NTNU
Det skapende universitet

Automated verification flow using Jenkins build server

Juan Camilo Parra Diaz

Embedded Computing Systems
Innlevert: juni 2016
Hovedveileder: Kjetil Svarstad, IET

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Automated verification flow using Jenkins build server

Juan Camilo Parra Diaz

Electronic and Telecommunication Institute

Norwegian University of Science and Technology

A thesis submitted for the degree of

Master of Science in Embedded Computing Systems

2016 June

1. Reviewer: Kjetil Svarstad

2. Co-Reviewer: Christoffer Amlo

Abstract

Context: An automated verification flow add several advantages, specially regarding the coverage closure evaluation, analysis of failures and re run of failed tests in debug mode, among others. When this automated verification process is performed in combination with a continuous integration practice (as in this project by using Jenkins build server) then the process gets enhanced from the capabilities of Ci practice and automated regression management systems like VRM.

Objectives The main aim of this process is to do an exploratory research of VRM and continuous integration practice in order to explore possible flows that gives facilities and features to an automated verification flow

Methods exploratory implementation, with its analysis and literature review. The literature review focuses on several aspects that this project faces (Continuous integration, Verification run manager and QME as principal topics). The exploratory implementation explores the possible improvements that VRM can bring to a CI practice and as special case QME of Mentor graphics that already has CI and VRM in its flow. These qualitative data is gathered trough observations of the tools involed in the verification process (VRM and jenkins build server) with its inductive contextual analysis (1)

Results from the exploratory implementation and its analysis, the integration of the CI practice and VRM brings to the flow the advantages that vrm offers (specially the verification metrics closure).

Conclusions The main conclusions is regarding the integration of the CI practice (under Jenkins build server) and VRM (Verification Run manager). In fact if both technologies are implemented in the same flow, the flow gets

the advantages of both, that gives as main characteristic, a metric driven methodology for coverage closure evaluation under a regular basis.

Acknowledgements

I would like to acknowledge professor Kjetil and Christoffer for their guidance trough the project and comments on the report

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

1

Introduction

It is important to integrate in the verification flow tools that allow the automation of regression testing systems since the duties of verification engineers will be focused on verification tasks rather than in maintenance duties. These improvements in the flow are given under the automation facilities that VRM (Verification Run Manager) brings to the process.

From the previous semester project, a linked approach was implemented as is depicted in the figure 1.1 in where an automated coverage analysis was performed.

This process is iterative as described below:

- First, when a bug is identified, the design is updated.
- Second, the verification environment could not exercise the entire design (maybe in the interested states), the verification requirements are not reflected in the verification environment. Therefore the test benches have to be updated, modified, refined or created.

Up to this point the linking approach ((6)) was an improvement that helps mainly in creating a clear tracking and traceability of requirements to the verification environment from last semester project (6).

However, there are several topics that have to be studied and explored in order to evaluate the possible higher automation flows with its respective advantages and disadvantages and this is the context under which this master thesis project is developed.

