ASML-AT:Turbo Scanner Software Architecture

Rick van Lierop and Joost Zonneveld

The ASML AT:Turbo Scanner component based layered software architecture is presented. It is shown by an example how new functional requirements are captured using UML Activity diagrams, how functionality is decomposed over layers in the software architecure and assigned to components in these layers, and how the presented software architecture fullfills its requirements.

Introduction

The ASML ATLAS[1] project is defining AT:Turbo systems, a new generation of Scanner Systems for the 300mm wafer market. Scanners are used in the production of Integrated Circuits. Scanners form only one step in the production process (Figure 1). The production process is explained next. Wafers are coated with a light-sensitive material in a so-called Track. Then they are exposed in a Scanner. Hereafter the coating is developed in the Track again. Coated Wafers are then processed in various ways in other parts of the Chip Factory. Ion implants, metal diffusion, and other process steps are performed here. This process is repeated a number of times, depending on the complexity of the Device at hand. After all Layers of a Wafer are processed, the Wafer is sawn into pieces corresponding to the Devices that were thus formed. These Devices are mounted in a carrier and sealed in a protective package. This is what we know as an Integrated Circuit, e.g. a micro-processor chip or DRAM-chip.

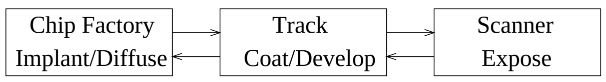

Figure 1: Scanner in production environment

[1]ATLAS is acronym for Advanced Technology for Large Area Substrates.

Scanner System Concept

The Scanner System is conceptually very simple (Figure 2). An Image is projected, using a Light source and a Lens, on a Wafer. The carrier for the Image is called a Reticle. The projection of the Image is called an Exposure.

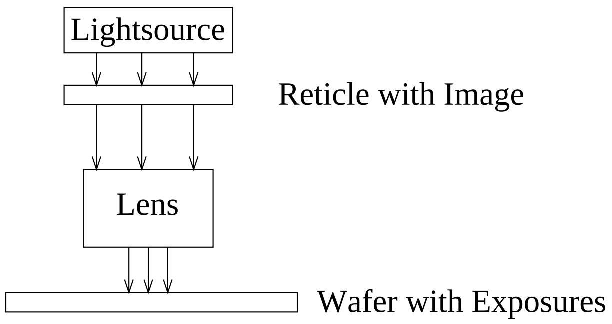

Figure 2: Scanner concept

An Image, and consequenlty an Exposure, can contain multiple Dies. The Dies correspond to the Devices formed on the Wafer after all Layers are processed. Wafers are processed in batches. A batch is called a Lot. Every Wafer in a Lot is undergoing the same process steps. From the Scanner point of view, a Lot is a number of Wafers that will be exposed according to the same Layer Definition. All Layer Definitions together for a Lot of Wafers, form a so-called Recipe.

User Requirements

The primary user requirements are straightforward:

- **Critical Dimension**: the smallest possible dimension of features in a chip design that can successfully be produced. The general rule of thumb is that Critical Dimension determines the amount of devices that can be produced per wafer.
- **Overlay**: the precision with which different layers of a wafer design can be placed on top of each other. Large Overlay value leads to short circuit or broken connection situations. Small Overlay values leads to low-power and/or high-frequency devices. Rule of thumb is that this determines how much money our customer can ask per device.

  Throughput: the number of Wafers that can be produced per scanner per time interval. High production speed, high uptime, and consequently low downtime (high MTBF, and low MTTR) of Scanners determine the Throughput value that can be reached.

