Generic Verification Platform

Definition And Implementation Of A
Generic Verification Platform
For Integrated Circuit Systems

Master's Thesis

at

Graz University of Technology

submitted by

Edwin Taferner

Institute for Software Technology
Graz University of Technology
A-8010 Graz, Austria

September, 12$^{\text{th}}$ 2010

© Copyright 2010 by Edwin Taferner

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
Advisor: Dipl.-Ing. Bernd Janger

sensordynamics
Leading in micro and wireless sensor products

TU Graz

Kurzfassung

Die Verifikation integrierter Schaltungen (IC) wird immer mehr zum Engpass der Produkteinführungszeit. Die sich ständig ändernde Technologie und die immer kompakteren und komplexeren Designs stellen immer höhere Anforderungen an die Verifikation von IC Systemen. Eine durchgängige Verifikation während den Phasen der Produkt-Entwicklung muss die zukünftige Fehlerfreiheit des Produkts in der Volumenproduktion sicherstellen.

Ein Vorschlag für eine systematische und strukturierte Vorgehensweise und Infrastruktur für die Verifikation ist die generische Verifikationsplatform. Sie bietet eine produkt-unabhängige Umgebung (Datenstrukturen, Dateiformate) und unterstützt die Verifikation mit Software-Tools wie Ausführungsautomatisierung. Durch diese einheitliche Verifikationsumgeben wird eine hohe Wiederverwendbarkeit der Verifikationselemente als auch eine enorme Steigerung der Effizienz der IC Verifikation gewährleistet. Durch das entwickelte Automatisierungskonzept kann die Verifikationszeit stark reduziert werden.

Der Einsatz der generischen Verifikationsplatform in einigen Projekten lässt auf eine vielversprechende Steigerung der Effizienz und Wiederverwendbarkeit schließen.

Schlüsselwörter: Verifikationsplatform, integrierte Schaltungen, System-on-a-Chip

Abstract

The improvement of efficiency in Integrated Circuit (IC) verification is an upcoming challenge the semiconductor industry is facing. The fast changing and shrinking technology of IC systems let the verification process become the bottle neck of time to market. The IC system verification is getting more and more time consuming because of the increasing compactness and complexity of IC systems like System-on-a-Chip (SoC)

A generic verification platform is a possible solution to overcome these challenges, by providing a project independent verification environment (data structures, file formats) with supporting software tools, like execution automation, based on this environment. These well-defined data structures and file formats decrease verification development time because of a high reuse of software, which improves over time. The provided project independent automation concept reduces verification time to a minimum.

The first projects using this generic verification platform concept confirm a very promising reuse and increasing efficiency.

Keywords: Integrated Circuit, Verification, Platform, System-on-a-Chip

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Contents

Contents ii

List of Figures iii

List of Tables iv

List of Acronyms vi

Acknowledgements vii

1 Introduction 1
- 1.1 Scope - Verification Platform . . . 1
- 1.2 Verification Environments . . . 2

2 Requirements 4
- 2.1 Requirements Elicitation . . . 4
  - 2.1.1 Elicitation Techniques . . . 4
  - 2.1.2 Technique Selection . . . 5
- 2.2 Concept Definition Requirements . . . 7
  - 2.2.1 Data Structures . . . 7
- 2.3 Software . . . 9
  - 2.3.1 Simulation . . . 10
  - 2.3.2 Evaluation . . . 10
  - 2.3.3 Production Test . . . 10

3 Status quo and Analysis 12
- 3.1 Verification Environments . . . 12
  - 3.1.1 Digital Simulation . . . 13
  - 3.1.2 Analog Simulation . . . 13
  - 3.1.3 Evaluation . . . 13
  - 3.1.4 Production Test / Characterization . . . 13
  - 3.1.5 Conclusion . . . 14
- 3.2 Data . . . 14
- 3.3 Work Flow . . . 17

4 Technology Overview **18**
4.1 Verification Platform . . . 18
4.2 Test Case/Data Management . . . 19
4.3 Test Description Languages . . . 20
4.4 Measurement Automation . . . 21

5 Specification **23**
5.1 Platform Concept . . . 23
5.1.1 Overview . . . 23
5.1.2 Description of Components . . . 23
5.1.3 Capabilities . . . 25
5.1.4 Boundaries . . . 25
5.2 Directory Structure and File Formats . . . 25
5.2.1 Embedding into IP Directory Structure . . . 25
5.2.2 Specs . . . 30
5.2.3 Plans . . . 38
5.2.4 Tests Content . . . 42
5.2.5 Results Structure . . . 50
5.2.6 Reports . . . 52
5.2.7 Verification Software . . . 52
5.3 Test Sequence Language . . . 53
5.3.1 Scope . . . 54
5.3.2 Test Data Definition . . . 55
5.3.3 Sequence Commands . . . 58
5.3.4 Variables . . . 65
5.3.5 Grammar Definition . . . 65
5.4 Verification Automation Concept . . . 69
5.4.1 Verification Platform Framework . . . 71

6 Results and Evaluation **77**
6.1 Improvements . . . 77
6.2 Risks . . . 78
6.3 Feedback . . . 78
6.4 Impact . . . 79

7 Conclusion **80**

Bibliography **83**

A Verification Automation Concept **84**

List of Figures

2.1 Logical tool organisation 11

3.1 SensorDynamics verification V-Model 12
3.2 Data model 15
3.3 Design verification work flow 17

5.1 Platform concept overview 24
5.2 SensorDynamics verification V-diagram 25
5.3 Embedding of verification platform directory in IP directory structure 27
5.4 Tests directory structure 28
5.5 Result directory structure 29
5.6 Design Verification Matrix - Header 31
5.7 Design Verification Matrix - specification parameters and test case definition . . 32
5.8 Design Verification Matrix - Simulation data 33
5.9 Design Verification Matrix - Product level definitions 34
5.10 Design Verification Matrix - Results 35
5.11 Design Verification Matrix - Parameter groups 36
5.12 Design Verification Matrix - Verification status 36
5.13 Design Verification Matrix - Chart tool 37
5.14 Design Verification Matrix - GR charts 37
5.15 Simulation plan spread sheet 39
5.16 Evaluation plan spread sheet 40
5.17 Lot list spread sheet 42
5.18 Test Case directory content 43
5.19 Relation of test files to entities of evaluation environment 43
5.20 External setup file format 46
5.21 Result directory structure 51
5.22 Sequence file - Relations 56
5.23 Automation concept - see Appendix A 69

A.1 Automation concept 85

List of Tables

5.1 Verification quantities . . . 33
5.2 Prefixes and abbreviations . . . 44
5.3 Units and abbreviations . . . 44
5.4 External setup conditions . . . 46
5.5 Compare operators . . . 59
5.6 Measurement quantities . . . 59
5.7 Measurement qualifiers . . . 61
5.8 Operator abbreviations . . . 61
5.9 Measurement operations . . . 62
5.10 Sequence language context free grammar rule set . . . 66
5.11 Sequence language command rule set . . . 67
5.12 Sequence language header terminals . . . 67
5.13 Sequence language command terminals . . . 68
5.14 Sequence language general terminals . . . 68
5.15 Measurement elements description . . . 74

List of Acronyms

AEC	Automotive Electronics Council
ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
ATG	Automatic Test data Generation
ATML	Automatic Test Markup Language
BNF	Backus-Naur-Form
CC	Critical Characteristic
CFG	Context Free Grammar
Cpk	Process Capability Index
CPU	Central Processing Unit
Csh	C-shell
DUT	Device Under Test
DV	Design Verification
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
GR	Gauge Repeatability
GRR	Gauge Repeatability and Reproducibility
GUI	Graphical User Interface
IC	Integrated Circuit
IP	Intellectual Property
JAD	Joint Application Development
JLCC	JTAG Like Control Chain
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench

NI	National Instruments
PCB	Printed Circuit Board
PXI	PCI eXtensions for Instrumentation
RTL	Register Transfer Level
SC	Significant Characteristic
SD	SensorDynamics
SI	Système International d'unitès
SoC	System-on-a-Chip
SWI	Single Wire Interface
TB	Test Bench
TC	Test Case
TCDL	Test Case Description Language
TCL	Tool Command Language
TMS	Test Management Solution
UART	Universal Asynchronous Receive/Transmit
VCS	Version Control System
VHDL	Very-high-speed integrated circuit Hardware Description Language
VSdE	Virtuoso Specification driven Environment
WCAG	Web Content Accessibility Guidelines
XML	Extensible Markup Language
XSD	XML Schema Definition

Acknowledgements

I would like to thank SensorDynamics AG who provided me the opportunity for my master's thesis. My special thanks goes to Prof. Franz Wotawa and Bernd Janger who supported and advised me during my research. My gratitude also goes to Oliver Gerler who proofread this thesis.

Last but not least, I would like to thank my parents and my girlfriend for their love, encouragement, emotional and financial support and understanding.

Edwin Taferner
Graz, Austria, August 2010

Chapter 1

Introduction

The semiconductors industry as fast growing market is facing challenges like shrinking technologies, the compactness of Integrated Circuit (IC) systems (e.g. system-on-a-chip) or the pressure to get the product to the market in time. The verification process takes up to fifty percent of the project development time. As the complexity of IC systems increases, the verification of these systems becomes more and more the bottleneck of time-to-market, since verification becomes more complicated and may also be error-prone. To overcome the challenge of complex System-on-a-Chip (SoC) verification, Electronic Design Automation (EDA) vendors provide solutions for parts of verification, like simulation or laboratory automation. Combining these solutions to a generic verification system is the next step to increase performance and decrease time-to-market.

Integrated circuits design From the beginning, SensorDynamics (SD) AG has been using a hierarchical approach in the IC design. Based on system specification, modules (Intellectual Property (IP)) are defined, which are specified and subdivided into sub-modules. This approach has the advantage of high reusability of the modules and an early start of verification. Therefore a detailed and clear module specification is required early on.

Integrated circuit verification The hierarchical design approach enables an early verification of sub systems, which leads to reduction of test time and reduction of debugging costs. Since this early verification can only be done using simulations, and simulation models are not 100% accurate, the verification after design and production is also very important. The evaluation of the target system uncovers these inaccuracies and helps to improve the next generation of the target system as well as the simulation models. Also the product needs to be tested in mass-production, which is the last stage of verification. A subset of the defined tests for simulation and evaluation are transferred to production test, where test time is a critical and costly factor which needs to be optimized.

1.1 Scope - Verification Platform

SD wants to establish a systematic verification approach which covers all three main steps in IC verification - Simulation, Evaluation, ATE Test - within an unified infrastructure. This

infrastructure should decrease time-to-market, improve verification efficiency and company processes related to verification. The platform should provide transferability of verification data, reproducibility and comparability of results and an automated approach to the verification process.

Task A generic verification platform needs to be defined, which provides all information, structures and data needed to verify/test an IC system. The verification data is subdivided into Test Case (TC) (or tests), which can refer to one or more IC system requirement parameters. Based on this defined verification platform a set of software tools is needed, which support and facilitate the use and acceptance of the platform.

These supportive software tools will **not** do any verification on their own. The intention of the tools is to provide data in a format a specific verification environment is able to use. Furthermore the automation of verification, handling of measurement results or the analysis of results could be done by software tools. The automation has to be limited to automated test execution. An Automatic Test data Generation (ATG)[16] can not be implemented, because of the high complexity of IC systems. Test data generation needs the know how of design engineers.

1.2 Verification Environments

Integrated circuit verification is done in several stages: design verification (simulation), product evaluation and production test.

Design Verification Design verification is using simulations on the analogue as well as the digital side. Digital simulations are based on test benches, where design modules are embedded and stimuli data is applied. Simulation results may be analysed fully automatically, since these simulations are script based. Analogue simulations work differently to digital, stimuli are applied to the design module, but results are analysed by the design engineer. SD has defined an IP module design guideline, which also defines the simulations required for analogue and digital modules.

Evaluation At evaluation the fundamental electrical characteristics (voltage, frequency, temperature behaviour etc.) of an IC system are determined based on laboratory measurements. This includes the check of electrical parameters versus specification, simulation and design parameters on golden samples.

Production test Production test is the in-limit check of electrical characteristics(voltage, frequency, temperature behaviour etc.) of an IC system using Automatic Test Equipment (ATE). The major focus is put on decreasing of test time with high test coverage, since this is the bottleneck and most expensive factor.

Verification result correlation The electrical characteristics (voltage, frequency, temperature behaviour etc.) are compared between simulations, laboratory measurements and production

test results. This includes the comparison of electrical parameters measured in the laboratory versus the production test environment on golden samples at least at three temperatures (product defined low, room and high temperature).

Chapter 2

Requirements

In this chapter the requirements to the verification platform are collected. The first section discusses the elicitation of the requirements. The following sections describe the requirements found via elicitation.

2.1 Requirements Elicitation

The phase of *Requirements Elicitation* is a significant part of a project. Insufficient knowledge about target system requirements will lead to failure of any project and this can be counteracted with a structured and well-defined requirements elicitation process. This section describes some basic as well as advanced requirements elicitation techniques and the selection of these techniques.

2.1.1 Elicitation Techniques

The most obvious approach to elicit target system requirements is taking **Interviews**. Interviews, in terms of requirements elicitation, could be done in different ways. Goguen and Linde [19] differentiate three types of interviews. These are *questionnaires, open ended interviews* and *focus group interviews*. *Questionnaires* can be well prepared, but may have the disadvantage that the respondent does just think about asked questions. For this reason, dependent on the content of the questionnaire, important requirements could stay covert. *Open ended interviews* prevent this issue, because the respondent answers not to a specific question, but explains requirements. For an acceptable output of this technique, the respondent needs to be encouraged by the interviewer to provide information. Therefore this technique relies on the interviewing quality of the requirement engineer. Different from above mentioned techniques, *focus group interviews* do not have dialogue character. The focus group members for the interview are the stakeholders of the project. The interaction inside the focus group could be on the one hand helpful on the other hand obstructive. The interviewer gives some input to the group and leads the discussion. To get a representative output from *focus group interviews*, the selection of the focus group members is a critical factor.

For a rough, unstructured requirements information gathering, **Brainstorming** could be the adequate approach. Raghavan et al. [35] describe brainstorming as "*a simple group tech-*

nique for generating ideas. It allows people to suggest and explore ideas in an atmosphere free of criticism or judgement". A brainstorming session consists of two phases, the *generation* and the *consolidation* phase. In the first phase as many ideas as possible, related to the topic, are collected without treatment. Afterwards the gathered ideas are structured and reviewed, to achieve a significant quality improvment to the outcome of the *generation* phase and overall outcome. The findings of this technique offer many different views of the target system. [35]

Discussion is a common and widely used technique for requirements elicitation. Assets and drawbacks are similar to *focus group interviews.*

Joint Application Development (JAD) is intended to develop a common understanding about what the target system shall look like. Raghavan et al. [35] state that *"Using that process, the developers help the users formulate problems and explore solutions, and the users gain a feeling of involvement, ownership, and commitment to the success of the system"*. Since this technique is intended to be used with customers, JAD will not be described here. This concept is also described by Wood and Silver [45].

The **PIECES framework** offers a set of categories, which should be treated by requirements elicitation. This framework represents a good starting point, if a requirements engineer has no idea about requirements elicitation. The PIECES categories are *Performance, Information (and Data), Economy, Control (and Security), Efficiency* and *Services*[35]. The category *Performance* addresses throughput or response time, for example tasks per time. *Information and data* covers input, output and stored data. The category *Economy* refers to costs and profits, whereas *Efficiency* addresses the reduction of wasted time (machine as well as human). The *Control and Security* category involves data safety and security, but also the control of input and output data etc. The behaviour of the target system, in terms of communications (to human as well as to machine), usability, compatibility, etc., is examined by the *Service* category.

Since every project has different aspects, the PIECES framework needs to be tailored. The framework provides guidance through the requirements elicitation process, especially for analysis of former projects. Merchant [33] provides a checklist which supports the use of the PIECES framework.

2.1.2 Technique Selection

To get a first overview of requirements, open ended interviews (no questionnaire) were done with responsible engineers of design, test and product engineering department. With the help of the adapted PIECES check list[33] a considerable set of requirements could be found. The PIECES framework is a very useful tool, because system verification had already been done before. With the PIECES check list significant requirements were found early in the elicitation process. The adapted check list includes the following items:

- Performance
  - Performance of currently used software
  - Needed/favoured performance

- Information and Data
  - Stored Data
    * How is test/verification data stored?
    * How should test/verification data be stored?
    * Redundancies
    * Revisions
    * Data organisation
    * Accessibility
    * Existing (fixed) data formats
  - Input/Output (referring software)
    * Existing interfaces
    * Type of exchanged data
- Economics
  - Current costs known? Licensed software?
  - Costs too high?
  - Budget
- Control and Security
  - Need security be considered?
  - Access control
  - Privacy
  - Are there inconveniences because of control/security?
- Efficiency
  - Redundancies at current verification process
    * redundant input or output
    * redundantly stored data
  - test/verification execution
  - test/verification automation
  - improvement opportunities
- Service
  - What software is provided at the moment?
  - Functionality of currently used software?
  - What kind of software should be provided? Functionality?
  - Usability (Graphical User Interface (GUI), command line based)
  - Has the required software to be compatible to existing data/software?

The outcomes of these interviews were used as a starting point for a brainstorming session. The so uncovered requirements were collected and reviewed in a discussion with all stakeholders. These requirements are described subsequently.

2.2 Concept Definition Requirements

The concept of a generic (multi-project) verification platform needs to be defined in detail, with attention to the illustration of boundaries and capabilities. The concept has to define data structures, data flow and control flow of verification.

There are general demands on the definition of the concept. The concept has to stay as clear and simple as possible without reduction of capabilities. No multiple implementations of basic software like file parsers etc. shall be allowed.

On top of all the requirements the following characteristics are to be achieved:

- **Reproducibility** of results
- **Comparability** of TC results of different environments
- **Automation** of system evaluation
- **Transferability** of TCs between environments

These characteristics need to be fundamental to all considerations.

2.2.1 Data Structures

A part of the concept is the definition of the directory and file structures inside an IP as well as file content formats. The specified files need to describe the subsequent aspects.

- **Description of test activities**
  The *What*, *When* and *How* of a test procedure.
  E.g. wait 5ms then measure voltage on Pin_x or apply stimuli to Pin_y.
- **Internal system configuration**
  Determined state of the Device Under Test (DUT).
  E.g. Register_x = value.
- **External system configuration** (at boundary of DUT)
  E.g. Supply Voltage is 5 Volts.
- **Measurement instrument setup** proposal (optional)
  Instrument name, type, measurement probe, etc.
- **Connections of instruments and system** proposal (optional)
  Connections to the DUT and connections to test and/or evaluation Printed Circuit Board (PCB).
- **Results**
  - Result data
    E.g. 3 volts measured at Pin_x

    - Measurement instrument setup
      E.g. Multimeter_x (serial_number) used
    - Connections of instruments and system
      E.g. Input A of Multimeter_x connected to Pin_x of DUT.
    - Condition (environment)
      E.g. Temperature: 25

The **measurement instrument setup** and **connections of instruments and system** are optional, because this is dependent on the implementation of a TC. These items can be added as proposals for the setup.

The **Analysis** needs to be separated from **Results**, because the interpretation may change due to different application limits, process changes, etc. The limits, needed for *Analysis*, could be either stored as a separate file (e.g. spreadsheet) or be part of one of the test specification files.

It is required that the content of each file type has to be **formally checkable**.

A scheme for TC identification needs to be considered. A TC identifier does not need to be unique through the whole system or all projects, but need to be unique inside an IP independent from verification environment. Also a kind of TC grouping should be taken into account.

The format of a **test cases list** has to be defined, with a short description, TC identifier, responsible engineer, etc.

The format and content of **Verification Plans** are to be specified. A verification plan represents the order of TCs, out of the *test cases list*, which need to be executed to verify system requirements. For each verification environment no, one or more plans can be created. A verification plan needs to include at least

- TC identifier
- Examined system requirement(s)
    - Requirement identifier (e.g. Name of parameter)
    - Description of requirement(s) (e.g. parameter to test)
- IP name the TC is associated with
- Responsible engineer
- Optional
    - Constraints
    - Status
    - Priority
    - Sequence number

– Notes/comments

Dependencies of TCs need to be considered. Especially for optimising verification plans (reducing execution time, increasing efficiency) this is essential.

All information and data of a TC has to stay human readable and has to be available without tool assistance. This forbids the use of a data base.

Aside functional requirements the internal SD quality procedures QM25 for device evaluation and golden sample preparation have to be taken into account[3]. The procedure regulates the IC design validation and data correlation of verification environments. The QM25 procedure is based on the Automotive Electronics Council (AEC) standards AEC-Q003 and AEC-Q100-009, the *Guidelines for Characterizing the Electrical Performance of Integrated Circuit Products*[12] and the *Electrical Distribution Assessment*[13]. Especially for reproducibility and correlation of verification environments the QM25 procedure has to be considered.

2.2.1.1 Data Model

A *conceptual data scheme*, which includes content and relations of data, is needed. This scheme visualises the association of files/file content in a TC as well as the relationship of TCs. Furthermore the scheme reflects the target of verification plans (TCs collected and sequenced).

2.2.1.2 Revision Control

A Version Control System (VCS) has to be used, which influences the data structures. With a VCS the data model gets one more dimension. For example a verification plan may refer to a specific version of a TC or a stored result makes no sense with the current version of a TC. Therefore it is important to consider revisions in the concept.

2.3 Software

There are several tools (software programs or scripts) needed based on the verification platform concept, which have to be of administrative nature. For example a tool should initiate a measurement, but not doing the measurement itself. Since the concept needs to stay project independent also these supporting tool need to stay generic. Therefore it is essential to specify well-defined interfaces to guarantee the distinction of verification platform (project independent) and outside world (project or measurement instrument dependencies, etc.). From economic point of view, any kind of purchase costs or license fee is restricted.

Based on the data model the *control flow* through a verification process, especially for tool support, has to be defined. This includes the relationship of the involved entities (tools) to data (e.g. Tool_x processes TC description). Furthermore the *control flow* specification has to distinguish executing and supporting tools, in terms of which is the master and which the slave.

2.3.1 Simulation

Since simulation within IC design is usually done script based, the following simulation tools need to be supported.

- Cadence® Ocean script
- Tool Command Language (TCL)
- C-shell (Csh)

As Figure 2.1 shows, these tools need to be executed by the verification platform software.