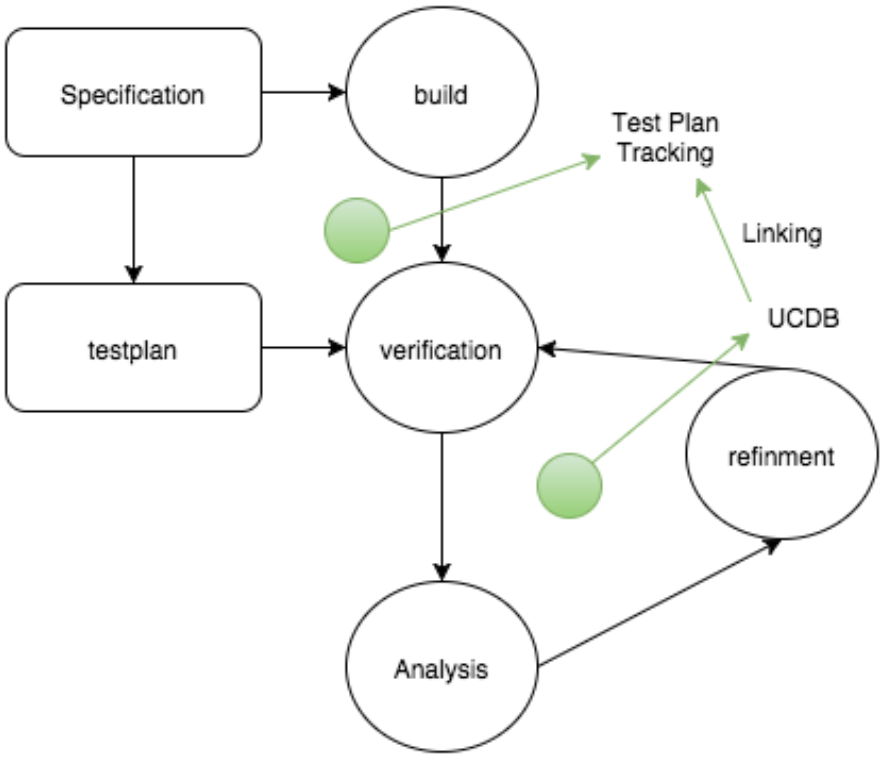


Figure 1.1: Verification Management flow with the implemented linking approach (6)

1. The linked approach is still done manually, therefore how can be this process fully automated

2. How can be this process integrated in a continuous integration (CI) approach (See section 2.1 for more information about Continuous integration). This is one the most relevant aspects of this project since several options were implemented in an exploratory way (see section 3.2 for more information of the possible options that were implemented). An important fact in this project is regarding to QME (Questa Makefile Environment 2.4) that takes advantages of the Continuous Integration and the Verification metrics as is described in section 2.1. In this sense, QME is strongly related to this project. However QME (Questa Makefile Environment as a regression management system) is not an end by itself but rather it has been used for the analysis that the automation capabilities brings to the flow when the verification is done under a Continuous Integration practice.

3. How triage reporting (automated result analysis) and debug capabilities can be

applied to an automated flow and what are the advantages it brings to the flow under an automated approach?

Figure 1.2 shows the general scope of the given problem that is faced. The upper part of the diagram shows what has been done through the linked based approach process (under the number 1 circle). Here it shows the testplan and the UCDB simulations that gets into the process. Then, although the extraction of coverage information is done automatically from the test plan, the steps of merging, report generation and if needed the process of testplan conversion to UCDB file format is done manually. In the circles that are under the number 2 is when the automation, integration (with the respective integration options) are faced (which is based on the automation features and capabilities for regression management systems that VRM supports) . In fact the process can be implemented using tools offered by Questa, which in this case is VRM (Verification Run Manager see section 2.3.3 for more information on VRM)

At the lower level there is the process of Continuous integration using Jenkins. The problem here stands on how this Continuous Integration process can be integrated to the linked based approach in such a way that once the regression suit pass, then get the coverage verification data, generate report for failing test (trend analysis see section 2.3.3.3) and if needed start a debug process for the failed tests.

Finally is the report generation depicted under the number 3. Since the report is generated in html files already, then research is performed in order to identify how this report can be shown in an easy way in the development flow in order to force a process that continuously evaluates coverage closure. From the given options for process improvement (see section 3.2) this html-verification report (with coverage data, trend analysis and triage reporting) is given under a directory tree while by using QME (See section 2.4 for more information about QME) the verification-report is besides updated to the Jenkins console.

With respect to the verification-report generation options using Jenkins, some options were considered also in order to be able to set coverage metrics in the report that jenkins sets as default in its console. In fact there are several plugins that could be implemented in order to achieve this goal. However since QME uses a plugin that integrates coverage reporting, then this projects goes further in the implementation that QME supports and do a further analysis. This analysis is done in section 2.4.4

1.1 Contributions

The contributions of this master project is based on the integration of VRM into Jenkins and how the features of VRM gives a higher order of automation to the verification process. This contributions are strongly related to the possible options of integration of Jenkins and VRM as shown in section 3 with its respective analysis and conclusions made from it. Also the triage and debug capabilities that can be potentially integrated to the flow. In this sense it is important to highlight the relevance that QME[1] brings to the flow since it already implements vrm and jenkins. The relevant analysis of QME is therefore made in section 6.3

These contributions are given under the possible implementation options for the integration of the CI practice (with Jenkins) with VRM (as a tool that support also regression management systems). From these exploratory study, the integration is valid and actually enhance the process of verification since, among other features, allows to have a Metric driven Verification (MDV) for a evaluation of coverage closure on regular basis. This main contribution is re-validated with the analysis and exploratory implementation of QME (that already has the integration of VRM and jenkins). The second contributions extends the analysis over features that VRM brings to the flow, regarding triage analysis and debug capabilities (e.g for re-run of failed tests).