Software Architecture Requirements

User requirements do not state the need for software architecture. So, there must be another reason. At ASML, a large number of software developers is working on a family of scanner products. Each of the family members differs in a number of aspects from other scanner family members. We do not want to invent the wheel over and over again. This is where software architecture comes into the picture. Software architecure must fulfil the following requirements:

- **Flexibility**: It must be easy to incorporate new functionality. Architecure should be prepared for future changes.
- **Modularity**: A number of software development teams must be able to work in parallel.
- **Reusability**: Reuse of documentation, design, and implementation must be possible. In all parts, commonalities shall be split off, and made available separately to increase reuse.
- **Modifiability**: Architecure must be easy to comprehend. Functionality is easily localised.
- **Correctness**: The architecture shall provide the means to verify correctness of the implementation, both beforehand (during design), and afterwards (after implementation).

Global Architecture

Layered Model

The Component Based [SZY] Layered [BUS] Scanner Architecture Model (Figure 3) provides the conceptual solution to the problem at hand (=requirements). Functionality is decomposed over components in the layers according to the description of the functionality for a layer. At one hand this significantly narrows down the amount of possible solutions, and at the other hand, this helps making decisions on how to realise the required functionality.

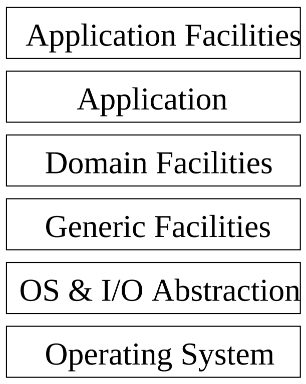

Figure 3: Layered Scanner Architecture model

- **Operating System Layer**: contains SW Components providing basic SW functionality, dedicated to a (range of) specific processor board(s), for multi-processor support, multi-process support, multi-thread support, interrupt support, interprocess-communication, error handling, file I/O, disk access, networking, I/O devices, printer output, etc. Generally these are commercial off-the-shelf (COTS) SW Components extensible with device specific drivers to support optional hardware components.
- **OS and I/O Abstraction Layer**: The SW Components in this layer provide the functionality to be independent of the specific underlying hardware and operating system, in the sense that one

interface to the functionality in the (various) Operating System(s) is provided. This may imply that: 1. only the common subset of available functionality of the various operating systems is supported 2. functionality that is required, but not supported on all underlying operating systems, is supplemented. 3. access to some functionality of the underlying operating systems is restricted, by not providing it on the interface

- **Generic Facilities Layer**: contains the SW Components providing facilities that are generic. Generic in the sense that they could be present in various applications, independent of the specific Application Domain (Toaster, Coffee Machine, Wafer Scanner).
- **Domain Facilities Layer**: contains the SW Components providing facilities that are specific for the Application Domain. Specifc in the sense that they would not be present in the various applications like Generic facilities, but only in the applications of the specific Domain (e.g our Competitors, Nikon, and Canon, products).
- **Application Layer**: Components in this layer implement the application functionality: sequencing of production/calibration/diagnostics/selftest Lots, sequencing of activities to execute Lots, scheduling of activities, executing synchronised activities, handling exceptions, and recovering from exceptions
- **Application Facilities Layer**: This layer contains the functionality needed to operate the application. It provides local/remote operator the interface to define Lots, inspect logs, test, diagnose, and calibrate.

Table 1 shows an overview of Global Architecture layered model and some typical components for ASML.

	Layer	**Typical Components for ASML Scanners**
6	Application Facilities Layer	Recipe Editor, Job Definition
5	Application Layer	refer to next paragraph Application Layer
4	Domain Facilities Layer	Motion Control
3	Generic Facilities Layer	Communication Network, Configuration Manager, Exception Handler
2	OS and IO Abstraction Layer	Portability Layer, Virtual Machine
1	Operating System Layer	Solaris, VxWorks

Table 1: Typical components in Global Architecture layers

Application Layer

The Application Layer itself consists of a layered model (Figure 4). Again, functionality is decomposed over the layers according to the description of the functionality for a layer, and again this helps in making decisions on how to realise the required functionality.

