                      bFlag := true;
90                      nextState <= ProcessData_ICA_Decompose_JudgeConvergence;
91                      transcend <= '0';
92                  end if;
93                  if((state=decompose)) then
94                      bFlag := true;
95                      nextState <= ProcessData_ICA_Idle;
96                      transcend <= '1';
97                      ProcessData_ICA_history <= currState;
98                  end if;
99                  if((state=idle)) then
100                     bFlag := true;
101                     nextState <= ProcessData_ICA_Idle;
102                     transcend <= '0';
103                 end if;
104             when ProcessData_ICA_Preprocess_Centre =>
105                 nextState <= ProcessData_ICA_Preprocess_Whiten;
106                 transcend <= '0';
```

Figure 6.2 (a): Excerpts of VHDL generated with Enterprise Architect

```
25 void OCARTA-C++::OCARTA( OATemplate, meg_clean, meg_raw)
26 {
27     // behavior is a Activity
28     self._picks = pick_types(meg_raw.info, meg=True, eeg=False, eog=False, stim=False, exclude='bads')
29     // Pick channels by type and names. meg: If True include all MEG channels.//
30     ;
31     //filter the data//
32     fi_mne_notch = jumeg_filter(fcut1=self.flow, fcut2=self.fhigh,
33                                 filter_type=filter_type,
34                                 remove_dcoffset=False,
35                                 sampling_frequency=meg_raw.info['sfreq'])
36     fi_mne_notch.apply_filter(meg_filt._data, picks=self._picks)
37     // default: flow=1;   default: fhigh=20//
38     ;
39     //initial training//
40     weights, idx_ca, idx_oa = self._initial_training(meg_filt)
41     //get some parameter//
42             nchan = self._picks.shape[0]
43             shift = int(self.shift_length * meg_filt.info['sfreq'])
44             nsteps = np.floor((self._nts1 - self._block)/shift) + 1
45             laststep = int(shift * nsteps)
46     ;
47     //decomposition and identify artifacts//
48     for istep, t in enumerate(range(0, laststep, shift)):
49     weights, idx_ca, idx_oa = self._update_cleaning_information(meg_filt, t, idx_end,
50                                                                 initial_weights=weights.T,
51                                                                 ca_idx=idx_ca, oa_idx=idx_oa)
52     ;
53     // get cleaning matrices//
54     iweights = pinv(weights)
55     iweights[:, idx_ca] = 0.     //remove columns related to CA//
```

Figure 6.2 (b): Excerpts of C++ code generated with Enterprise Architect

6.2 Implementation of PCA

A commonly used method for PCA is the eigenvalue decomposition (EVD) of the data covariance matrix. It is comprised of four steps:

1) the original data set is centered by subtracting the mean vector,

2) the covariance matrix is computed,

3) EVD is applied to the covariance matrix where eigenvalues and eigenvectors are computed,

4) an orthogonal transformation is applied onto the original data to get the principal components (PC).

Task 3) is the most computationally demanding step of performing PCA. There are various techniques to perform EVD. Considering the large dimensionality of MEG data, QR algorithm for PCA is adopted [34]. The QR algorithm is the decomposition of the covariance matrix into an orthogonal matrix and a triangular matrix.

According to the computation steps, the FPGA implementation is divided into four modules, as plotted in Fig. 6.3. The MEG raw data are stored in RAM external to FPGA.

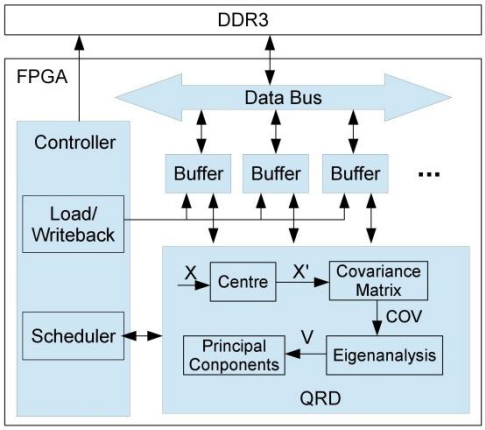

Figure 6.3: FPGA implementation architecture of PCA

6.2.1 Centre module

As shown in Fig. 6.4, the centre module consists of an accumulator, a divider and a subtractor. The data are centered by subtracting the relevant mean vector. Mean values are calculated along the dimensions, which indicates the number of elements in the mean vector is equal to the number of dimensions of the data set. The accumulator consists of an adder and an accumulator register with a feedback loop to the adder. In the design, the mean value of a dimension is computed by dividing the sum by the numerator. Finally, the data are centred by the subtractor.

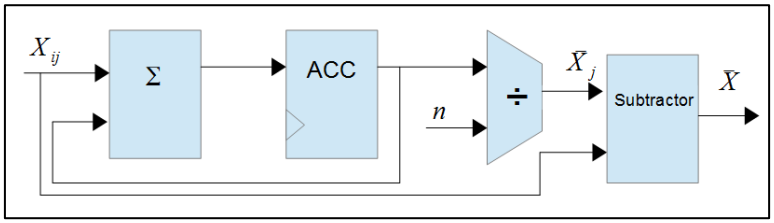

Figure 6.4: Data flow of centre module

6.2.2 COV module

Based on the definition of covariance matrix and Eq. 2.4, the covariance matrix of normalized MEG data is a square symmetric matrix. Therefore, only the diagonal elements and the $\boldsymbol{n(n-1)/2}$ entries above or below the main diagonal need to be calculated, where $\boldsymbol{n}$ is the number of dimensions. Covariance matrix computation involves multiplication, addition and division.

For the sake resource saving, the data are processed in serial structure, as shown in Fig. 6.5 (clock signal is not shown). Every clock cycle 2 data points of 2 out of the 248-channels are read from RAM. They are multiplied and the product is accumulated. When all the data points are processed, the divider is activated and the covariance matrix is computed. Pipeline structure is utilized for acceleration.

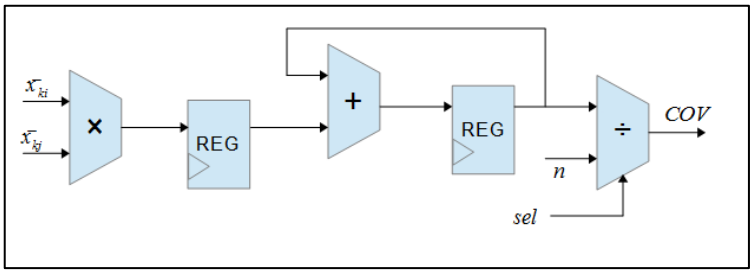

Figure 6.5: Architecture of covariance matrix module

6.2.3 EVD module

A widely used method for EVD is the QR decomposition. Three QR decomposition approaches (Gram-Schmidt, Householder transformation and Givens rotation) are widely used in signal processing systems. Each method has its own advantages and disadvantages. Gram-Schmidt algorithm [88] is adopted in this work.

Estimating the eigenvalues is the most computation demanding part among the four steps of PCA. The design of EVD module is composed of multipliers, dividers, multiplexers, adders, accumulators, a square-root unit, comparators and some registers, as shown in Fig. 6.6. The $\boldsymbol{n \times n}$ data covariance matrix is first decomposed into a product $\boldsymbol{COV = Q_1 R_1}$ of an orthogonal matrix $\boldsymbol{Q_1}$ and an upper triangular matrix $\boldsymbol{R_1}$. Then new matrix $\boldsymbol{COV_1 = R_1 Q_1}$ is got by multiplying the two factors in the reverse order. The two steps are repeated until the bottom triangle of the data covariance matrix becomes zero.

During the EVD process, the input data are first multiplied and accumulated. Then the intermediate results of the accumulation is forwarded to the BRAM. These results are also transmitted to the temporary register for further operations or to the comparator to judge the convergence. Secondly, result of the multiply-and-accumulate from the previous step is processed by the square-root block, and the final result is forwarded to the BRAM. The results are transmitted to the temporary register for further processing and also to the comparator to determine if the results are zero. Afterwards, the data

are processed by the multipliers and sub-tractors before the result is transferred to BRAM. Finally, the data are processed by the divider.

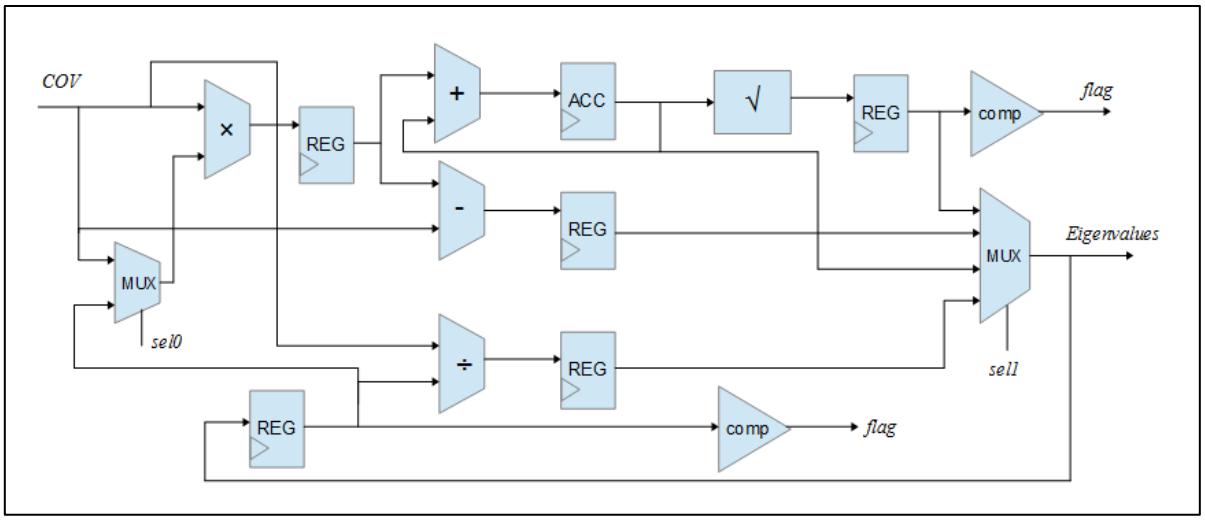

Figure 6.6: Architecture of EVD module

6.2.4 PC module

PC module performs data compression by right multiplying the centred data with the eigenvector matrix. The design of the PC module is simple, including a multiplier and an accumulator. The centred data are already computed by the centre module. The dimension of the original data set is reduced to the number of eigenvectors.

6.3 Implementation of ICA

6.3.1 Implementation architecture

In this study, the natural-gradient version of Infomax ICA is chosen for hardware implementation. It can be learned from Eq. 2.11 that not only is the convergence speed of $\boldsymbol{\Delta W}$ increased, but also that the computation of $[\boldsymbol{W^T}]^{-1}$ is avoided, which facilitates the FPGA realization.

Traditionally, the whole data set is decomposed using the ICA method all at once. In our experiments, the dataset account for about 170,000 time samples for the 4-D-Neuroimaging equipment [9]. To do real-time MEG signal processing, the whole dataset is divided into small data segments for data decomposition. The computation is performed using a sliding time window with a window width of 12 s and 10 s overlap [7].

$$\tilde{\boldsymbol{u}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{W_0}, \text{ and}$$

$$\boldsymbol{W_{i+1}} = \boldsymbol{W_i} + \boldsymbol{\varepsilon}\left[\boldsymbol{BI} + \left(1 - 2g(\tilde{\boldsymbol{u}})\right)\tilde{\boldsymbol{u}}^{\boldsymbol{T}}\right]\boldsymbol{W_i} = \left[\boldsymbol{I} + \boldsymbol{\varepsilon BI} + \boldsymbol{\varepsilon}\left(1 - 2g(\tilde{\boldsymbol{u}})\right)\tilde{\boldsymbol{u}}^{\boldsymbol{T}}\right]\boldsymbol{W_i}, \quad (6.1)$$

where $\boldsymbol{B}$ is the segment size, $\boldsymbol{\varepsilon}$ is the learning rate whose initial value is 10^{-3}. In general, demixing matrices are estimated for segments separately, i.e. whenever the estimation of $\boldsymbol{W_i}$ is completed, the next demixing matrix $\boldsymbol{W_{(i+1)}}$ is estimated on basis of the last 12 s. During the estimation of $\boldsymbol{W_{(i+1)}}$,

$\boldsymbol{W_i}$ is used for data decomposition. In the computation process, $\boldsymbol{I + \varepsilon BI}$ is seen as a constant matrix. Based on the features of FPGA design, the Infomax implementation can be divided into input module, learning module, matrix multiplication module (data update), judge module and output module, among which learning module is the core of the design. The learning module initializes demixing matrix $\boldsymbol{W}$ and performs the learning iterations in Eq. 6.1. The workflow of Informax is plotted in Fig. 6.7.

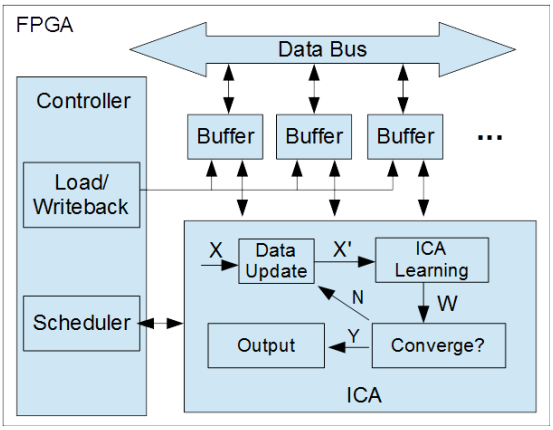

Figure 6.7: Architecture of data decomposition

The input module reads out segments of data from the memory. The chosen sliding window with a window width of 12 s translates to 12,206 data points. After reading the 12,206 data points, the output of the input module is set to zero and waits for the trigger signal of the learning module.

The learning module performs the learning process of Infomax ICA and computes the demixing matrix $\boldsymbol{W}$ of the data segments. The identity matrix was chosen as the initial matrix when calculating the demixing matrix $\boldsymbol{W_1}$ of the first data segment. The previous demixing matrix $\boldsymbol{W_{i-1}\ with\ i = 2, 3, \ldots, n}$ is used as the initial matrix to compute the demixing matrix $\boldsymbol{W_i\ with\ i = 2, 3, \ldots, n}$ of the current segment, where $\boldsymbol{n}$ is the number of data segments. Then the demixing matrix is transmitted to the matrix multiplication module for data update.

The matrix multiplication module multiplies the MEG data segment with the corresponding demixing matrix $\boldsymbol{W}$ and transmits the demixing matrix $\boldsymbol{W}$ to the judge module.

The judge module checks the convergence of the demixing matrix $\boldsymbol{W}$. In the case of convergence, the computation stops; otherwise, computation is repeated.

6.3.2 Input module

The input module reads MEG data from the memory and transmit it the matrix multiplication module. The MEG signals are processed block by block. The block size is [25×12,206]. When the input module finishes reading data, it outputs zero and waits for the demixing matrix computation. When the demixing matrix of the current data segment is done, the trigger signal *start* is sent to the input module. The input module then begins reading the next data segment.

6.3.3 Data update module

The computation involved in data update module is matrix multiplication. Matrix multiplication is often used as a core algorithm in the fields of image processing and signal processing. Thus, the efficiency of matrix multiplication directly affects the processing speed. The matrix multiplication module also influences the resource consumption, etc.

In the flow of the Informax algorithm, the matrix multiplication module calculates the matrix multiplication of [25×25] × [25×12,206]. The data segment from the input module is multiplied with the demixing matrix from the learning module. The matrix multiplication module implements $\tilde{\boldsymbol{u}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{W}_{\boldsymbol{0}}$.

Matrix multiplication can be realized with high speed in hardware. However, hardware resources and efficiency have to be taken into account. Matrix multiplication is implemented using serial structure. The demixing matrix $\boldsymbol{W}$ is fed to the matrix multiplication module as a whole and the MEG segment is read column by column. Each column of the data segment is serially multiplied with the demixing matrix rows and accumulated, which generates columns of the updated data segment. The demixing matrix $\boldsymbol{W}$ is provided by the learning module and the initial value is identity matrix, as shown in Fig. 6.8.