1.2 Summary

This Thesis project focuses further on a higher level of automation in the verification flow and in the reporting of coverage data. For doing that, two scenarios have to be faced. The first one is related to the continuous integration using Jenkins, where the design projects are developed with their respective integration tests. Second is the verification run manager (VRM) of questasim as a tool that extends the possibilities of regression management systems in an automated approach. In this way, a research (which is done in an exploratory way) is performed based on the possible options for verification information flow, taking into account the utilities of both tools Jenkins and VRM. Besides an analysis of an open software (QME) that integrates VRM and Jenkins is done (trough an exploratory way)Since this project is scoped under the automation

[1]QME is an automated regression system that uses Jenkins and VRM, see section 2.4 for more information

of the flow by doing research in the Jenkins flow and in the VRM capabilities, then in order to get full benefits of an automated approach , the regression system needs to be able to do management of seeds (constrain random seeds), rerun failed tests automatically (with more debug visibility), merging coverage across multiple runs and interface to compute resources. However the scope of this research is restricted to the analysis and study of rerun of failed tests automatically (with debug capabilities and triage analysis), merge coverage, in special for report generation (which in this case a strong research has been done, specially in the sense on how this coverage merge and report generation can be done using jenkins and vrm under the CI methodology).

In the implementation (exploratory) section, it is studied the options and the QME approach to the FPU and SPI design. From this exploratory section, the respective analysis has been done in section 6. This analysis process is mainly related to the advantages that VRM brings to the flow (that are currently missed in the given flow that is implemented using CI). This analysis goes around debug capabilities, triage analysis, report generation and merging capabilities (report for coverage, trending, triage and regression status) and the integration to the CI methodology with Jenkins. An important aspect in the context of this project is that Jenkins says if a test pass or fails, and then it will report some basic reporting of results. However the lack of metrics for verification can be improved in the process that is achieved based on the VRM capabilities

1.3 Outline

Therefore this report is divided as described below:

Chapter 2 presents the background chapter. Here it is described the sections for Continuous Integration, the description of VRM, verification management, the study cases for triage report and for implementation of regression systems with VRM.

Chapter 3 present the possible flows that were created in where integration of VRM and Jenkins has been presented

Chapter 4 presents the actual implementation (which is exploratory implementation) of the flows proposed in section 3

Chapter 5 presents the exploratory section of QME (Questa Makefile environment) that is an open source implementation of VRM and Jenkins given by Mentor Graphics

Chapter 6 presents the discussion of the exploratory sections and goes further in the analysis of the verification flow using VRM.