2.3.2 Evaluation

For evaluation measurements a well-defined software interface needs to be designed. Evaluation hardware and software will need to be able to read TC data. Typical evaluation systems are based on National Instruments (NI) Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW)™[24], Python or C software modules. Since one of the main targets is automation of the system evaluation, the following items need to be considered and defined

- Data flow
- Control flow (as described above)
- NI PCI eXtensions for Instrumentation (PXI) support

Especially for golden sample preparation it is required that the execution of TCs can be done automatically (reproducibility).

2.3.3 Production Test

The interface to the ATE is a SD custom defined excel spreadsheet, therefore a *TC data to ATE test specification translator* tool is needed. This tool converts TC data to specific spreadsheet formats (in combination with the *verification plan*). This *TC data to ATE test specification translator* has to be designed format independent, which means that if the ATE changes it should be easy to adapt the tool.

Figure 2.1 shows how the platform should be separated from exemplary external software.

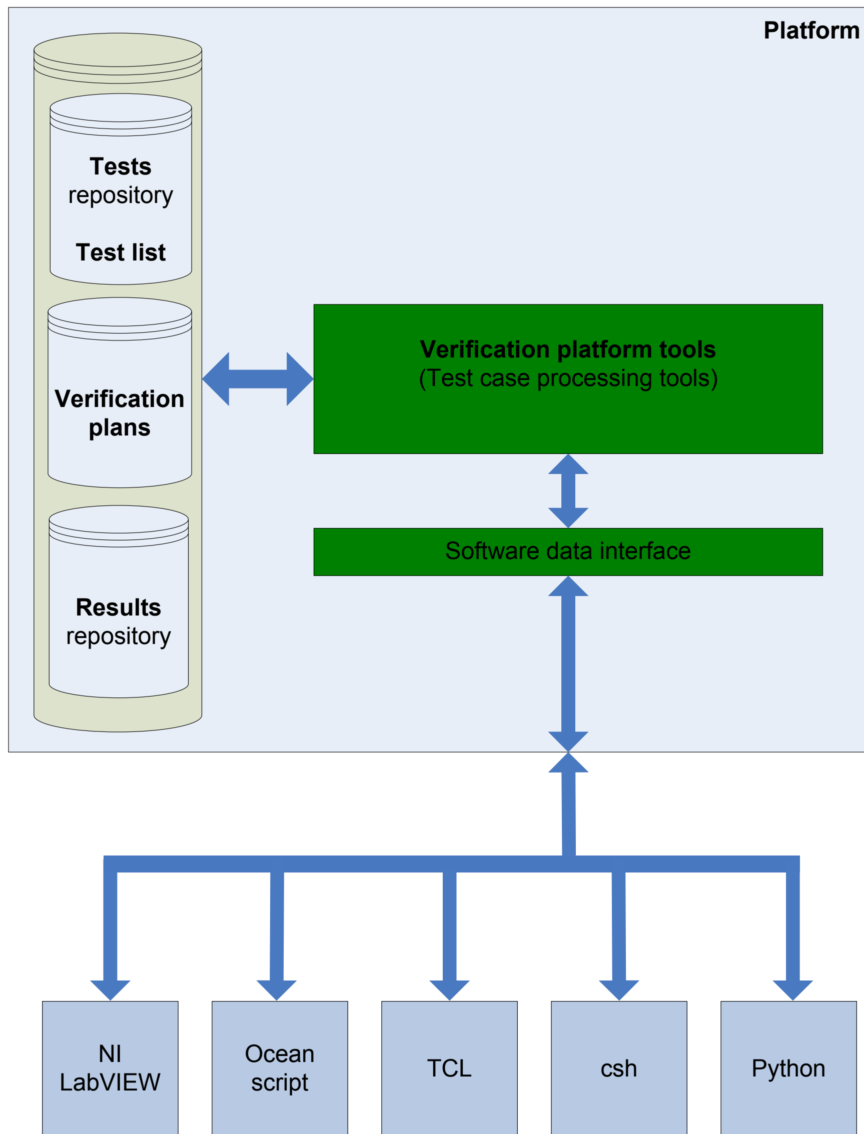


Figure 2.1: Logical tool organisation

Chapter 3

Status quo and Analysis

This chapter analyses which data structures and tools are currently in use at SD. The first step here was the examination of data and relations. This also include the observation of the work flow from the early stages of a project to production test and how these processes influence the verification of a project.

Based on requirements found in Section 2 the existing verification methodologies were examined, including the characteristics and differences of the verification environments.

3.1 Verification Environments

The procedures and methodologies of verification environments are described. Finding interfaces and relations between environments is the second focus in this section.

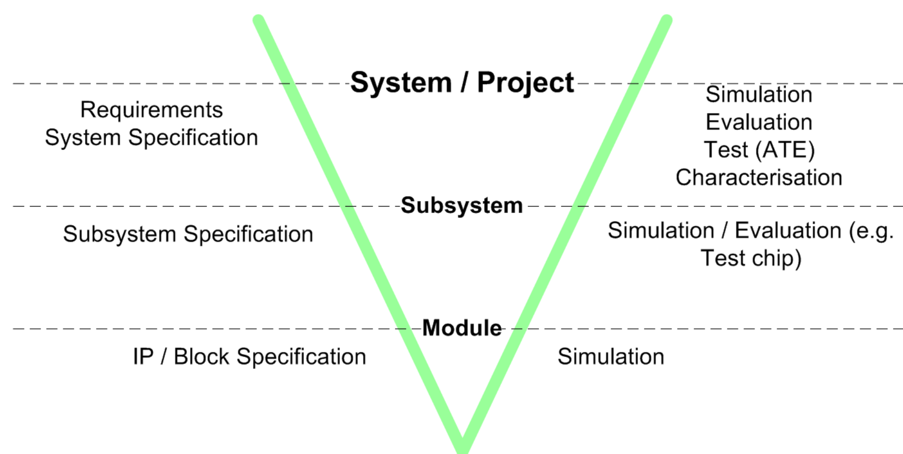


Figure 3.1: SensorDynamics verification V-Model

As Figure 3.1 shows, design and verification is performed from IP module level up to system level. Simulation (design verification) is done in every stage of the project and evaluation and production test is realised for the assembled product.

3.1.1 Digital Simulation

Digital design verification is basically done by simulation, using Very-high-speed integrated circuit Hardware Description Language (VHDL) test benches at Register Transfer Level (RTL) and gate level (synthesized VHDL code). The next step is to transfer the design, or parts of it, to a Field Programmable Gate Array (FPGA) to also verify the design physically. The test bench data from simulation is not used in physical verification(except top level simulation), because these test benches are module/IP based, but physical verification works on higher compound level. Digital verification data is not used by other environments as is, except firmware. Configuration data (setup) of digital modules/IPs is transfered via scripts or manually for evaluation and production test usage.

3.1.2 Analog Simulation

The basis for analog simulations are module/IP schematics and layouts (parasitic extraction). There are different kinds of simulations done, the most important are

- typical parameter simulation
- Corner simulation
  The designed analog circuit is simulated with a model of the production process, which parameters are set to worst or best case. E.g. slowest behavior of transistors
- Monte Carlo simulation
  A statistical method through modifying model parameters randomly. [7]

To be able to verify analog design parameters also on silicon (assembled product), the designer need to consider a possibility to route these signals on a test bus or equal, which is basically done via an external configuration interface JTAG Like Control Chain (JLCC).

3.1.3 Evaluation

Evaluation is the measurement of the products electrical characteristics, which is done with laboratory equipment and mostly done semi-automatic at the end of measurement development. The DUT is configured with data out of simulation and results are compared against simulation. Since simulation models are not completely accurate, the configuration may not stay exactly the same. For evaluation all parameters are verified in very detail, especially critical ones, to get a clear knowledge of the DUTs behaviour.

3.1.4 Production Test / Characterization

Production test is done with the help of ATE, which is very flexible and with high performance. Test time is a very expensive factor within the production of an Application Specific Integrated Circuit (ASIC) or SoC, which implies that this needs to be as optimised as possible. This is also the major difference to evaluation, where test time is not an important factor. At production test measurements are done similar to evaluation, with the configuration data verified by evaluation, but parameters are just checked if they are within their specified limits

and statistical data is analysed. As example the Process Capability Index (Cpk) is calculated, which is a statistical benchmark of a production process and the design.

Characterisation of a DUT is done via results of production test and is only possible if enough parts are tested to get statistical DUT parameter data.

3.1.5 Conclusion

Up to now the IC verification is done for each verification environment separately. Information exchange is performed, but not a transfer of verification input data. A common data storage for verification input data or results is not implemented at all.

Test execution automation for evaluation is done by each engineer on her/his own, if required. Since an automation is only done for a specific module of the project, evaluation is never fully automated. Furthermore, up to now, no reuse of automation software could be observed, neither between project nor engineers.

Since ATE test data (spread sheet) is not checked by software or an other verification environment, the production test data is error-prone and needs debugging. The format of the spread sheet should only be changed slightly, if required.

The script based simulation works very well and should not be touched at all.

To establish comparability of DUT parameters, a common view of the DUT and common data source is needed. Since this is not implemented at the moment, comparability can not be guaranteed. Verification data, input as well as output, is not collected at a single point, but roughly collected for each verification environment inside the IPs directory structure. Furthermore evaluation automation is done for each project differently, which leads to different output formats and report styles. The test development needs to be started before evaluation is finished, because automation of measurements is implemented again for each project, which is very inefficient.

3.2 Data

Figure 3.2 shows the complete verification data model, with all data needed within the verification platform. The figure is not an image of the current status, but the required data model. This data model needs to be considered for the specification of the generic verification platform.

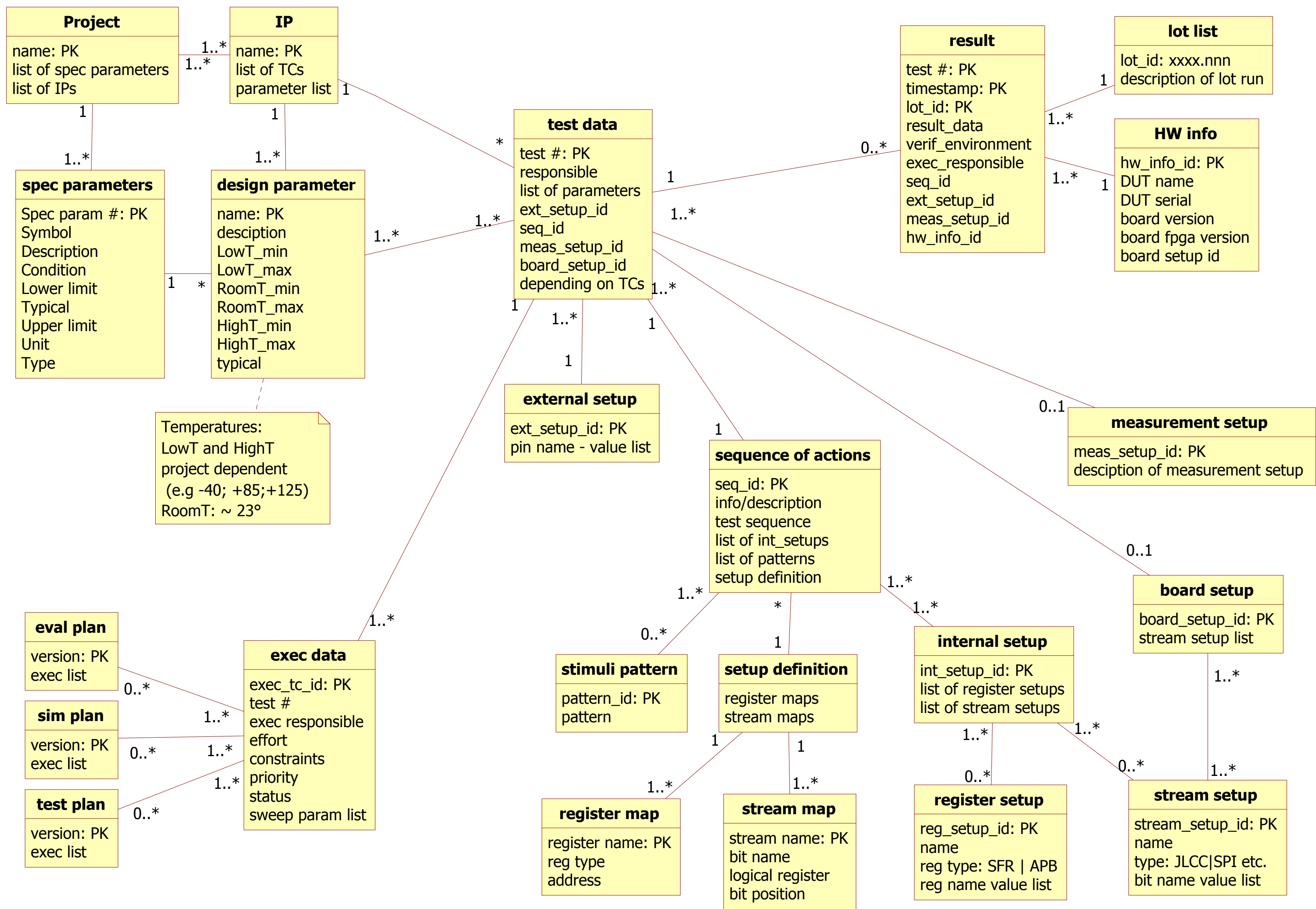


Figure 3.2: Data model

DUT requirements/specifications For verification the most important information are the project requirements and parameter specifications, which defines what has to be tested. The data model shows these data with following entities:

- Project
- IP
- spec parameter
- design parameter

Test data is any information required to run a single test. This includes external as well as internal conditions of the DUT and is modeled in Figure 3.2 by the subsequent entities:

- sequence of actions
- external setup
- internal setup
- setup definition
- register/stream map
- register/stream setup
- stimuli pattern
- board setup
- measurement setup

Test execution data Each verification environment needs to execute different tests and therefore execution information, where overlaps are comparable. The data model shows these information with

- exec data
- eval plan
- sim plan
- test plan

Result data At last results need to be stored with additional information to test results, which is essential to ensure reproduceability.

- result
- lot list
- hw info

3.3 Work Flow

The project work flow refering to verification can be separated into three major steps, the *definition* of requirements and specifications, the *design* phase and the *evaluation and characterisation* of the project. The work flow diagram (see Figure 3.3), in terms of verification, starts at the definition of requirement and specification parameters. The outcome are specification limits, which are used for acceptance tests and information for implementation of design. After and during design implementation simulations are done, where the results are crosschecked against specification limits. Design parameters and corresponding limits are defined, which are more tight than specification limits. These design parameters are evaluated and characterised (test), but for production only a subset of these parameters are tested to guarantee functionality of the product.

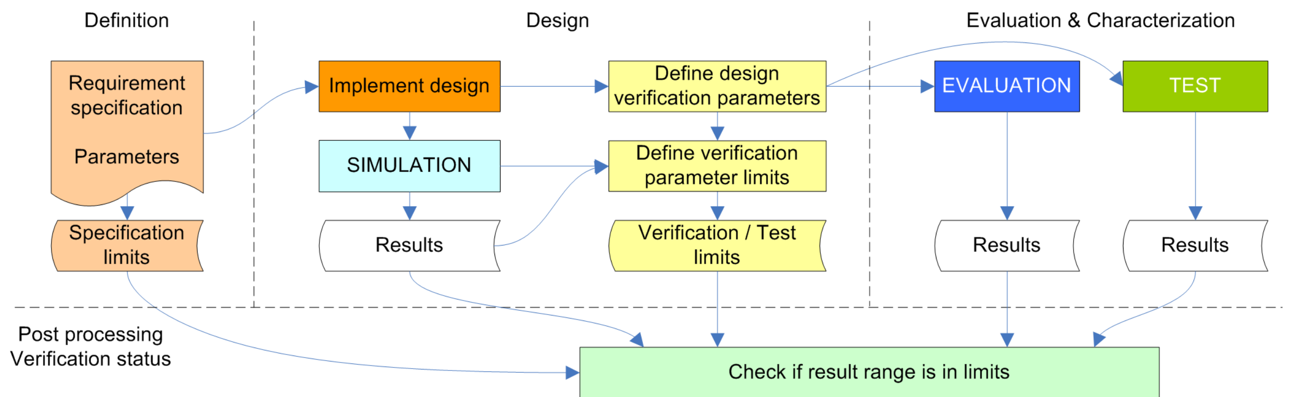


Figure 3.3: Design verification work flow

Chapter 4

Technology Overview

A rough market analysis was done to identify what is available related to IC verification platforms, test activity description languages, test case management and measurement automation.

4.1 Verification Platform

This first section of this overview treats *verification platforms* provided by well known suppliers such as Mentor Graphics, MathWorks or Cadence.

Calibre - Physical Verification Platform "*Mentor's IC verification and sign-off includes not only traditional rule-based physical verification and parasitic extraction, but also new capabilities and automated technologies that help improve yield by enhancing the design itself.*"[32]

Mentors verification platform is meant to be used during design and especially before tape-out (final design process stage - design is sent to manufacture). It offers tools for the verification of the design (circuit verification) and of the IC layout (physical verification). Mentors platform does not cover design evaluation nor production test, therefore this can not be used as a complete solution.

Synopsis - Discovery Verification Platform "*The Discovery™ Verification Platform is an integrated portfolio of functional, AMS, formal and low-power verification tools. Discovery provides high performance, high accuracy and efficient interactions among best-in-class technologies, including mixed-HDL simulation, mixed-signal simulation, assertions, coverage, testbench automation, verification IP, formal analysis, unified debug, equivalence checking and rapid prototyping. Discoverys components support industry standards including SystemVerilog, SystemC, VHDL, UPF, OpenVera, Verilog-A, Verilog-AMS, SPICE, and more.*"[39]

Similar to Mentors platform Synopsis also provides a design verification platform, which is used before tape-out. It does not cover evaluation or production test.

MathWorks - SystemTest "*SystemTest™ software lets you develop and execute tests that exercise MATLAB® algorithms and Simulink® models. It includes predefined test elements*

that let you build and maintain standard test routines. You can map test variables into a result set for analysis." [30]

This framework is used for digital design verification, but as the verification platforms from Mentor and Synopsis it does not handle verification of the 'real' silicon (evaluation, production test).

Cadence - Specification Driven Environment "*The Cadence® Virtuoso® Specification-Driven Environment is the advanced design and simulation environment for the Virtuoso custom design platform. By supporting extensive exploration of multiple designs against their objective specifications, Virtuoso Specification-Driven Environment sets the standard for fast, accurate design verification."* [8]

The Cadence Virtuoso Specification driven Environment (VSdE) provides design and simulation tools as well as the needed environment (data structures, etc.). Equal to above mentioned verification platforms it just handles design verification.

Conclusion The described verification platforms are not the kind of platform SD is searching for. An overall solution is needed which covers simulation (design verification), evaluation and production test. It seems, that the term 'verification' is just used in the sense of 'design verification', which is more or less simulation, by these EDA vendors.

4.2 Test Case/Data Management

An important part of a verification platform is the management of verification related data. The most promising providers for this kind of solution are OptimalTest, GridTool and VI tech.

OptimalTest - Test Management Solution "*OptimalTests new generation of Test Management Solution (TMS) gives you unprecedented improvements in Yield, Test Time Reduction, Reliability, Quality and Productivity based on our:*

- *Innovative advanced adaptive testing methodology*
- *Patented reference die technology*
- *Data Feed Forward/Data Feed Backward capability*
- *Centralized, engineering-oriented database designed for rapid data retrieval*
- *Unparalleled data integrity and actionable data throughout the IC lifecycle*
- *Unique open test-rule generator*
- *Open IT infrastructure and seamless connectivity*
- *Real-time automation and control*

High performance test management at low cost of test allows you to adapt and correct your test processes in real time and continuously improve and evolve products, processes, and operations to compete more effectively."[43]

OptimalTest's TMS is a powerful and promising tool for production test data management, but this is also the drawback of the solution; it is only meant to be used for production test.

GridTool - Datamaker "*The Datamaker™ suite offers a complete end-to-end solution for creating secure test and development data across multiple projects and test environments. The Datamaker™ test data management solution is easy to use, easy to learn, flexible and fast.*

The Datamaker™ tool set allows you to quickly create and store new, high-quality data. The new data can be "tokenized" whereby, at create time, the data will be built with the specific data you need to test your requirements.

GT Datamaker™ allows you to build test sets of data which can be published directly into development or testing environments. The version of each test set can be easily upgraded, allowing test cases which are built for one version to easily be used in later versions of an application."[21]

Datamaker™ would be a very powerful and well fitting solution, but is too overloaded for SD's needs and does not fit into the budget of the generic verification platform.

VI Tech - Enterprise Test Solution "*Enterprise Test is a term that encompasses the requirements, complexities, and best practices associated with todays product testing and characterization. These include multiple and often disparate technologies; the large quantities of data that are typically produced, analyzed, and shared; and the geographic and work distribution of team members - from department managers and design engineers to test operators - across departments and the globe.*"[42]

Equal to OptimalTest's TMS this solution is only useable for production testing.

4.3 Test Description Languages

To get a common understanding between all involved verification environments, a formal description of test data and activities needs to be introduced. There is a variety of well defined and promising solutions available.

IEEE - Automatic Test Markup Language "*Automatic Test Markup Language (ATML) was born out of the thought that it is "the information" and a way to "communicate that information" may be the way to improve upon the inefficiencies of testing in both time and budget. The intent of ATML is to establish an open standard for this "information" as the basis for test investments such that the description would be sufficient to allow transportability of test programs between testers and to provide a means to communicate test outcomes up and down the maintenance chain. The new XML-based standards for ATE and test information*

data exchange are the ATML family of standards. ATML is emerging with widespread support among test and measurement industry leaders as well as major government programs."[36][1]

ATML is an industry standard for structuring and interchanging test data through different kind of information systems, including results and analytical data (e.g. limit information). Since ATML is based on the XML standard, test data is very well structured.

The drawback of introducing this IEEE standard is the demand for design engineers to describe their defined verification routines (tests) in much detail, which will lead to rejection of ATML. Parts of the standard should be considered for this definition of the generic verification platform and in a next generation the complete IEEE ATML standard may be realised.

Test Case Description Language 2.0 "*Test Case Description Language (TCDL) is an XML vocabulary for describing test files and evaluation scenarios for such test files, intended to be included in test suites for user interface guidelines such as Web Content Accessibility Guidelines (WCAG) 2.0.*"[38]

Similar to ATML this description language needs description of tests in large detail, but ATML is already an industry (IEEE) standard, it would be prefered.