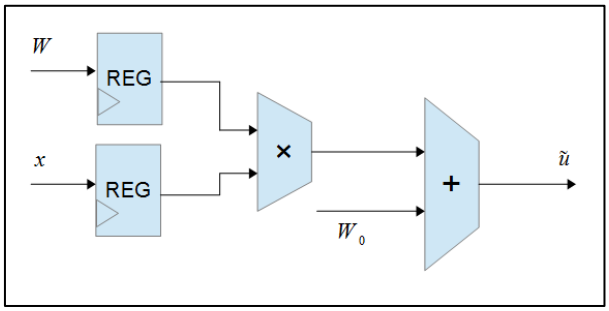

Figure 6.8: Architecture of matrix multiplication

6.3.4 Judge module

The main function of the judge module is to determine the completion of the data processing cycle through the control signal *finish*. If it is not completed (i.e. the number of input data points does not reach 167846), the selection module *Multiplexer* selects input zero and the result of the matrix multiplication module is sent to the learning module for the next iteration of data operation. If the cycle is completed, the selection module *Multiplexer* selects input one and the demixing matrix is multiplied with the mixed signal to obtain the estimated signal of the source signal. The judge module structure is shown in Fig. 6.9.

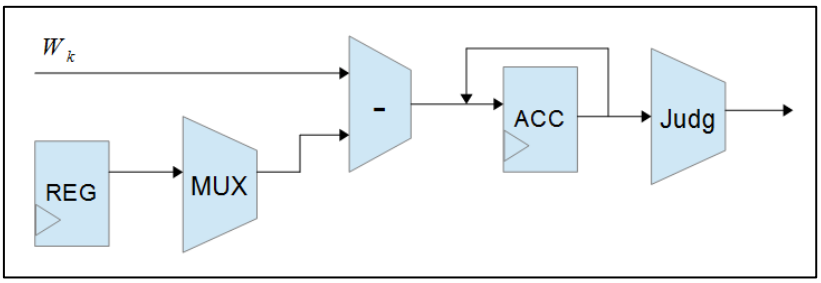

Figure 6.9: Architecture of judge module

6.3.5 Learning module

Learning module performs demixing matrix $\boldsymbol{W}$ initialization and the learning cycle Eq. 6.1. It is the core module of Infomax ICA. We further divide the learning module into the following parts, look up table module, computation module, update module and buffer module, shown in Fig. 6.10.

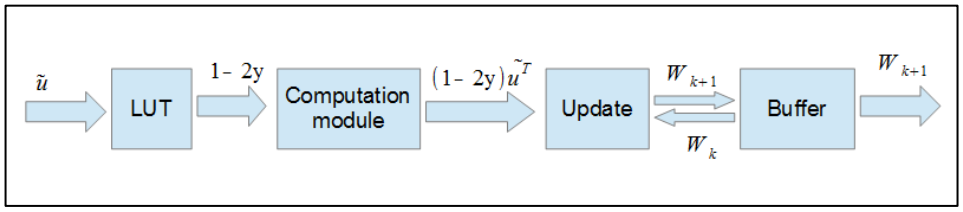

Figure 6.10: Top architecture of learning module

According to Eq. 6.1, when the updated data from the matrix multiplication module is sent to the learning module, the look up table module performs the non-linear transform $\mathbf{1} - \mathbf{2}\boldsymbol{y}$ (plotted in Fig. 6.11). The computation module completes the transposition of $\tilde{\boldsymbol{u}}$ and the matrix multiplication of $\mathbf{1} - \mathbf{2}\boldsymbol{y}$ and $\tilde{\boldsymbol{u}}^{\boldsymbol{T}}$. Update module calculates $\boldsymbol{W_{i+1}} = \left[\boldsymbol{I} + \boldsymbol{\varepsilon BI} + \boldsymbol{\varepsilon}\left(1 - 2g(\tilde{\boldsymbol{u}})\right)\tilde{\boldsymbol{u}}^{\boldsymbol{T}}\right]\boldsymbol{W_i}$. The buffer module ensures the demixing matrix to be unchangeable during the next iteration.

Look up table module

Look up table module performs the calculation of $\mathbf{1} - \mathbf{2}(\mathbf{1} + \boldsymbol{e}^{-\tilde{\boldsymbol{u}}})^{-\mathbf{1}}$. The curve of the non-linear function is shown in Fig. 6.11.

As shown in Fig. 6.11, $\mathbf{1} - \mathbf{2}(\mathbf{1} + \boldsymbol{e}^{-\tilde{\boldsymbol{u}}})^{-\mathbf{1}}$ is an odd function within the input range of [-2, 2]. We designed a symmetrical piece-wise look up table to implement the non-linear function. It can save half of areas by storing half of the function values. The LUT in this design stores the function values within the range of [0, 2]. And the judgment of whether the input is less than zero or not is necessary. The function values for $\mathbf{1} - \mathbf{2}(\mathbf{1} + \boldsymbol{e}^{-\tilde{\boldsymbol{u}}})^{-\mathbf{1}}$ are (-2*(l./(l+exp(-[0:l/65536:2])))+l)*65536, which indicates that the range [0, 2] is divided into 65536 pieces and normalization is performed.

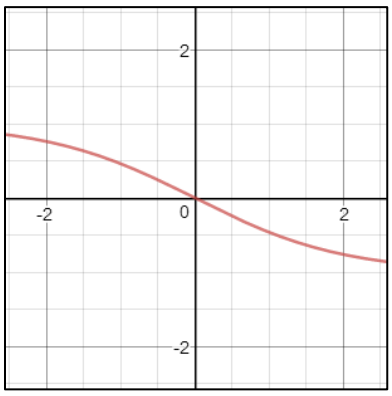

Figure 6.11: Curve of the non-linear function

Computation module

Computation module realizes the function $(\mathbf{1}-\mathbf{2}\boldsymbol{y})\widetilde{\boldsymbol{u}}^{\boldsymbol{T}}$, which is the design of a Multiply-Accumulator.

Update module

The update module calculates the demixing matrix, which is $\boldsymbol{W_{i+1}} = \left[\boldsymbol{I}+\boldsymbol{\varepsilon BI}+\boldsymbol{\varepsilon}\left(1-2g(\widetilde{\boldsymbol{u}})\right)\widetilde{\boldsymbol{u}}^{\boldsymbol{T}}\right]\boldsymbol{W_i}$. The module is divided into 2 parts, addition part and multiplication part, as shown in Fig. 6.12. The addition part computes $\boldsymbol{I}+\boldsymbol{\varepsilon BI}+\boldsymbol{\varepsilon}(\mathbf{1}-\mathbf{2}\boldsymbol{y})\widetilde{\boldsymbol{u}}^{\boldsymbol{T}}$, which updates the coefficients. The multiplication part implements $\boldsymbol{W_k}(\boldsymbol{I}+\boldsymbol{\varepsilon BI}+\boldsymbol{\varepsilon}(\mathbf{1}-\mathbf{2}\boldsymbol{y})\widetilde{\boldsymbol{u}}^{\boldsymbol{T}})$.

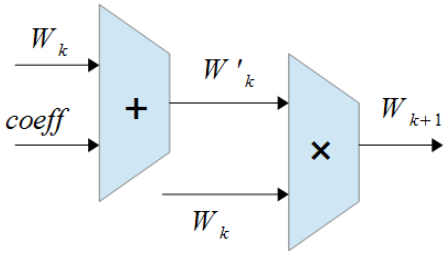

Figure 6.12: Architecture of update module

Buffer module

The buffer module initializes and stores the demixing matrix $\boldsymbol{W}$. In this module, three control signals, which are *first*, *wait* and *finish*, are provided by the input module. *first* controls the initialization of the demixing matrix. *wait* controls the input of each data segment. *finish* is the flag of data processing completion.

6.3.6 Output module

The output module stores the results of matrix multiplication of the judge module. The design of output module is simple and thus not detailed.

6.4 Implementation of CTPS

6.4.1 Implementation architecture

The first step of CTPS is to transform the MEG signals into phase space. Our design adopted Hilbert transformation to extract phase information. Then, the phase distribution is compared with a uniform distribution using the Kuiper test to identify the components related to cardiac artifacts.

6.4.2 Hilbert transform module

Hilbert transform is a widely used and important technique in signal analysis and processing. It performs a phase shift with a radian of $\pi/2$ on the input signal, which shifts the phase of positive frequency by $-\pi/2$ and the phase negative frequency components by $\pi/2$ while leaving the magnitude of the inputs unchanged. A detailed description of Hilbert transform can be found in [89].

Hilbert transform can be viewed as a special digital filter. Hence it is possible to approximate the Hilbert transform with a digital filter. However, the non-causal and infinite impulse response of that filter makes it difficult to realize a good approximation with low hardware resource usage [90]. Nevertheless, Hilbert transform can be implemented as a finite impulse response (FIR) filter, which is a preferred approach [90],[91],[92],[93].

FIR filtering brings delays to signals of different levels. Typically, an *n*-th order FIR filter requires *n+1* multiplies and *n* adds [94]. When the length of unit impulse response is large, the computation is demanding and the delay can be very long, limiting the signal processing speed.

Using the rich register resources on an FPGA chip, multi-stage pipeline technique can be easily implemented in the Hilbert transform realization process. The flow of a FIR filter using a one-stage pipeline structure is shown in Fig. 6.13.

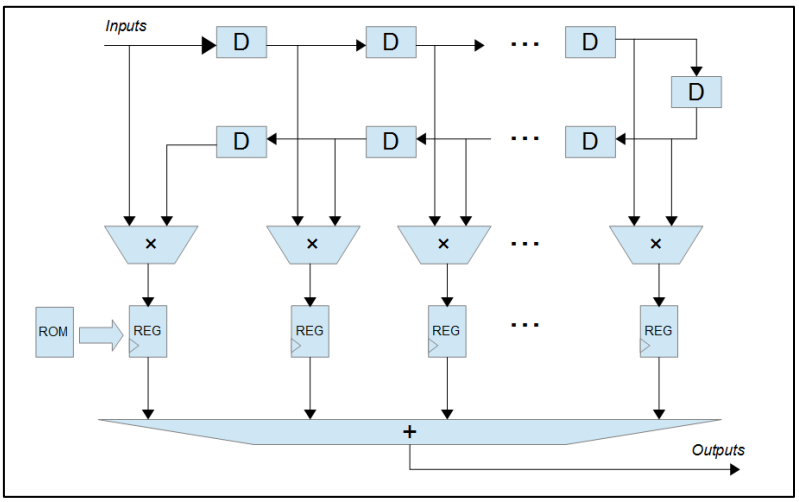

Figure 6.13: Architecture of Hilbert transform

During execution, the pipeline registers store the results of the multiplication of the previous cycle and completes the addition in the next cycle. Therefore, the FIR of one-stage pipeline structure can

simultaneously perform the accumulation of the previous cycle and the multiplication of the current cycle, so as to improve the overall speed of the system. If much higher operating speed is desired, further multi-stage pipeline structure can be applied to multiplication and accumulation operations.

In Fig. 6.14, P(0)-P(25) represent the data channels. The data are serially written to the FIR converter in this module. The filter coefficients are multiplied and accumulated by the data of each channel in turn. Because of the use of the pipeline technology, the data of one channel is processed while reading, rather than processing after reading is complete.

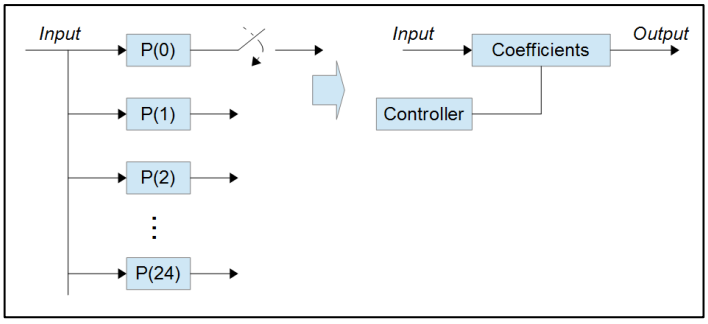

Figure 6.14: Internal architecture of FIR Hilbert transformer

6.4.3 Kuiper test module

The Kuiper test [95] is a statistical test that refines the well-known Kolmogorov-Smirnov test (K-S test) [96]. The Kuiper test is the adaption of the K-S test for cyclic problems. It quantifies the probability that two data sets are samples of the same distribution. For checking stimuli-related phase responses, the cross-trial cumulative phase distribution is compared with a cumulative uniform distribution [38]. The Kuiper test algorithm was realized in Verilog using the Xilinx IP core library to ease the implementation of blocks as memories or arithmetic operators.

First the cumulative distribution function (CDF) of the phase samples is estimated. During CDF estimation, the value range of phase data is divided into $\boldsymbol{M}$ ($\boldsymbol{M}$ is the square root of the number of data points) segments. The number of data samples from each given segment is counted and accumulated. A simplified diagram for CDF estimation from the measured data is shown in Fig. 6.15. Two RAM memories are used to store data. RAM1 is for the input data, RAM2 is for the values of estimated CDF approximated by the sum. The absolute values of the data samples are calculated first. Then the absolute values are compared with the intervals (value ranges of segments) in the comparator. If one absolute value belongs to a specified segment, the result of comparison is one, then the relevant input of RAM2 is incremented by one. Otherwise, the input of RAM2 stays constant. This computation process is iterated until all data samples pass through. A Finite State Machine (FSM) is used to control the read and write operation of RAM2.

After the CDF of the phase values is calculated, the CDF is tested against a uniform distribution by the operations of subtraction and the maximum search, as plotted in Fig. 6.16. The CDF values are

subtracted with the pre-defined exponential calculation values stored in RAM3, then the maximum of the results is searched and buffered in the register MAX. The eventual result is obtained based the error probability. Computation of the error probability involves accumulation and exponential calculation. Exponential calculation is performed using LUT (See Section 6.2.5).

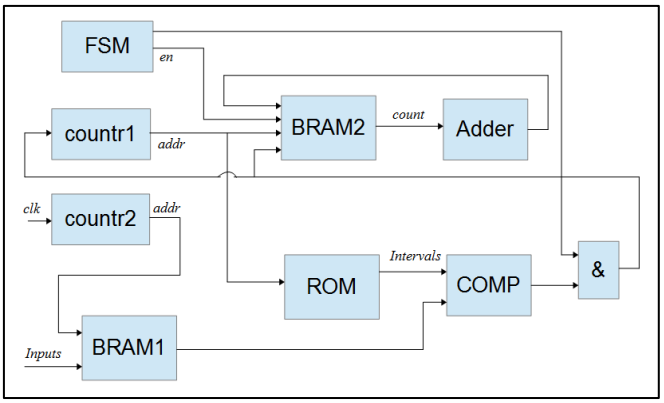

Figure 6.15: Architecture of CDF estimation

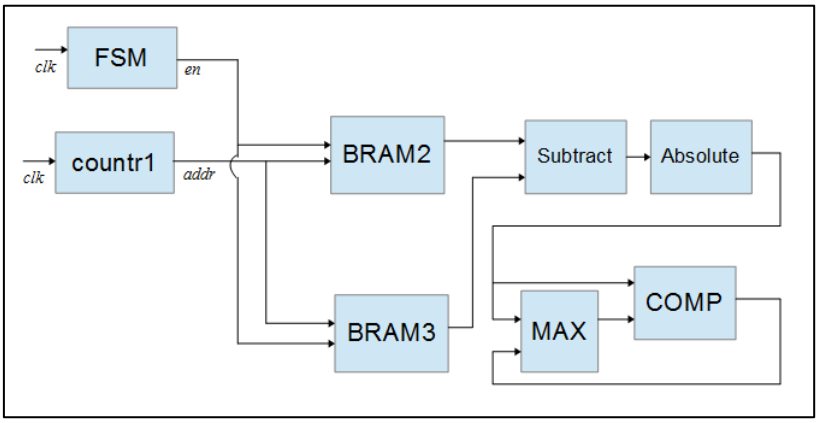

Figure 6.16: Architecture of maximum search

6.5 Implementation of temporal and spatial correlation

6.5.1 Implementation architecture

Ocular artifacts are automatically identified by estimating the Pearson linear correlation between each independent component of MEG signal and the reference EOG signal. As columns of the demixing matrix contain the spatial information, it is also taken into account by calculating the Pearson linear correlation between the template T and the field maps as extracted by ICA [9].

Suppose $\boldsymbol{a}$ and $\boldsymbol{b}$ are both zero-mean real-valued random variables. The Pearson correlation coefficient (PCC) is defined as [97]:

$$\boldsymbol{\rho^2(a,b) = \frac{E^2(ab)}{\sigma_a^2 \sigma_b^2}}$$

where $\boldsymbol{E(ab)}$ is the cross-correlation between $\boldsymbol{a}$ and $\boldsymbol{b}$. $\boldsymbol{\sigma_a^2 = E(a^2)}$ and $\boldsymbol{\sigma_b^2 = E(b^2)}$ are the variances of the signals $\boldsymbol{a}$ and $\boldsymbol{b}$, separately. The PCC quantifies the strength of the linear relationship between the two random variables $\boldsymbol{a}$ and $\boldsymbol{b}$. The computation of PCC involves the calculation of covariance and variances of the signals $\boldsymbol{a}$ and $\boldsymbol{b}$.