Chapter 7 presents the conclusions and recommendations

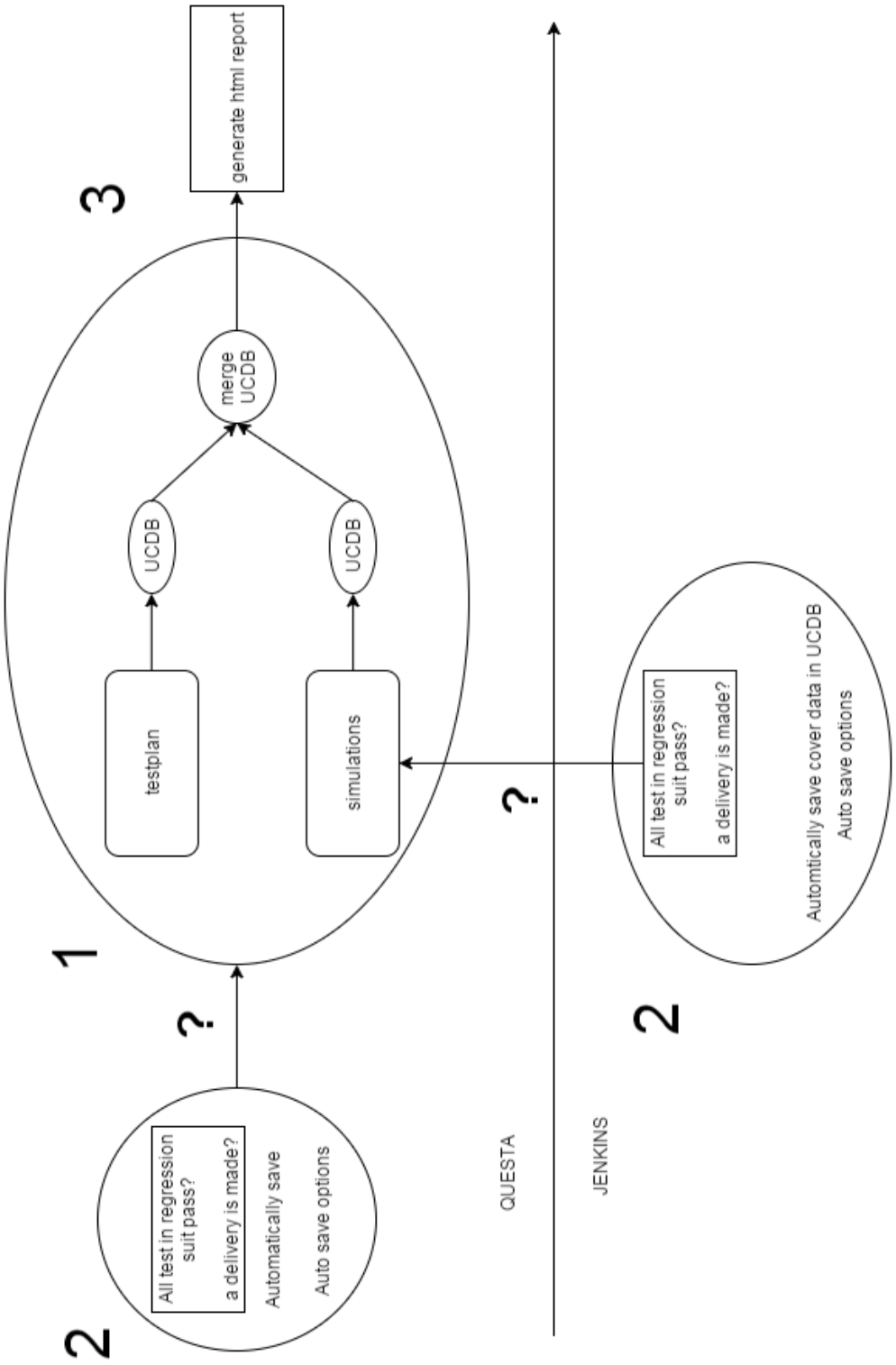


Figure 1.2: General Problem description for the given verification flow

2

Background

This is the background chapter that contains the information regarding the topics that are faced in the project. The topics are Continuous integration CI, Approaches to the automated flow, verification management (this section was extracted from the background chapter of (6)), Verification run manager (VRM), QME (Questa Makefile Environment)

2.1 Continuous integration

In software engineering, the basic model for software development is represented as the waterfall model depicted in figure 2.1 in which the faces of the development process are organized in linear order (7). Usually in this development process, the integration phase requires a great investment in time and energy (9). One of the most relevant problems at this stage is that the changes that are made by individual developers or small teams could evolve to even months of conflicting changes. Even it could be the case of reworking code that was written weeks or months before. This process usually lead to delivery delays and unplanned costs (9). Under this context, Continuous Integration (CI) is a concept that faces the integration problem. The main idea is that whenever a new change is done for any of the developers teams, the whole project is updated, compiled and tested. In this way if something is not right, then the development team is notified and the process of fixing it is performed. So the idea here is that this relative-small integration steps are much more easy to do, when they are done in a regular basis, rather than performing it at once almost at the end of the project when

this has to be delivered. In this way CI allows to give a fast feedback, regression and integration approach that ends up in fewer bugs and quicker delivery.

