Module Guide (MG)
STEM Moiré GPA

Alexandre Pofelski
macid: pofelska
github: slimpotatoes

November 3, 2017

1 Revision History

Table 1: **Revision History**

Date	**Version**	**Notes**
31/10/2017	1.0	First Draft

Contents

List of Tables

List of Figures

2 Introduction

The module guide document is providing to the reader the decomposition of STEM Moiré GPA into smaller pieces to help the implementation phase. The decomposition is based on the principle of information hiding [2] which interest relies on its flexibility to design change. The module design follows the rules layed out by [1], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

The module guide (MG) follows the Software Requirements Specification (SRS) available in the documentation tree. Terminologies, symbols and acronyms used in the document are described in the SRS and TestPlan documents. The rest of the document is organized as follows. Section 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section section 7 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 8 describes the use relation between modules.

3 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in section 3.1, and unlikely changes are listed in section 3.2.

3.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision.

AC 1 Hardware running STEM Moiré GPA

AC 2 SMH, ICREF, inputs file format

AC 3 Small strain and small gradient of strain assumption

AC 4 SMH simulation from a reference algorithm

AC 5 GPA algorithm

AC 6 Mask function for frequency isolation (in GPA)

AC 7 Fitting algorithm to get unstrained reference (in GPA)

AC 8 Output format

AC 9 Data plotting

3.2 Unlikely Changes

The unlikely changes are design decisions fixing some aspects of STEM Moiré GPA. The lack of modularity is compensated by a simplification in the software design. If any of the elements are modified, important changes would be expected in the design. Therefore the following decisions are expected to stay unchanged.

UC 1 SMH data type (2D arrays)

UC 2 Perfect 2D periodic scanning grid assumption

UC 3 Pixel data type (real number)

UC 4 Keyboard/mouse interface device with user

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M 1 Hardware Module

M 2 Input Module

M 3 Output Module

M 4 Fourier Transform Module

M 5 Mask Module

M 6 Gradient Module

M 7 Least Square Fitting Method Module

M 8 Phase operation Module

M 9 Strain calculation Module

M 10 Plotting Module

M 11 GUI Module

M 12 GPA Module

M 13 SMH simulation Module

M 14 Unstrained reference calculation Module

M 15 Moiré to Crystal conversion Module

M 16 Control Module

M 17 Data Structure Module

Level 1	**Level 2**	**Level 3**
Hardware-Hiding Module		
Behaviour-Hiding Module	Input	
	Output	
	Control Module	
	SMH Simulation	Fourier Transform
	Unstrained reference calculation	Least Square Fitting Method
	Moiré to Crystal conversion	
	Strain calculation	
	GPA	Fourier Transform Mask Phase operation Gradient
Software Decision Module	Data StructureSS GUI Plotting	

Table 2: Module Hierarchy

5 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by [1]. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

6.1 Hardware Hiding Modules (M 1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

6.2.1 Input Module (M 2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: STEM Moiré GPA

6.2.2 Output Module (M 3)

Secrets: Output format

Services: Transforms data to be usable by the plotter and the hardware device

Implemented By: STEM Moiré GPA

6.2.3 Fourier Transform Module (M 4)

Secrets: Fourier transform algorithm

Services: Transforms data into its Fourier transform

Implemented By: External Python library

6.2.4 Least Square Fitting Method Module (M 7)

Secrets: Least Square Fitting algorithm

Services: Fit set of data into a 2D linear function using the Least squares method

Implemented By: External Python library

6.2.5 Gradient Module (M 6)

Secrets: 2D Gradient algorithm

Services: Perform the 2D derivative of a data set.

Implemented By: External Python library

6.2.6 Phase Module (M 8)

Secrets: Algorithm to wrap/unwrap the phase

Services: Wrap/unwrap the phase by adding/removing discontinuity in the interval [0,2π]

Implemented By: External Python library

6.2.7 SMH simulation Module (M 13)

Secrets: Algorithm to simulate the SMH from the reference image

Services: Simulate the SMH in Fourier space using the pixel size p of $I_{SMH_{\text{exp}}}$ using the Fourier transform the image reference $I_{C_{\text{ref}}}$

Implemented By: STEM Moiré GPA

6.2.8 Strain calculation Module (M 9)

Secrets: Algorithm to calculate the strain from distortion matrix

Services: Calculate the strain level on each pixel of the data

Implemented By: STEM Moiré GPA

6.2.9 Control Module (M 16)

Secrets: Execution flow of STEM Moiré GPA

Services: Calls the different modules in the appropriate order

Implemented By: STEM Moiré GPA

6.2.10 Moiré to Crystal conversion Module (M 15)

Secrets: Algorithm to convert a Moiré wave vector to a crystalline wave vector

Services: Affine vectorial transformation using the Moiré data from GPA and the sampling vectors user inputs.