6.5.2 Covariance

In probability theory, covariance specifies the joint probability for two random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ [98]. To estimate the covariance, the expected values for $\boldsymbol{X}$ and $\boldsymbol{Y}$ (i.e. $\bar{\boldsymbol{X}}$ and $\bar{\boldsymbol{Y}}$) are calculated. Afterwards, the differences of $\boldsymbol{X}$ values from their expected value are multiplied by the differences of $\boldsymbol{Y}$ values from their expected value. Then the products are accumulated, which is divided by the number of examples in the population to get the final covariance $\boldsymbol{E(XY)}$. The simplified architecture diagram of the covariance estimation module is shown in Fig. 6.17.

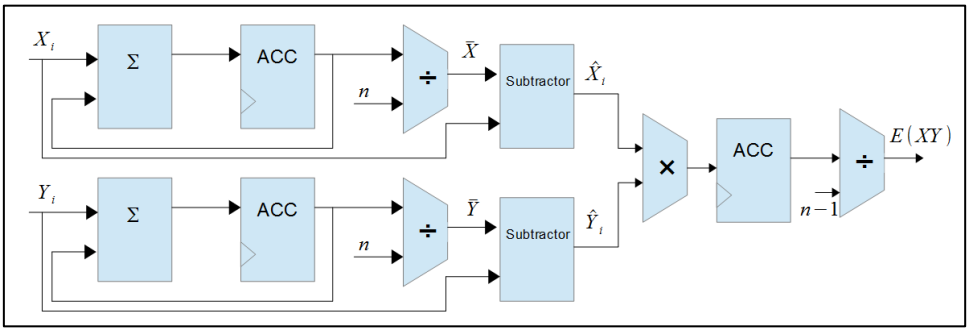

Figure 6.17: Block diagram of covariance estimation module

6.5.3 Variance

In probability theory, the variance specifies how much a random variable $\boldsymbol{X}$ deviates from its mean [98]. Suppose $\bar{\boldsymbol{X}}$ is the expected value of the variable $\boldsymbol{X}$. Each value in the data set is subtracted by the expected value $\bar{\boldsymbol{X}}$. Then the obtained differences $\widehat{\boldsymbol{X_t}}$ are squared and accumulated. The variance $\boldsymbol{\sigma_X^2}$ is estimated by dividing the sum of the squares by the number of examples in the data set. The simplified architecture diagram of the variance estimation module is shown in Fig. 6.18.

After estimating the Pearson's linear correlation, both the temporal and spatial correlation computations are incorporated into a single threshold parameter and compared with a pre-defined threshold (0.8) [7] to determine the components related to ocular artifacts. Eye blinks are detected by applying the Teager-Kaiser-Energy-Operator on each time point of the filtered EOG signal (for details please refer to [7]). The implementation of the eye blink detection only involves a multiplier and a subtractor, which is easy and not shown here.

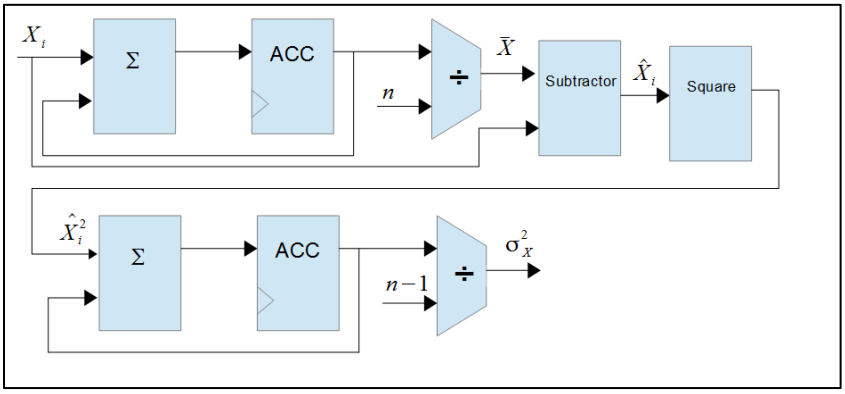

Figure 6.18: Architecture of variance calculation

7 Results and Analysis

Cost analysis of resource and performance was carried out to assess the efficiency of the proposed prognostic (pre-implementation) model. The design was implemented on a Virtex-6 FPGA (running at 100 MHz) by optimizing the Verilog HDL code automatically generated by Enterprise Architect for further synthesis using the Xilinx ISE 14.1 tool. Questasim 10.5c and Xilinx Chipscope Pro 14.1 were used to validate the quantitative requirements and functional correctness. The test MEG data set was recorded with a whole-head magnetometer system (Magnes-3600WH, 4D-Neuroimaging) [7]. The configurable hardware design of MEG signal pre-processing was carried out by extensive parameterization of functional blocks to be generally applicable to other MEG systems.

7.1 Experiment data set

The design was tested using MEG data provided by the Institute of Neuroscience and Medicine (INM-4) in Jülich Research Center, where a 248-channel whole-head magnetometer system (Magnes 3600WH, 4D-Neuroimaging) is located..

During the experiments, MEG signals from five subjects were measured. The experiment setup is to record neuro-magnetic field changes due to finger tapping cued by auditory stimulation. The auditory stimulations were 50-ms sinusoidal tones (single clicks) of 1000 Hz [9]. The data set is chosen because the experimental paradigm studies the auditory cortex, a well-known area in the brain. Detailed information on the dataset is listed in Tab. 5.1. Relevant SysML models about the dataset are provided in Fig. 5.3 and Fig. 5.8.

7.2 Results and discussion of RDD & MBD methodology

7.2.1 Traceability in the system model of MEG signal pre-processing

Consistency is ensured in RDD & MBD by establishing traceability links between the modeling elements in the design, as shown in Fig. 22. On the one hand, links between the requirements and the other model elements are built, which makes sure that all the requirements are covered in the design. On the other hand, the elements from different aspect-oriented models are related to each other to ensure consistency of the design.

In Fig. 7.1, dependency relationships including *trace*, *satisfy* and *allocation* enable a modeling tool to navigate through modeling elements and perform automated downstream impact analysis. The *satisfy* relationship is another kind of dependency indicating an instance of the block to fulfill the linked requirement. "*The allocation relationship can provide an effective means for navigating the model by establishing cross relationships, and ensuring the various parts of the model are properly integrated*" [44]. The *allocation* relationship can be established between the system functions (dynamic behaviors) and the components (static structure) that implement the functionalities of the system. The *traceability*

relationships shown in Fig. 7.1 facilitate the developers to evaluate the impacts of changes in the design. Any changes of these modeling elements will automatically be propagated back to their instances linked by other models.

The RDD & MBD methodology provides multiple constructs that fulfill the tasks of modeling in systems engineering. It is a requirement driven method. In the requirements formalization process, non-functional requirements like performance and safety requirements are added besides the major mode requirements. Additionally, the complex physical flows in MEG signal pre-processing are modeled with the help of self-defined value types, which facilitate checking the consistency of data dimensions in the models. RDD & MBD supports extension capabilities and mechanisms, which increases the adaptability of the methodology. Users can extend its semantics with domain specific concepts to model a much larger variety of systems.

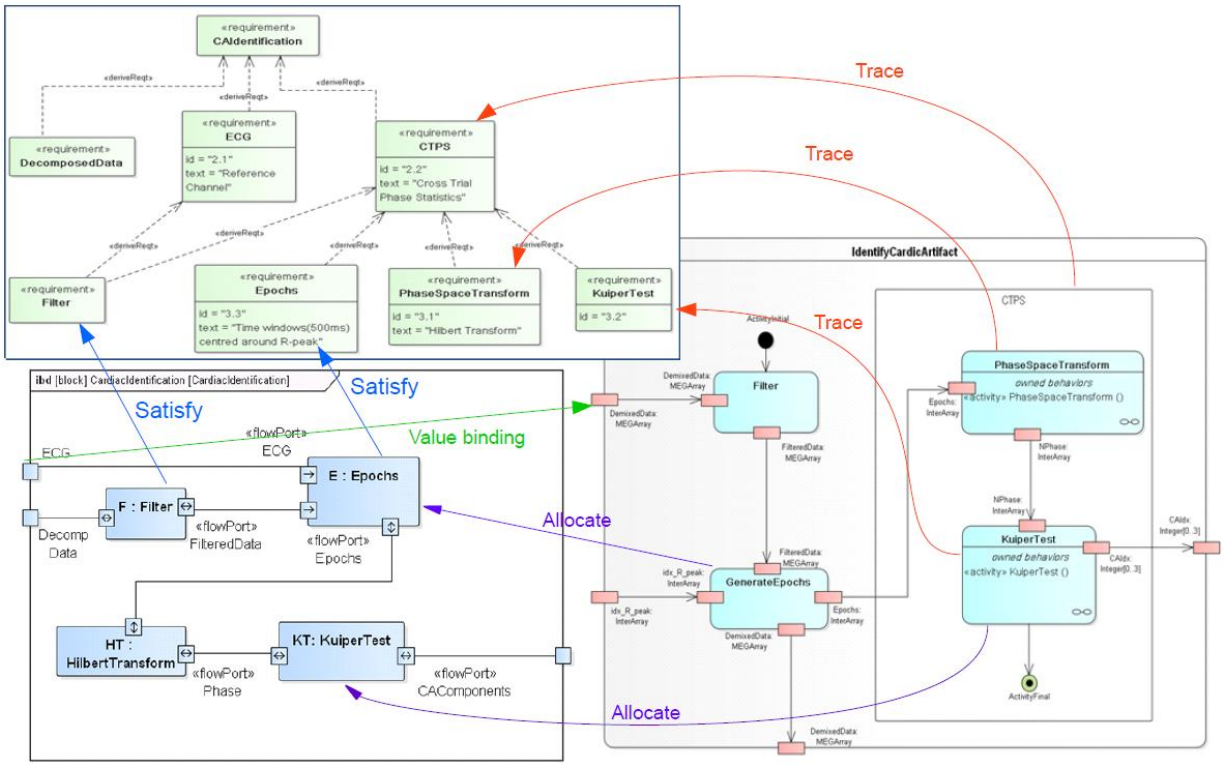

Figure 7.1 An example of traceability in the systems models of MEG signal pre-processing. Note that the arrows are added just for illustration. They are not shown in real SysML diagrams as the relationships are hidden under the modeling elements

7.2.2 Benefits of applying RDD & MBD to MEG signal pre-processing

The RDD & MBD methodology addresses several problems in MEG signal pre-processing:

Efficient requirements management. RDD & MBD to requirement elicitation is based on distinguishing phases of a product life-cycle and categorizing the questions that can be stated in each phase to computing and controlling systems. As many requirements as possible are extracted based on the concepts and needs concerned by the MEG users and engineers and formalized. Then they are

further refined with requirement diagrams of different levels, showing their involved relationships. Full or close to full coverage can be achieved by grouping and generalizing the requirements with subsequent classification and categorization of them.

On the other hand, the functionality of MEG signal pre-processing are described by capturing all the architectural concepts and dynamic behaviors and their relationships, which is performed using multiple SysML behavioral models. Quantifiable requirements are detailed with mathematical relationships and imposed on the relevant elements of structural models, producing predictable implementation parameters such as performance, resource budget and power consumption to implement MEG signal pre-processing on relevant platforms.

RDD & MBD supports great reusability. For all the modeling work, both the behavioral and structural modeling elements are defined using a hierarchical approach to facilitate reusability of modeling components at different levels of abstraction. In MEG signal pre-processing, the modeling work consists of different abstraction levels, in which higher-level models instantiate lower-level elements to present a detailed view of the system. In addition, this systems model of MEG signal preprocessing is prepared as a basis for the Forschungszentrum Jülich MEG-RT 2.0 project.

7.3 Results and analysis of PCA design

7.3.1 Results of PCA design

The system design of PCA was completed successfully. The RDD & MBD method describes the system with great simplicity using SysML. It decreases the effective complexity for large system design and reduces the development time by creating a system model with different abstraction levels.

In my experiment, captured waveforms indicate that the dimension of the input data is reduced from 248 to on average 25 components (This is specific in our experiment; results may vary in other designs), explaining > 95% of the original data variance. The PCA computation results match the results in [7]. As shown in Tab. 7.1, the design occupies 42% of total slices (56880 available) and 8% of DSP48E1s (576 available), which highly correlates with the results of the prognostic models created for system-level modeling in Tab. 5.2. The execution time is measured for each data segment and the average is 21,526,273 clock cycles.

Table 7.1: Resource utilization of PCA design

Item	Occupied	Available	Percentage
Number of slices	23,889	56,880	42%
Number of DSP48E1s	46	576	8%

7.3.2 Result analysis and discussion

Requirements of the design are covered in two aspects. Firstly, the requirements are traced to different parts of the models to point how each requirement is addressed by the models. The requirements

(*Preprocessing* and *Method*) of PCA design are refined (linked) to model elements including a state machine model *PCA*, a block definition diagram *PCA-BDD* and a sequence diagram *Preprocessing*, which are colored yellow in Fig. 7.2. The other linked models are not shown for the readability of the diagram.

Secondly, the PCA requirements are covered by implementation and verification on an FPGA board. The covered requirements are colored green in Fig. 7.2. Note the requirements colored in grey are covered in relevant sections as PCA is the focus of this specific section. This is performed to ensure the traceability of requirements within the system models to final realization.

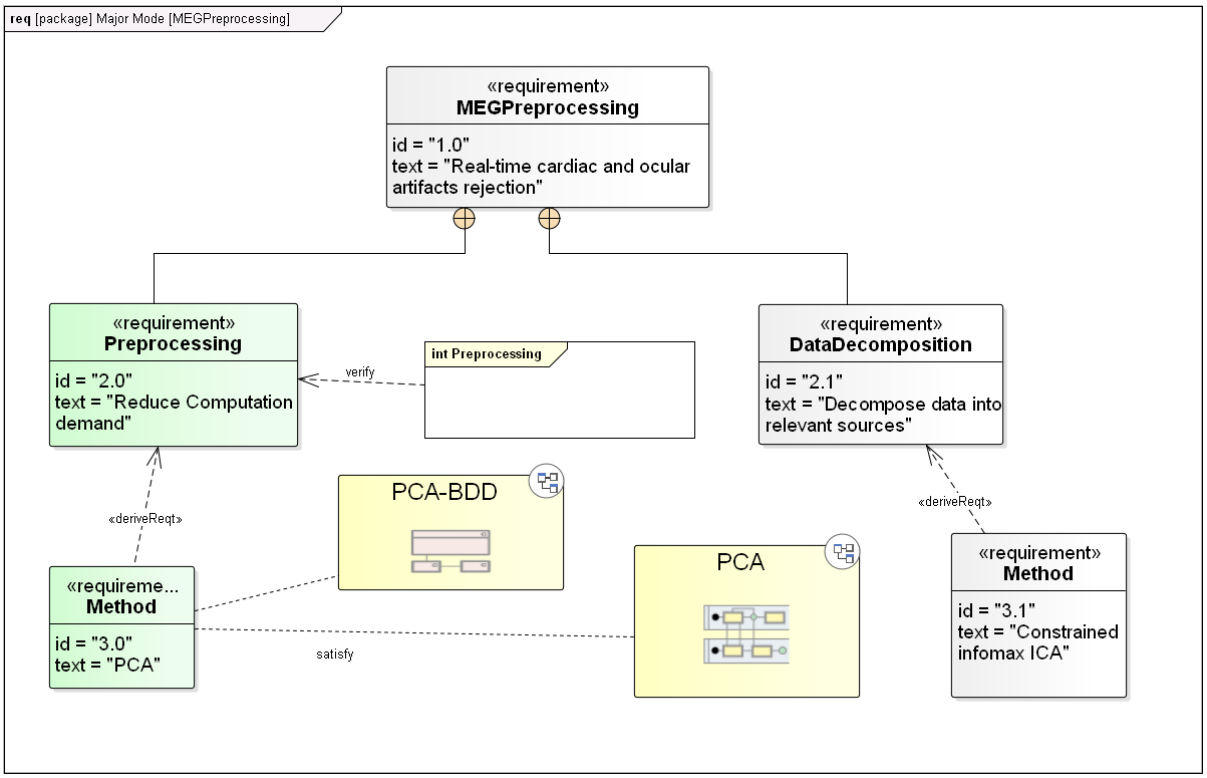

Figure 7.2: Requirement coverage of PCA design (See Fig. 5.4 for the complete requirement diagram)

PCA is widely used in many fields including image analysis, web data processing and machine learning. Previous studies in [99],[100],[101],[102] have investigated reaching high throughput and performance by utilizing the parallel computations on FPGA or ASIC devices; however, the resource consumption grows exponentially with the increase of matrix dimensions, which limits matrix size only up to 8×8. Based on our knowledge, the latest and only scalable architecture for FPGA-based EVD/SVD design was proposed in [103]. However, the proposed approach utilized low-cost FPGA devices and the matrix dimensions are limited with in the size of 32×128 because of the memory available. The large latency comes from SVD computation of large matrices which requires the iterative architecture in the hardware implementation. In our design, PCA is well performed on dataset

of larger dimensions (248 × 12,206). Besides, our design is parameterized and easily adaptable to data of different dimensions.