In the context of CI there is the idea of Continuous Deployment in which every successful build integration that passes the tests can go directly to the production. When the notions of a more predictable release cycle and more stable versions are taken into account also, then it refers to Continuous delivery. In this way the deployment process has to be automated (no manual steps) with emphasis in strong quality tests.

In order to implement a continuous integration approach in the development process of software systems (which in this case is the development of IP-SoC), then it is necessary to use a Integration tool (Jenkis Build Server)

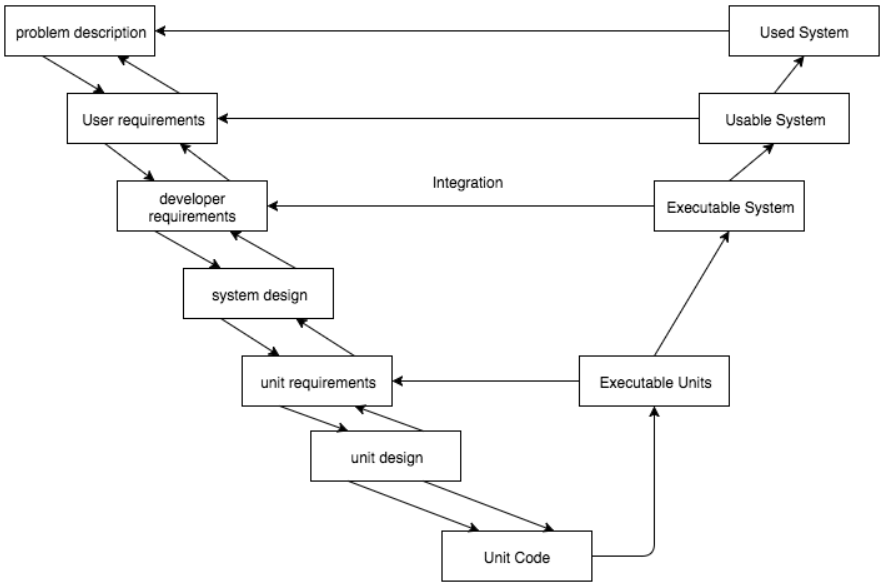


Figure 2.1: Waterfall Model (7)

In the area of software engineering, Quality assurance is about the verification of functional requirements, identification of defects (Bugs) in order to create market credibility (10). When this automated testing is implemented, then the business credibility increases and promotes customer reliance (10)

A generic build process in the life cycle of software systems is as described below (10):

1. Get a source-code copy from source control.

2. Fetch dependencies.

3. Version stamp.

4. Compile source code.

5. Unit tests execution.

6. Get objects in output directory.

7. Create package with binaries and deliverables.

8. Publish the deliverable in an artifact repository.

This Continuous Integration Approach is already implemented but in the context of hardware development. Here all the developers integrate on a regular basis their work to the whole project. From now on This methodology will be referred as Continuous Integration or CI

2.1.1 Continuous Integration in the Verification Environment - Study Case

Figure 2.2 shows a picture that reflects how Continuous integration is performed as a Jenkins Environment Overview.

The first step is to check in the code from the developers into the Source control version management (like git or Subversion). After it is done, the regression process starts. The results from it will then be managed by a Continuous Integration Server that will notify and indicates the changes to the developers and managers based on the pass or failed status. Therefore the study case presented in (8) that is summarized in this section, it studies the integration of CI and EDA tools in the steps 3, 4 and 5. In fact, EDA tools have several features for the design and optimization of the verification flow. Specially these EDA tools helps in the results reporting of the metrics that helps the debug and analysis of coverage closure under a Metric Driven Verification MDV environment