4.4 Measurement Automation

SD is basically using two technologies of software for laboratory measurements: National Instruments LabVIEW™ and the Python programming language. Since SD does not want to change these technologies, this section concentrates on improvement of the current systems or complete systems using one of them.

National Instruments LabVIEW™ "*National Instruments LabVIEW™ is a graphical programming language that has its roots in automation control and data acquisition. Its graphical representation, similar to a process flow diagram, was created to provide an intuitive programming environment for scientists and engineers.*"[17]

National Instruments LabVIEW™ is a very powerful programming language especially in combination with laboratory equipment. National Instruments provides a measurement automation environment called NI TestStand.

"*NI TestStand is ready-to-run test management software designed to help you develop automated test and validation systems faster. You can use NI TestStand to develop, execute, and deploy test system software. In addition, you can develop test sequences that integrate code modules written in any test programming language. Sequences also specify execution flow, reporting, database logging, and connectivity to other enterprise systems. Finally, you can deploy test systems to production with easy-to-use operator interfaces.*"[25]

Unfortunately NI TestStand licenses do not fit to the verification platform budget and can therefore not be considered.

Python The Python programming language is also a standard technology at SD. Since the access of laboratory equipment is much more complicated compared to LabVIEW™ it is rarely used, but some problems may be solved quicker by using a traditional programming language instead of using a graphical language (such as LabVIEW™). Laboratory measurement automation may be done using python, but measurements may stay in LabVIEW™. Feasibility studies about using Python for measurement automation came to the conclusion that it is useful, but an in-depth knowledge of the programming language is required (multi-tasking, etc.).[9]

Chapter 5

Specification

This chapter specifies the generic verification platform in detail. Based on the information found by requirements engineering and analysis of current situation, test data and directory structure, embedding in IP, test identification, file formats etc. are defined subsequently.

5.1 Platform Concept

5.1.1 Overview

All verification environments (simulation, evaluation, production test) in the verification process of integrated circuits need to have a common source of input data and a common sink for results. Within the generic verification platform this is realised by three major entities: test data, results and the DV-Matrix (Design Verification Matrix). As Figure 5.1 shows, simulation and evaluation get their input data via the *Verification Platform Framework*, which may change the format of data (but not the content) and handles the automation of verification (see Section 5.4.1).

5.1.2 Description of Components

Test data is divided into three parts, *Tests*, *Plans* and *Test spec*. *Tests* is the set of test cases used for verification. Each test has a *sequence* file wherein the activities for the test are described (see Section 5.3). Also pattern files, firmware files, scripts etc. are kept here. The *Test spec* excel spread sheet is a SD's custom defined format, which can be transfered to ATE format. Within this spreadsheet all DUT setup data is kept. This has the considerable advantage, that no format conversion is needed between different environments of verification. This guarantees that the DUT setup is equal for each environment, which is essential for evaluation and production test correlation. If results differ, one likely source of errors can be excluded. Moreover the efficiency of production test ramp-up can be improved dramatically. *Plans* define the order of test execution and all data belonging to execution (see 5.2.3).

Results the output of all simulations and measurements are collected in here. *Test* writes its results to the *Yield data base*, which is a special ATE storage for production test results, reports and any other related data.

Design Verification Matrix is an excel spread sheet for defining tests formally, defining test names, storing limits and storing and correlating results. Since this spread sheet has to be maintained manually, the only software which reads this spread sheet and distributes its data is the verification platform framework(see 5.2.2.1).

Verification Platform Framework is the common interface for verification environments, except production test environment which just uses Testspec. The tasks of the framework are acquiring data, data format conversion and verification automation (see 5.4).

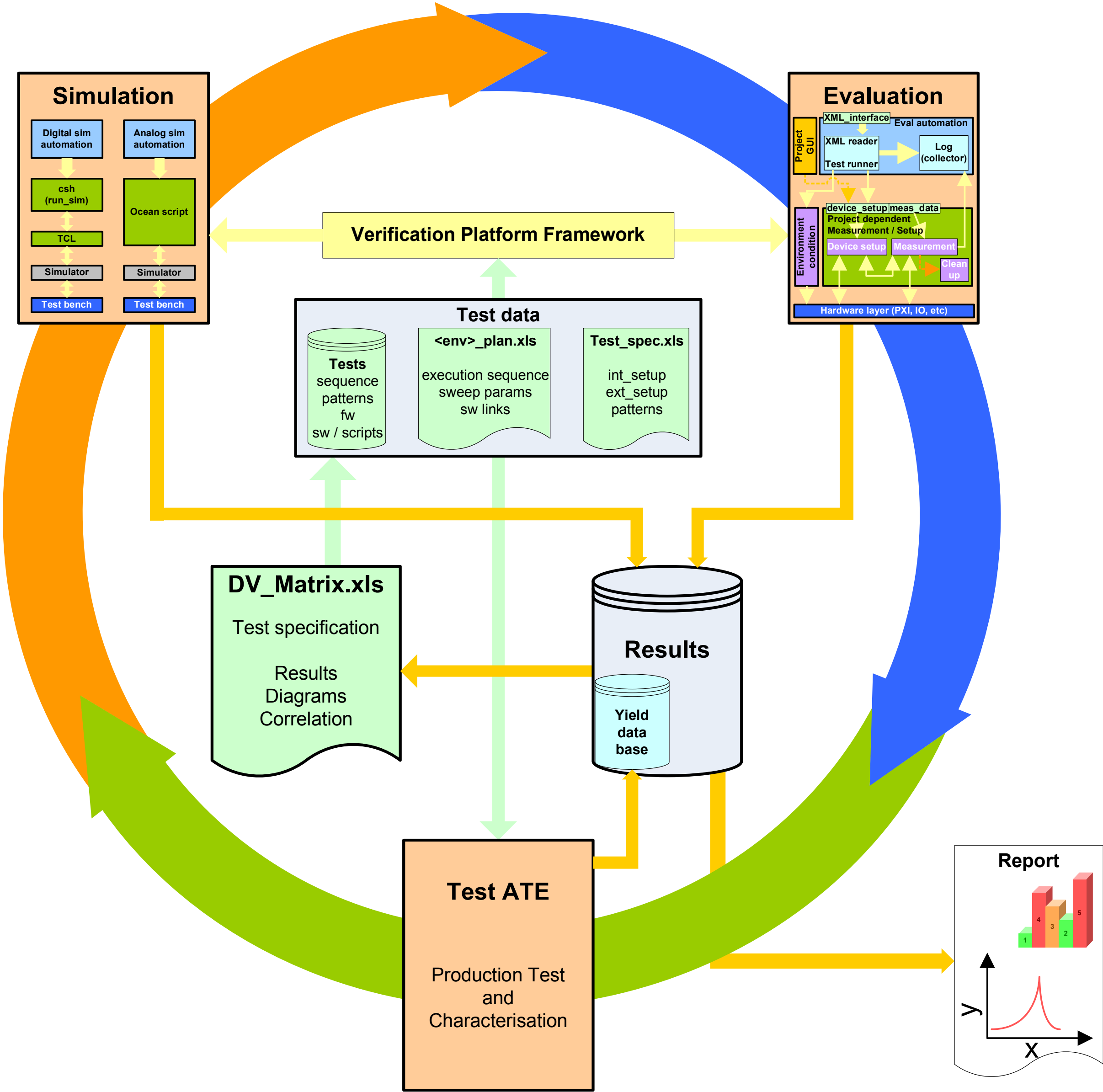


Figure 5.1: Platform concept overview

5.1.3 Capabilities

The main task of the generic verification platform is to provide a common layer for any kind of verification environment. The components are used to share input data and correlate result data to guarantee a maximum of efficiency and transparency in all stages of integrated circuit design verification. From block level circuit simulation up to high volume production test, all parties involved in the verification process base their verification on the same data input and output. This leads to a maximum of reproducibility and comparability in the verification progress. Standardised data formats are fundamental for automation and tools support, which is provided by the verification platform framework.

5.1.4 Boundaries

The generic verification platform provides the environment, interfaces and tools for integrated circuit design verification, but is not implementing tests itself. It defines structures, formats and provides templates for the definition of tests and specifies the data flow and control flow for the execution of test implementations. The verification platform does not interpret the test definitions or test activities, which is has to be done by the responsible engineer.

5.2 Directory Structure and File Formats

This section defines data structures and formats of the generic verification platform in detail.

5.2.1 Embedding into IP Directory Structure

SD projects are structured with IP modules hierarchically (see Section 1). Since verification is done from lowest hierarchy level (block level) across compound levels up to the project top level (all levels organised via IP modules), the embedding of the verification structure has to be considered at every stage of the project. Figure 5.2 shows the relation of project hierarchy to verification type.

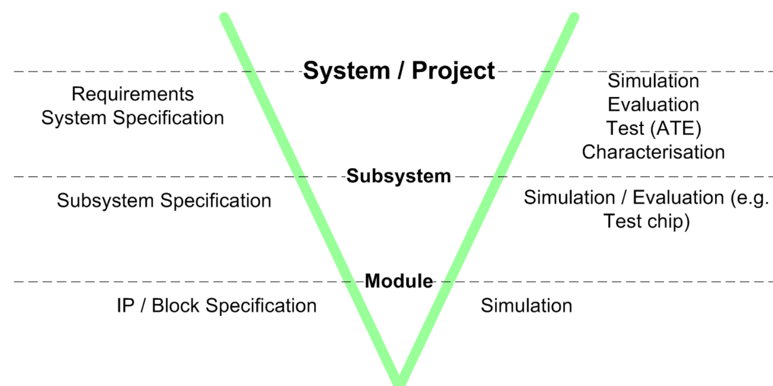


Figure 5.2: SensorDynamics verification V-diagram

Each IP at any hierarchy level stores its verification data in a directory called *verif* at the top level of the IP directory. This directory builds the root directory for any kind of verification data

related to this IP. The next directory level down separates the major parts of verification, which are definitions (specs, plans, sweeps), verification data (tests) and results and post processing data (results, reports). This level in the directory structure equals the structure of Cadence VSdE[40], which may be used for analog simulations in future. Figure 5.3 shows the specified directory structure with its content. Subdirectories and content are described subsequently. Furthermore the *verif* directory contains the *sw* directory, where any software related to an IP has to be stored, and a *hw* directory where PCB schematics, layouts or equal are stored. Evaluation software needs to be separated into IP dependent and independent software with high reuse.

specs Any kind of verification specification has to be stored here.

plans In here execution plans are stored, which describe the order of TCs and related information.

sweeps This directory could be used as sweep specification storage. Sweep parameters are defined in verification plans, but for complex analog, digital or mixed signal simulations more information could be needed (e.g. Cadence VSdE[40]).

tests This directory collects the test data itself.

results All results of a test case execution are stored here. Each verification environment has its own result directory inside.

reports The reports directory collects evaluated results of test cases, plots, diagrams etc.

sw All software related to verification need to be stored in the *sw* directory, where this is subdivided into verification environment directories (e.g. eval).

hw PCB layouts and schematics developed for verification and all other hardware information referring to verification have to be placed here.

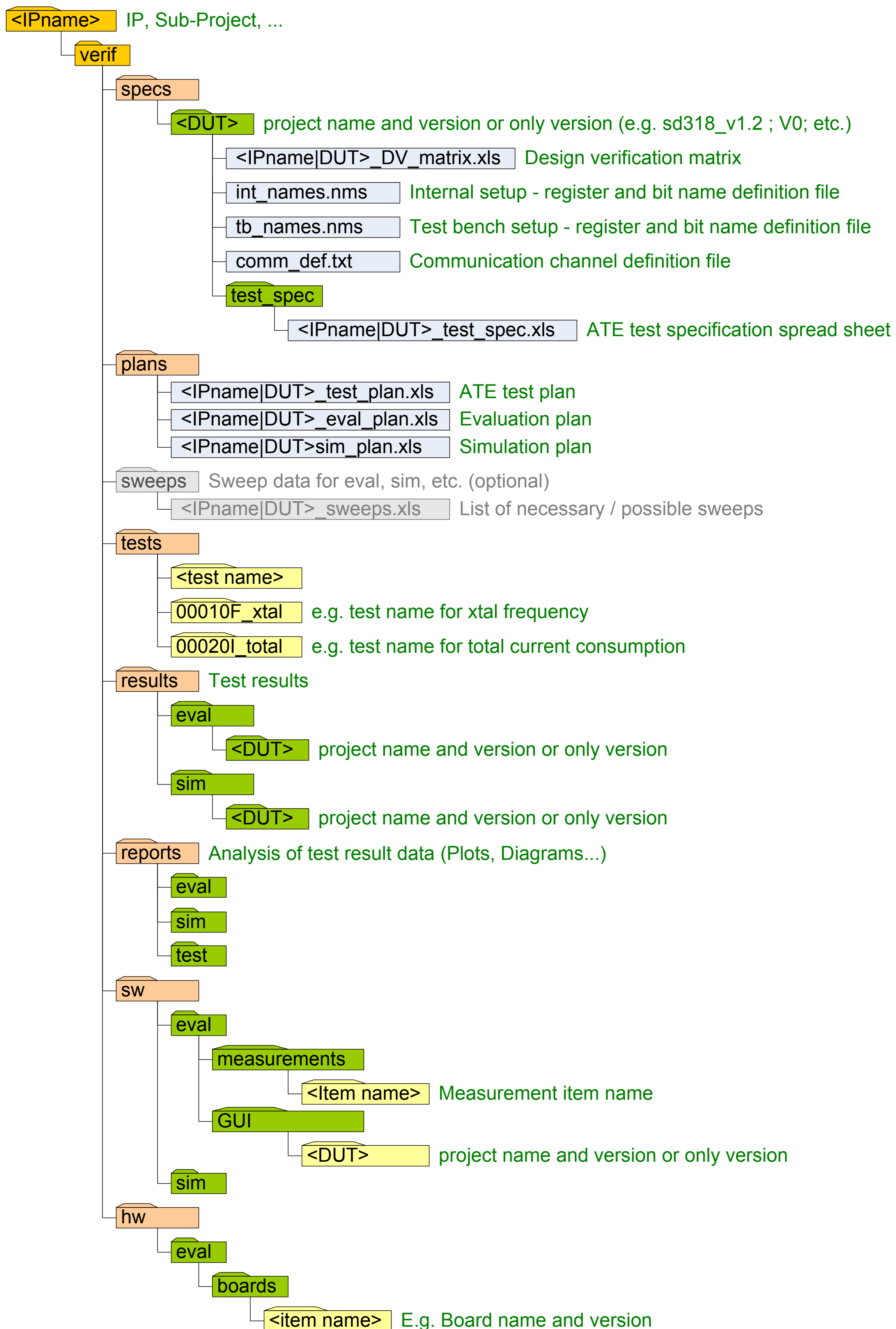


Figure 5.3: Embedding of verification platform directory in IP directory structure

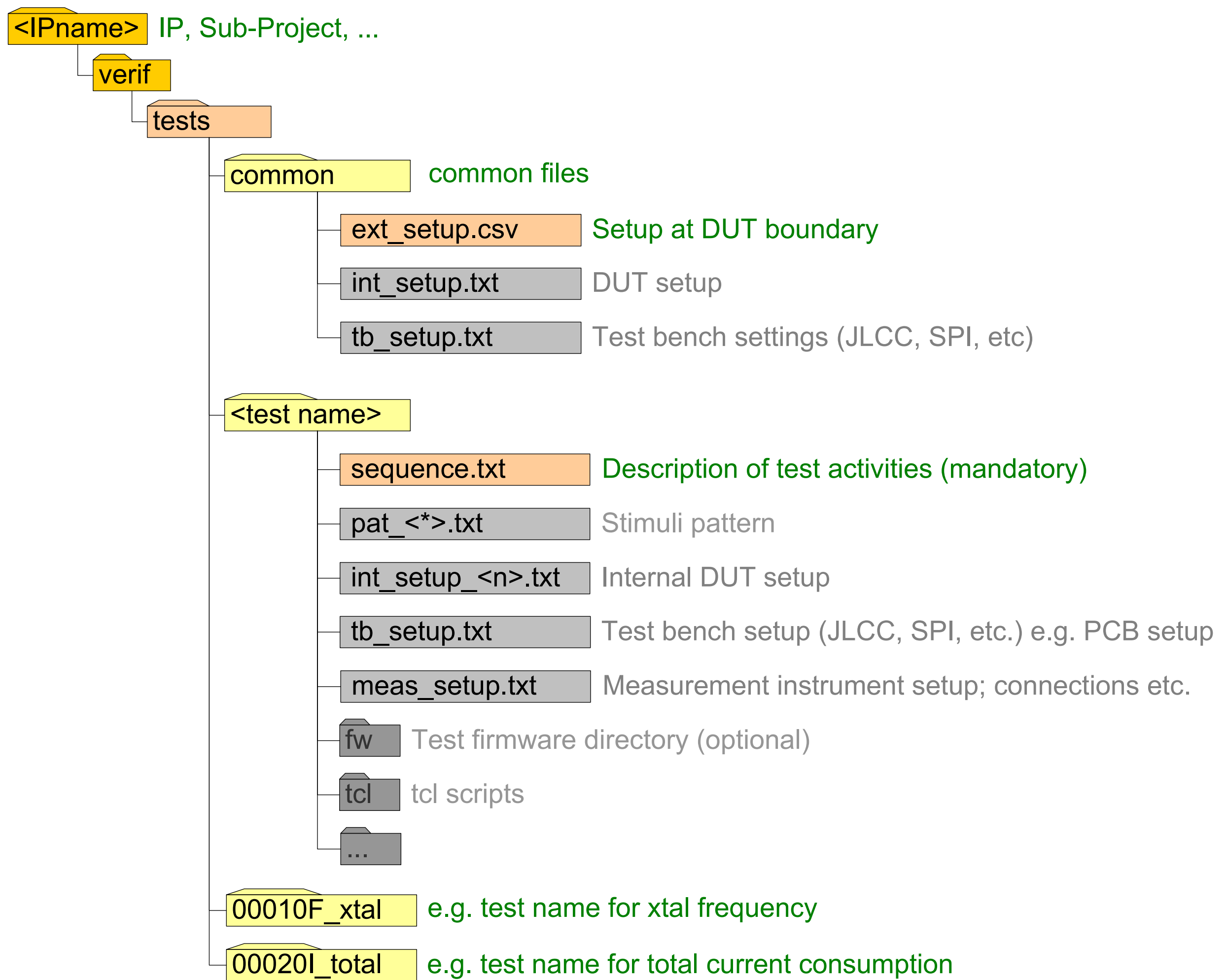
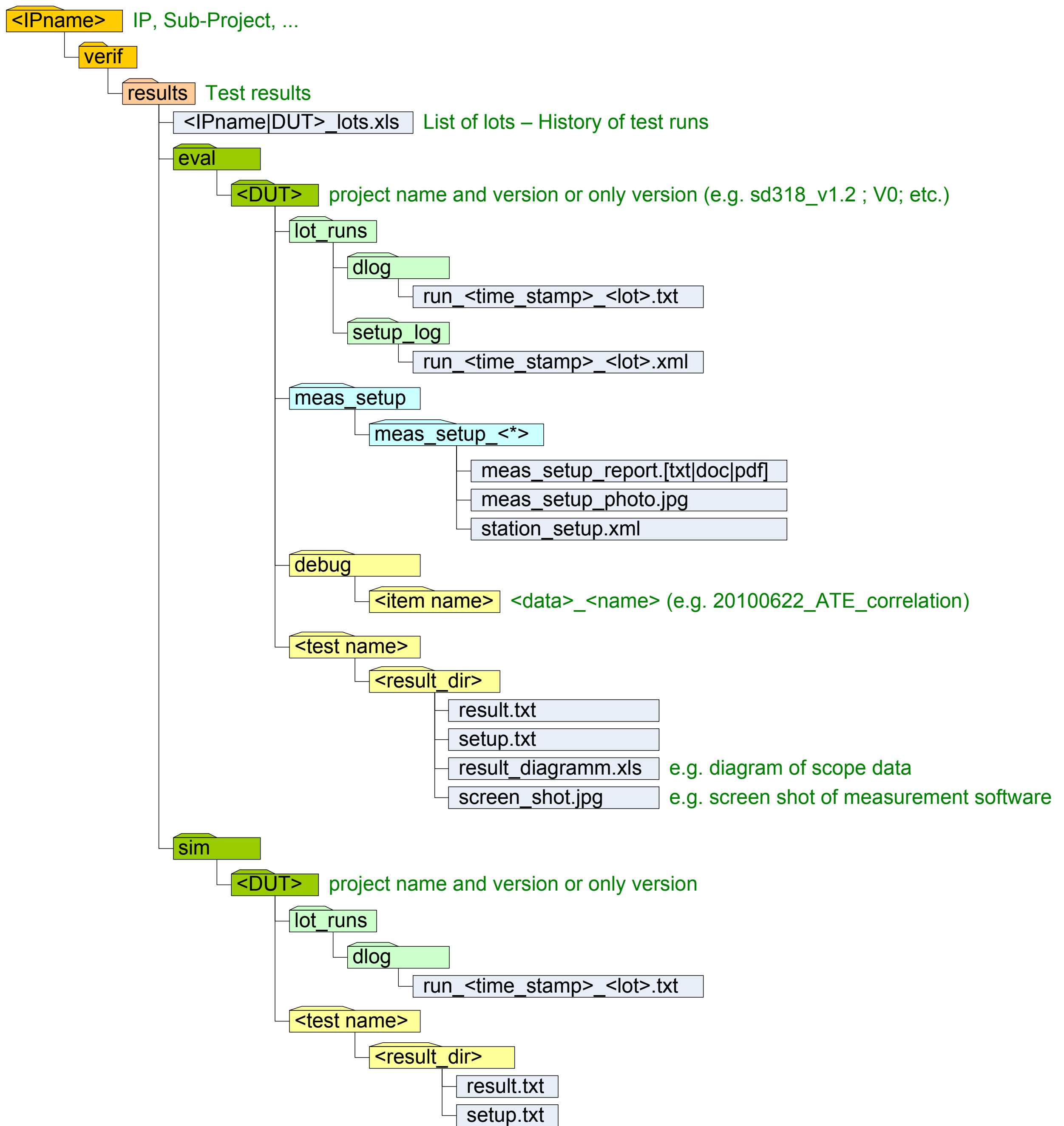


Figure 5.4: Tests directory structure



<result_dir>: [<token>_]<YYYYMMDD>[HHMMSS]

<token>:
- typ
- parm_<parm_name(s)>
- corn
- mc
- rcx

examples for result directory names:
- 20100220120100
- 20100225
- typ_20100221
- parm_vdd_20100115172355
- corn_20090430

Figure 5.5: Result directory structure

5.2.2 Specs

Any kind of verification related specification has to be stored in the *specs* directory. Although the Design Verification (DV) Matrix is hybrid (mixture of definition and results) it should be stored here, because specification parameter definition and verification parameter definition should only be done in this document.