7.4 Results and analysis of ICA design

7.4.1 Results of ICA design

For design evaluation, the 25 channels of mixed signals are stored in the block RAM on FPGA. Each channel consists of 12,206 data points. The ICA modules decompose the mixed signals into relevant components.

Tab. 7.2 shows the summary of synthesis for Infomax ICA implementation. As shown in Tab. 7.2, the design occupies 60% of total slices (56880 available) and 54% of DSP48E1s (576 available), which correlates highly with the results of the prognostic models created for system-level modeling. The reconfigurable hardware designs of Infomax ICA in this work are parameterized and easily adaptable to data of different dimensions, which results in different resource utilization.

Table 7.2: Resource utilization of ICA design

Item	Occupied	Available	Percentage
Number of slices	34,128	56,880	60%
Number of DSP48E1s	311	576	54%

The proposed Infomax ICA hardware design is driven by a system clock of 100 MHz; the target MEG data has a window size of 12 s and a sampling rate of 1017.25 Hz. The first 12 s of the data from each measurement was used as training data set. The data segments overlap within a time range of 10 s, so the dynamics of the underlying sources in two overlapping data segments are analogous. The processing of one segment is based on the prior calculations of the last segment, which speeds up the decomposition, and the signal separation can be done in one iteration. The maximum hardware computation time is 17,326,971 clock cycles. Thus, the throughput is 6103 points/s. Therefore, the implemented Infomax ICA architecture can support the real-time requirement of our application well.

7.4.2 Result analysis and discussion

The requirements of ICA design are first traced to different models to ensure that all requirements are covered in systems modeling. The requirements (*DataDecomposition* and *Method*) of ICA design are linked to model elements including a state machine model *InfomaxICA*, a use case diagram *ICA* and a sequence diagram *DataDecomposition*, which are colored yellow in Fig. 7.3. Note the requirements colored in grey are covered in relevant sections as ICA is the focus of this specific section. Various other traced model elements are not listed for the readability of the figure. Afterwards, the ICA requirements are realized and verified on an FPGA board. The implemented (covered) requirements are colored green in Fig. 7.3. Final realization is traced to relevant requirements to ensure the requirement coverage of systems engineering.

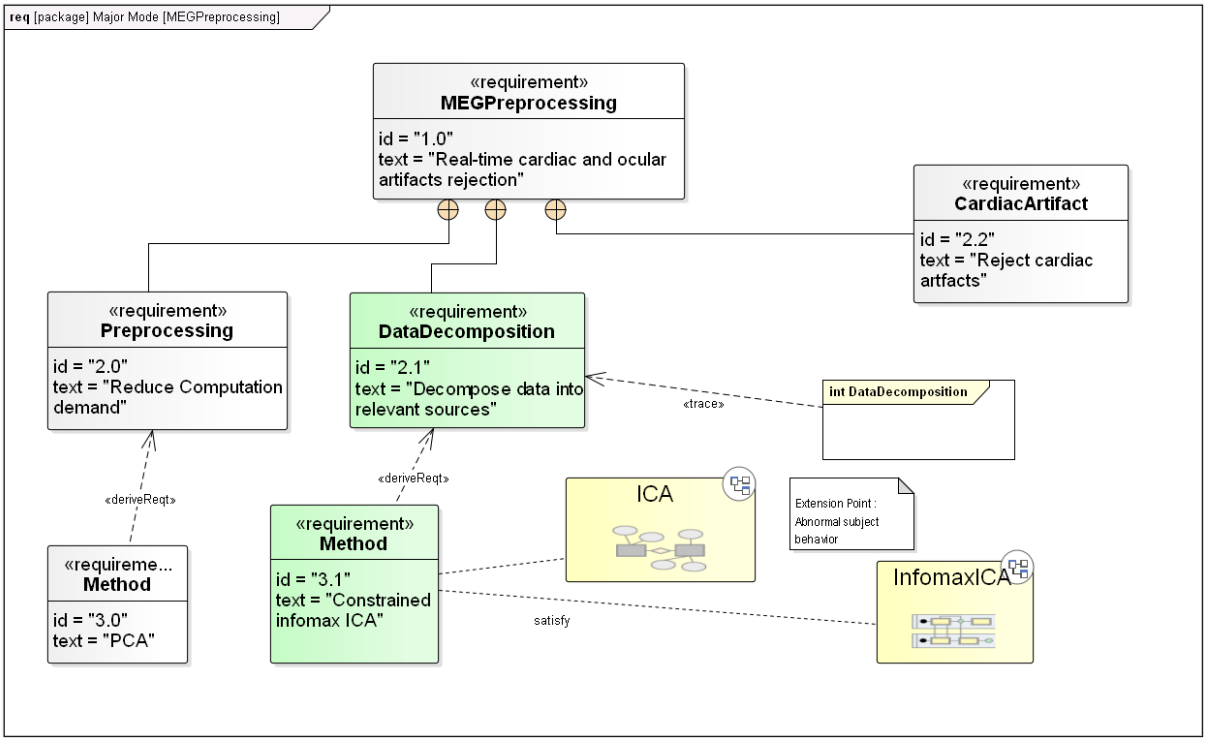

Figure 7.3: Requirement coverage of ICA design (See Fig. 5.4 for the complete requirement diagram)

Many studies have been devoted to ICA hardware implementations. Hardware realizations of ICA algorithms are challenging with respect to resource, flexibility, and speed. Many studies on the real-time implementation of ICA targeted data of much lower channel and sample numbers. Higher channel and sample numbers increase the computation time for ICA processing because of the greater computational complexity. In [104],[105],[106], the FPGA implementations can only achieve 2-channel ICA processing for speech signals. In [107],[108],[109], the proposed hardware architectures can perform ICA processing with a channel number of 4. In [110],[111],[112], 8-channel ICA implementations are reported. In [113],[114], the ICA implementation can achieve 16-channel processing. Thirty-two–channel convolutive ICA is first implemented on FPGA and applied to real world signals in [115]. However, the post-layout results of the design are not reported. Another 32-channel ICA implementation on FPGA used for EEG signal processing is reported in [116] but the implementation information is currently not accessible. In this study, the proposed hardware implementation of Infomax ICA can decompose MEG signals with a channel number of 25, which is parameterized and can be scaled to applications with channel numbers > 25.

Speed is also an important criterion for evaluating a hardware implementation of ICA. The ICA designs implemented in [106] and [115] work at the operating frequencies of 12.3 MHz and 50 MHz, respectively. The implementation in [106] is as short as 0.003 s for lower-dimension ICA computation, while the realization in [115] takes > 60 s for adaptive noise canceling. A parallel ICA is implemented on FPGA in [107],[108]; it has the operating frequency of 20.2 MHz used for hyperspectral image

analysis. The 4-channel Infomax ICA is implemented on FPGA in [23] with the operating frequency of 68 MHz applied to EEG signal processing. In our work, the operating frequency of the proposed hardware architecture is 100 MHz with the computation time of 1.72 s.

The present work has greater resource consumption than previous studies in [31], as the existing implementations are only for signal processing of much lower channel and sample numbers. The processing capacity in the present study can be up to 25 channels and 12,206 samples, which are both higher than previous implementations. Additionally, our design can process data with dimensions > 25 × 12,206, as it is adaptable.

7.5 Results and analysis of artifact rejection design

7.5.1 Results of artifact rejection

Artifact components are identified and removed from recorded signals using CTPS. The design is evaluated using the windowed MEG data around R-peak latencies of the QRS complex from ECG signals, which has a window size of 500 ms. Each data window consists of 508 data points, which is of small volume and buffered in the block RAM on FPGA. The CTPS modules first transform the data into phase space using Hilbert transform, and then identify the cardiac components.

Ocular components are identified by computing the Pearson linear correlation between the MEG component and the reference EOG signal and also between the template T and the field maps. Mathematically, the calculations mainly involve divisions and vector–vector multiplications.

Tab. 7.3 shows summary of synthesis for the artifact rejection implementation. The design occupies 27% of total slices (56880 available) and 16% of DSP48E1s (576 available). There is strong correlation between the resource prediction model in Tab. 5.4 and the synthesis results. Meanwhile, it is difficult for the Resource Model to obtain the absolute accuracy in prediction as the results of synthesis depend on the optimization algorithms, e.g., logic replication, typically exploited by modern synthesis tools for improving the timing characteristics of synthesized designs.

Table 7.3: Resource utilization of artifact rejection design

Item	Occupied	Available	Percentage
Number of slices	15,742	56,880	27%
Number of DSP48E1s	95	576	16%

The proposed artifact rejection hardware design is driven by a system clock of 100 MHz; the data epochs (segments) during CTPS computation are buffered in on-chip BRAM, which reduces the memory access time. For ocular artifact rejection, the data channels are pre-fetched, which improves the data access speed. There is no data dependency between the covariance calculation and the computation of the standard deviations, which largely improves the computation parallelism. The design adopts the pipeline structure, further enhancing the throughput. Thus, the computation modules

are are fully utilized. The data is processed by one module, while the results are transmitted to the next unit for further computation in every clock cycle. Additionally, the computations are designed to overlap with the memory access to make best use of the pipelined architecture. The execution time is measured for each data segment during CTPS computation, and the average is 223 clock cycles. In the testing, an average two components identified as related to cardiac activity. For ocular artifact rejection, the execution time is measured for each data channel, and the average is 106 clock cycles. In the testing, 1–3 components (average, 1.5) were identified as being attributed ocular activity. The rejection performance measure as introduced in [9] is used to evaluate the cardiac artifact rejection results. Cardiac artifacts were sufficiently removed with the hardware implementation of CTPS while keeping the signal of interest unchanged.

7.5.2 Result analysis and discussion

The requirements of CTPS design are satisfied in two ways. On the one hand, traceability between the requirements and different models is established, which ensures requirement coverage during the systems modeling process. The requirements (*CardiacArtifact* and *Method*) of CTPS design are traced to model elements including a state machine model *CAStructureAllocate*, a use case diagram *CAIdentification* and a sequence diagram *ArtifactIdentification*, which are colored yellow in Fig. 7.4. The other traced model elements are not shown for the readability of the diagram.

Subsequently, the CTPS is realized and verified on an FPGA board to further cover the relevant requirements. The covered requirements by CTPS design are colored green in Fig. 7.4. The hardware implementation of CTPS ensures the traceability of requirements to the final realization.

Again, the requirements of temporal and spatial correlation design (*OcularArtifact*, *IdentifyEyeBlinks* and *Method*) and data cleaning (*DataCleaning*) are first traced to different model elements guarantee the requirements coverage in systems modeling. They are refined by tracing to both structural and behavioral diagrams.

Secondly, the requirements of temporal and spatial correlation and data cleaning are covered by implementation and verification on an FPGA board. The covered requirements in this section are colored green in Fig. 7.5. The requirements covered in previous sections are colored yellow in Fig. 7.5. Up to now, all the requirements of MEG signal pre-processing are fully covered, not only by traceability to different system model elements, but also by realization and verification on an FPGA board.

Real-time capability is achieved in this work by using a parallel computation platform FPGA, which makes best of the parallelism of the matrix–matrix multiplications or matrix–vector multiplications in MEG signal pre-processing. The computation modules of lower levels are designed in the pipeline structure, further enhancing the throughput. Data cleaning is performed in parallel to estimating a new demixing matrix (see Fig. 5.15), leading to the possibility to massively parallelize computation on FPGA. The data cleaning procedure is performed with a time delay of < 1 ms.

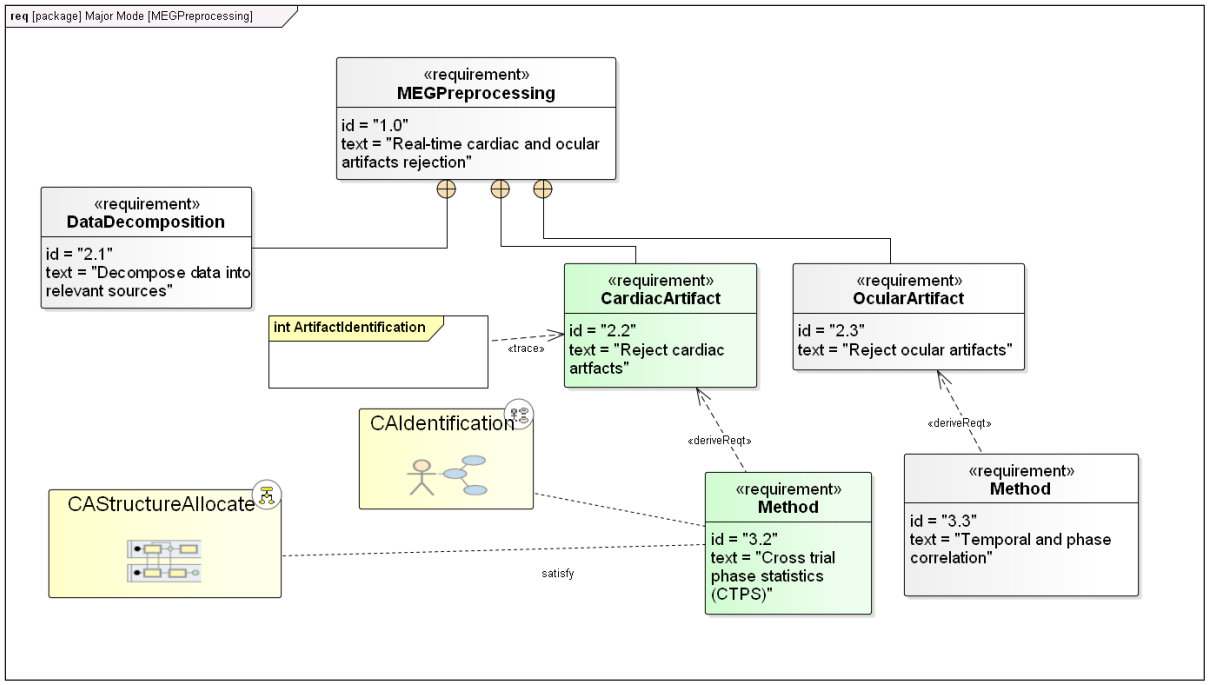

Figure 7.4: Requirement coverage of CTPS design (See Fig. 5.4 for the complete requirement diagram)

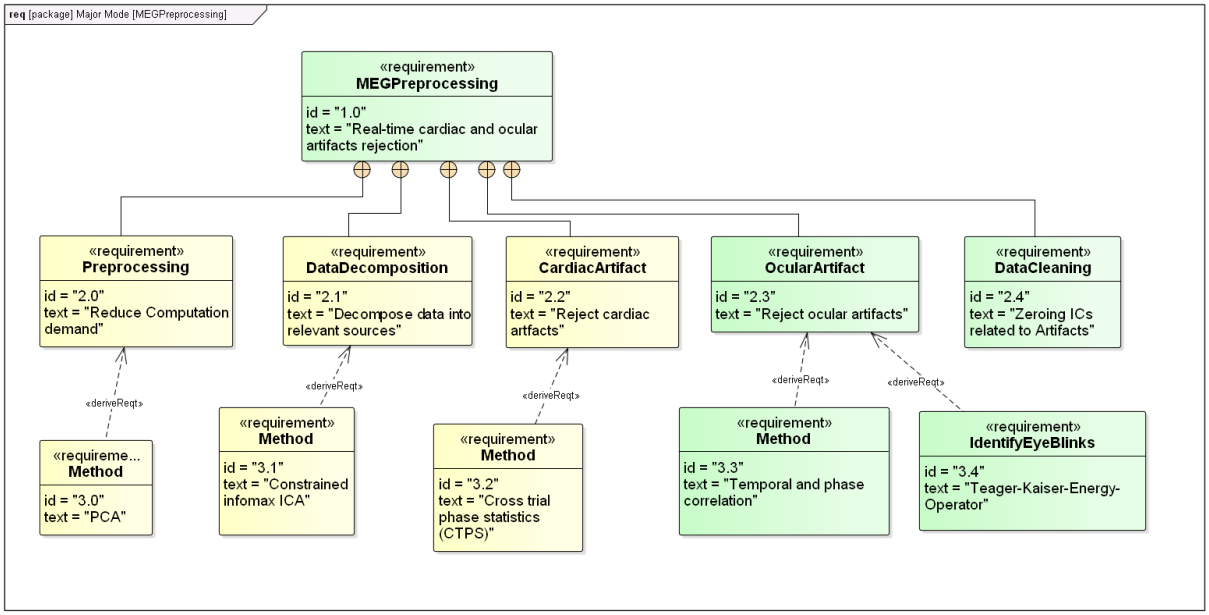

Figure 7.5: Requirement coverage of temporal and spatial correlation design

In [9], MEG data is tested on the Intel i5-2410M Core with the power dissipation of 35W. The reported processing time, including data decomposition and artifact rejection, is 1.1 s without taking the data training into account. The average computation time in this study is 1.4 s but with much lower power dissipation (maximum 7.5W). The CPU implementation (on an Intel Core i5-2410M, 2.3 GHz, 6 GB RAM) is not application specific and often slowed by the computation tasks of other threads. While the FPGA dedicated design has advantages in terms of both speed and resource consumption.