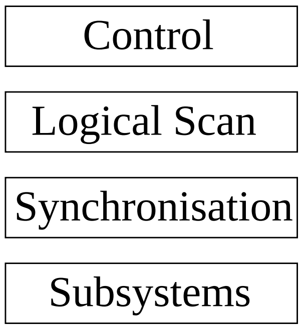

Figure 4: Application Layer structure

- **Subsystems Layer**: contains drivers for the subsystems defined in the System Design phase.
- **Synchronisation Layer**: realises tightly synchronised physical scan actions between subsystems.
- **Logical Scan Layer**: translates logical scans into physical scan actions.
- **Control Layer**: realises required functional behaviour by the sequencing of logical scans.

Table 2 shows an overview of Application Layer layered model and some typical ASML components, that will beencountered in the next example.

	Sub-Layer	Typical Components for ASML Scanners
4	Control Layer	Measure Control
3	Logical Scan Layer	Alignment Logical Scan
2	Synchronisation Layer	Synchronisation Control
1	Subsystems Layer	Stage Driver, Alignment Sensor Driver, Interferometer Driver

Table 2: Typical components in sub-layers of Application Layer

Alignment example

The Alignment function is primarily leading to the realisation of the Overlay requirement. In this (simplified) example one particular part of the Alignment function is worked out: Wafer to Chuck Alignment. The example shows how functional system requirements are gathered and translated into software functionality, decomposed over components and sub-layers of the Application Layer.

Wafer to Chuck Alignment Function

The purpose of the Wafer to Chuck Alignment function is to determine the horizontal relation (in X, Y, and Rz[2]) between the Wafer surface and the Wafer Chuck[3] (Figure 5). This information is used when exposing images to reach required Overlay values. Fiducial 1/Fiducial 2 are present to allow determining Chuck position. Wafer Mark1 and Wafer Mark2 allow determining Wafer position. The Wafer is placed on a Chuck. The Chuck is positioned (in X, Y, and Rz) by the Wafer Stage subsystem. The Wafer Stage subsystem is controlled by the Wafer Stage subsystem driver. An Interferometer subsystem provides very accurate information about the Chuck position.

Within ASML, Activity Diagrams [RUM] have proven to be successfull in capturing requirements on the process steps to be performed. Next to the diagram, a description (omitted in this example) for every Activity is made, containing Pre-/Post-condition, In/Output parameters, Exceptions, and Component assignment. Figure 6 presents an Activity Diagram that shows what has to be done to perform the Wafer to Chuck Alignment function.

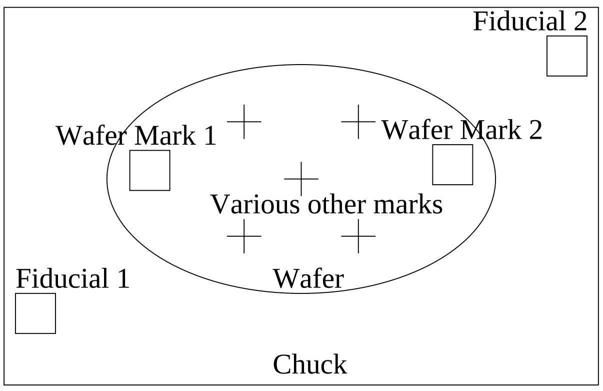

Figure 5: Wafer placed on a Chuck

Description of the Acitivies:

- **ACF1/ACF2**: Align on Chuck Fiducial1/Fiducial2: these activities determine the position (X, Y, and Rz) of the Chuck Fiducials with respect to the Interferometer Coordinate System. Together they determine Translation, Rotation, and Magnification (expansion/shrinkage of several nano-meters because of some milli-Kelvin temperature changes) of the Chuck.
- **CAM1/CAM2**: Coarse Wafer Align on Primary Wafer Mark1/Mark2: these activities determine the position (X, Y, and Rz) of the Wafer Marks with respect to the Interferometer Coordinate System. Together they determine Translation, Rotation, and Magnification (expansion/shrinkage) of the Wafer.
- **FA**: Fine Wafer Alignment on extra alignment marks defined by customer. This function determines the position (X, Y, and Rz) of the extra alignment marks with respect to the Interferometer Coordinate System. The gathered information is used to determine local deformation of

[2] Rz denotes rotation around the Z-axis

[3] Though the term Alignment suggests a displacement of the Wafer with respect to the Chuck, this function only determines the position of the Wafer with respect to the Chuck

the Wafer surface.