5.2.2.1 Design Verification Matrix

Since the DV Matrix is one of the essential components of the generic verification platform it is described in this section in very high detail.

The DV Matrix is an excel spread sheet which combines the following data and functionality:

- project specification parameters
- specification parameter limits
- verification parameter definition
- verification parameter limits (temperature related)
- test name (relational key between verification environments)
- simulation results (incl. corner and Monte Carlo simulation[7] if available)
- evaluation and test (ATE) results
- automatically calculated verification status
- chart tools

The DV Matrix also provides the possibility of verification environment results correlation in post processing. Evaluation and Test (ATE) measurements are done for five golden samples for at least three temperatures, low, room and high temperature (project dependent)[3]. Each measurement has to be repeated three times with power cycling at the beginning. This procedure should demonstrate the *reproducibility* of measurements and fulfills the requirements of SDs internal quality procedure QM25 (AEC based)[3]. Post processing actions, like calculation of Cpk [26] may be done in a separate work sheet.

The DV Matrix has to be created for each silicon version of a project (test chip 0, engineering sample 1, etc.) as well as for all sub IP versions, because requirements may change in the time line of a project (e.g. moving target, verification results of former versions show unsolvable problems, etc.) or an IP is reused in a different project. Furthermore this approach guarantees a high level of verification coverage through all project stages.

The subsequent screen shots show the DV Matrix template with some dummy data (screen shots from left to right in the spread sheet, except header).

The header includes all basic information relating to the IP or the project. As Figure 5.6 shows the header includes a project/IP information summary as well as a summary of the file content.

<Project Name - Dash> - DESIGN VERIFICATION MATRIX

Project:	<Project Name - Dash>	Number parameters:	8
Last update:	01.01.2000	Number of tests:	13
PM:	<Name>	By:	<user>
Template:	DV_Matrix_template_v0.2.xls	Today:	22.07.2010
Spec Rev:	<Name>	Calender week:	30
Notes:			
Testcase Root:	..\tests		

Figure 5.6: Design Verification Matrix - Header

The initial part of the DV matrix is the *Requirement parameter specification*. Out of the project specification/requirements document (negotiated with the customer) all project parameters need to be added in this first part, this needs to be done for IPs and their specification. The column *Type* lists the type of the parameter e.g. Supply or General Purpose Input Output (GPIO). *CC/SC* stands for Critical Characteristic (CC) and Significant Characteristic (SC) (the definition of critical and significant can be found in the SD project management handbook[4] and is not discussed here). Furthermore internal specification parameters need to be defined here (e.g. design parameters). Numbering of *spec param #* is defined later on.

In the *Design IP* column the IP is defined corresponding to the requirement parameter with an IP responsible. Functional requirements will be tested on top level IP, but other requirement parameters may be fulfilled within sub IPs.

Test case # and *Test case sub #* are numbers used for unique identification of tests. The four digits test case number may match with the *spec param #*, where sub tests have the equal *Test case #* and a running *Test case sub #*. *Test case sub#* is needed for test cases which can not be done within one single test (e.g. differential voltage). The test case sub # '0' identifies the master test of these test cases. For example two voltages 'A' and 'B' need to be measured, the master test calculates the difference $A - B$. Of course in simulation this could be done in a single step, so just the master test (test case sub # = '0') is executed and results compared with evaluation and production test.

The first character of the *test item name* indicates the verified quantity (capital letter obligatory). Table 5.1 shows valid quantities. The second character of the name optionally qualifies the quantity in detail (e.g. 'D' for differential). The test item name should be selected to transport at first glance the intention of the test/parameter. The name is furthermore restricted to a length of ten characters (including prefix) with allowed characters [a..zA..Z0..9_]. *Test item description* is just a very brief description of what this test does.

Test name needs to be an unique name of a test uniquely in the respective IP structure. Furthermore grouping of tests (analog tests, digital tests, etc.) is required, which is done through the *test case #*. The test name combines the test case #, the test case sub # and the test item name, where the restrictions of the single items guarantee an unique test name with a maximum length of fifteen characters.

```
<spec param#><verif param#><quantity><[qualifier]><param name>
```

Requirement parameter specification									
spec param #	Symbol	Parameter description	Specified Condition / Notes	min	typ	max	Unit	Type	CC/SC
0000	F_XTAL	x_tal clock	room temperature	12.00	16.00	20.00	MHz		
0001	I_CON_TOTAL	total current consuption	CPU active, RX/TX active	15.00	30.00	35.00	mA		CC
0002	I_CON_SLEEP	current consuption in sleep mode	CPU inactive; interrupt unit active; x_tal on	1.10	2.40	3.50	mA		CC
0003	I_CON_DSLEEP	current consuption in deep sleep mode	only Interrupt unit active; clock inaktive; CPU inactive	45.00	50.00	53.00	uA		
0025	V_DIFF_AB	voltage difference between A and B	room temperature	3.00	5.00	10.00	mV		
0026	RVDD	RF supply voltage		3.00	3.30	3.55	V	Sup	SC
0027	SVDD	Synthesizer supply voltage	Load current at SVDD pin to external load (not allowed)	1.70	1.80	2.00	V	Sup	
0150	CPU_TIMER_CNT	timer 1 counter							

Design IP		Test case specification				Automatic naming do not edit
IP name	Responsible	Test case #	Test case sub #	Test item name	Test item description	Test name
sd_top_ip	abc	0000	0	F_XTAL	x_tal clock	00000F_XTAL
sd_top_ip	xyz	0001	0	I_total	total current consuption	00010I_total
sd_top_ip	abc	0002	0	I_sleep	current consuption in sleep mode	00020I_sleep
sd_top_ip	mno	0003	0	I_dsleep	current consuption in deep sleep mode	00030I_dsleep
sd_ip_vout	mno	0025	0	V_DIFF_AB	voltage difference between A and B	00250V_DIFF_AB
sd_ip_vout	mno	0025	1	V_outa	voltage A	00251V_outa
sd_ip_vout	mno	0025	2	V_outb	voltage B	00252V_outb
sd_reg_r	rsa	0026	0	RVDD	RF supply voltage	00260RVDD
sd_reg_s	rsa	0027	0	SVDD	Synthesizer supply voltage	00270SVDD
sd_timer_1	sde	0150	0	D_timer	timer 1 counter	01500D_timer
sd_timer_1	sde	0150	1	D_cntdown	check timer counting down	01501D_cntdown
sd_timer_1	sde	0150	2	D_cntup	check timer counting up	01502D_cntup
sd_timer_1	sde	0150	3	D_cntirq	check timer IRQ ok	01503D_cntirq

Figure 5.7: Design Verification Matrix - specification parameters and test case definition

For example 00000V_vbg1; 02010F_xtal_out; 10099I_itot; etc.

NOTE: At IP level the DV Matrix is used only up to Simulation, because neither evaluation nor production tests are done at this level.

A simulation test bench may cover more than one test, therefore a simulation master column has been introduced. Different to *test case sub #* this simulation master test is not necessarily depending on other tests, but executes all these tests (see Figure 5.8).

On top level, sub IP requirement parameters are not simulated, because this has already been done. The column *Link to simulation result* is used to indicate the test name on sub IP level, which may not be equal to test name on top level (e.g. reused IPs) otherwise this column

Quantity	Abbreviation
Voltage	V
Current	I
Frequency	F
Power	P
Ratio	R
Gain	G
Discrete/digital value	D
Sensitivity	S

Table 5.1: Verification quantities

Simulation											
Simulation master test [test name] (e.g. test bench with equal options) (fill only if not equal to Test name)	Link to simulation result [Test name] (fill only for product level DV - if not equal Test name)	Simulation min	process typ LT	typical / mean or PASS / FAIL	process typ HT	Simulation max	MC sigma	Cpk	Corner done	Monte Carlo done	Responsible
		15.05	15.00	15.25	16.20	15.35	0.30	3.61	Y	N	gro
		21.00		31.90		32.60	0.50	2.07	Y	Y	gro
		1.50		2.44		2.90			Y	N	gro
		48.56	49.20	49.10	49.01	51.00	1.30	1.00	Y	Y	gro
				5.55			0.20	4.25	N	Y	oge
		3.15		3.30		3.39	0.75	0.11	Y	Y	oge
		1.75		1.86		1.89	0.03	1.56	Y	Y	oge
				PASS							sde
01500D_timer				PASS							sde
01500D_timer				PASS							
01500D_timer				PASS							

Figure 5.8: Design Verification Matrix - Simulation data

stays empty.

The columns *Simulation min* to *Simulation max* are filled with corresponding simulation results. *process typ LT* and *process typ HT* are the simulation results at typical production process parameters at low and high temperature. For Monte Carlo simulations[34] the mean value and the sigma are entered into the according columns. The Cpk[26] is automatically calculated for correlation with production test/characterisation.

Product level definitions are needed for evaluation and production test. *Parameter observability* describes how the parameter can be measured and how to control this parameter has to be described in *Parameter controllability* column. For example an internal signal could be measured via a test bus and controlled with a JLCC bit. Based on simulation data, measurement limits for evaluation and test are defined for low, room and high temperature (discrete temperature values are project dependent).

As a last step at product level definition the responsible test engineer needs to review these definitions for testability. If, up to here, everything is filled correctly measurement and data can be set up and implemented.

Product level definitions											
			LT limits: <low T>		RT limits: <room T>		HT limits: <high T>			Test review	
Parameter Observability Test Description	Parameter Controllability Configuration	Responsible	min	max	min	max	min	max	Unit	Param. testable on ATE	Responsible
Measure on test bus	JLCC configuration	abc	13.70	16.40	14.10	16.50	13.70	19.20	MHz	Y	tfa
Measure on supply pin	JLCC and CPU config	xyz	20.60	33.50	20.80	33.70	20.95	33.50	mA	Y	tfa
Measure on supply pin	JLCC and CPU config	abc	1.30	3.05	1.35	2.90	1.50	3.07	mA	Y	tfa
Measure on supply pin	JLCC and CPU config	mno	47.00	50.30	48.00	49.70	48.50	52.00	uA	N	tfa
calculation of A - B	No configuration.	mno	3.20	7.78	3.30	7.90	3.35	8.00	mV	N	tfa
Measure output voltage on Testbus.	JLCC configuration	mno	20.00	30.00	20.00	30.00	20.00	30.00	mV	Y	tfa
Measure output voltage on Testbus.	JLCC configuration	mno							V	Y	tfa
Measure Output Voltage on RVDD Pin	No configuration.	rsa	3.10	3.40	3.10	3.45	3.10	3.50	V	Y	tfa
Measure output voltage on SVDD pin	No configuration.	rsa	1.70	1.99	1.70	1.90	1.70	1.95	V	Y	tfa

Figure 5.9: Design Verification Matrix - Product level definitions

Figure 5.10 shows how measurement results are entered into the DV matrix. Each measurement needs to be done for five parts, three times and for low, room and high temperature. This is a requirement to fulfill SDs QM25 quality procedure[3]. These are ninety measurements for evaluation and test (not all results shown in Figure 5.10) in total.

In a separate worksheet the group definition needs to be done by the project leader. For each group a start and end of (*spec param #*), a group name and a group description has to be specified (see Figure 5.11).

The worksheet *Status* includes the current verification status of the product or IP. First all results are compared versus *Requirement parameter specification* limits and against *Product level definition* limits. A further status quantity is the appraiser variant Gauge Repeatability (GR)[37], which is a measurement of the repeatability of measurements. For Gauge Repeatability and Reproducibility (GRR) (also Gr&R)[37] the measurement needs to be executed twice with new insertion of the DUT into the socket (if used). This results in ninety measurements per environment (evaluation and test). Since this is not necessarily required by the QM25 procedure [3], it is not implemented but could be easily added to the spread sheet.

The correlation of simulation against evaluation is done for each temperature, where low temperature evaluation results are compared against *process typ LT* (typical production process at low temperature). Similar for room temperature against mean/typical and high temperature against *process typ HT*. The limits for this comparison are calculated from the range between high and low temperature, where the limit can be entered in percent (see "enter limit here").

For the comparison of simulation against production test (ATE), the Cpk[26] value is used. For simulation this is a calculated value and for production test it is a statistical value.

At last the GR and GRR values are calculated, but GRR values have no valid information if the measurement is not executed twice with new insertions of the DUT.

Status colouring:

Evaluation Responsiblity														
Low Temperature														
Part1			Part2			Part3			Part4			Part5		
Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3
14.9	15.01	15	14.95	15.02	15.05	14.89	14.95	14.99	15.02	15.05	14.98	14.9	15.02	14.98
25	25.1	26	30.5	30.6	30.1	25	25.1	26	30.5	30.6	30.1	25	25.1	26
2.5	2.6	2.7	2.1	2	2.6	2.6	2.7	2.1	2.35	2.333333	2.316667	2.3	2.283333	2.266667
5	49 4.6	49.5 5.9	49 6.066667	48.75 6.516667	50 6.966667	49.25 4.916667	49.21429 5.295238	49.17857 5.263095	49.14286 5.230952	49.10714 5.19881	49.07143 5.166667	49.03571 5.134524	49 5.102381	48.96429 5.070238
25	25.6	24.9	25.06667	25.01667	24.96667	24.91667	24.86667	24.81667	24.76667	24.71667	24.66667	24.61667	24.56667	24.51667
20	21	19	19	18.5	18	20	19.57143	19.55357	19.53571	19.51786	19.5	19.48214	19.46429	19.44643
3.31	3.33	3.22	3.196667	3.38	3.289333	3.29	3.290667	3.291333	3.292	3.292667	3.293333	3.294	3.294667	3.295333
2	1.856	1.857	1.85	1.856	1.845	1.857	1.893	1.856	1.845	1.857	1.893	1.856	1.845	1.857

Test Responsibility														
Room temperature														
Part1			Part2			Part3			Part4			Part5		
Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3	Meas 1	Meas 2	Meas 3
15.26	15.29	15.25	15.3	15.32	15.29	15.22	15.29	15.27	15.3	15.24	15.26	15.32	15.28	15.27
30.5	30.6	30.1	25	25.1	26	30.5	30.6	30.1	25	25.1	26	30.5	30.6	30.1
2.25	2.233333	2.216667	2.2	2.183333	2.166667	2.15	2.133333	2.116667	2.1	2.083333	2.066667	2.05	2.033333	2.016667
48.92857 4	48.89286 5.446429	48.85714 4.857563	48.82143 4.816492	48.78571 4.77542	48.75 4.734349	48.71429 4.693277	48.67857 4.652206	48.64286 4.611134	48.60714 4.570063	48.57143 4.528992	48.53571 4.48792	48.5 4.446849	48.46429 4.405777	48.42857 4.364706
26	25.5	24.95	24.94779	24.94559	24.94338	24.94118	24.93897	24.93676	24.93456	24.93235	24.93015	24.92794	24.92574	24.92353
22	20.05357	20.09244	20.1313	20.17017	20.20903	20.2479	20.28676	20.32563	20.3645	20.40336	20.44223	20.48109	20.51996	20.55882
3.296	3.296667	3.297333	3.298	3.298667	3.299333	3.3	3.300667	3.301333	3.302	3.302667	3.303333	3.304	3.304667	3.305333
1.856	1.845	1.857	1.893	1.856	1.845	1.857	1.893	1.856	1.845	1.857	1.893	1.856	1.845	1.857

Figure 5.10: Design Verification Matrix - Results

- Green: Pass
- Orange: Pass, but needs review (not all measurements done, GR above ten percent)
- Red: Fail (out of limits, GR above thirty percent)

With the chart tool, worksheet 'Charts' (see Figure 5.13), the results of all environments are visualized including limits. With the drop down box in the upper left corner the test is selected, the box below selects the environments and DUT parts. Limits are displayed as discrete values and in the chart.

Spec parameter groups (need to be defined by PM)			
Spec Param #			
Start	End	Groupname	Description
0000	0049	ANA_RX	analog receive
0050	0099	ANA_TX	analog transmit
0100	0129	ANA_BG	analog bandgap
0130	0159	ANA_LNA	analog lna
0500	0549	DIG_CPU	digital CPU
0550	0599	DIG_BIST	digital BIST
0800	0899	EVAL_ONLY	just internal evaluation
0900	0999	DEBUG	debugging

Figure 5.11: Design Verification Matrix - Parameter groups

	Verification Status (Automatically calculated)																	
							enter limit here [%]											
	Results vs. Limits						10				10		Gr&R % 100 x r&R / Tolerance			GR% 100 x R / Tolerance		
test_name:	Spec vs. Sim	Spec vs. Eval	Spec vs. Test	Verification vs. Eval	Verification vs. Test	GR% max 100 x R / Tolerance	limit ±10% of sim range	Sim vs. Eval LT	Sim vs. Eval RT	Sim vs. Eval HT	limit ±10% of sim range	Sim CPK vs. Test Cpk	LT	RT	HT	LT	RT	HT
00000F_XTAL	PASS	PASS	PASS	PASS	PASS	4.3436	0.03	PASS	PASS	FAIL	0.3611	FAIL	8.4711	8.5201	17.663	0	0	4.3436
00010I_total	PASS	PASS	PASS	PASS	PASS	0	1.16		FAIL		0.2067		18.915	16.55	19.442	0	0	0
00020I_sleep	PASS	PASS	PASS	PASS	PASS	0	0.14		FAIL				51.124	6.5591	56.985	0	0	0
00030I_dsleep	PASS	PASS	PASS	PASS	PASS	0	0.244	PASS	FAIL	FAIL	0.1			12.815	42.581		0	0
00250V_DIFF_AB	PASS	PASS	PASS	PASS	FAIL	0					0.425		46.128	23.538	5.3879	0	0	0
00251V_outa				PASS	PASS	0							6.71	6.5126	0.1346	0	0	0
00252V_outb																		
00260RVDD	PASS	PASS	PASS	PASS	PASS	0	0.024		PASS		0.0111		60.458	1.1619	1.0167	0	0	0
00270SVDD	PASS	PASS	PASS	FAIL	PASS	0	0.014		PASS		0.1556			47.885	38.308		0	0
01500D_timer	PASS																	
01501D_cntdown	PASS																	
01502D_cntup	PASS																	
01503D_cntirq	PASS																	

Figure 5.12: Design Verification Matrix - Verification status

The GR chart, worksheet 'GR Charts' (see Figure 5.14), is a graphical output of the GR values from *Status* worksheet. Colouring is equal to Status colouring.

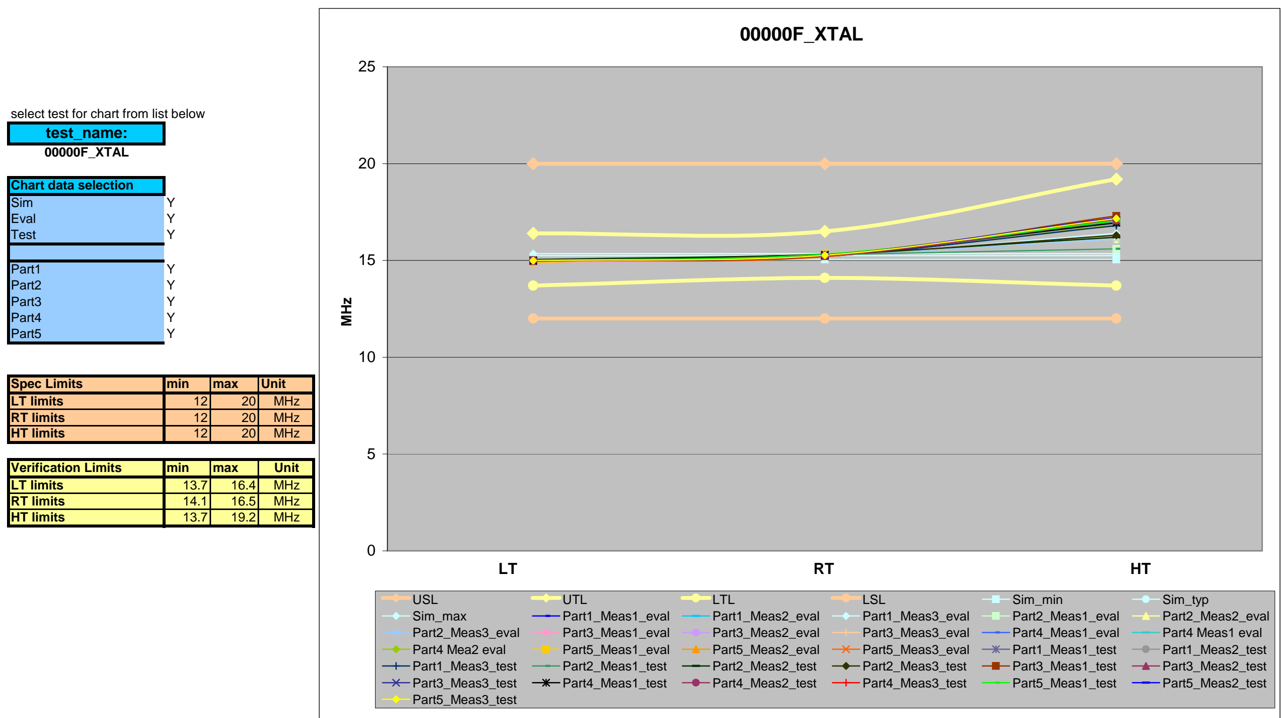


Figure 5.13: Design Verification Matrix - Chart tool

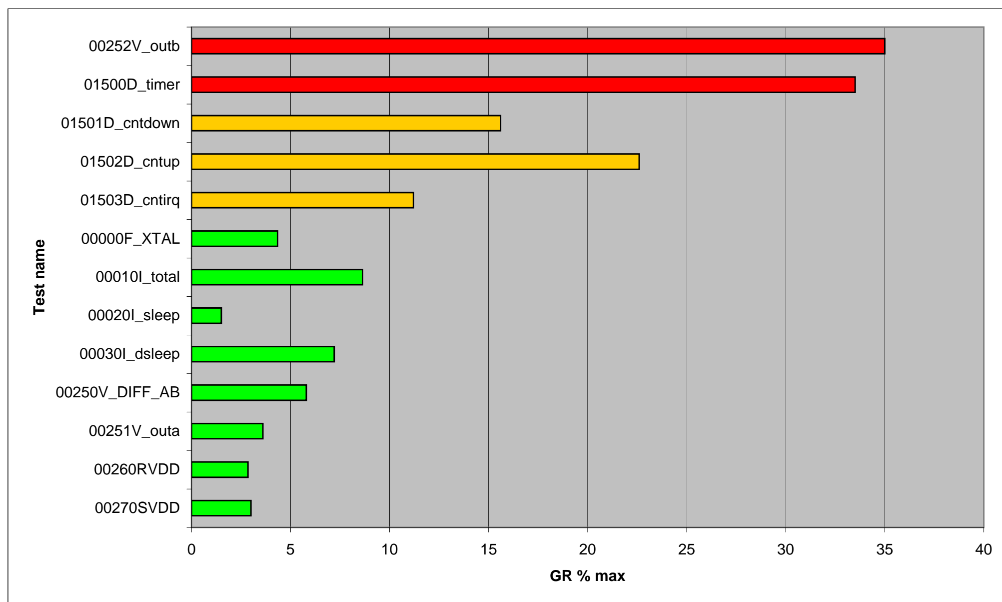


Figure 5.14: Design Verification Matrix - GR charts

5.2.2.2 Communcation Channel Definition

The communication definition file (*comm_def.txt*) is used to define all channels that can be used to configure the DUT. Furthermore a default channel can be specified for each memory type, where memory type can be a Central Processing Unit (CPU) register as well as a bit stream chain (e.g. JLCC).

```
[header]
IP Name: <ip name>

[channels]
<channel name> ; [<type name>;...]
```

The example below shows a comm_def file, which defines SPI as default communication channel for APB and SFR. JLCC is specified as default channel for 'Digital_chain_0' configuration chain. Furthermore the file defines Universal Asynchronous Receive/Transmit (UART) and Single Wire Interface (SWI) to be available interfaces.

```
[header]
IP Name: SD_IP_1

[channels]
SPI ; ABP ; SFR
JLCC ; Digital_chain_0
UART
SWI
```

5.2.3 Plans

During the verification process not each environment executes each test case. Furthermore optimisations may be defined especially for *production test (ATE)* verification environment.