As described in section 5.1.1, signal processing systems with a delay of < 200 ms are considered to be real-time in this thesis. Different computation times for real-time feedback design are indicated in previous studies. In the studies in [8],[12],[20], real-time designs a feedback update every 500 ms are reported. A real-time system with a feedback update every 300 ms is provided in [11]. Despite the differences in feedback delay, all authors described their systems as being real-time capable. In this thesis and in [9], the artifact rejection, not including data decomposition and data training, takes < 1 ms. In summary, the hardware implementation of MEG signal preprocessing offers effective real-time capabilities, with low power dissipation, high speed and reasonable resource consumption.

8 Summary and Outlook

8.1 Summary

In this thesis, a requirement-driven model-based development methodology for real-time computation systems was introduced. It handles the ever-increasing complexity of computation systems by increasing the abstraction level in the design.

The proposed RDD & MBD methodology was applied to the MEG-RT 2.0 project in the Jülich Research Center (Forschungszentrum Jülich). In the MEG-RT 2.0 project, an MEG signal processing device capable of advanced MEG signal processing in real-time including source reconstruction will be developed and used as an add-on to existing MEG systems, which enables e.g., neuro-feedback applications.

The requirements of the MEG-RT 2.0 project were first defined and formalized based on the workflows of the RDD & MBD approach, as shown in Fig. 8.1. The requirements are structured in a tree-like manner and defined on several levels of abstraction following a functional decomposition. The MEG-RT 2.0 design is capable of real-time (and thus automated) artifact rejection: MEG data are first decomposed into underlying informal components (requirement *DataDecomposition*), then artifact components are identified (requirement *ArtifactIdentification*) and rejected (requirement *DataCleaning*). Furthermore, the design can perform source reconstruction (requirement *SourceLocalization*) and neuro-feedback (requirement *Neuro-feedback*) in real-time. Corresponding computation methods (requirement *Methods* and *IdentifyEyeBlinks*) are derived from mission statement requirements.

However, previous studies have either neglected or simplified real-time artifact rejection [8],[11],[12],[19], as it is computationally extremely demanding. This thesis focused on the design of a **S**ystem-**o**n-**C**hip (SoC) capable of performing real-time artifact rejection, which is a first and essential component of the Research Center Jülich MEG-RT 2.0 project. The reconfigurable hardware designs of MEG signal pre-processing in this paper were parameterized and could be modified externally within the same hardware architectures to be generally applicable to other MEG systems. The requirements in blue in Fig. 8.1 were fulfilled in this work, while the requirements in yellow are to be covered in future developments.

The development process within this thesis is briefly summarized as follows.

The design began by studying the challenges in MEG data processing. The analysis of recordings from modern multi-channel MEG system is exceedingly costly in computation. MEG data processing usually requires filtering, artefact rejection, source localization and connectivity analysis. Processing of such a high dense spatio-temporal data stream, is usually in the range of a few days for a single MEG experiment. SoPC reconfigurable computing has been a good choice for data processing acceleration. However, with increasing complexity of SoPC design, to find a balance between

performance and resource adds more design difficulties. Therefore, an efficient and fast way of designing is required.

As an approach to solve these challenges, the first contribution presented in this thesis was proposing a requirement-driven model-based development methodology, which focused on a global system scope to cope with the complexity. The approach began by capturing and formalizing the requirements of MEG signal processing. The requirements were analyzed and real-time artifact rejection was chosen as the focus of this work. Then the system was modeled with respect to block structure and behavior. As many SysML diagrams as necessary were created, including top-level and lower-level diagrams, to detail different views of MEG signal pre-processing from different perspectives. Moreover, traceability was established by allocating the model elements to requirements.

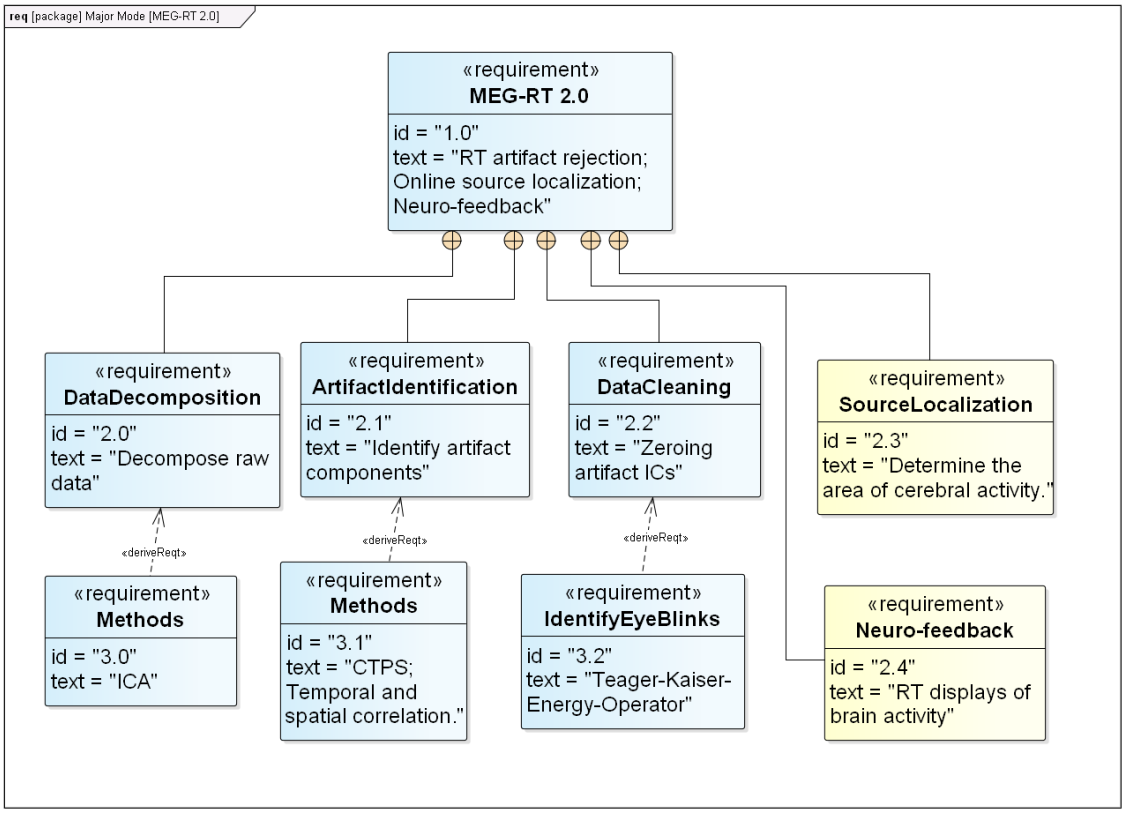

Figure 8.1: Requirement coverage of MEG-RT 2.0. The circled plus sign with a solid line indicates containment relationship

Subsequently, the constraints and parametric characteristics of MEG signal pre-processing were defined. Parametric models enabled the trade-off studies of different design configurations to find an optimal set of solutions. It also enabled checking violation of the quantifiable constraints of a design, and performing early functional prototyping (mathematical modeling). The created models for MEG signal pre-processing worked as a golden reference for further modular hardware implementation and functional verification.

After the systems engineering work, we established MEG signal pre-processing configuration generation chain, which was based on the Enterprise Architecture environment. The purpose of this chain was to automatically generate the synthesizable Verilog code of a high-level model of the MEG signal pre-processing system. A model-to-text transformation was developed generating the Verilog code representing the modeled MEG signal pre-processing configuration. This code could then be simulated or implemented on FPGA using the available simulation and synthesis tools. The automation tool significantly reduced design time of real-time MEG signal pre-processing system with increased productivity.

Finally, in the last part of the thesis, the design was justified in the real context of an FPGA implementation. This made it possible to validate the methodology developed as well as the implemented real-time MEG signal pre-processing system. This case study was also an opportunity to measure the performance of real-time MEG signal pre-processing system and deduce the efficiency against the tested applications. A comparison was conducted between the implemented work and other relevant studies, which showed the sufficient performance and efficiency of the developed solution and the RDD & MBD approach.

8.2 Outlook

The MEG-RT 2.0 project can be continued in many directions. Some are presented here:

Exploring source localization configurations

In the final part of the thesis, we exposed the MEG-RT 2.0 automation from a high-level modeling. The RDD & MBD design flow simplifies the construction of a SoC by providing simple graphical notations that are easily understood by the user. These notations make it possible to abstract the details and techniques of implementation which accelerates the design of the system.

Source localization is an important approach to image the electrical activities of deep brain structures in both fields of EEG and MEG. Different source modeling techniques have been proposed to deal with the source localization problem. Source localization is a procedure of high computational burden. However, all approaches have certain limitations [9].

The RDD & MBD approach is ideal working on the source localization problem, because real-time and embedded systems can be easily designed using SysML [117]. In this context, the high level exploration process helps the developer effectively manage the design complexities. Additionally, the choice of high-speed hardware platforms well meets the demand of providing data processing results in real-time.

Towards an MEG pre-processing optimization model

Power consumption is a very important criterion for high-performance on-chip systems. Indeed, the MEG pre-processing system is to be optimized for lower power consumption and higher clock rate. Optimization techniques can then be added in the Verilog code to maximize the performance of MEG pre-processing. For instance, the power consumption of the system can be reduced by integrating a power management module, which disables the processing elements that are in an idle state when

executing a parallel instruction. This context is particularly valid in the case of conditional statements (if-then-else). Communication between cores could be similarly accelerated. These communications can be controlled by a hardware control module instead of being guided by the instructions.

9 Appendix

9.1 SysML

SysML is a UML-based modeling language standardized by the Object Management Group (OMG). SysML takes the form of a UML profile that redefines the semantics and forms of some UML elements while defining new concepts and elements.

Like UML, SysML allows the specification, analysis, design, verification and validation of many systems. However, SysML is specifically dedicated to the processes and business characteristics implemented to model systems such as those from the fields of automotive engineering, acoustic engineering and aeronautics. Specifically, this standard takes 7 out of 13 UML 2.0 diagrams (among them, three are modified) and includes two new types of diagrams. These diagrams can be grouped into three categories. The first category is the requirement diagram, which makes it possible to represent the requirements of the system. Then there are four types of so-called structural diagrams. They allow giving a static view of the system. And finally, four behavioral diagrams that allow to represent the dynamic part of the system.

An overview of the SysML diagrams, using examples from the MEG-RT 2.0 project, is presented in following sections. First, section 9.2 presents the structural diagrams. Then, section 9.3 introduces the behavioral diagrams and an overview of the requirement diagram is given in section 9.4. Finally, traceability of diagrams is introduced in section 9.5. A very detailed description of the SysML diagrams and corresponding elements can be found in [44].

9.2 Structural diagrams

9.2.1 Block Definition Diagram

The BDD is derived from the UML class diagram. Likewise, it allows the representation of the block hierarchy of the system and a black box view of the system.

The UML class is enriched in SysML by the concept of blocks which enables the powerful expression of SysML in the field of systems engineering. A system is broken down into several parts or components. Each of them is thus modeled by a block. A block represents a part of the system, software or hardware. The symbol of block is a rectangle sub-divided into compartments. The name of the block appears in the top compartment headed by the stereotype *«block»*, as illustrated in Fig. 9.1.

A BDD has the same concepts as those used in the UML class diagram, including associations, compositions, attributes (termed properties in SysML), ports or operations.

Each block can contain properties to convey the features of the block. Structural properties include part properties, reference properties, value properties, constraint properties and ports. Part properties specify the composition of a block, which indicates a block is composed of its part properties. A reference property represents an external structure that interacts with the block. In Fig. 9.1, the block

InfomaxICA has two part properties, *Preprocess* and *Decompose*. The block *MEMORY* is a reference property to the block *Preprocess*. Value properties represent the quantities of a block. Constraint properties stand for mathematical relationships imposed on the value properties. A new type of port introduced in SysML is the flow port. It allows representing the exchanges of all kinds (electrical signals, matter or data) between the blocks in addition to the services required or offered by the components. The notation of a flow port is a small square straddling the border of a block. It is possible to specify the direction of the exchange (in, out or in/out). In Fig. 9.1, the block *Preprocess* is attached with a flow port *PCs* of data type *MEGVector*. Additionally, a block can be attached with two types of behavioral properties: operations and receptions. Behavioral properties represent a set of behaviors that this block can perform. Different properties are accommodated in the corresponding compartments of the block rectangle denoted with related headings. For example, the heading for flow port of the block is *flow ports*.

In order to connect the blocks together, two kinds of associations are used: reference association and composite association. The reference association represents a link, a simple relationship between elements. The corresponding notation is a solid line between two elements. The link between the block *MEMORY* and the block *Preprocess* in Fig. 9.1 is an instance of reference association. The composite association indicates a nesting and the fact that an element is divided into several components. It is graphically represented by a solid line with a solid diamond on the composite block side, as shown in Fig. 9.1.

The block diagram may also contain signals. A signal represents an element that can be transmitted from one block to the other. In comparison with operations, signals represent a stimulus while operations represent services offered by blocks.

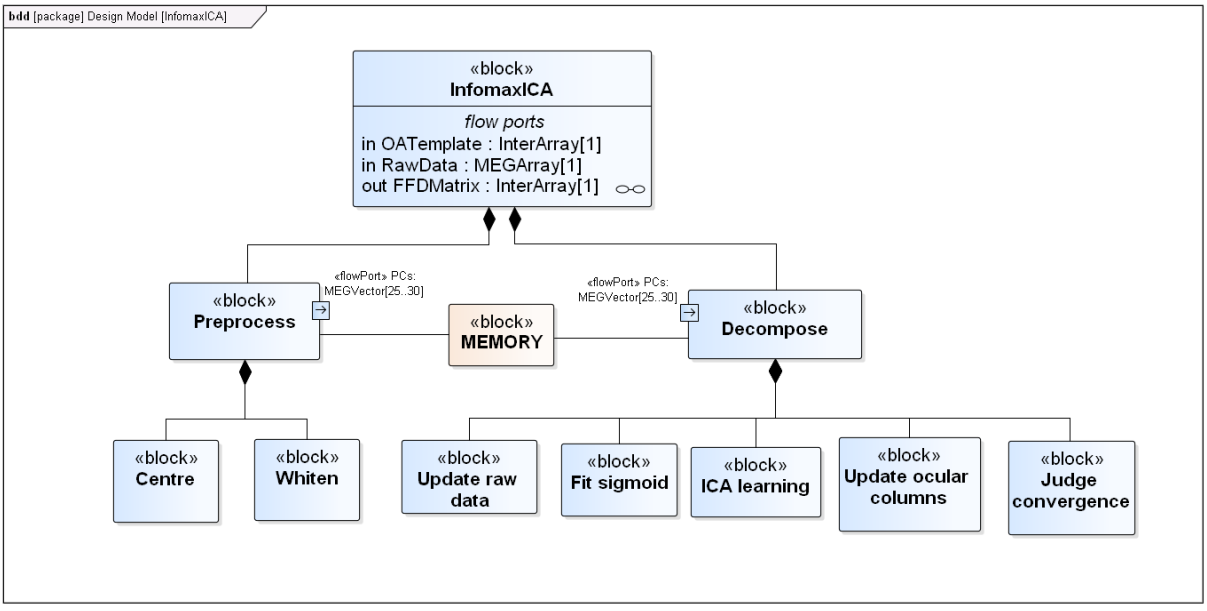

Figure 9.1: BDD of data decomposition

9.2.2 Internal Block Diagram

The internal block diagram (IBD) represents the communication links between the different parts of the system. It allows representing both software information exchanges (data structures) and exchanges materials (e.g., matter or energy). An IBD is attached to a block. The part properties of the block are instantiated and assembled by connectors through the ports on an IBD.