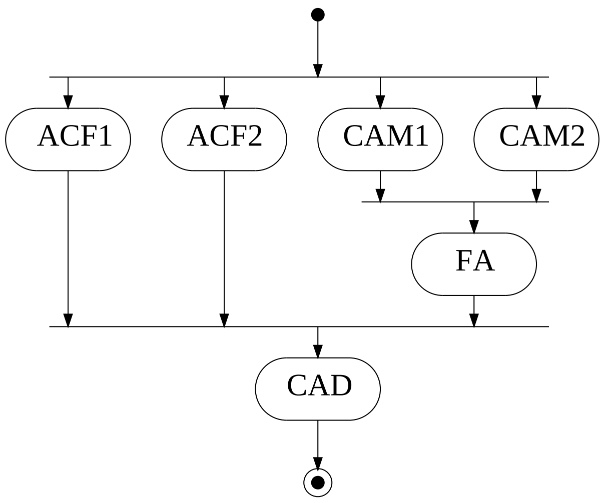

Figure 6: Activity Diagram for Wafer to Chuck Alignment Function

- **CAD**: Calculate Alignment Data. The Position of the Wafer Surface (X, Y, and Rz) with respect to the Wafer Chuck is determined, based on the information gathered during the other activities.

From the Activity Diagram above it can be seen that for a correct implementation of this function:

- Calculate Alignment Data (CAD) can only be performed after all other steps are performed. It needs the results of all other steps.
- Fine Wafer Alignment (FA) can only be done after Coarse Wafer Alignment on Primary Wafer Mark1 (CAM1) and Coarse Wafer Alignment on Primary Wafer Mark2 (CAM2) is ready.
- Alignment on Chuck Fiducial1/Fiducial2 (ACF1/ACF2), Coarse Wafer Alignment on Wafer Mark1/Mark2 (CAM1/CAM2) can be done in any order.

However, there is only one Chuck. The Chuck is needed for all of these four activities. This together with knowledge about the system geometry and subsystems performance leads to an optimal sequence of activities (Figure 7). After a Wafer is loaded on the Chuck, Fiducial 1 is nearest to the Alignment Sensor. Then Wafer Mark1 is nearest, followed by Wafer Mark2. After Aligning on Fiducial 2, Fine Alignment can take place, which in turn solves the preconditions for the calculation.

Architectural Design

The optimal sequence from the previous paragraph is implemented by Measurement Control Component [Control Layer]. The Sequence consists of (4 + N) Logical Alignment Scans (2 Fiducials + 2 Primary Wafer Marks + N extra Wafer Marks) plus a calculation. The Alignment Logical Scan Component [Logical Scan Layer] translates these logical scans to physical scans, indicating which subsystems need to do what. The logical scan of a Fiducial results in one or more physical scans (e.g. one in X- and one in Y-direction).

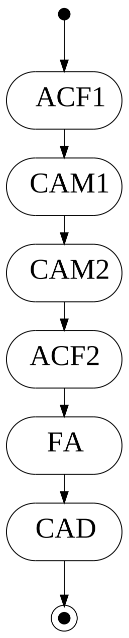

Figure 7: Throughput optimal Wafer to Chuck Alignment sequence

Synchronisation Control [Synchronisation Layer] is requested to determine the exact timing of the physical scans. The Wafer Stage Subsystem [Subsystem Layer] performs the movement involved in the scan. The Interferometer System [Subsystem Layer] determines the exact position of the Wafer Stage with respect to the Interferometer System during the movement and supplies this information to the Alignment Sensor Driver. The Alignment Sensor Driver [Subsystem Layer] takes intensity samples of the Alignment Sensor. Together with the supplied Wafer Stage position information, the position of the Alignment Mark being scanned can be accurately determined using a dedicated algorithm.