Implemented By: STEM Moiré GPA

6.2.11 Unstrained reference calculation Module (M 14)

Secrets: Algorithm to calculate the unstrained reference

Services: Calculation of the unstrained reference based on the user input U and the result of the fitting method.

Implemented By: STEM Moiré GPA

6.2.12 Mask Module (M 5)

Secrets: Algorithm to isolate a portion of an image

Services: Function isolating and weighting a sub area of an image

Implemented By: STEM Moiré GPA

6.2.13 GPA Module (M 12)

Secrets: GPA method

Services: Map the variation of a vector in Fourier space related the the variation of the periodicity in real space

Implemented By: STEM Moiré GPA

6.3 Software Decision Module

6.3.1 Plotting Module (M 10)

Secrets: Method to transforming the data format into plot format

Services: Transform data into format usable by the hardware to display data into readable format for the user

Implemented By: External Python library

6.3.2 GUI Module (M 11)

Secrets: Method to interact with the user

Services: Provide an interactive interface for the user to feed the information needed for the proper execution of STEM Moiré GPA.

Implemented By: External Python library

6.3.3 Data Structure Module (M 17)

Secrets: Data format for an image

Services: Provide convenient format to store, read and manipulate all elements (pixel) from an image

Implemented By: External Python library

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [3] said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

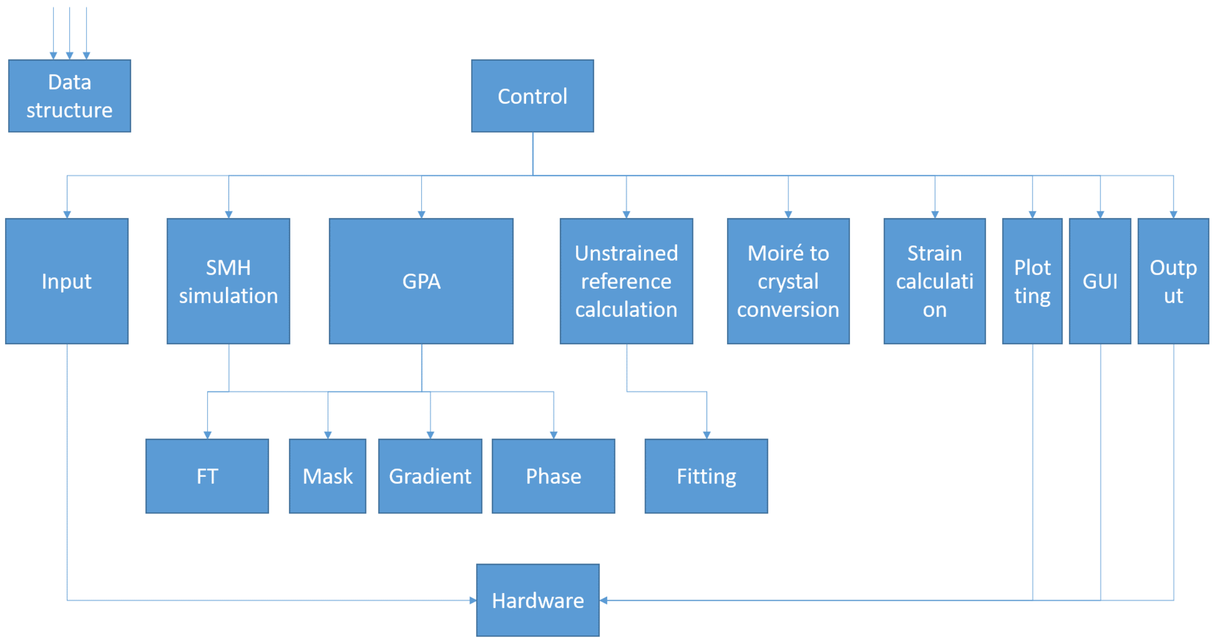

Figure 1: Use hierarchy among modules

References

[1] D. Parnas, P. Clement, and D. M. Weiss, "The modular structure of complex systems," in *International Conference on Software Engineering*, pp. 408–419, 1984.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Comm. ACM*, vol. 15, pp. 1053–1058, December 1972.

[3] D. L. Parnas, "Designing software for ease of extension and contraction," in *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, (Piscataway, NJ, USA), pp. 264–277, IEEE Press, 1978.