The internal view of a block is presented on an IBD. It displays the different properties of a block that complement the aspects conveyed on BDDs. Each property also has a type that matches the element it represents. A part property can have nested sub-properties inside to represent lower level structures. A part property on an IBD is graphically represented by a rectangle. As shown in Fig. 9.2, the part property *Decompose* is an instantiation of the block *Decompose* from Fig. 9.1. It has five nested sub-properties, *UpdateArtifactCostFunc*, *Fitsigmoid*, *UpdateRawData*, *ICALearning* and *Judge*.

The ports defined in the block are represented as small rectangles on the frame of the IBD. On the other hand, the part property ports are placed on the borders of the part properties. There are two types of ports: standard port and flow port. The graphical representation of the flow ports (the arrow inside the square indicates direction) is visible in Fig. 9.2. The connectors represent the communication paths between the elements. The corresponding symbol of a connector is a solid line between two elements.

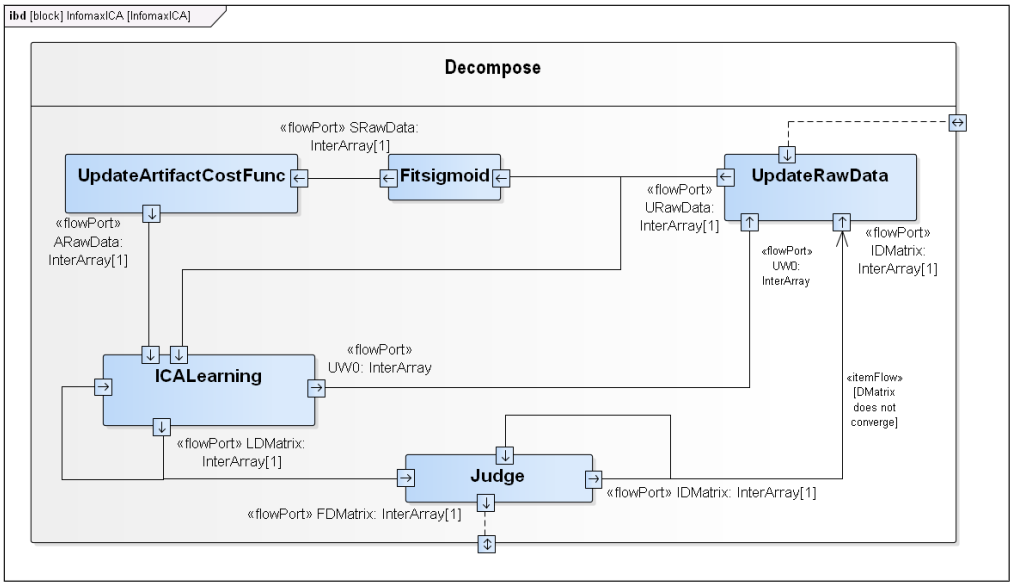

Figure 9.2: IBD (excerpt) of data decomposition

9.2.3 Parametric Diagram

The parametric diagram is used to model the physical and quantitative constraints of the system. It represents the constraints imposed on the physical parameters of the system. For this purpose, constraint blocks are used to represent mathematical expressions where each parameter can refer to an element of the model (block property for example). A parametric diagram that represents the PCA computation process of the constraint block *InfomaxICA* is shown in Fig. 9.3.

The notation of a constraint property on a parametric diagram is a round-angle. The corresponding name is defined by the developer. A constraint property is always typed by a constraint block. A constraint parameter, which represents a variable in a constraint expression, has the notation of a small square put to the boundary of a property round-angle. A constraint parameter of a constraint block straddles the frame of the corresponding parametric diagram. When a parametric diagram represents a block, the symbol of a value property is represented by a rectangle.

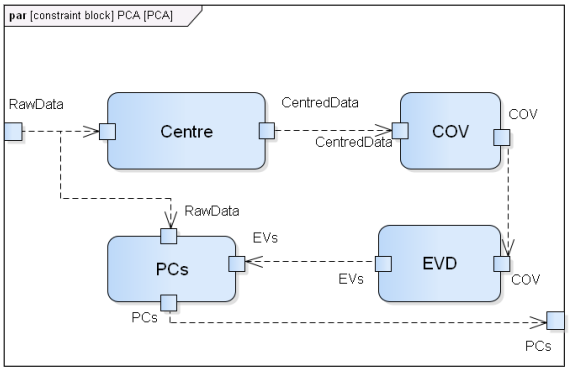

Figure 9.3: Parametric diagram of principal component analysis

9.2.4 Package Diagram

The package diagram shows the general organization of the model. The organization of the system model is determined by the package hierarchy that partitions the model elements into logically cohesive groups [44]. It allows a very simple representation of the packages used during modeling and the different links that exist between them. The notation for a package is a folder symbol. The notation with a solid line and a circled plus sign conveys that the *Exploitational* package contains four packages, *Deployment Platform*, *Life-cycle*, *System Coupling* and *Disposal*, as shown in Fig. 9.4.

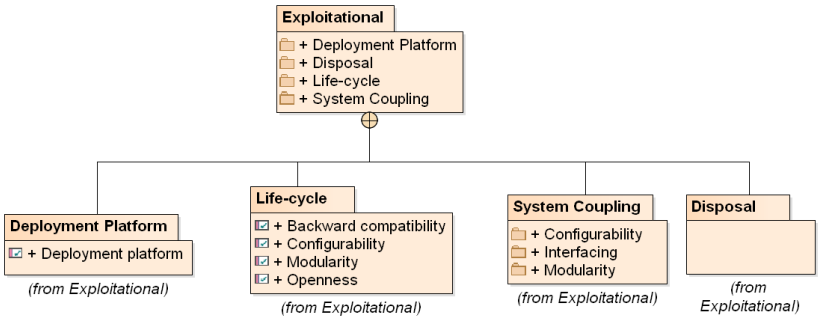

Figure 9.4: Exploitational requirements are categorized into four groups using a package diagram

9.3 Behavioral diagrams

Behavioral diagrams allow the representation of the dynamic view of the system. In SysML, there are four types of behavioral diagrams: the use case diagram, the sequence diagram, the activity diagram and the state machine diagram. The activity diagram has been extended with respect to the UML version, other behavioral diagrams come directly from the UML notation.

9.3.1 Use Case Diagram

A use case diagram represents the services that the system must provide to users (humans or machines). It is used to represent the functionality and overall operation of a system.

An example use case diagram is shown in Fig. 9.5. A use case is graphically represented by an ellipse (oval). There are two symbol representations for an actor: (1) a stick figure or (2) a rectangle applied with the stereotype *«actor»*. Typically, the stick figure notation is used to represent a person and the rectangle notation a system. An association between an actor and a use case indicates the actor interacts with the system. Three actors are shown in Fig. 9.5, two *BufferMEM*s and one *LUT*.

There are two relationships between two use cases: *include* and *extend*. The symbol of an *include* relationship is a dashed line with an open arrowhead and with the stereotype *«extend»* applied to it. It indicates when the use case at the tail end of the relationship gets invoked, the included use case at the arrowhead end is also executed. The notation for an *extend* relationship is the same as that of an *include* relationship but with the stereotype *«extend»* applied to it. It conveys that when the use case at the arrowhead end gets invoked, the extending use case at the tail end may be executed optionally [46].

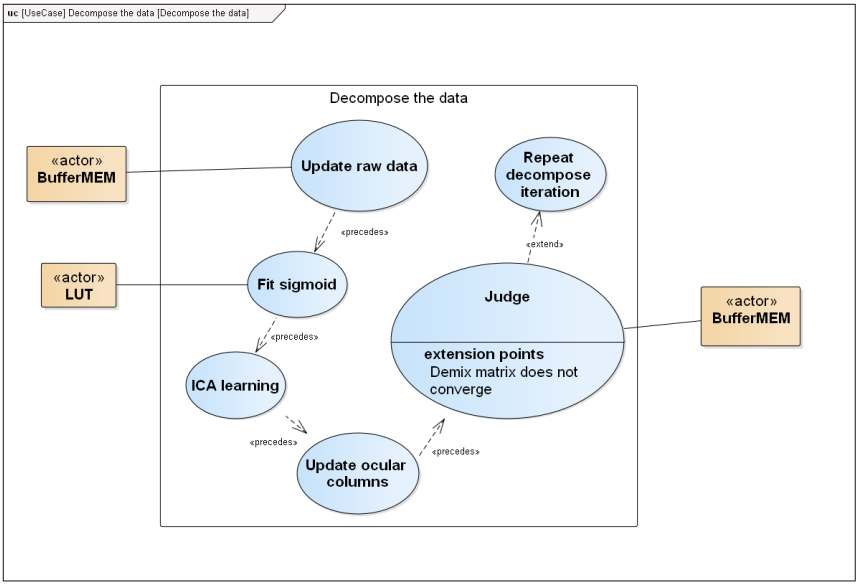

Figure 9.5: Use case diagram of data decomposition

9.3.2 Sequence diagram

The sequence diagram is used to represent the order of execution of the actions represented in the use case diagram. The sequence diagram models the chronology of interactions between system elements or between the system and its environment. In order to model a scenario (a sequence), each of the actions is represented using lifelines. A lifeline is an element that represents either roles or object instances that participate in an interaction. The lifelines are corresponding to part properties in a block that the interaction models. The graphical representation of a lifeline is a rectangle with a dashed line attached to it (see Fig. 9.6), flowing down the sequence diagram [44]. The dashed line represents the execution time of the relevant part property.

A message specifies a data exchange between two lifelines. The data exchange can be a behavior invocation, a reply, a lifeline creation or destruction. The symbol for a message is a solid line with an arrowhead that flows from the sending lifeline to the receiving lifeline. For example, in Fig. 9.6 the lifeline *Preprocess* sends a message *WhitenData()* to the lifeline *UpdateRawData*.

The thin and vertical rectangles that straddle the lifelines represent execution specifications. An execution specification defines the period of time when the lifeline is actively executing a behavior [44].

A combined fragment is a mechanism that models more complex interactions using interaction constraints (such as decisions, loops, and parallel behaviors). A combined fragment is graphically represented by a rectangle that is placed over one or more lifelines and encapsulates one or more relevant messages. The encapsulated messages that pass between those lifelines are subject to the control logic defined by that combined fragment. The interaction constraint is specified by a string and placed in a pentagon in the upper-left corner of the rectangle. The string is termed an interaction operator. For example, in Fig. 9.6 a combined fragment with a *loop* interaction operator (in the upper-left corner) specifies that the sub-modules of *DataDecomposition* could happen multiple times before the computation converges. *(0, 200)* specifies the minimum and maximum number of iterations of the loop.

9.3.3 Activity diagram

The activity diagram models the information flow dependencies and control flows of the system as flowcharts. It makes it possible to graphically detail a computational or business process (or the running of a use case) in terms of sequences of activities. An activity diagram accommodates a series of actions linked by control flows based on the execution order, optionally emphasizing the information transports using object flows. An action represents the fundamental unit of a behavior specification and cannot be further decomposed within the activity. A basic action is graphically represented by a round-cornered rectangle, as shown in Fig. 9.7. An activity may be composed of a set of actions coordinated sequentially, concurrently, or a combination of these.

Control nodes steer the execution of an activity along paths other than a simple sequence of actions. There are seven kinds of control nodes: initial nodes, activity final nodes, flow final nodes, decision

nodes, merge nodes, fork nodes, and join nodes. An initial node marks the starting point within an activity. The symbol of an initial node is a small and filled-in circle, which is allowed to have one outgoing control flow. An activity final node represents the termination of a control token flow. An activity final node is graphically represented by a circle that encapsulates a small and filled-in circle. The initial node *ActivityInitial* and final node *ActivityFinal* are shown in Fig. 9.7.

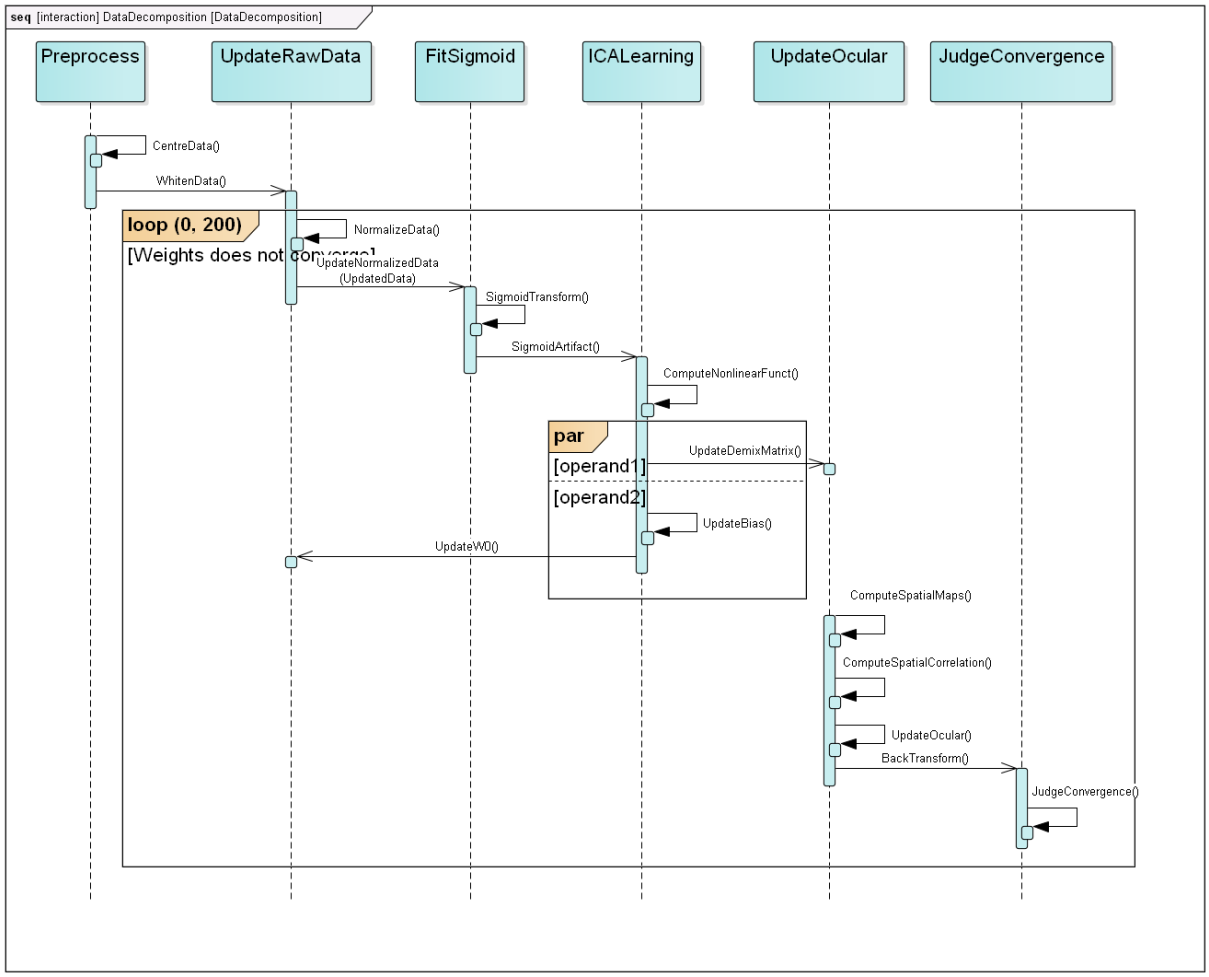

Figure 9.6: Sequence diagram of data decomposition

Furthermore, it may involve synchronization and branching. Concurrency and synchronization are modeled using fork and join nodes, whereas branching is modeled using decision and merge nodes. A decision node specifies a choice between two possible paths based on the evaluation of a guard condition (The notation is a hollow diamond, as shown in Fig. 9.7), a fork node indicates the beginning of multiple parallel control threads. Moreover, a merge node specifies a point from where different incoming control paths have to follow the same path, while a join node allows multiple parallel control threads to synchronize and rejoin.