5.2.3.1 Simulation Plan

The simulation plan defines which tests need to be implemented and executed during design simulation. Yellow columns ('Test name' to 'Responsible') in Figure 5.15 contain data linked from DV matrix, blue coloured columns are simulation specific data.

The simulation specific data includes responsibilities as well as duration estimations and simulation execution data ('test bench name', 'regression file' and 'command line'). The command line column specifies the command line for execution of the according test, which is especially needed for repeating tests (reproducibility) and automation.

<IPname> - tests **Do not edit grey fields!**

Project:	<Project Name - Dash>	Number of requirement parameters:	8
Last update:	01.01.2000	Number of tests:	13
PM:	<Name>	By:	<user>
Spec Rev:	<Name>	Notes:	
Testcase Root:	..\tests		

Lookup index	17	34	35	36
Test name	**Verification parameter desciption**	**Parameter Observability** **Test Description**	**Parameter Controllability** **Configuration**	**Responsible**
00000F_xtal	Xtal frequency	Measure on test bus	JLCC configuration	abc
00100I_total	total current consuption	Measure on supply pin	JLCC and CPU config	xyz
00200I_sleep	sleep current consuption	Measure on supply pin	JLCC and CPU config	abc
00300I_dsleep	deep sleep current consuption	Measure on supply pin	JLCC and CPU config	mno
02500VD_outab	v_diff = ABS(v_A - v_B)	calculation of A - B	No configuration.	mno
02501V_outa	voltage A	Measure output voltage on Testbus.	JLCC configuration	mno
02502V_outb	voltage B	Measure output voltage on Testbus.	JLCC configuration	mno
02600V_rvddreg	regulator output voltage	Measure Output Voltage on RVDD Pin	No configuration.	rsa
02700V_svddreg	regulator output voltage	Measure output voltage on SVDD pin	No configuration.	rsa

<name> - Simulation Plan

IP/Sub-Project:	<name>				Number of tests:	9
Update:	01.01.2010	By:	<user>		Implemented	1
Responsible:	<Name>	Today/CW:	22.07.2010	30	Link Errors:	0
Notes:						
Results Root:	..\results\eval\				Total effort:	**4.5 days**

Resp. Execution	Effort [days]	Constraint	CW	Priority / Sequence	Status	Self check	Test bench name	Regression file / FW file	Command line
abc	0.5 3.0 1.0	22.05.2008	21		done			boot_n_times.c	-tcl ../scripts/test_00000.tcl

Figure 5.15: Simulation plan spread sheet

5.2.3.2 Evaluation Plan

This plan is very similar to the simulation plan. Equal to the simulation plan, the yellow columns are linked from the DV Matrix and the blue ones are evaluation execution specific.

<IPname> - tests — **Do not edit grey fields!**

Project:	<Project Name - Dash>	Number of requirement parameters:	8
Last update:	01.01.2000	Number of tests:	13
PM:	<Name>	By:	<user>
Spec Rev:	<Name>	Notes:	
Testcase Root:	..\tests		

Lookup index	17	34	35	36
Test name	**Verification parameter desciption**	**Parameter Observability** **Test Description**	**Parameter Controllability** **Configuration**	**Responsible**
00000F_xtal	Xtal frequency	Measure on test bus	JLCC configuration	abc
00100I_total	total current consuption	Measure on supply pin	JLCC and CPU config	xyz
00200I_sleep	sleep current consuption	Measure on supply pin	JLCC and CPU config	abc
00300I_dsleep	deep sleep current consuption	Measure on supply pin	JLCC and CPU config	mno
02501V_outa	voltage A	Measure output voltage on Testbus.	JLCC configuration	mno
02502V_outb	voltage B	Measure output voltage on Testbus.	JLCC configuration	mno
02500VD_outab	v_diff = ABS(v_A - v_B)	calculation of A - B	No configuration.	mno
02600V_rvddreg	regulator output voltage	Measure Output Voltage on RVDD Pin	No configuration.	rsa
02700V_svddreg	regulator output voltage	Measure output voltage on SVDD pin	No configuration.	rsa

<name> - Evaluation Plan

IP/Sub-Project:	<name>				Number of tests:	9
Update:	01.01.2010	By:	<user>		Implemented tests:	4
Responsible:	<Name>	Today/CW:	22.07.2010	30	Link Errors:	0
Notes:						
Results Root:	..\results\eval\				Total effort:	**6 days**

Sweep parameters	**Resp. Execution**	**Effort [days]**	**Constraint**	**CW**	**Priority / Sequence**	**Status**	**Measurement SW Link**
dvdd:I:3.0;5.0;0.5 V JLCC_reg1:L:0x00;0xFF D	abb	1.0	22.03.2010	13		done	..\sw\labview\meas_F.vi
	abb	0.5				done	..\..\sw\sw\meas_I.py
vbat:I: 10;14;0.5 V	abb	0.0				done	..\..\sw\sw\meas_I.py
	aab	0.0				done	..\..\sw\sw\meas_I.py
	opx	1.0	21.02.2010	8			..\..\sw\labview\meas_V.vi
	opx	0.0	21.02.2010	8			..\..\sw\labview\meas_V.vi
	opx	0.5	22.02.2010	9			
	jao	1.5	30.03.2010	14			
	jao	1.5	30.03.2010	14			

Figure 5.16: Evaluation plan spread sheet

The 'sweep parameter' is a special column for evaluation, which defines a parameter sweep for a specific test. The syntax for a sweep is defined as

```
# as Interval
<param name>:I:<start>;<stop>;<step>;<unit>
```

```
# as List
<param name>:L:<val_0>;<val_1>;...;<val_n>;<unit>

Examples:
dvdd0:I:2.5;3.9;0.1 V

vdd:L:1.8; 2.5; 3.3 V
```

Columns 'Resp. Execution' to 'Status' are equal to simulation plan. 'Measurement SW Link' is the path to the test implementation for evaluation. For optimisation purpose the column 'Predecessor' may be used to define dependencies of tests by entering test names.

5.2.3.3 Production Test Plan

The production test plan is not described here with figures, since it is equal to evaluation plan except for the columns 'Sweep parameters' and 'Measurement SW Link' which are not needed within production test (ATE) execution.

5.2.3.4 Lot List

The lot list is used to uniquely identify verification execution runs (lot ID). The lot ID is stored within each result file, which is used to gather all results from one single run. This mechanism is required for report generation and to reproduce results.

Lot list

Lot ID	Description
000.0001_RF_run	RF tests done

Figure 5.17: Lot list spread sheet

The list shown in Figure 5.17 simply stores the lot identifier and a description.

5.2.4 Tests Content

Any data referring to a test is stored in the directory *'verif/tests/<test name>/'*, which is shown in Figure 5.18 with proposed content of a test. Each test needs to provide a *Sequence* file, which describes the activities of this test formally using the sequence language (see Section 5.3). For special setups, which can not be represented in an excel spread sheet, an optional *int_setup* file is proposed. As shown in Figure 5.18 this can be done for internal, Test Bench (TB) setup and external setup. It is also possible to store these setups in a *common* directory, if the setups are used within more than one test. Special external setups are always stored in this *common* directory. Other items like firmware or TCL scripts may be stored in separate directories like *'fw'* or *'tcl'*.

With the content of the tests directory and the *test spec* spread sheet, engineers should be able to reproduce a result of a certain test.

As described above the *sequence* file specifies the actions during a test. For setup description the *test spec* spread sheet or optional files are needed. As Figure 5.19 shows *int_setup* specifies the internal state of the DUT. *Ext_setup* specifies the initial and after-reset conditions at DUT boundary. Different to *ext_setup*, *patterns* specifies not a single state, but also defines conditions at DUT boundary. *Meas_setup* describe the used instruments, connections and PCB setup, but this should be a result documentation or proposal in tests directory. The test result has to be independent from its implementation (see Section 5.2.5).

5.2.4.1 General Guidelines

Here some general guidelines and restrictions for all type of files are defined.

Comments '#' is the begin of a comment (except vector files), beginning from the '#' character up to the end of the line.

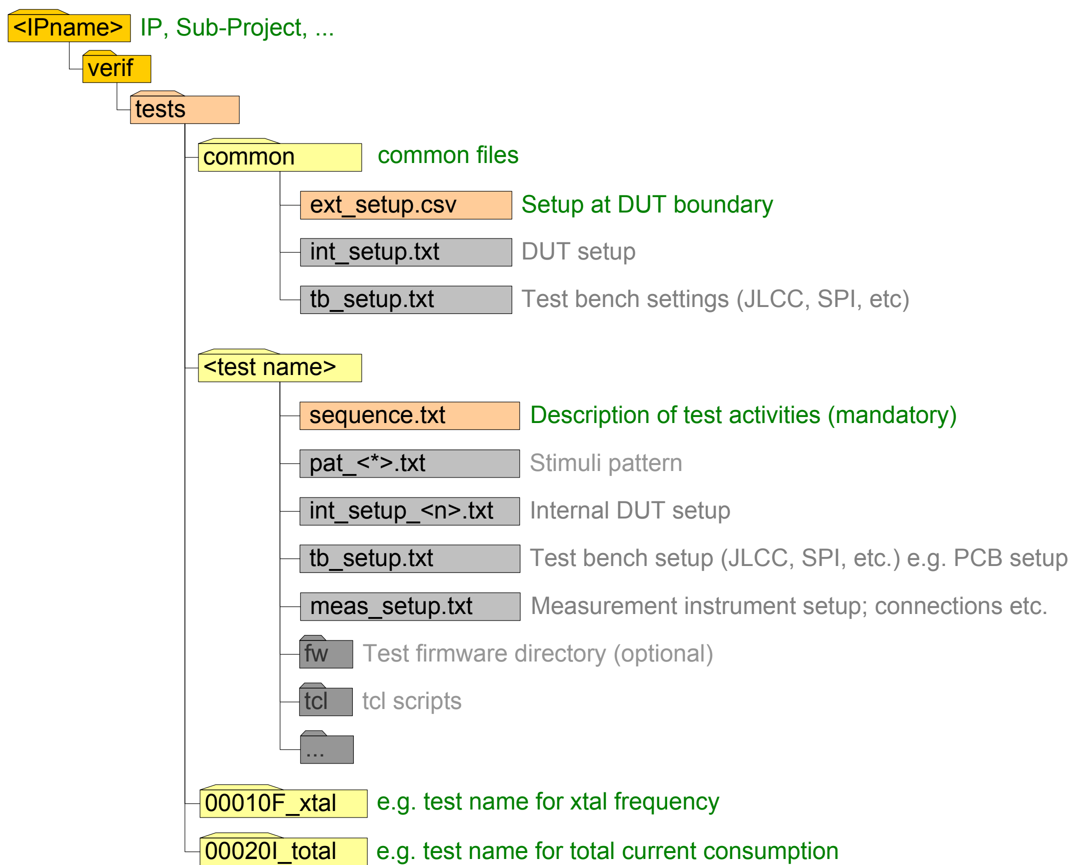


Figure 5.18: Test Case directory content

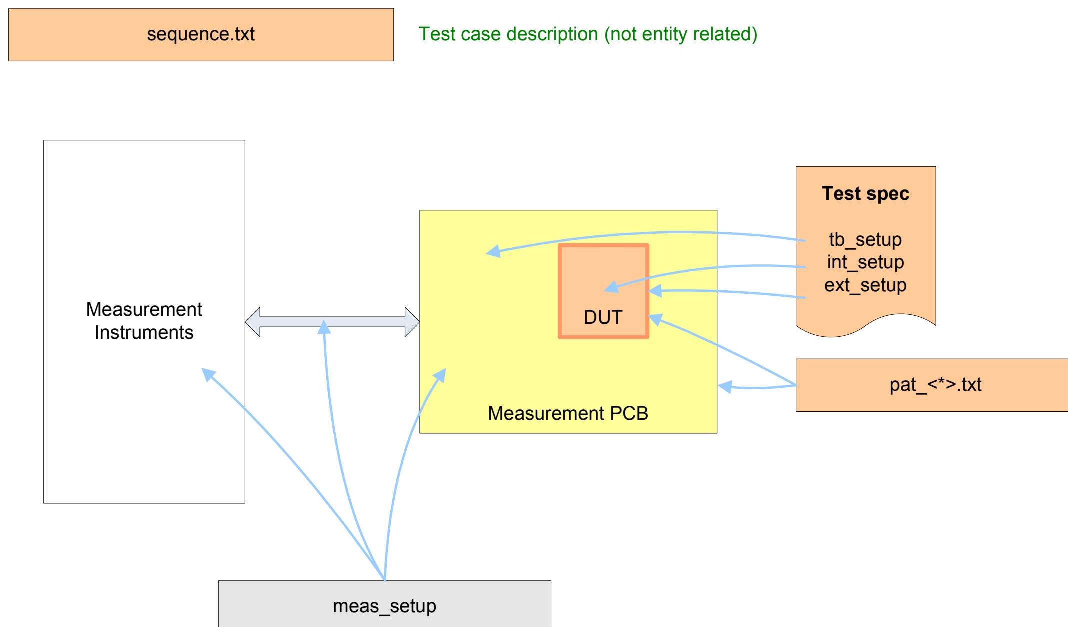


Figure 5.19: Relation of test files to entities of evaluation environment

Empty lines are allowed for all file types.

SI prefixes A general file format definition requires specification of unit prefixes to be used. Table 5.2 shows the valid Système International d'unitès (SI) units, abbreviations and alternatives.

Units Valid units can be found in Table 5.3 subsequently.

Name		Abbrev.	Alternative
Peta	10^{15}	P	e15
Tera	10^{12}	T	e12
Giga	10^{9}	G	e9
Mega	10^{6}	M	e6
Kilo	10^{3}	k	e3
Milli	10^{-3}	m	e-3
Micro	10^{-6}	u	e-6
Nano	10^{-9}	n	e-9
Pico	10^{-12}	p	e-12
Femto	10^{-15}	f	e-15
Atto	10^{-18}	a	e-18

Table 5.2: Prefixes and abbreviations

Name	Abbrev.
Volt	V
Ampere	A
Watt	W
Voltampere	VA
Second	s
Hertz	Hz
Ohm	Ohm
Farad	F
Henry	H
Decibel	dB or dBm

Table 5.3: Units and abbreviations

5.2.4.2 Internal Setup

The internal setup (int_setup) is either specified in the *test spec* spread sheet (SD custom defined ATE spread sheet) or in a separate file, where the file format is specified subsequently. Since DUT setup could be done via CPU firmware (register access) or an external DUT configuration interface, there are two upper-level setup types defined - *REG* and *STREAM*. *REG* is used to specify any kind of registers, which need to be configured with the use of a firmware running on an embedded CPU. The *STREAM* type is used to describe configuration data, which is set up through external bit stream interfaces. The internal DUT setup needs two files: the int_setup and the names file, where registers (names and addresses) and streams (logical register, bitnames and postions) are specified. The int_setup only specifies the value of a certain register or stream bit.

Template

```
[header]
IP Name:   <ip name>
Test Name: <test name>

[<reg name>]
# REG  always set via CPU firmware
<register name>;<value>

[<stream name>]
# via external interface
<logical register name>;<bit name>;<0|1>
```

The file is subdivided into sections, indicated by squared brackets. The header section is obligatory for an int_setup file, but [<name>] sections are optional. The type of the section is project dependent, for example a *SFR* will be a *REG* (defined in names file). The example below illustrates a possible int_setup file.

```
[header]
IP Name:   sd_ip_1
Test Name: 00010F_Xtal

[SFR]
timer_0;0xF1

[APB]
flywheel_conf;0xFF000000

[Digital_chain_1]
digital_conf_reg;bit_1;1

[analog_config]
regulator_config;supply_en;1
```

For the mapping of names to addresses or bit positions a further file type is introduced, the names (int_names.nms) file. This file contains the name-address pair for all CPU registers and all bits of configuration streams. Furthermore here the section names are mapped to either STREAM or REG, which is done once for a project.

names file

```
[header]
IP Name: <ip name>

[REG;<reg name>]
<register name>;<address>
<register name>;<address>
<register name>;<address>
...

[REG;<reg name>]
<register name>;<address>
<register name>;<address>
<register name>;<address>
...

[STREAM;multiplexer_settings]
<stream name>;<1st mux setting>;<2nd mux setting>; ... <n-th setting>
<stream name>;<1st mux setting>;<2nd mux setting>; ... <n-th setting>
<stream name>;<1st mux setting>;<2nd mux setting>; ... <n-th setting>
...

[STREAM;<stream name>]
<logical register name>;<bit name>;<bit position>
<logical register name>;<bit name>;<bit position>
<logical register name>;<bit name>;<bit position>
...
```

It is common to use multiplexers for configuration streams, therefore the *names file* includes a section 'multiplexer_settings' which specifies the setting for all streams, if needed.

5.2.4.3 External Setup

Equivalent to the internal setup, the external setup (ext_setup) is either specified in *test spec* spread sheet or as a separate file in the *'verif/tests/common'* directory. This should only be the case if setups cannot be represented in the test spec spread sheet.

[header]
IP Name : <ip_name>

	setup1	setup2	...	setup<n>
pin0	FV 4V MAX_I 500mA	FV 4.1V MAX_I 500mA		
pin1	FV 3.3V MAX_I 200mA	FV 3.3V MAX_I 200mA		
pin2				
pin3				
.				
.				
.				
pin<n>				

Figure 5.20: External setup file format

Figure 5.20 shows the ext_setup format, where the header is similar to int_setup. The syntax of conditions can be found in Table 5.4.

Condition	Desciption
NC	not connected
$FV < value > MAX_I < value >$	force voltage with current limited
$FI < value > MAX_V < value >$	force current with voltage limited
$LC < value > CONN_TO < pin >$	connect capacitance to pin
$LR < value > CONN_TO < pin >$	connect resistance to pin
$LL < value > CONN_TO < pin >$	connect inductance to pin
DCH	digital channel connected
$ARB < patternref >$	pattern reference from sequence file
$SINE < patternref >$	pattern reference from sequence file

Table 5.4: External setup conditions

5.2.4.4 Stimuli Patterns

Patterns are either provided by vector definitions or by signal specification. Vector definitions can be processed by the ATE as is, but this vector format can only be used for digital signals. Analog signals (e.g. sine waves) are defined with a signal specification. Since the vector file has a fixed file format defined previously, comments are marked with '//' instead of the usual '#'.

```
//
// header information
//
import tset <name>;
vector( $tset, <p_name_0>, <p_name_1>, <p_name_2>, ... , <p_name_n>)
{
  [repeat <n>] > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
  [repeat <n>] > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
               > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
               > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
  [repeat <n>] > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
  halt         > <t_set>     <v_pin_0> <v_pin_1> <v_pin_2> ... <v_pin_n> ;
}
```

The vector signature defines which pins (as for ext_setup) are used. The first parameter defines the time set (indicated by '$'), which is needed for signal generation and may change for each vector specification. The time set needs to be specified in the vector file by the command 'import'. Since the time set is just needed at ATE the definition of a time set is not discussed here, because it depends on the used ATE.

The optional command 'repeat' can be used to apply the current set of pin values 'n' times, where a set of values consists of a single line. The command 'halt' indicates the end of the vector.

An example for a vector file can be found below.

```
import tset pads;
vector ($tset, reset, gpio0_0, gpio0_1, gpio0_2, gpio0_3)
{
//                               g g g g
//                               p p p p
//                             r i i i i
//                             e o o o o
//                             s 0 0 0 0
//                             e _ _ _ _
//                             t 0 1 2 3

repeat 2       > pads          1 X X X X ;
repeat 2       > pads          1 X X X X ;
repeat 2       > pads          1 X X X X ;
               > pads          0 0 0 1 0 ;
               > pads          0 0 1 0 1 ;
repeat 2       > pads          0 0 1 0 0 ;
repeat 29      > pads          0 1 0 0 0 ;
repeat 2       > pads          0 0 0 0 1 ;
               > pads          0 0 1 0 1 ;
  .
  .
  .
}
```

The second alternative for specifying stimuli patterns is the signal specification. The keyword 'sig_spec' in square brackets indicates this alternative. The 'type' of the signal is not limited to special types, for example the signal can be of type 'SINE' or 'SQUARE', but also other signal types. The keywords 'sampling_rate', 'frequency', 'amplitude' and 'offset' need to be specified for each kind of signal separately.

```
[sig_spec]
type;[SINE|ARBITRARY|SQUARE|...]
sampling_rate;<Hz>
frequency;<Hz>
amplitude;<V>
offset;<V>
```

5.2.4.5 Stream Pattern File

The stream pattern file combines configuration of DUT and read back, for comparison or storage to a variable, in one pattern file. This pattern file is similar to internal setup files (int_setup) with the extensions of comparison or storage. The read back stream can be compared against H ('1'), L ('0') and X (don't care) bit for bit. Furthermore a variable letter may be specified to store a part or the complete read back stream (see Section 5.3.4 for detailed information on variables). The bit name definition is done in the int_names.nms file ('verif/specs/<DUT>/').