In the Collaboration Diagram [RUM] depicted in Figure 8, the involved Components, Layers, and the functions they use and provide, are shown.

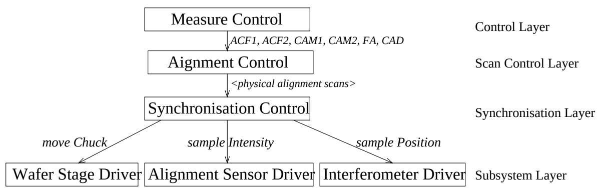

Figure 8: Collaboration Diagram for Wafer to Chuck Alignment Function

Conclusions

From the example[4] above, it can be seen that the software architecture requirements are fulfilled:

- **Flexibility**: It is easy to add extra functionality according to the defined Software Architecure. A new subsystem would require a new Subsystem Driver and, if new Logical Scans must be defined, possibly a new Logical Scan Component.
- **Modularity**: All Subsystem Drivers and Logical Scan Components can be developed in parallel.
- **Reusability**: Though not specifically addressed in this short example, interfaces and behaviour are re-used over components in a layer. Several design patterns [GAM] are applied to realise this.
- **Modifiability**: Functionality is easily localised. The specific Logical Scan Component can be found in the Logical Scan layer, while the specific Subsystem Drivers can be found in the Subsystem layer.
- **Correctness**: Though UML Activity Diagrams are not very strictly formally defined, it is possible to define this yourself. This way, formal specification languages (e.g. CCS, CSP, Chi) can be used to formally verify correctness of the production process design beforehand. Currently a post graduate student of the Technical University of Eindhoven is verifying the production process design of the new AT:Turbo systems, using Chi [ROO]. Not shown in this short example is that all Components incorporate a tracing mechanism to allow run-time tracing of the implemented behaviour. This allows verifying the implemented behaviour against the designed behaviour.

References

[BUS] F. Buschman et. al, *A System of Patterns: Pattern Oriented Software Architecture* John Wiley and Sons. ISBN 0471958697

[SZY] Clemens Szyperski, *Component Software: Beyond Object-Oriented Programming* Addison-Wesley Pub Co. ISBN 0201178885

[GAM] E.Gamma et. al, *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley Pub Co ISBN 0201633612

[BRO] W. J. Brown et. al, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis* John Wiley and Sons ISBN 0471197130

[RUM] J. Rumbaugh et. al, *OMG Unified Modeling Language Specification, version 1.3* Obtainable at: www.rational.com

[FOW] M. Fowler, *UML Distilled: Applying the Standard Object Modeling Language* Addison-Wesley Pub Co. ISBN 0201325632

[ROO] J. E. Rooda, *The Modeling of Industrial Systems* Eindhoven University of Technology, Netherlands. Lecture Notes

About the authors

Rick van Lierop. University of Eindhoven 1981-1988 (Elektrotechniek). Graduate of Prof. dr. ing. D. K. Hammer/Prof. ir. M.P.J. Stevens (Digital Communication Systems). Software Architect and System Architect for Stork Digital Imaging 1988–1997. Working since September 1997 with ASM Lithography for the ATLAS Software Development

[4]The new AT:Turbo systems being described contain 7 logical Scan Components and 12 Subsystem Drivers. This example shows only 1 logical Scan Component and 3 Subsystem Drivers, but the same principles apply to the other Logical Scan Components and Subsystem Drivers.

Department in the function of Software Architect and Software Architecture Teamleader.

Joost Zonneveld. University of Delft 1986-1991 (Technische Natuurkunde). Software Architect and System Architect for Philips Medical Systems 1991–1998. Working since August 1998 with ASM Lithography for the ATLAS Software Development Department in the function of Software Architect.