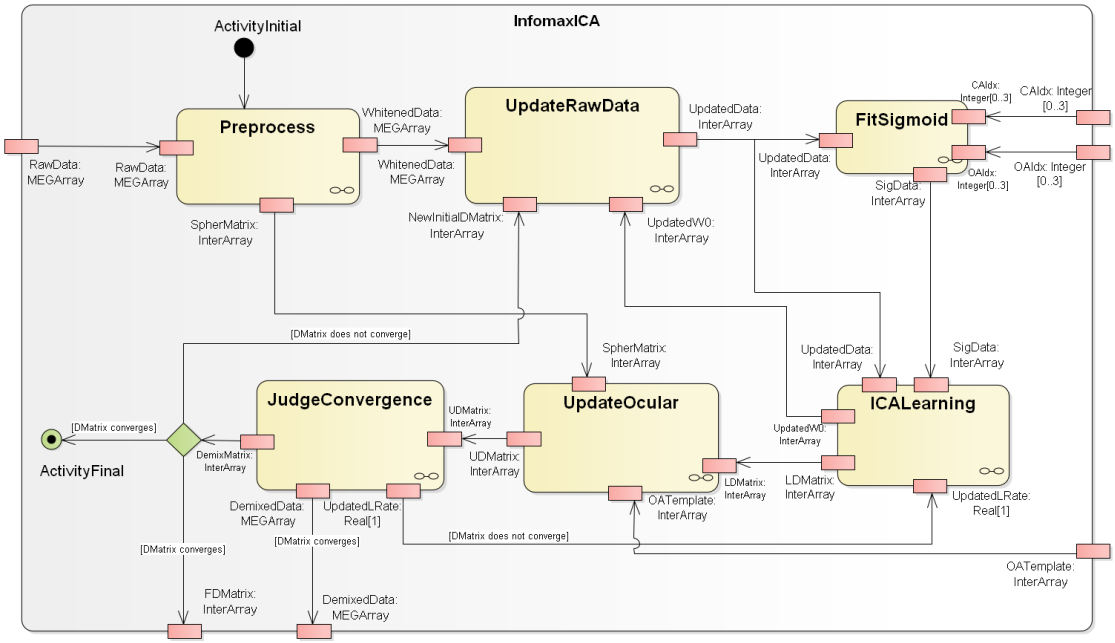

Figure 9.7: Activity diagram of data decomposition

9.3.4 State Machine Diagram

The state machine diagram describes the transitions between system states using a finite state machine. In addition to the transitions, the authorized structures are the simple state (a round-cornered rectangle), the final state (a black spot surrounded by a circle) and the composite state (a round-cornered rectangle with nested sub-states). Additionally, there are the pseudo-states such as the initial state, the choice point, the fraction bar and the junction bar. The pseudo-states are temporary states in which the system is not allowed to park.

Each state machine diagram allows only one initial state and only one transition must start from it to avoid indeterminacy. At the level of the final states, there may be several but no transition can start.

A transition is a solid line with an open arrowhead that represents a change from one state to another. Since a transition cannot be trans-hierarchical, a composite state must contain an initial state. Thus, when a transition arrives at the wall of a composite state, it goes to its initial state. If a transition is crossed, it leaves the composite state regardless of its situation. For example, in Fig. 9.8 the state *ICALearning* is a composite state, which nests four actions.

The fraction and junction bars make it possible to set up a parallelization of transitions. Only one transition is allowed to arrive on a fraction bar while several transitions can leave at the same time. The transitions that start from this bar are execution paths that run in parallel. Each of these paths evolves in its own way. In order to merge these paths (so as to interrupt the parallelism), a junction bar is used. Execution paths point to this bar, each using a transition. Only one transition is allowed to start from this bar. This transition is able to be executed on condition that all the transitions that arrive on this bar must be crossed. Even if one of the paths reaches a final state, the others continue their

execution. The notations for fraction and junction bars are the same: a line segment. A fraction bar and a junction bar are shown in the composite state *InitialTraining* in Fig. 5.14. When a transition arrives at the fraction bar, it gets duplicated on the three outgoing transitions. The three transitions then proceed to their respective downstream states in parallel: *CostFuncCA*, *CostFuncOA* and *SpatialTemplate*. The concurrent transitions finally merge at the junction bar.

The transition that arrives at a decision point (or point of choice) realizes a dynamic conditional branch. A decision point must have one single incoming transition and two or more outgoing transitions. The guards of the outgoing transitions are evaluated and only one outgoing transition is selected. On the other hand, the outgoing transitions must not carry a trigger but must obligatorily carry a guard.

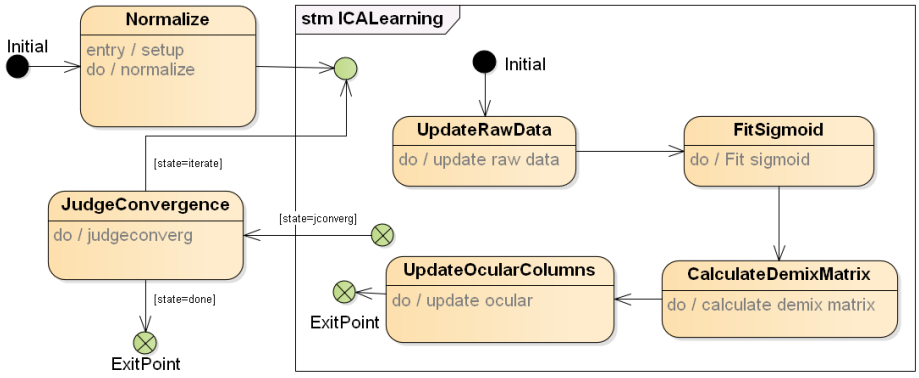

Figure 9.8: State machine diagram of data decomposition

9.4 Requirements diagram

The requirement diagram does not exist in UML. It allows you to collect and organize all stakeholder needs and system requirements. It can be used to model the technical, legal, physical, commercial, normative or other requirements of a project. The requirements are intended to ensure the adequacy of the solution (the realized system) with the needs. This diagram makes it possible to organize these requirements in a structured and hierarchical manner. This diagram has the particularity of being transversal to the model. Each requirement of the requirements diagram is defined by a description and an identifier. A hierarchy can be established between the requirements using the relationships *containment* (composition) or *derive* (one requirement derives from another). An example is represented graphically as in Fig. 9.9. Crosshair notation is used to indicate that *Deployment platform* requirement is composed of *Software platform* requirement and *Hardware platform* requirement. A *derive* relationship is a kind of dependency. The notation is a dashed line with an open arrowhead and attached with the stereotype *«deriveReqt»*. It conveys that sub-requirements can be derived from a requirement. The requirement diagram in Figure 9.9 displays several derive relationships.

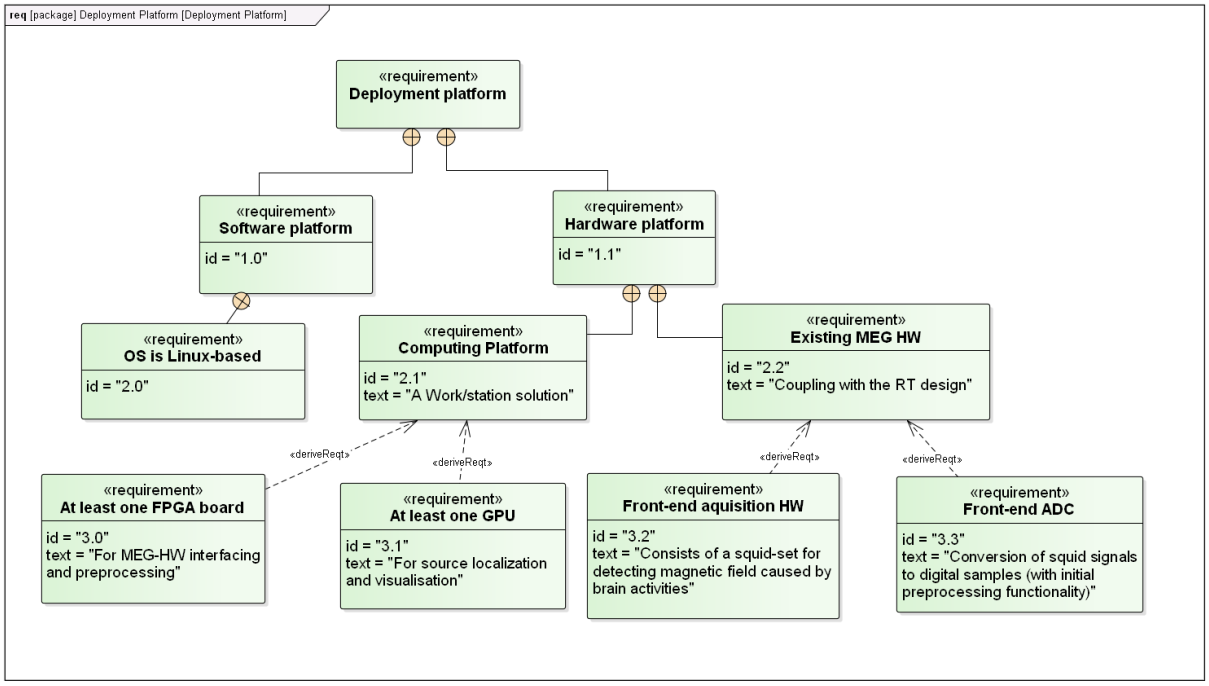

Figure 9.9: Requirement diagram of platform deployment

9.5 Traceability of diagrams

It is crucial to trace the model elements back to the requirements and make sure every requirement is satisfied. SysML support managing traceability throughout the system development process. For example, Fig. 9.10 represents the traceability from SysML design models back to the requirements in MEG signal processing.

The requirements can be linked to both structural and behavioral diagrams and the corresponding model elements through the relationships including *satisfy*, *verify*, *refine*, etc.

A *refine* relationship is a kind of dependency. The notation is a dashed line with an open arrowhead and attached with the stereotype *«refine»*. It specifies that the linked model element is more concrete than the requirement at the arrowhead end. The diagram in Fig. 9.10 conveys that the *MEGSignalProcessing* state machine diagram refines the *MEG-RT 2.0* requirement.

A *satisfy* relationship is another kind of dependency. The notation is a dashed line with an open arrowhead and attached with the stereotype *«satisfy»*. There is no constraint on the kind of the model element linked a *satisfy* relationship. By convention, however, the linked model element is always a block. The *satisfy* link is used to represent the fact that a model element satisfies a requirement. The requirements diagram in Fig. 9.10 specifies that the *DataCleaning* block satisfies the *DataCleaning* requirement.

A *verify* relationship is another kind of dependency. The notation is a dashed line with an open arrowhead and attached with the stereotype *«verify»*. The *verify* link represents the fact that a requirement is verified by a model element. The requirements diagram in Fig. 9.10 presents that the data type named *MEGSignalProcessing* verifies the *MEG-RT 2.0* requirement.

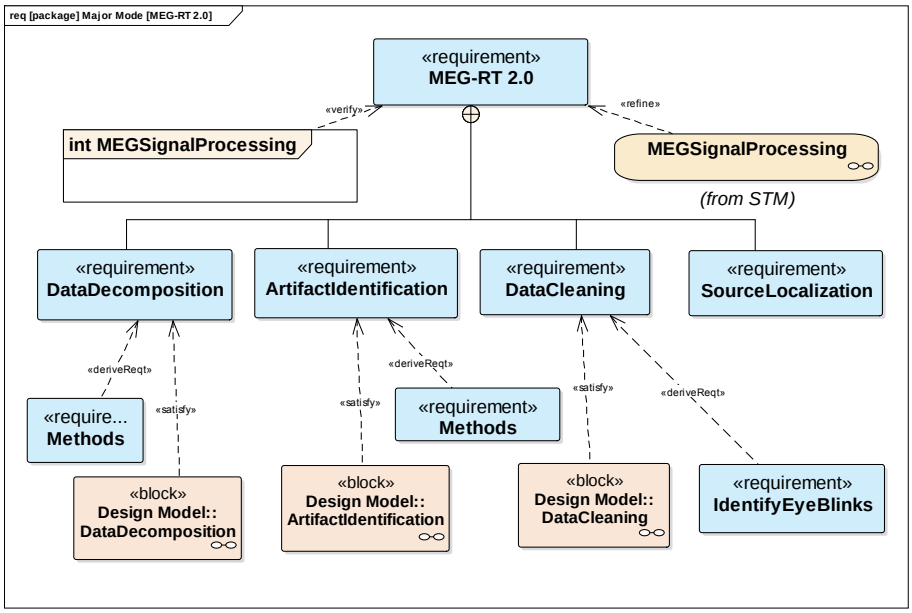

Figure 9.10: Traceability of MEG signal processing

List of Publications

Conference Proceedings

Chen, T., Suslov, S., Schiek, M., Shah, N. J., van Waasen, S., & Dammers, J. Model-Driven Development Methodology Applied to Real-Time MEG Signal Preprocessing System Design. *11th European Modelling Symposium on Mathematical Modelling and Computer Simulation*, Manchester, UK, 2017, pp. 28 - 33.

Chen, T., Suslov, S., Schiek, M., Dammers, J, Shah, N. J., & van Waasen, S. Real-time MEG data-processing unit for online medical imaging and brain-computer interface: a model-based approach. *11th International Conference on Bioelectromagnetism (ICBEM)*, Aachen, Germany, 2018.

Bibliography

[1] Weilkiens, Tim. *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[2] Office of the Deputy under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering. Systems Engineering Guide for Systems of Systems, Version 1.0. Washington, DC: ODUSD (A&T) SSE, 2008.

[3] Suslov, Sergey. *Parallelisation Potential of Image Segmentation in Hierarchical Island Structures on Hardwareaccelerated Platforms in Real-time Applications*. Forschungszentrum Jülich, 2013.

[4] Suslov, Sergey. Presentation: SysML for computing controlling system development, https://www.fz-juelich.de/SharedDocs/Downloads/ZEA/ZEA-2/DE/SysML.pdf?__blob= publicationFile, 2017.

[5] Biomagnetic Technologies Inc., "MAGNES 2500 WH-X and 3600 WH Hardware Reference Manual," BTi San Diego, California, USA, 2006.

[6] Dammers, Jürgen, et al. "Integration of amplitude and phase statistics for complete artifact removal in independent components of neuromagnetic recordings." *IEEE transactions on biomedical engineering* 55.10 (2008): 2353-2362.

[7] Breuer, Lukas, et al. "Ocular and cardiac artifact rejection for real-time analysis in MEG." *Journal of neuroscience methods* 233 (2014): 105-114.

[8] Sudre, Gustavo, et al. "rtMEG: a real-time software interface for magnetoencephalography." *Computational intelligence and neuroscience* 2011 (2011): 11.

[9] Breuer, Lukas, "Identification of Neuromagnetic Responses for Real-Time Analysis in Magnetoencephalography", RWTH Aachen University, Aachen, Germany, 2015.

[10] Rongen, H., V. Hadamschek, and M. Schiek. "Real time data acquisition and online signal processing for magnetoencephalography." *Real Time Conference, 2005. 14th IEEE-NPSS*. IEEE, 2005.

[11] Buch, Ethan, et al. "Think to move: a neuromagnetic brain-computer interface (BCI) system for chronic stroke." *Stroke* 39.3 (2008): 910-917.

[12] Mellinger, Jürgen, et al. "An MEG-based brain–computer interface (BCI)." *Neuroimage* 36.3 (2007): 581-593.

[13] Hämäläinen, Matti, et al. "Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain." *Reviews of modern Physics* 65.2 (1993): 413.

[14] Proudfoot, Malcolm, et al. "Magnetoencephalography." *Practical neurology* (2014): practneurol-2013.

[15] Zimmerman, J. E., Paul Thiene, and J. T. Harding. "Design and Operation of Stable rf - Biased Superconducting Point-Contact Quantum Devices, and a Note on the Properties of Perfectly Clean Metal Contacts." *Journal of Applied Physics* 41.4 (1970): 1572-1580.

[16] Singh, Sanjay P. "Magnetoencephalography: basic principles." *Annals of Indian Academy of Neurology* 17.Suppl 1 (2014): S107.

[17] Wheless, James W., et al. "Magnetoencephalography (MEG) and magnetic source imaging (MSI)." *The neurologist* 10.3 (2004): 138-153.

[18] Zamrini, Edward, et al. "Magnetoencephalography as a putative biomarker for Alzheimer's disease." *International journal of Alzheimer's disease* 2011 (2011).

[19] Hesse, C., et al. "On the development of a brain-computer interface system using high-density magnetoencephalogram signals for real-time control of a robot arm." (2007).

[20] Florin, Esther, Elizabeth Bock, and Sylvain Baillet. "Targeted reinforcement of neural oscillatory activity with real-time neuroimaging feedback." *Neuroimage* 88 (2014): 54-60.

[21] Ora, Hiroki, et al. "Implementation of a beam forming technique in real-time magnetoencephalography." *Journal of integrative neuroscience* 12.03 (2013): 331-341.