NOTE: This comparison is just possible for *STREAM* types, which work like shift registers (write and read at the same time).

```
[header]
IP Name:   <ip_name>
Test Name: <test name>

[<stream name>]
# logical register; bit name; write value; reference/read value/letter
# variable letters may be R, S, T, V
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
...

[<stream name>]
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
...

[<stream name>]
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
<logical register>;<bit name>;<0|1>;<H|L|X|<pattern variable letter>
...
```

5.2.4.6 Description of Activities - Sequence

The sequence file describes the activities that need to be done by any test implementation. Equal to the file formats above, sections (defined by squared brackets) are used here as well. The section *info* is used for documentation and for informal description of test activities. Sections *int_setup*, *ext_setup* and *tb_setup* are used to specify required setups, which need to be placed in the corresponding test directory. Furthermore *pattern* files may be used within the test. The *sequence* section formally describes test activities, referring to sections specified above. The sequence language (description of activities) itself is defined in Section 5.3 in detail.

```
[header]
IP Name:   <ip name>
Test Name: <test name>

[info]
# All additional information regarding TC execution.

[int_setup]
<int_setup_0>
<int_setup_1>
..
<int_setup_n>

[ext_setup]
<ext_setup>

[tb_setup]
<tb_setup_0>

[pattern]
<pattern_0>

[sequence]
# defines sequence of TC execution -> sequence language
```

5.2.4.7 Measurement Setup

The measurement setup file needs to document the used measuring equipment and the connections between them. This setup only needs to be described once for a test run (if the setup does not change). This description of the measurement setup guarantees the reproducibility of each test. Below the format of the measurement setup file is provided.

```
[header]
IP Name:   <ip name>
Test Name: <test name>

[<instrument name>]
<setup>

# setup:
# The setup is different for every instrument.
# Therefore a detailed documentation of the used
# instrument setup is required. The level of detail
```

```
# is up to the engineer, but the test case execution has
# to be reproducible! The basic instrument setup is
# done by the executing engineer.
#
# e.g. used measuring probe; measurement range; trigger level; etc.
#

[connections]
<instrument name>;<connector name>;<DUT pin name>
<instrument name>;<connector name>;<measurement PCB pin name>

# instrument names need to match sections in instrument setup file

[pcb]
# Setup of PCB described informally
# e.g. Jumper J8 closed; Switch S1 on; etc.
```

5.2.4.8 Test Bench Setup

Equal to the DUT also parts of the PCB could be configurable by a stream interface (e.g. JLCC). This could be specified either via *test spec* spread sheet or inside a *TB setup* file. The file format is very similar to *int_setup* file.

```
[header]
IP Name:   <ip name>
Test Name: <test name>

[<stream name>]
<logical register name>;<bit name>;<0|1>
<logical register name>;<bit name>;<0|1>
<logical register name>;<bit name>;<0|1>
```

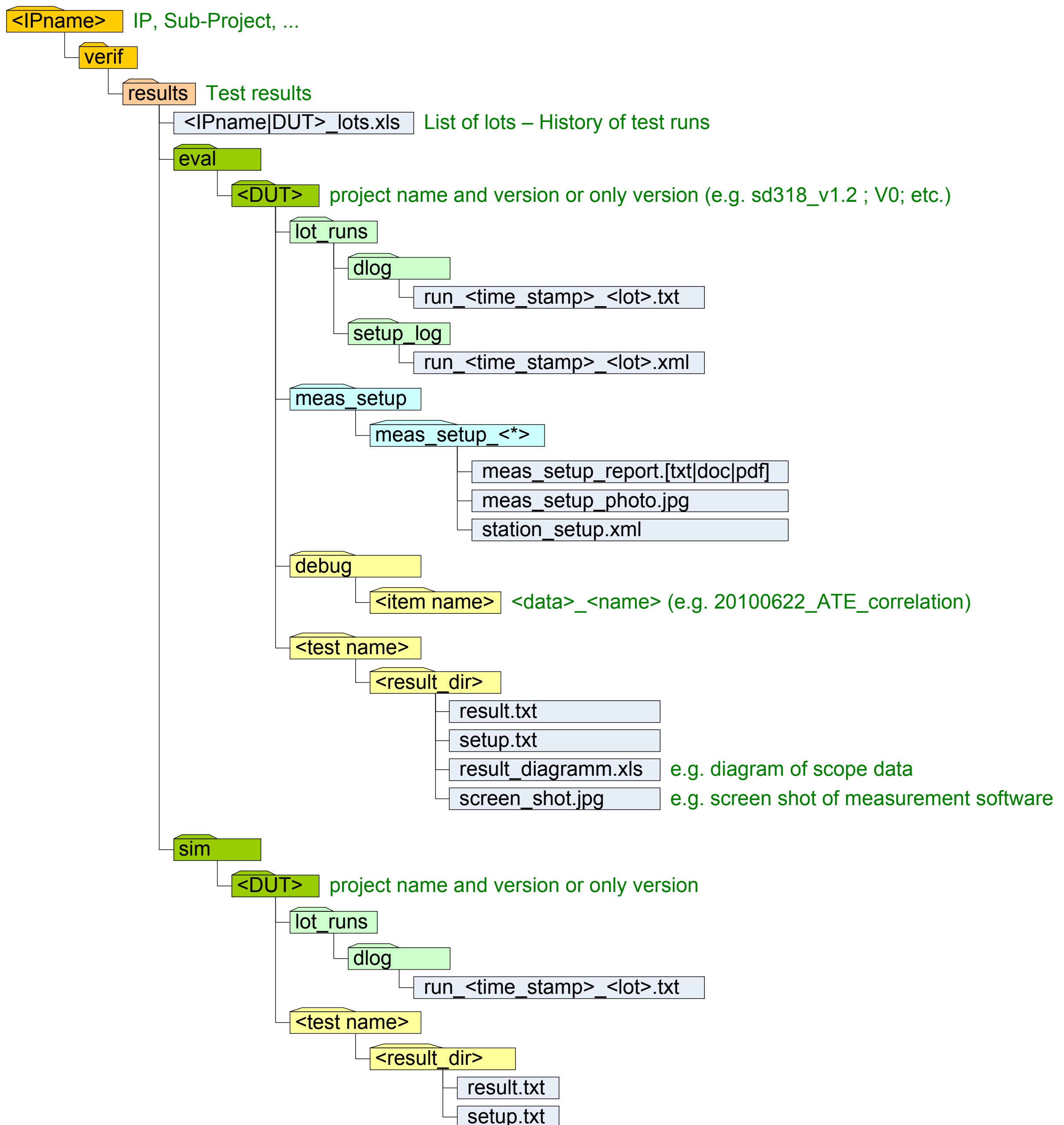
As for *int_setup* also the test bench setup needs to be defined in a names file (see Section 5.2.4.2), which is named tb_setup.nms ('/verif/hw/boards/<item>/').

5.2.4.9 Station Setup

The station setup file specifies all included measurement equipment formally in Extensible Markup Language (XML) file format. The structure of this setup file is described in Section 5.4.1.2 since it is related to evaluation automation.

5.2.5 Results Structure

All TC result data is stored in the *verif/results* directory of each IP. Inside this directory a separate directory is created for each environment (e.g.eval), where each environment directory follows the same structure inside. The <DUT> directory should be named according to the tested device (e.g. version of product). The *lot_runs* directory collects all data of automatic measurement runs or simulation runs. *meas_setup* stores the measurement setup reports. For each executed test a separate directory is created to save its results (see Figure 5.21).



<result_dir>: [<token>_]<YYYYMMDD>[HHMMSS]

<token>:
- typ
- parm_<parm_name(s)>
- corn
- mc
- rcx

examples for result directory names:
- 20100220120100
- 20100225
- typ_20100221
- parm_vdd_20100115172355
- corn_20090430

Figure 5.21: Result directory structure

5.2.5.1 Result File Format

The raw result data is stored in the *result.txt* file, which needs to fulfill the subsequent format.

```
[header]
lot_id:                   <lot_id>
time_stamp:               <YYYY><MM><DD>[<hh><mm><ss>]
responsible:              <executing engineer>
board_name:               <board name>
board_version:            <board version>
board_serial:             <serial number>
board_fpga_name:          <fpga name>
board_fpga_version:       <fpga version>
fw_framework_name:        <firmware framework name>
fw_framework_version:     <firmware framework version>
meas_setup:               <measurement setup>
test_spec:                <test spec name and version>

[VALUES]
<test>;<cond1>;<cond_1_unit>;...<cond_n>;<cond_n_unit>;<value>;<unit>

# OR (for 2 dimensional data)
<test>;<conditions>;<names>;<units and scales>
      ;x_val1;y1_val1;y2_val1 ...
      ;x_val2;y1_val2;y2_val2 ...

# <conditions>      : <cond1>;<cond1_unit>;...;<cond_n>;<cond_n_unit>;
# <names>           : <x_name>;<y1_name>;...;<yn_name>;
# <units and scales>: <x_unit>;<y_unit>;<x_scale>;<y_scale>
```

NOTE: Leading ';' for CSV/Excel presentation of table.

5.2.6 Reports

Out of measurement and simulation results any kind of report could be created (textual, graphical, etc.). Because of the well defined formats it is recommended to implement a report generator tool in future.

5.2.7 Verification Software

Any kind of verification software needs to be stored in IP structure it belongs to. The common path of verification software inside an IP is

```
<path to IP>/verif/sw/
```

where simulation and evaluation software is further separated into

```
<path to IP>/verif/sw/eval/
<path to IP>/verif/sw/eval/measurements/<measurement item>
<path to IP>/verif/sw/eval/GUI/<DUT>  # configuration GUI

<path to IP>/verif/sw/sim/
```

NOTE: If the functionality of measurements needs to be changed (for example for a new DUT version), they need to be copied, modified and saved with a different name. A VCS should not be used for this purpose, because the affected measurement may be needed for the evaluation of an other DUT or DUT version. The measurements VCS revision will likely not be checked by engineer which causes debug time. The VCS can be used for the new created measurement software as usual. The influence of changes on modules of one measurement to others needs to be considered because of the hierarchical design of measurement software (e.g. hardware drivers).

Implementation of measurements is not discussed in this document, because this is not related to the generic verification platform concept.

5.3 Test Sequence Language

For a common format and a formal description of test activities a sequence language is introduced, which is stored in the *sequence.txt* file (see Section 5.2.4.6). This language is formally defined as Context Free Grammar (CFG) in Backus-Naur-Form (BNF) in Section 5.3.5, to easily verify the syntax of the language. The structure of a sequence file is presented in the example below. The available elements of the sequence language are described in the subsequent sections.

```
[header]
IP Name:   sd_ip_1
Test Name: 00010F_Xtal

[info]
# All additional information regarding TC execution.

[int_setup]
init    './int_setup_0.txt'
jlcc_01 './int_setup_1.txt'

[tb_setup]
tb_init './tb_setup_0.txt'

[pattern]
sti_001 './stimuli.txt'
pat_001 './stream_pat.txt'

[ext_setup]
# ext_setup out of '../common/ext_setup.csv'
ext_setup_001

[sequence]
reset ...

int_setup ...

measure ...

write_results ...
```

The sequence file is separated into sections specified by square brackets (e.g. [header]). The *header* section contains general data like IP and test name. The *info* section gives an informal description of the test activities and other comments referring to the test. All other sections are defined in detail subsequently.

Formatting To guarantee a well structured format the following restrictions are defined:

- Sections need to be kept in order
  1. header
  2. info
  3. int_setup
  4. tb_setup (optional)
  5. pattern (optional)
  6. ext_setup
  7. sequence
- Each statement (section, command incl. data, etc.) in a single line
- White spaces are ignored

Although line break is usually not part of a grammar, it is used to guarantee a well structured format. The CFG defines line break as '$\backslash r?\backslash n$', which also include Unix style line break.

5.3.1 Scope

This sequence language is indented to formally describe the test setup (DUT, PCB, etc.), test input (stimuli pattern, DUT communication etc.) and the measurement activities and the sequential order of these items. It is not meant to be a programming language for test execution and will not be interpreted by tools of the generic verification platform.

Sequence language elements:

- Test setup (int/ext/tb/stimuli)
- measurement activities (measure/calculate/write to variable)
- pin to measure
- DUT communication (read/write incl. pattern download)
- flow description (order of statements)

Implementation details are not part of the sequence file.

5.3.2 Test Data Definition

The test data needs to be defined in separate sections for a clear overview of what is needed within the test. There are four possible data sections (specified in square brackets):

- **int_setup**
- **ext_setup**
- tb_setup
- patterns

There needs to be at least one **int_setup** and exactly one **ext_setup** defined inside a sequence file, which guarantees a defined initial state of the tested system. Figure 5.22 shows the relations of the sequence file to other verification platform files, but not to test_spec spread sheet which is a single data source instead of multiple files (referenced by label names).

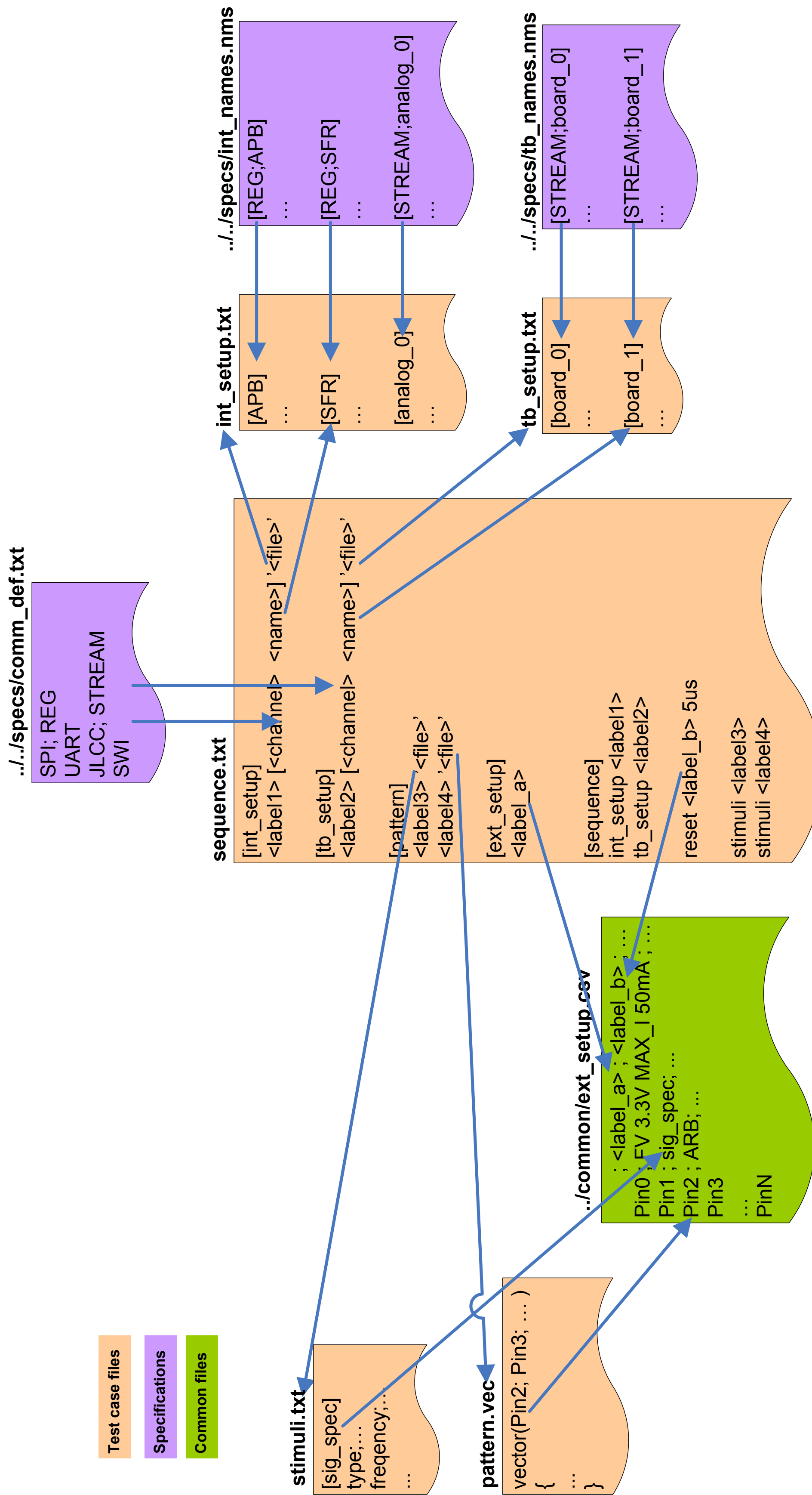


Figure 5.22: Sequence file - Relations

5.3.2.1 int_setup / tb_setup

An internal or test bench setup can be simply specified as label and according file name pair. Furthermore there is the possibility to define the part of the referenced file (with a communications channel specified), which should be used for this setup only.

```
[int_setup|tb_setup]
<label> [<comm channel> <name>] '<file name>'

# <comm channel>: Name of communications channel e.g. JLCC or SWI
# <name>: name of setup in file
```

Examples:

```
[int_setup]
  initial_0 'int_setup_0.txt'
  initial_1 '\\10.1.2.16\proj\costumer\test_spec.xls'

  xtal_on SPI SFR 'int_setup_1.txt'

  all_regs JLCC analog_0 './test_spec.xls'

  [tb_setup]
  # definition of test bench setup
  pcb_0 'tb_setup_0.txt'
```

The label is the reference of the setup used in the sequence later on, where the specified setup can be applied. As already described for internal setup files (*int_setup* see Section 5.2.4.2) streams and registers are differentiated, where registers are accessed via firmware and streams via an external configuration interface. These setups may be referenced by <name>, for example [SFR] section of int_setup file (<name> is 'SFR'). Furthermore the *comm channel* can be specified, which needs to be defined in a communication definition file (see Section 5.2.2.2).

5.3.2.2 pattern

A pattern file, either stimuli or stream data, can be specified as

```
[pattern]
<label> <path to file> [<pin name>;<pin name>; ...]
```

Examples:

```
  stim01 './patterns/pattern_file.vec'
  stim02 './patterns/pattern_file.vec' gpio_0 ; gpio_1 ; gpio_2
```

Equal to int_setup and tb_setup the label is used as reference in the sequence section. For integrity reasons the pin names may be specified here, but do not need to. The stimuli pattern itself is defined as pattern file (see Section 5.2.4.4).

5.3.2.3 ext_setup

Here the name of the external setup, defined in ext_setup.csv or Test-spec, is specified. The external setup describes the setup at DUT boundary at the beginning of a test as well as after a reset.

```
[ext_setup]
<label>
```

5.3.3 Sequence Commands

The test activities are described as single statements which are treated sequentially. The statements below are valid inside a sequence section, which is defined as

```
[sequence]
```

5.3.3.1 reset

A *reset* could be done via an external setup *label* (name of setup in ext_setup.csv or Test-spec) or via a pin name and value tuple list. It is mandatory that the values of this list keep the *ext_setup* pin value format (e.g. FV 3.3V MAX_I 50mA - see Section 5.2.4.3).

```
reset <ext_setup label>|(<pin_name value>[;<pin_name value>;...]) <duration>
```

Examples:

```
  reset ext_label_01 50.0 ms
  reset reset_pin FV 3.3 V MAX_I 50.0 mA 3.0 s
```

A reset must have a specific duration, which is defined as

```
<duration>: <float> [<SI prefix>]s
  e.g. 5.5ms or 100.0e-6s
```

Valid prefixes can be found in Table 5.2. After the reset the DUT boundary setup, which is specified in ext_setup section, is recovered.

5.3.3.2 int_setup / tb_setup

An internal setup can simply be applied via the statement

```
int_setup <label>
```

which is equal for test bench setup

```
tb_setup <label>
```

where the *label* is specified in section *[int_setup]* or *[tb_setup]*.

5.3.3.3 wait

This statement is used to wait for specific duration or on events.

```
wait  <duration> | <pin state> | <register value>
```

Examples:

```
wait 1.0 ms
wait 1.0e-3 s
wait gpio_pin = 3.3V
wait clk_pin > 1 MHz
wait sfr_reg < 0x1F
```

The duration specification of the *wait* statement is equal to the reset duration. Furthermore it is possible to wait for two different events: either a pin state or a register value. For waiting on events the subsequent compare operators are valid:

Compare operator	Description
=	equal
>	greater than
<	less than
>=	greater and equal than
<=	less and equal than

Table 5.5: Compare operators

A *wait* statement for a pin state is defined as:

```
wait <pin name> <compare op> <value> [<SI prefix>]<quantity>
```

Example:

```
wait gpio0 >= 3.2V
```

Table 5.6 shows valid pin state quantities to wait on.

Quantity	Abbreviation
Volt	V
Ampere	A
Watt	W
Volt-Ampere	VA
Hertz	Hz
Ohm	Ohm
Farad	F
Henry	H
Decibel	dB or dBm
Phase noise	dBc/Hz

Table 5.6: Measurement quantities

The wait statement for a register value is defined as:

```
wait <register name> <compare operator> <compare value hex>
```

Example:

```
wait timer_0_count > 0x8A
```

5.3.3.4 stimuli

Stimuli can be applied to pin(s) via the statement

```
stimuli <label>
```

NOTE: How the pattern is applied and to which pins it is applied is defined within the pattern file itself.