[22] Lee, Te-Won. "Independent component analysis." *Independent Component Analysis*. Springer, Boston, MA, 1998. 27-66.

[23] Huang, Wei-Chung, et al. "FPGA implementation of 4-channel ICA for on-line EEG signal separation." *Biomedical Circuits and Systems Conference, 2008. BioCAS 2008. IEEE*. IEEE, 2008.

[24] Zhao, Dongsheng, et al. "FPGA Implementation of FastICA Algorithm for On-line EEG Signal Separation." *CCF National Conference on Compujter Engineering and Technology*. Springer, Berlin, Heidelberg, 2014.

[25] Torti, Emanuele, et al. "Custom FPGA processing for real-time fetal ECG extraction and identification." *Computers in biology and medicine* 80 (2017): 30-38.

[26] Kim, Chang-Min, and Soo Young Lee. "A digital chip for robust speech recognition in noisy environment." *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*. Vol. 2. IEEE, 2001.

[27] Falco, Nicola, Jon Atli Benediktsson, and Lorenzo Bruzzone. "A study on the effectiveness of different independent component analysis algorithms for hyperspectral image classification." *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7.6 (2014): 2183-2199.

[28] Knuth, Kevin H. "Difficulties applying recent blind source separation techniques to EEG and MEG." *Maximum Entropy and Bayesian Methods*. Springer, Dordrecht, 1998. 209-222.

[29] Jadhav, Sangeeta D., and Anjali S. Bhalchandra. "Blind source separation: trends of new age-a review." (2008): 251-254.

[30] Pal, Madhab, et al. "Blind source separation: A review and analysis." *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*. 2013.

[31] Karamizadeh, Sasan, et al. "An overview of principal component analysis." *Journal of Signal and Information Processing* 4.03 (2013): 173.

[32] Ali, Amine Ait Si, et al. "Hardware PCA for gas identification systems using high level synthesis on the Zynq SoC." *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*. IEEE, 2013.

[33] Perera, Darshika G., and Kin Fun Li. "Analysis of single-chip hardware support for mobile and embedded applications." *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. IEEE, 2013.

[34] Shahrouzi, S. Navid, and Darshika G. Perera. "Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices." *EURASIP Journal on Embedded Systems* 2017.1 (2017): 25.

[35] Bell, Anthony J., and Terrence J. Sejnowski. "An information-maximization approach to blind separation and blind deconvolution." *Neural computation* 7.6 (1995): 1129-1159.

[36] Amari, Shun-Ichi. "Natural gradient works efficiently in learning." *Neural computation* 10.2 (1998): 251-276.

[37] Brown, Glen D., Satoshi Yamada, and Terrence J. Sejnowski. "Independent component analysis at the neural cocktail party." *Trends in neurosciences* 24.1 (2001): 54-63.

[38] Dammers, Jürgen, and Michael Schiek. "Detection of artifacts and brain responses using instantaneous phase statistics in independent components." *Magnetoencephalography*. InTech, 2011.

[39] Hahn, Stefan L. *Hilbert transforms in signal processing*. Vol. 2. Boston: Artech House, 1996.

[40] Stephens, Michael A. "Use of the Kolmogorov-Smirnov, Cramér-Von Mises and related statistics without extensive tables." *Journal of the Royal Statistical Society. Series B (Methodological)* (1970): 115-122.

[41] Arsham, H. "Kuiper's P-value as a measuring tool and decision procedure for the goodness-of-fit test." *Journal of Applied Statistics* 15.2 (1988): 131-135.

[42] Abboud, Shimon, Omer Berenfeld, and Dror Sadeh. "Simulation of high-resolution QRS complex using a ventricular model with a fractal conduction system. Effects of ischemia on high-frequency QRS potentials." *Circulation research* 68.6 (1991): 1751-1760.

[43] Friedenthal, Sanford, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[44] Delligatti, Lenny. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[45] Object Management Group. OMG Systems Modeling Language (OMG SysML), v1.3. June 2012. Available at http://www.omg.org/spec/SysML/1.3/.

[46] Object Management Group. OMG Systems Modeling Language (OMG SysML), v1.2. June 2010. Available at http://www.omg.org/spec/SysML/1.2/.

[47] Estefan, Jeff A. "Survey of model-based systems engineering (MBSE) methodologies." *Incose MBSE Focus Group* 25.8 (2007).

[48] Estefan, Jeff. "MBSE methodology survey." *Insight* 12.4 (2009): 16-18.

[49] Royce, Winston W. "Managing the development of large software systems: concepts and techniques." *Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, 1987.

[50] Boehm, Barry W. "A spiral model of software development and enhancement." *Computer* 21.5 (1988): 61-72.

[51] Rook, Paul. "Controlling software projects." *Software Engineering Journal* 1.1 (1986): 7-16.

[52] Forsberg, Kevin, and Harold Mooz. "The relationship of system engineering to the project cycle." *INCOSE International Symposium*. Vol. 1. No. 1. 1991.

[53] Kevin, Forsberg, and Hal Mooz. "Application of the Vee to Incremental and Evolutionary Development." *Proceedings of the International Council on Systems Engineering (INCOSE) Conference, St. Louis, MO*. 1995.

[54] Yadav, Ravi Shanker. "Improvement in the V-Model." *International Journal of Scientific & Engineering Research* (2013).

[55] Rosenberg, Doug, and Sam Mancerella. "Embedded systems development using SysML: an illustrated example using enterprise architect." *Sparx Systems Pty Ltd and ICONIX* (2010): 4-14.

[56] Fuller, Samuel H., and Lynette I. Millett. "Computing performance: Game over or next level?." *Computer* 44.1 (2011): 31-38.

[57] Rahman, Rezaur. *Intel® Xeon Phi™ Coprocessor Architecture and Tools*. Apress, 2013.

[58] Hamblen, James O., Tyson S. Hall, and Michael D. Furman. *Rapid prototyping of digital systems: SOPC edition*. Vol. 1. Springer Science & Business Media, 2007.

[59] Farmahini-Farahani, Amin, Sied Mehdi Fakhraie, and Saeed Safari. "SOPC-based architecture for discrete particle swarm optimization." *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*. IEEE, 2007.

[60] Altera, M. "DS-EXCNIOS-01" Nios Soft Core Embedded Processor Data Sheet"." (2000).

[61] Inc, Xilinx. "Zynq-7000 all programmable SoC overview." (2012).

[62] Santarini, Mike. "Zynq-7000 EPP sets stage for new era of innovations." *Xcell journal* 75 (2011): 8-13.

[63] Mittal, Sparsh, and Jeffrey S. Vetter. "A survey of CPU-GPU heterogeneous computing techniques." *ACM Computing Surveys (CSUR)* 47.4 (2015): 69.

[64] Youness, Hassan, et al. "Accelerated Processing Unit (APU) potential: N-body simulation case study." *Computer Engineering & Systems (ICCES), 2016 11th International Conference on*. IEEE, 2016.

[65] Daga, Mayank, Ashwin M. Aji, and Wu-chun Feng. "On the efficacy of a fused CPU+ GPU processor (or APU) for parallel computing." *Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on*. IEEE, 2011.

[66] Vanderperren, Yves, Wolfgang Mueller, and Wim Dehaene. "UML for electronic systems design: a comprehensive overview." *Design automation for embedded systems* 12.4 (2008): 261-292.

[67] Murphy, Brett, Amory Wakefield, and Jon Friedman. *Best practices for verification, validation, and test in model-based design*. No. 2008-01-1469. SAE Technical Paper, 2008.

[68] Linehan, Eamonn, and Siobhán Clarke. "An aspect-oriented, model-driven approach to functional hardware verification." *Journal of Systems Architecture* 58.5 (2012): 195-208.

[69] Davies, Jim, et al. "Compositionality and refinement in model-driven engineering." *Brazilian Symposium on Formal Methods*. Springer, Berlin, Heidelberg, 2012.

[70] Gajski, Daniel D., et al. *High—Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.

[71] System Level Design Development Working Group. "VSI Alliance: Model Taxonomy." (2001).

[72] Bailey, Brian, Grant Martin, and Thomas Anderson, eds. *Taxonomies for the Development and Verification of digital systems*. Springer Science & Business Media, 2005.

[73] Martin, Grant, Brian Bailey, and Andrew Piziali. *ESL design and verification: a prescription for electronic system level methodology*. Morgan Kaufmann, 2010.

[74] Mohanty, Srikant Kumar, Suchismita Sengupta, and S. K. Mohapatra. "Test bench automation to overcome verification challenge of SOC Interconnect." *Man and Machine Interfacing (MAMI), 2015 International Conference on*. IEEE, 2015.

[75] Nidhra, Srinivas, and Jagruthi Dondeti. "Black box and white box testing techniques-a literature review." *International Journal of Embedded Systems and Applications (IJESA)* 2.2 (2012): 29-50.

[76] Khan, Mohd Ehmer. "Different approaches to white box testing technique for finding errors." *International Journal of Software Engineering and Its Applications* 5.3 (2011): 1-14.

[77] Sagdeo, Vivek. "Standard Delay Format." *The Complete Verilog Book* (1998): 321-363.

[78] Golshan, Khosrow. *Physical Design Essentials*. Springer Science+ Business Media, LLC, 2007.

[79] Sivaraman, Mukund, and Andrzej J. Strojwas. *A unified approach for timing verification and delay fault testing*. Springer Science & Business Media, 2012.

[80] Dale, Corby L., et al. "Timing is everything: neural response dynamics during syllable processing and its relation to higher-order cognition in schizophrenia and healthy comparison subjects." *International Journal of Psychophysiology* 75.2 (2010): 183-193.

[81] Lauer, Richard T., et al. "Applications of cortical signals to neuroprosthetic control: a critical review." *IEEE transactions on rehabilitation engineering* 8.2 (2000): 205-208.

[82] Geoffrey Sparks, "Enterprise Architect User Guide", Sparx Systems, 2014.

[83] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1998.

[84] Charoensak, Charayaphan, and Farook Sattar. "Design of low-cost FPGA hardware for real-time ICA-based blind source separation algorithm." *EURASIP Journal on Advances in Signal Processing* 2005.18 (2005): 173453.

[85] Li, Zhongfeng, and Qiuhua Lin. "FPGA implementation of Infomax BSS algorithm with fixed-point number representation." *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on*. Vol. 2. IEEE, 2005.

[86] Wang, Jia-Ching, et al. "VLSI Design for Convolutive Blind Source Separation." *IEEE Trans. on Circuits and Systems* 63.2 (2016): 196-200.

[87] Gupta, Suyog, et al. "Deep learning with limited numerical precision." *International Conference on Machine Learning*. 2015.

[88] Gander, Walter. "Algorithms for the QR decomposition." *Res. Rep* 80.02 (1980): 1251-1268.

[89] Hahn, Stefan L. *Hilbert transforms in signal processing*. Vol. 2. Boston: Artech House, 1996.

[90] Romero, David Ernesto Troncoso, and Gordana Jovanovic Dolecek. "Digital FIR Hilbert transformers: fundamentals and efficient design methods." *MATLAB-A Fundamental Tool for Scientific Computing and Engineering Applications-Volume 1*. InTech, 2012.

[91] Lim, Yong Ching, and Ya Jun Yu. "Synthesis of very sharp Hilbert transformer using the frequency-response masking technique." *IEEE Transactions on Signal Processing* 53.7 (2005): 2595-2597.

[92] Lehto, Raija, Tapio Saramäki, and Olli Vainio. "Synthesis of wideband linear-phase FIR filters with a piecewise-polynomial-sinusoidal impulse response." *Circuits, Systems and Signal Processing* 29.1 (2010): 25-50.

[93] Romero, David Ernesto Troncoso, Miriam Guadalupe Cruz Jimenez, and Gordana Jovanovic Dolecek. "A new Pipelined-Interleaved structure for FIR Hilbert transformers based on frequency transformation technique." *Communications Control and Signal Processing (ISCCSP), 2012 5th International Symposium on*. IEEE, 2012.

[94] Wang, Avery, and J. O. Smith. "On fast FIR filters implemented as tail-canceling IIR filters." *IEEE Transactions on Signal Processing* 45.6 (1997): 1415-1427.

[95] Stephens, Michael A. "Use of the Kolmogorov-Smirnov, Cramér-Von Mises and related statistics without extensive tables." *Journal of the Royal Statistical Society. Series B (Methodological)* (1970): 115-122.

[96] Smirnov, Nickolay. "Table for estimating the goodness of fit of empirical distributions." *The annals of mathematical statistics* 19.2 (1948): 279-281.

[97] Benesty, Jacob, et al. "Pearson correlation coefficient." *Noise reduction in speech processing*. Springer Berlin Heidelberg, 2009. 1-4.

[98] Henderson, Charles R. "Estimation of variance and covariance components." *Biometrics* 9.2 (1953): 226-252.

[99] Wang, Yue, et al. "Singular value decomposition hardware for MIMO: State of the art and custom design." *2010 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2010.

[100] Senning, Christian, et al. "Hardware-efficient steering matrix computation architecture for MIMO communication systems." *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*. IEEE, 2008.

[101] Snopce, Halil, and Ilir Spahiu. "Parallel Computation of the SVD." *In practice* 1 (2011): 2.

[102] Rahmati, Masih, Mohammad Sadegh Sadri, and Mehdi Ataei Naeini. "FPGA based singular value decomposition for image processing applications." *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*. IEEE, 2008.

[103] Ledesma-Carrillo, Luis M., et al. "Reconfigurable FPGA-Based unit for Singular Value Decomposition of large mxn matrices." *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*. IEEE, 2011.

[104] Sattar, F., and C. Charayaphan. "Low-cost design and implementation of an ICA-based blind source separation algorithm." *ASIC/SOC Conference, 2002. 15th Annual IEEE International*. IEEE, 2002.

[105] Charoensak, Charayaphan, and Farook Sattar. "A single-chip FPGA design for real-time ICA-based blind source separation algorithm." *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*. IEEE, 2005.

[106] Shyu, Kuo-Kai, et al. "Implementation of pipelined FastICA on FPGA for real-time blind source separation." *IEEE transactions on neural networks* 19.6 (2008): 958-970.

[107] Du, Hongtao, Hairong Qi, and Gregory D. Peterson. "Parallel ICA and its hardware implementation in hyperspectral image analysis." *Independent Component Analyses, Wavelets, Unsupervised Smart Sensors, and Neural Networks II*. Vol. 5439. International Society for Optics and Photonics, 2004.

[108] Du, Hongtao, and Hairong Qi. "An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images." *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*. Vol. 5. IEEE, 2004.

[109] Huang, Wei-Chung, et al. "FPGA implementation of 4-channel ICA for on-line EEG signal separation." *Biomedical Circuits and Systems Conference, 2008. BioCAS 2008. IEEE*. IEEE, 2008.

[110] Van, Lan-Da, Di-You Wu, and Chien-Shiun Chen. "Energy-efficient FastICA implementation for biomedical signal separation." *IEEE transactions on neural networks* 22.11 (2011): 1809-1822.

[111] Yang, Chia-Hsiang, Yi-Hsin Shih, and Herming Chiueh. "An 81.6 uW FastICA Processor for Epileptic Seizure Detection." *IEEE transactions on biomedical circuits and systems* 9.1 (2015): 60-71.

[112] Shih, Wei-Yeh, et al. "An effective chip implementation of a real-time eight-channel eeg signal processor based on on-line recursive ica algorithm." *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*. IEEE, 2012.

[113] Roh, Taehwan, et al. "A wearable neuro-feedback system with EEG-based mental status monitoring and transcranial electrical stimulation." *IEEE transactions on biomedical circuits and systems* 8.6 (2014): 755-764.

[114] Kuriakose, Jini, and Jayan K. George. "Multilevel Power Optimization for ICA Processor."

[115] Kim, Chang-Min, et al. "FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling." *IEEE Transactions on Neural Networks* 14.5 (2003): 1038-1046.

[116] Chen, Tsan-Yu. *A System-on-Chip Design of 32-Channel EEG Acquisition System with Automatic Artifacts Rejection.* MS Thesis. National Chiao Tung University, 2016. Web. 20 Jun. 2018.

[117] Basit-Ur-Rahim, Muhammad Abdul, Fahim Arif, and Jamil Ahmad. "Modeling of real-time embedded systems using SysML and its verification using UPPAAL and DiVinE." *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*. IEEE, 2014.

DOI: 10.17185/duepublico/70157
URN: urn:nbn:de:hbz:464-20190603-144731-7