5.3.3.5 measure

The *measure* statement is one of the key elements of the sequence section, but is not mandatory. This statement is used to define measuring on one or more pins and defined as

```
measure <PinToMeasure> <type of meas> [<variable>] [<duration|timeout>]


# <PinToMeasure> = <pin name1>[; <pin name2>; <pin name3>; ...]

# <type of measurement> = [SI prefix]<quantity>[<qualifier>]

# <variable>: e.g. LV:R:var_name:1 (see section 'Variables')

# <duration|timeout>: e.g. 3.0 ms (equal to duration for wait command)
```

Examples:

```
  measure out_pin V
  measure pin1; pin2; pin3 mApp
  measure f_out MHz 10ms
  measure gpio_1 Vrms LV:M:voltage:2
```

The result of the measurement may be stored in a variable (var name) or is implicitly written into the result file if no variable is specified. The optional arguments *duration* and *timeout* may be used for frequency measurements. Valid SI prefixes, quantities and qualifiers are specified in tables 5.2, 5.6 and 5.7. An additional measurement quantity is Pattern, which may be recorded (abbreviation 'pat').

5.3.3.6 operation

Since not every parameter can be measured within one test, operations on results of tests need to be done. These are specified as

```
operation <Op1> <operator> <Op2>
```

Abbrev	Description
p	peak
pp	peak to peak
dc	mean
rms	root mean square
_d	differential _d may be combined with other qualifiers above

Table 5.7: Measurement qualifiers

Examples:

```
operation 01011V_outa -ii 01012V_outb
operation 100 /md nop
operation 02001I_out1 //mid 3.14
```

where the *Op* may be a test name, which need to be a valid test name out of the DV matrix (see Section 5.2.2.1) or a constant value.

Tables 5.8 and 5.9 show valid operators and operands.

Abbrev.	Description
i	indirect
d	direct
M	the measurement taken in this test
M1	A previous measurement number
M2	A previous measurement number
K	a constant (double number)
nop	no operand or no operator

Table 5.8: Operator abbreviations

The operation result is always implicitly written to the result file, because very complex or nested calculations should not be done within this sequence language.

Operator	Op1	Op2	Result
/ii	M1	M2	M1 / M2
/id	M1	K	M1 / K
/di	K	M2	K / M2
/mi	M1	nop	M / M1
/im	M1	nop	M1 / M
/md	K	nop	M / K
/dm	K	nop	K / M
//mid	M1	K	M / M1 / K
//mii	M1	M2	M / M1 / M2
//imd	M1	K	M1 / M / K
//iim	M1	M2	M1 / M2 / M
*ii	M1	M2	M1 * M2
*id	M1	K	M1 * K
*di	K	M2	K * M2
*mi	M1	nop	M * M1
*im	M1	nop	M1 * M
*md	K	nop	M * K
*dm	K	nop	K * M
-ii	M1	M2	M1 - M2
-id	M1	K	M1 - K
-di	K	M2	K - M2
-mi	M1	nop	M - M1
-im	M1	nop	M1 - M
-md	K	nop	M - K
-dm	K	nop	K - M
+ii	M1	M2	M1 + M2
+id	M1	K	M1 + K
+di	K	M2	K + M2
+mi	M1	nop	M + M1
+im	M1	nop	M1 + M
+md	K	nop	M + K
+dm	K	nop	K + M
%ii	M1	M2	(M1 - M2) / M2 * 100
%id	M1	K	(M1 - K) / K * 100
%di	K	M2	(K - M2) / M2 * 100
%mi	M1	nop	(M - M1) / M1 * 100
%im	M1	nop	(M1 - M) / M * 100
%md	K	nop	(M - K) / K * 100
%dm	K	nop	(K - M) / M * 100

Table 5.9: Measurement operations

5.3.3.7 read / write

The two commands *read* and *write* can be used for any type of memory. These commands take three arguments, the *communication channel* (hardware interface e.g. UART, SPI, etc.), the memory name and the according data or pattern file which depends on the memory type.

```
write <comm channel> <mem name> <data>|<pattern label> [<variable>]
  # data: byte list (space separated)
  # e.g. 0x00 0x01 0x02
read <comm channel> <mem name> <data> <var letter> <variable>

# <comm channel>: SPI|UART|JLCC|SWI|3WI|...

# <mem name>: APB|SFR|XDATA|CODE|JLCC| ...

# <data>: hexadecimal raw data (e.g. FF 01 C8 )

# <pattern label>: label defined in [pattern] section

# <variable>: e.g. LV:R:var_name:1 (see section 'Variables')
```

NOTE: JLCC can be a communication channel as well as a memory type, the JLCC memory may also be written via CPU.

Examples:

```
  write SPI SFR B8 00
  write SPI ABP 10 00 WW WW VV VV GV:W:var1:2 GV:V:var2:2
  read UART ABP 0C 04 RR RR LV:R:psw:2

  write JLCC JLCC pattern_label   # label defined in [pattern] section
  read JLCC JLCC pattern_label    # the pattern file defines what to read
```

For each memory type the *data* has the following structure

```
<start address> <raw data>
```

where the size of the *start address* depends on the memory type and the underlying CPU addressing scheme (e.g. 8-bit SFR; 16-bit APB). The raw data needs to be according to the software protocol, which is product dependent. The format of the pattern file is described in Section 5.2.4.5 or may be an Intel HEX file [11].

5.3.3.8 force/release

The force command is used to set a pin to a specific value, where the value need to have the ext_setup definition format (e.g. FV 3.3V MAX_I 50mA). To recover the initial external setup the release statement with the corresponding pin list can be used.

```
force <pin> <value>[;<pin> <value>;...] [duration]
release <pin>[;<pin>;...]
```

Examples:

```
force UVDD FV 1.8V MAX_I 20mA
force RVDD FV 3.3V MAX_I 50mA ; UVDD FV 1.8V MAX_I 20mA 3.0s
release UVDD
```

5.3.3.9 Firmware Related Commands

For products with an embedded CPU it might be useful to implement a firmware framework for verification. The subsequent commands are defined to communicate with such a framework. The communication itself is not discussed here, since this is not project independent.

The *ping* command should be used to check if the DUT is 'alive'. A timeout or a trial count may be specified optionally. The result is stored into a variable.

```
cmd_ping <var_letter> <variable name> [<timeout>|<num trials>]
```

Examples:

```
cmd_ping RR GV:R:ping_var:1
cmd_ping RR GV:R:ping_var:1 10
```

The next four commands are used to call functions, where the first step is setting a function input, if required. Here the data is depending on the required function. The function can be called via its name, whereas the resolution from function name to function pointer needs to be done by a project dependent abstraction layer. It is required that functions can be stopped, which is necessary if the function just sends data in a loop as example. At last the function output can be stored into a variable.

```
cmd_set_func_input  <hex byte data>
cmd_get_func_output <var letter> <variable name>

cmd_call_function <function name>
cmd_stop_func
```

Examples:

```
cmd_set_func_input 03 00 01 01
cmd_get_func_output WW WW LV:W:out_var:2
cmd_call_fuction test_func
```

The DUT firmware framework status can be read via the *cmd_get_status_byte*, while the content of this byte is not defined in this document. Likewise, the OK and version data can be read, but are not defined here (project dependent).

```
cmd_get_status_byte <var letter> <variable name>
cmd_get_ok_byte     <var letter> <variable name>
cmd_get_version     <var letter> <variable name>
```

Examples:

```
cmd_get_status_byte WW LV:W:status:1
cmd_get_ok_byte VV LV:V:ok_var:1
cmd_get_version UU UU LV:U:version:2
```

5.3.3.10 write_result

Writing a result is either implicitly done within the measure or operation command or with the *write_result* command, which takes a variable name as optional arguments.

```
write_result [<var letter> <variable name>]
```

Examples:

```
  write_result # result of measurements, operation or equal
  write_result WW LV:W:result_var[4:12]:2
```

5.3.4 Variables

Within the test activities variables are possibly needed, which are defined as

```
GV:<letter>:<name>[<num>:<num>]:<bytes>
LV:<letter>:<name>[<num>:<num>]:<bytes>

<letter>: A capital letter, which is not used for hex representation
          logic states nor 'G' and 'L' (used for variable definition GV,LV).

<name>: Variable name. Valid characters: [a-z0-9_]

<bytes>: The length of the variable, which is not limited to one byte.

Optional:

[<num>:<num>]: The bits may define just a part of a variable.

E.g. [3:0] for the lower nipple of a byte.
```

Examples:

```
# read SFR register 0xB8 to RR (variable psw)
read SPI SFR B8 RR  LV:R:psw:1

#write variable psw[3:0] to result file
write_result R LV:R:psw[3:0]:1

# write global variable to APB register at address 0x0C00
write SPI APB 0C 00 TT TT TT TT GV:T:a_var:4
```

5.3.5 Grammar Definition

Since the sequence language defined above needs formal specification, which can be used for checking sequence files, a CFG was developed.

"A context-free grammar consists of terminals, nonterminals, a start symbol and productions." [5]

The terminals and nonterminals are not explicitly shown here. The start symbol is 'S'.

The subsequent tables 5.10, 5.11, 5.12, 5.13 and 5.14 show the CFG definition of the sequence language.

<table>
<tr><td>S</td><td>→</td><td>header_sec
info_sec
int_setup_sec
[tb_setup_sec]
[pattern_sec]
ext_setup_sec
sequence_sec
END</td></tr>
<tr><td>header_sec</td><td>→</td><td>HEADER_SEC_NAME
IP_NAME ID
TN T_NAME</td></tr>
<tr><td>info_sec</td><td>→</td><td>INFO_SEC_NAME</td></tr>
<tr><td>int_setup_sec</td><td>→</td><td>INT_SETUP_SEC_NAME
(LABEL [ID ID] 'FILE' NEW_LINE)+</td></tr>
<tr><td>tb_setup_sec</td><td>→</td><td>TB_SETUP_SEC_NAME
(LABEL [ID ID] 'FILE' NEW_LINE)+</td></tr>
<tr><td>pattern_sec</td><td>→</td><td>PATTERN_SEC_NAME
(LABEL 'FILE' NEW_LINE)+</td></tr>
<tr><td>ext_setup_sec</td><td>→</td><td>EXT_SETUP_SEC_NAME
LABEL</td></tr>
<tr><td>sequence_sec</td><td>→</td><td>SEQUENCE_SEC_NAME
(seq_command NEW_LINE)+</td></tr>
<tr><td>seq_command</td><td>→</td><td>reset
| int_setup
| tb_setup
| wait
| stimuli
| measure
| operation
| write
| read
| force
| release
| write_result
| fw_ping
| fw_set_func_in
| fw_get_func_out
| fw_call_func
| fw_stop_func
| fw_get_status
| fw_get_ok
| fw_get_version</td></tr>
</table>

Table 5.10: Sequence language context free grammar rule set

reset	→	CMD_RESET ID [EXT_SETUP_VALUE]
		(SEMI_COL ID EXT_SETUP_VALUE)* DURATION
int_setup	→	CMD_INT_SETUP LABEL
tb_setup	→	CMD_TB_SETUP LABEL
wait	→	CMD_WAIT (DURATION | ID COMPARE_OP
		((NUM SI_PREF QUANTITY)|HEX_NUM))
stimuli	→	CMD_STIMULI LABEL
measure	→	CMD_MEASURE ID (SEMI_COL ID)*
		SI_PREF QUANTITY QUALIFIER [VARIABLE] [DURATION]
operation	→	CMD_OPERATION (T_NAME|NUM) OPERATOR (T_NAME|NUM|NOP)
write	→	CMD_WRITE COM_CHANNEL MEM_NAME
		[RW_DATA] [LABEL] [VARIABLE+]
read	→	CMD_READ COM_CHANNEL MEM_NAME RW_DATA (VARIABLE)+
force	→	CMD_FORCE ID EXT_SETUP_VALUE
		(SEMI_COL ID EXT_SETUP_VALUE)* [DURATION]
release	→	CMD_RELEASE ID (SEMI_COL ID)*
write_result	→	CMD_WRITE_RES (RW_DATA)* (VARIABLE)*
fw_ping	→	CMD_FW_PING V_LETTER+ VARIABLE [(DURATION|DIGIT+)]
fw_set_func_in	→	CMD_FW_SET_FUNC_INPUT RW_DATA
fw_get_func_out	→	CMD_FW_GET_FUNC_OUTPUT (V_LETTER)+ (VARIABLE)+
fw_call_func	→	CMD_FW_CALL_FUNCTION ID
fw_stop_func	→	CMD_FW_STOP_FUNCTION
fw_get_status	→	CMD_FW_GET_STATUS_BYTE (V_LETTER)+ VARIABLE
fw_get_ok	→	CMD_FW_GET_OK_BYTE (V_LETTER)+ VARIABLE
fw_get_version	→	CMD_FW_GET_VERSION (V_LETTER)+ VARIABLE

Table 5.11: Sequence language command rule set

HEADER_SEC_NAME	→	'[*header*]' NEW_LINE
INFO_SEC_NAME	→	'[*info*]' NEW_LINE
INT_SETUP_SEC_NAME	→	'[*int_setup*]' NEW_LINE
TB_SETUP_SEC_NAME	→	'[*tb_setup*]' NEW_LINE
PATTERN_SEC_NAME	→	'[*pattern*]' NEW_LINE
EXT_SETUP_SEC_NAME	→	'[*ext_setup*]' NEW_LINE
SEQUENCE_SEC_NAME	→	'[*sequence*]' NEW_LINE

Table 5.12: Sequence language header terminals

CMD_RESET	→	’*reset*’
CMD_WAIT	→	’*wait*’
CMD_INT_SETUP	→	’*int_setup*’
CMD_TB_SETUP	→	’*tb_setup*’
CMD_STIMULI	→	’*stimuli*’
CMD_MEASURE	→	’*measure*’
CMD_OPERATION	→	’*operation*’
CMD_READ	→	’*read*’
CMD_WRITE	→	’*write*’
CMD_FORCE	→	’*force*’
CMD_RELEASE	→	’*release*’
CMD_WRITE_RES	→	’*write_result*’
CMD_FW_PING	→	’*cmd_ping*’
CMD_FW_SET_FUNC_INPUT	→	’*cmd_set_func_input*’
CMD_FW_GET_FUNC_OUTPUT	→	’*cmd_get_func_output*’
CMD_FW_CALL_FUNCTION	→	’*cmd_call_function*’
CMD_FW_STOP_FUNCTION	→	’*cmd_stop_function*’
CMD_FW_GET_STATUS_BYTE	→	’*cmd_get_status_byte*’
CMD_FW_GET_OK_BYTE	→	’*cmd_get_ok_byte*’
CMD_FW_GET_VERSION	→	’*cmd_get_version*’

Table 5.13: Sequence language command terminals

ID	→	’$[a-zA-Z][a-zA-Z0-9_]+(<[0-9]+>)?$’
IP_NAME	→	’$IP\backslash sName:$’
TN	→	’$Test\backslash sName:$’
T_NAME	→	’$[0-9][0-9][0-9][0-9][0-9][A-Z][a-z_]+$’
FILE	→	’$[a-zA-Z0-9_\backslash/:.]+$’
LABEL	→	’$[a-z][a-zA-Z0-9_]+$’
NEW_LINE	→	’$\backslash r?\backslash n$’
SEMI_COL	→	’;’
NOP	→	’*nop*’
END:	→	’$\$$’
DIGIT	→	’$[0-9]$’
HEX_NUM	→	’$0x[0-9A-Fa-f]+$’
COM_CHANNEL	→	’$[a-zA-Z][a-zA-Z0-9_]+$’
MEM_NAME	→	’$[a-zA-Z][a-zA-Z0-9_]+$’
SI_PREF	→	’$[PTGMkmunpfa]?$’
QUALIFIER	→	’$(pp?\vert dc\vert rms)?(_d)?$’
QUANTITY	→	’$VA?\vert A\vert W\vert Hz\vert Ohm\vert F\vert H\vert dbc/Hz$’
NUM	→	’$[+-]?[0-9]+(.[0-9]+)?(\backslash s*[eE][+-]?[0-9]+)?$’
COMPARE_OP	→	’$(>=?\vert <=?\vert =)$’
DURATION	→	$NUM\ SI_PREF\ '\backslash s+s'$
FV	→	$'FV\backslash s+'\ NUM'\backslash s+MAX_I\backslash s+'\ NUM$
FI	→	$'FI\backslash s+'\ NUM'\backslash s+MAX_V\backslash s+'\ NUM$
CONN_TO	→	$('LC'\vert 'LR'\vert 'LL')'\backslash s+'\ NUM'\backslash s+CONN_TO\backslash s+'\ NUM$
EXT_SETUP_VALUE	→	$'NC'\vert FV\vert FI\vert CONN_TO\vert 'DCH'\vert ('ARB'\vert 'SINE')'\backslash s+'\ ID$
VARIABLE	→	’$[GL]V':V_LETTER:ID('['DIGIT+:DIGIT+']')?:DIGIT+$’
V_LETTER	→	’$[H-KM-SU-Z]$’
RW_DATA	→	’$((HEX_NUM\ HEX_NUM])\vert (V_LETTER\ V_LETTER))$ $(\backslash s+((HEX_NUM\ HEX_NUM)\vert (V_LETTER\ V_LETTER)))*$’
OPERATOR	→	’$((+\vert -\vert *\vert \%\vert /)(i[idm]\vert m[id]\vert d[im]))\vert (//(mi[id]\vert imd\vert iim))$’

Table 5.14: Sequence language general terminals

5.4 Verification Automation Concept

Requirements of the generic verification platform are *reproducibility* and *automation*. Therefore a detailed concept for automation was developed which guarantees implicitly that measurements and simulations are reproducible. Furthermore the Test-spec data is checked to be correct which decreases debug time at production test development. Figure 5.23 shows the concept which is described subsequently.

Figure 5.23: Automation concept - see Appendix A

The components *<env>_plan.xls*, *test spec* and *tests* repository are the common data storage for all kinds of verification throughout a project. Content and structure of these components can be found in Section 5.2.

User Interface The user of the verification automation software should not have to care about test data, measurement software or suchlike. Therefore a user only needs to load a verification plan (*<env>_plan.xls* see Section 5.2.3) and select tests which need to be executed. Additional data like environmental sweeps (temperature, humidity, etc.) or other parameter sweeps (e.g. power supply sweep) may be specified. Environmental sweeps may be done by automation software, but this can not be guaranteed especially for simulation. Parametric sweeps need to be implemented in test software anyway, since this parameters are product dependent. This implies that specified sweeps may be ignored if they are not implemented. The selected tests

and appropriate data is passed to *Verification Platform Framework* (Mark (1) in Figure 5.23), where the test name and the link to measurement software are obligatory items.

Verification Platform Framework The main tasks of the framework are collecting and converting data out of the common storage into appropriate formats on one hand and automation of test execution on the other. For simulation the task is mostly automation, which is done by sequentially calling external software with options, where the software link is passed from *User Interface*. Since simulation is finished before the *test-spec* is created no data needs to be read from it.

For evaluation, tests and according data need to be passed to evaluation automation software. The interface defined between the verification platform framework and the evaluation automation (2) is described in Section 5.4.1.1 in detail.

The optional *Project - Device setup* GUI needs to be created for each project and can be used for debug and bring-up purposes. This GUI is not related to the verification automation concept.

Simulation automation For digital as well as analogue simulation, scripts need to be implemented which are callable from *Verification Platform Framework*. This is Csh on the digital side and ocean script for analogue simulation. These scripts are not discussed in this document, because they are project dependent (test implementation).

Evaluation automation The *evaluation automation software* takes as input the *XML interface*, which includes all test data required for evaluation. Based on this data the *measurement handler* sequentially executes all tests. The second input for software, is the *station setup*, which describes the available measurement equipment (see Section 5.4.1.2) and its connections.

Environmental condition is project independent and therefore controlled by the *measurement handler*.

Device setup needs to be implemented once for each project. *Device setup* includes DUT internal setup as well as TB setup in software and hardware protocols.

Tests For each test an implementation has to be created using any programming language as desired. The implementation needs to be callable by the *measurement handler*. The implementation needs to include the test activities (described sequence file) and the result formatting. Optionally parametric sweeps may be implemented here, which can be specified in the *eval_plan* or by the user.

Hardware layer hides measurement equipment from test implementation, which should gain a high degree of exchangeability of equipment where possible.

Result and setup log All results need to be stored in the results log and setup log (results directory structure see Section 5.2.5 and Figure 5.3).

5.4.1 Verification Platform Framework

5.4.1.1 XML Interface Specification

The XML interface contains all test data needed for evaluation. Since this XML interface needs to be checkable against a XML Schema Definition (XSD), created data needs to be a valid XML document. Therefore the interface content is embedded in the subsequent tag.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lot_run>
  .
  .
  .
</lot_run>
```

Inside this *lot_run* block the structure includes the following items.

- Run information
- Environmental conditions
- Measurement(s)

The XML interface needs to have one run information and one environmental condition element, but there is no limitation for measurements.

Run information The *run_info* XML element is used for documentation purposes as well as for common information like base paths. This is required for reproducibility of measurement results. Parts of this information are also stored in the measurement result files.

```
<run_info>
    <lot_id>xxx.yyyy</lot_id>
    <time_stamp>YYYYMMDDhhmmss</time_stamp>
    <responsible>xyz</responsible>

    <board>
        <name>eval_board</name>
        <version>v1.0</version>
        <fpga_name>eval_fpga</fpga_name>
        <fpga_version>v1.0</fpga_version>
    </board>

    <meas_setup>meas_setup_1</meas_setup>
    <sw_link_root> path to measurements </sw_link_root>

    <test_spec>
        <path> path to test spec </path>
        <version> path to test spec </version>
    </test_spec>

    <firmware_framework>
        <name>gvp_fw</name>
        <version>v1.0.0</version>
    </firmware_framework>
</run_info>
```

All elements except *firmware_framework* (not all products have an embedded processor) are mandatory within the run_info section.

Environmental conditions This section describes the conditions all tests need to run through. Within this context environmental conditions are air temperature, air pressure, humidity. These conditions are swept through all values of a specified list or an interval.

```
<env_cond_sweep>
    <sweep id="0">
        <name>Temperature</name>
        <type>list</type>
        <values type="t" unit="C">-40;25;85</values>
    </sweep>
    <sweep id="1">
        <name>Temperature</name>
        <type>interval</type>
        <values type="t" unit="C">-40;85;5</values>
    </sweep>
</env_cond_sweep>
```

The first element shows a list of temperatures, the second element specifies the identical temperature range but a step size of five degree.

Measurement section The measurement element contains the information for a single test. The *measurement* tag has an attribute called *name*, which needs to be equal to the test name defined in the *DV Matrix* (see Section 5.2.2.1). The measurement element has a *device setup* (optional) and a *meas_data* element, which are defined in detail below.

```
<measurement name="00010V_sup">
    <device_setup>
        .
        .
        .
    </device_setup>

    <meas_data>
        .
        .
        .
    </meas_data>
</measurement>
```

Device setup The device setup includes the initial setup for a test, which includes the setup of the DUT (internal setup - int_setup) as well as the configuration of the measurement TB. The *device_setup* are either streams or registers (embedded processor registers).

```
<device_setup>
    <streams>
        <JLCC>
            <chain name="chain_name1__00010-0">
                <bit_stream>000000010110100010010000</bit_stream>
                <mux>011</mux>
```

```
            </chain>
            <chain name="chain_name2__00010-1">
                <bit_stream>0000000000100</bit_stream>
                <mux>110;0110</mux>
            </chain>
        </JLCC>
    </streams>

    <registers>
        <SFR>
            <reg>
                <address>0x8B</address>
                <value>0x00</value>
            </reg>
        </SFR>
    </register>
</device_setup>
```

Streams are separated by the type of configuration interface (e.g. JLCC). Within the type element chains are defined with the name as attribute. The *bit_stream* defines the data, where the left most bit is sent first to the corresponding device. The *mux* element defines the setup for possibly used stream multiplexer(s).

Registers are simply specified by an address/value pair and also differentiated by the type of register (e.g. SFR, APB).

The order from top to bottom of device setup defines the sequence of the configuration. In the example above streams are processed before registers and 'chain_name1__00010-0' before 'chain_name2__00010-1'.

Measurement data Within this XML element the *sw_link* element is mandatory, which is the path to the measurement software relative to *sw_link_root*. This measurement software is the implementation of the test (described in *sequence file*) and may not need additional information.

Element	Description
pins_to_measure	DUT pins that need to be measured.
ext_setup	Setup at DUT boundary (see Section 5.2.4.3). The attributes 'name' and 'num' of the 'pin' element identify the pin at DUT boundary. The 'type' attribute specifies the kind of pin (e.g. FV - force voltage), where the values and limitations are specified by value tags.
int_setup	Possibly the internal setup changes within the test implementation (for example a parameter sweep of a CPU register) This internal setup could be referenced by the implementation via the attribute 'id'.
sweep	Within a single test it is possible to implement external or internal setup sweeps (e.g. supply voltage, register values, etc.). These are defined within this element, where either a list or interval is specified equal to environmental conditions (see example below).
operation	A test may consist of an operation on the result of other tests only (e.g. differential voltages). Possible operations are specified in Table 5.9. 'op1' and 'op2' need to be valid test names (defined in *DV matrix*).
stimuli_ref	This element includes needed stimuli pattern file paths, which are referenced by the 'name' attribute.

Table 5.15: Measurement elements description

```
<meas_data>
    <sw_link>'path to test implementation'</sw_link>

    <pins_to_measure>
        <pin>gpio1_5</pin>
    </pins_to_measure>

    <ext_setup name="ext_setup__A">
        <pin name="vdd" type="FV">
            <value type="FV" unit="V">3.3</value>
            <value type="MAX_I" unit="mA">50</value>
        </pin>
        <pin name="iovdd" type="FV">
            <value type="FV" unit="V">3.3</value>
            <value type="MAX_I" unit="mA">50</value>
        </pin>
    </ext_setup>

    <int_setup id="0"> <!-- id is running number -->
      .
      .
      .
```

```
    </int_setup>

    <sweep type="int" id="0">
      <int_setup id="0">
        <registers>
          <SFR>
            <reg>
              <address>0x8B</address>
              <type>list</type>
              <value>0x00; 0x01; 0x02; 0x04; 0x08 </value>
            </reg>
          </SFR>
        </registers>
      </int_setup>
    </sweep>

    <operation>
        <opcode>-ii</opcode>
        <op1>00011V_sup_n</op1>
        <op2>00012V_sup_p</op2>
    </operation>

    <stimuli_ref>
        <stimuli name="Stimuli__00010-0">
            <path>'path to stimuli file'</path>
        </stimuli>
    </stimuli_ref>
</meas_data>
```

5.4.1.2 XML Station Setup

Similar to the XML interface the station setup also needs to be a valid XML file to be XSD checkable. Therefore all elements are embedded in following the body.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<setup version="1.0">
 .
 .
 .
</setup>
```

Station The stations are differentiated by the type of their connection to the measurement host. A station has slots which represent measurement equipment. A test implementation (measurement software) may reference the described measurement equipment by the alias specified in the XML slot element (e.g. Timing01). The 'type' tag needs to be the the real name of the measurement equipment, which could be read out by software or from the equipment display. Below examples for specified stations can be found.

```
<station type="PXI" id="LB02-PXI-002">
  <slot id="2" type="TIMING">
    <required>no</required>
    <name>PXI1Slot2</name>
    <alias>Timing01</alias>
    <type>NI-5122</type>
```

```
  </slot>

  <slot id="3" type="AWG">
    <required>False</required>
    <name>PXI1Slot3</name>
    <alias>AWG01</alias>
    <type>NI-5421</type>
  </slot>
</station>
```

A PXI station may have an additional attribute 'id' which identifies the equipment uniquely, this is foreseen for future purposes to reduce to one station setup file (for all products), where just the 'required' tag needs to be changed.

```
<station type="GPIB">
  <slot id="07" type="Power Supply">
    <required>yes</required>
    <name>PS01</name>
    <alias>PS01</alias>
    <type>Agilent E3631A</type>
  </slot>
  <slot id="9" type="THC">
    <required>yes</required>
    <name>THC01</name>
    <alias>THC01</alias>
    <type>TP 04310</type>
  </slot>
</station>

<station type="USB">
  <slot id="01" type="FTDI">
    <required>yes</required>
    <name>FTP0E1B9A</name>
    <alias>SPI</alias>
    <type>FT2032LM</type>
  </slot>
  <slot id="02" type="FTDI">
    <required>yes</required>
    <name>FTP0DKR5A</name>
    <alias>JLCC</alias>
    <type>FT2032LM</type>
  </slot>
</station>
```

Chapter 6

Results and Evaluation

The introduction of a generic verification platform offers improvements and opportunities, but also implicates risks. This section provides an overview of achievements, drawbacks and risks of the generic verification platform. Furthermore the impact to SDs IC verification process is discussed.

6.1 Improvements

The major improvements by introducing the generic verification platform are the tremendous increase of software reuse, the enhancement of verification data quality and the high increase of efficiency which saves time and costs.

In the course of the status quo analysis nearly no reuse of evaluation software could be observed. With the implementation of the *Verification Automation Concept* (see Section 5.4) the reuse can be increased by a multiple. A concrete percentage rate can not be provided, since the analysis of status quo has not been done in that high detail for evaluation software reuse. Furthermore, there is not enough empirical data about verification software reuse of the provided *Verification Automation Concept*. A high reuse implies also an improvement of software quality, performance and functionality, since it will be used within different projects with individual requirements. Verification software like laboratory equipment drivers, will be reviewed by engineers which may uncovers errors. The observed multiple implementations of equal software can also be reduced to a minimum.

The considerable increase of reuse is also supported by the defined data structures and file formats. Supporting software like report generation or results analysis is possibly implemented in the future, which may be used by many projects.

Because of the common data storage, verification data is less error-prone. Equal input data is used for all verification environments, which reduces debug time and improves quality of verification data. With the specification of data formats and structures, two of the major requirements, *comparability* and *transferability*, can be provided. Furthermore, the verification data structure and formats improve efficiency of engineers at the specification of TCs as well as for the search for previous results. The common use of the generic verification platform may

also improves the quality of project modules (IPs) by the enhancement of data quality and the reuse of the data within different project. Furthermore different requirements to modules may improve the module itself as well as its documentation.

The defined DV Matrix provides a summary of the used verification data and results. Such a document has not been used at all within the previous verification process. The use of the DV Matrix offers an overview of the verification process and makes the management easier for project leaders. It may also be used for the acceptance procedure with customers.

6.2 Risks

As already mentioned above the major achievement of the specified structures and formats is the *transferability* of verification data, which may also involves some risks. If the equal verification data is used by all verification environments, equal results may be produced. These equal results do not guarantee that the executed test is correct and proves the achievement of customer requirements as example. It is likely that the quality of test data is increased, but this does not imply that the quality of the DUT can be increased. Since all verification environments use the equal data, probably less errors of the DUT are uncovered. Therefore it is essential that the saved time of test data debugging is used to check transfered data against description and requirements. These checks may be done quicker than the debugging, which will save costs. Existing verification software will need adoption to the specified data structures and formats, which may cause debug time.

The use of automatic software may lead engineers to loose their sense of responsibility for executed tests. By the high reuse of verification software the feeling of responsibility will decrease over time and may also end in a loss of verification accuracy. Engineers may get a feeling of "That'll go off all right" if verification software has already been used within other projects. But the reused software may not cover the requirements of the currently needed verification software.

The possibly loss of product quality and the decreasing sense of responsibility of engineers needs to be managed by project team leaders or group leaders.

6.3 Feedback

The first implementation of the *Verification Automation Concept* promises a high increase in efficiency in evaluation test execution. This can be enhanced by an improvement of the implementation performance. Because of the hierarchical design, which provides a very generic usage, the performance does not satisfy engineers.

The DV Matrix has been used for two new projects and became a SD standard document. The DV Matrix has been inspected by SDs quality and product engineering departments who decided to accept the DV Matrix as standard document for all new projects. Furthermore a very promising feedback of users can be observed.

The specified data structures and file formats were accepted by the users, because of the promising increase of efficiency and reuse of verification software.

6.4 Impact

The use of the generic verification platform has many advantages. The major requirements are fulfilled which are *comparability, reproducibility, transferability* and *automation*. This is a tremendous improvement to the previous verification process. Since only parts of the verification platform are used up to now, the overall impact to SDs IC verification can not be estimated. To give a concrete percentage rate at least five projects need to implement the specified verification platform and use and improve the created software. With the empirical data of the implemented projects, the weaknesses of the generic verification platform may be uncovered and resolved. Up to now, no concrete issues have been uncovered. Possible drawbacks and risks of the verification platform are discussed in Section 6.2.

Chapter 7

Conclusion

The definition of a generic verification platform has been successfully completed. The verification platform provides the required generic, well-defined verification approach. The defined structures and file formats guarantee the required *reproducibility*, *transferability* and *comparability*. This generic approach will lead to a high degree of reusability of verification software. The developed *automation* concept provides an efficient, time and cost saving approach to verification.

SD has implemented the general specification of the generic verification platform for two new LF communication products. For these projects the full directory structure has been implemented as well as the design verification matrix (*DV matrix*). The matrix has only been filled up to the verification parameter definition, since the design is currently in production. For this reason no conclusion about parameter correlation can be done.

The automation concept is implemented in LabVIEW™ for an existing project SD340 (multichannel RF transceiver), where the concept has been inspected and satisfies the requirements. Therefore the automation framework is also used for the two new LF communication products. The considerable effort put into the definition of the generic verification platform has turned out to pay off, since a very high reuse can be observed within the three projects. The quality of the *Test-spec* spreadsheet has been improved by rechecking it via the laboratory automation, which was very valuable for SD340 production test issues solving.

The future work for this project is on one hand the development of supportive tools like structure checking scripts, a report generator and software for increasing usability and user acceptance. To raise the definition of the generic verification platform to a higher level of maturity, the implementation of IEEE standard 1671 (ATML) may be considered which will increase the detailedness of test data if this is needed. The use of the generic verification platform should be mandatory for all new projects, since this will increase reuse and efficiency considerably. It is essential, that the use of a standardized verification platform does not release engineers from their responsibility of an accurate and faithful verification (test definition) of their design and evaluation.

Bibliography

[1] IEEE Standards Coordinating Committee 20. IEEE standard for automatic test markup language (ATML) for exchanging automatic test equipment and test information via XML. Technical report, IEEE, 3 Park Avenue, New York, NY 10016-5997, USA, 2009.

[2] Mellik A. Automated XML-based test modelling for mixed-signal circuits. Technical report, Department of Electronics, TTU, Ehitajate tee 5, 19086 Tallinn, Estonia, 2006.

[3] SensorDynamics AG. QM25 - device evaluation and golden sample preparation. 2009.

[4] SensorDynamics AG. Project management handbook. 2010.

[5] Alfred V. Aho, Monica Lam, Ravi Sethi, and Ullman Jeffrey D. *Compilers - Principles, Techniques & Tools*. Pearson Education, Inc., second edition, 2007.

[6] Alfredo Benso, Stefano Di Carlo, Paolo Prinetto, and Yervant Zorian. A hierarchical infrastructure for SoC test management. In *IEEE Design & Test of Computers*, pages 32–39, 2003.

[7] Kurt Binder and Dieter W. Heermann. *Monte Carlo Simulation in Statistical Physics*. Springer-Verlag Berlin Heidelberg New York, 4th edition, 2002.

[8] Cadence. Virtuoso specification driven environment. http://www.cadence.com/rl/Resources/datasheets/VirtuosoSpecDriven_ds.pdf, February 2010.

[9] Matthew H. Cahn and Mark F. Russo. Python and automated laboratory system control. Technical report, The Association for Laboratory Automation, 2007.

[10] Eugene Charniak. Parsing with context-free grammars and word statistics. Technical report, Department of Computer Science, Brown University, Rhode Island, 1995.

[11] Intel Corporation. *Hexadecimal Object File Format Specification*, A edition, 1988.

[12] Automotive Electronics Council. *AEC - Q003 Guidelines for Characterizing the Electrical Performance of Integrated Circuit Products*, July 2001.

[13] Automotive Electronics Council. *AEC - Q100-009 Electrical Distribution Assessment*, B edition, August 2007.

[14] Jonathan David. Efficient functional verification for mixed signal IP. In *Behavioral Modeling and Simulation Conference*, pages 53 – 58, 2004.

[15] Hiren D. Desai. Test case management system (TCMS). Technical report, Research Engineer, Science and Technology, BellSouth Telecommunications Inc, Atlanta, GA, 1994.

[16] J. Edvardsson. A survey on automatic test data generation. In *Proceedings of the 2nd Conference on Computer Science and Engineering*, pages 21–28, 1999.

[17] Chance Elliott, Vipin Vijayakumar, Wesley Zink, and Richard Hansen. National Instruments LabVIEW: A programming environment for laboratory automation and measurement. Technical report, The Association for Laboratory Automation, 2007.

[18] Gregor Erbach. *Tools for Grammar Engineering*. http://acl.ldc.upenn.edu/A/A92/A92-1039.pdf, April 2010.

[19] Joseph A. Goguen and Charlotte Linde. Techniques for requirements elicitation, 1993.

[20] Chris Gorringe, Teresa Lopes, and Dan Pleasant. ATML capabilities explained. Technical report, EADS Test & Services, Teradyne Inc., Agilent Technologies Inc., 2007.

[21] Grid-Tools. How do grid-tools use data creation and data generation techniques? http://www.grid-tools.com/solutions/test_data_creation.php, March 2010.

[22] D. Grune and Ceriel J.H. Jacobs. *Parsing techniques a practical guide*. Ellis Horwood Limited http://citeseer.ist.psu.edu/grune90parsing.html, Chichester, England, 1990.

[23] IEEE. IEEE guide to software requirements specification, 1993. Std 830-1993.

[24] National Instruments. *Introducing NI LabVIEW 2009*. http://www.ni.com/labview/, November 2009.

[25] National Instruments. What is NI TestStand. http://zone.ni.com/devzone/cda/tut/p/id/6073, March 2010.

[26] Samuel Kotz and Norman L. Johnson. *Process Capability Indices*. Chapman & Hall, first edition, 1993.

[27] Donn Jr. Le Vie. Writing software requirements specification. http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html, 2009.

[28] P. Lu, D. Glaser, G. Uygur, S. Weichslgartner, K. Helmreich, and A. Lechner. Mixed-Signal Test Development using Open Standard Modeling and Description Languages. Technical report, Friedrich-Alexander-University Erlangen-Nuremberg, Paul-Gordan-Str 5, 91052 Erlangen, Germany.

[29] Ralf Lumel. *Grammar Testing*. Springer Berlin/Heidelberg, 2001.

[30] MathWorks. Systemtest 2.5. http://www.mathworks.com/products/systemtest/, January 2010.

[31] Ian McSweeney and Aristides Garces. Software requirements specifications document, 2004.

[32] MentorGraphics. IC verification and signoff using calibre. http://www.mentor.com/products/ic_nanometer_design/verification-signoff/, January 2010.

[33] Sylnovie Merchant. The pieces problem-solving framework and checklist. Technical report, Department of Management Information Science. California State University, Sacramento, http://www.csus.edu/indiv/m/merchants/pieces.pdf, December 2009.

[34] Christopher Z. Mooney. *Monte carlo simulation.* Sage publications, Inc., Thousand Oaks, California, 116th edition, 1997.

[35] S. Raghavan, G. Zelesnik, and G. Ford. Lecture notes on requirements elicitation. *Educational Materials. CMU/SEI-94-EM-10. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA*, 15213, 1994.

[36] Mike Seavey and Tamara Einspanjer. ATML: What "it" is, what "it" is not, and an example of how "it" can be applied. Technical report, Northrop Grumman Corporation, Defensive Systems Division, 600 Hicks Road, Rolling Meadows, Illinois 60008-1098, 2005.

[37] SiliconFarEast.com. Gauge repeatability and reproducibility (GR&R). http://www.siliconfareast.com/grr.htm, April 2010.

[38] Christophe Strobbe. Test Case Description Language 2.0: Specification and guide. Katholieke Universiteit Leuven, http://www.bentoweb.org/refs/TCDL2.0.html, March 2010.

[39] Synopsis. Discovery verification platform. http://www.synopsys.com/Solutions/EndSolutions/DiscoveryVerification/Pages/default.aspx, January 2010.

[40] Cadence Design Systems. *Virtuoso Specification-driven Envrionment User Guide*, January 2006.

[41] Gordon F. Taylor and Steven M. Blumenau. A pragmatic test data management system. In *International test conference*, pages 338–344, 300 Baker Ave, Concord MA 01742-2174, USA, 1991.

[42] VI Technology. Enterprise your test. http://www.vi-tech.com/solutions/enterprisetestsolutions/default.aspx, March 2010.

[43] Optimal Test. Test management solutions. http://www.optimaltest.com/, February 2010.

[44] Kevin Thompson and Ladd Williamson. Hardware verification with the unified modeling language and vera. Technical report, Cypress Semiconductor, 2008.

[45] Jane Wood and Denise Silver. *Joint Application Development.* Wiley, New York, 1995. ISBN 0471042994.

Appendix A

Verification Automation Concept

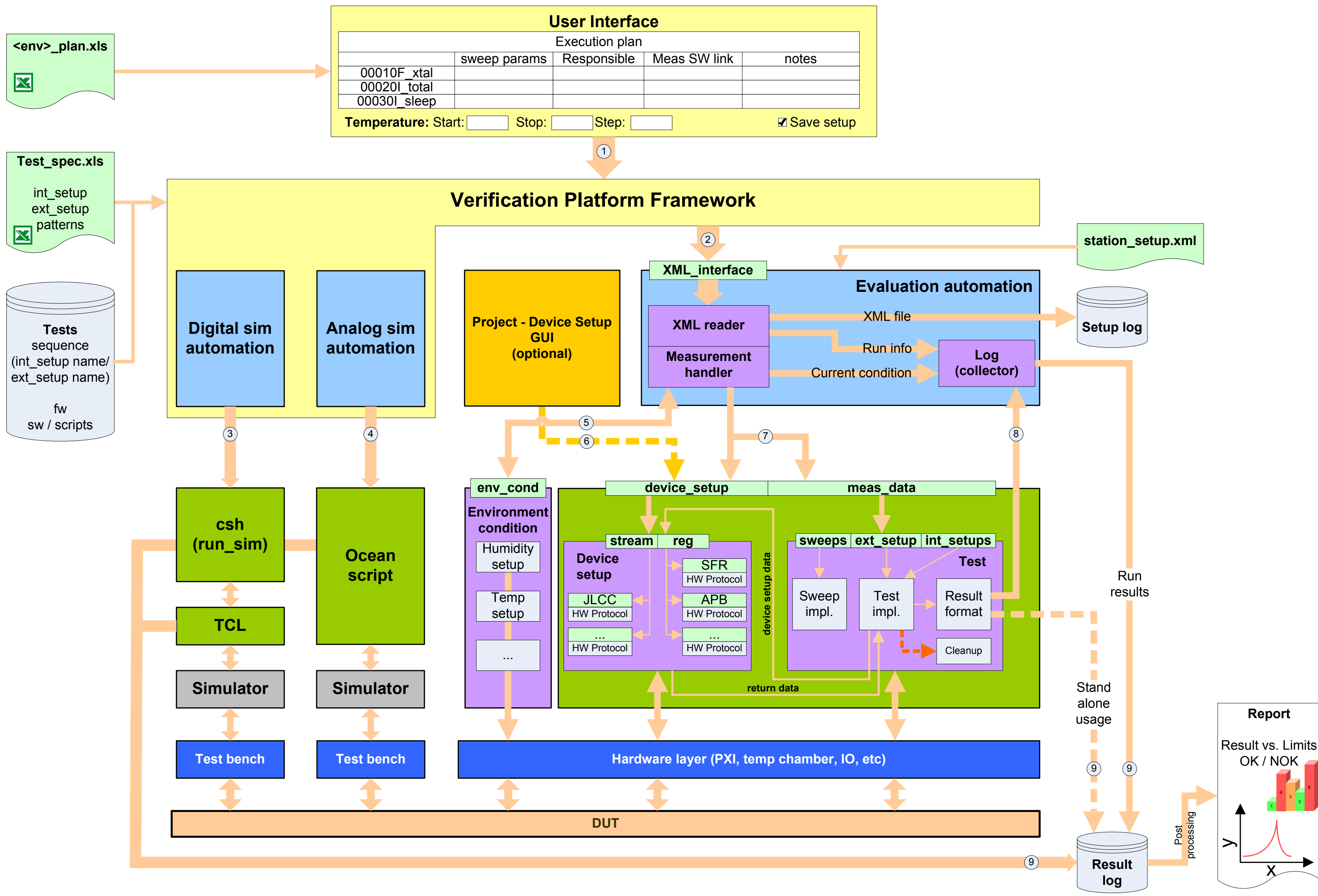


Figure A.1: Automation concept