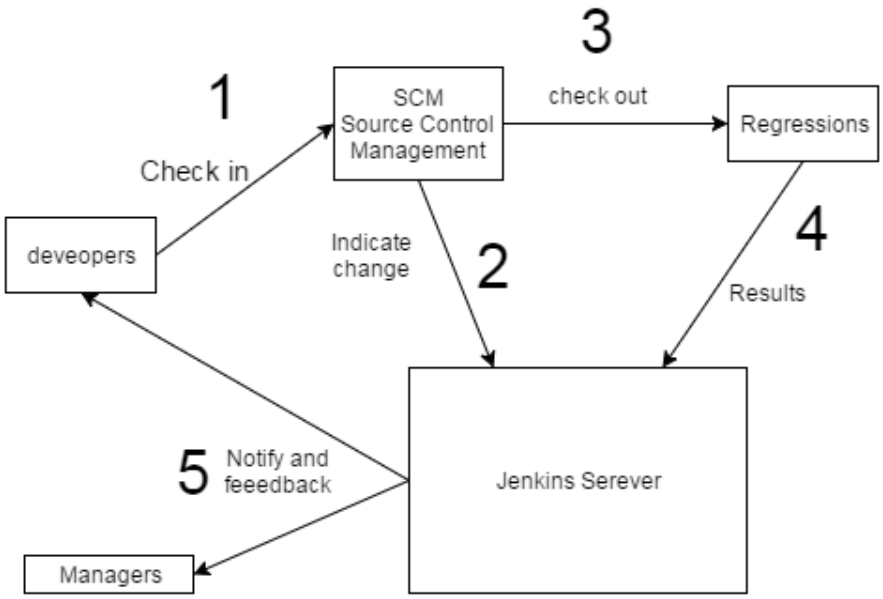


Figure 2.2: Jenkins Environment Overview (8)

Regression tools (EDA tools) and CI (like Jenkins) both have in common that they provide analysis of results (report of coverage and trend reporting for analysis) and submission of simulations (actually building and running the regression suite).

In fact DMV environments as typical verification environments, they have to analyze, filter a lot of information, (log error files) and perform code coverage, functional coverage, plan coverage. Therefore the idea of a DMV environment is actually to reduce the debug effort trough triage features in order to evaluate and achieve verification closure. On the other hand, jenkins checks in an automatic way the check in code and then spread out the results to the developers and managers that are interested in this information.

In order to integrate CI and DMV is to let each of them to contribute in its respective strengths (in fact this has been one of the results that have been observed from the exploratory implementation- section 4 - and from the QME analysis - section 5), The implementation presented in this study case (8) has concepts that can be applied to any integration of Continuous integration practice with the DMV environment. The implementation has two mayor parts:

- Generic: General library to provide reliable integration to the CI server (Jenkins)

- specific: specific tools that implement a EDA tool (in this study case **??** is Vmanager and Vmanager C/S)

The particular study case (8) presents the integration of Cadence vManager into the CI practice. vManager is a tool specific that offers verification management features (like VRM in this master project. See section 2.3.3 for more background information of VRM).

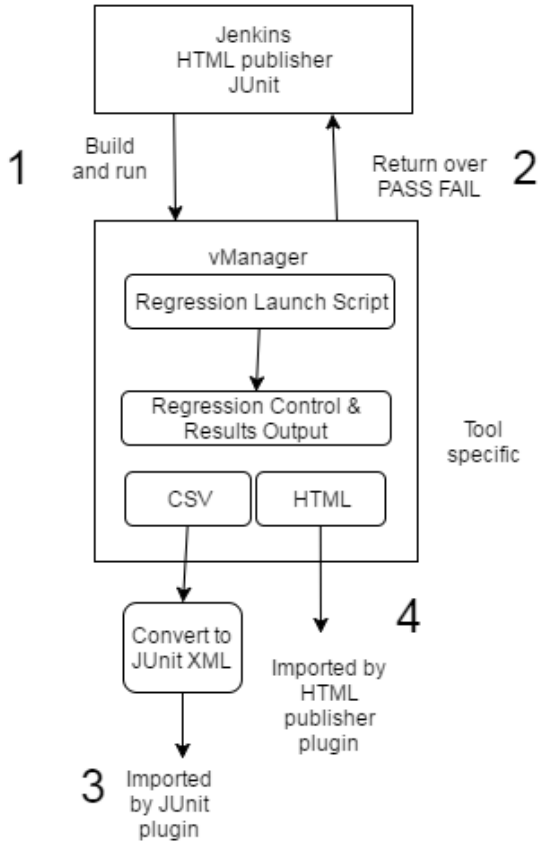


Figure 2.3: Integration of Jenkins and vManager (8)

In this sense, the first step (shown in picture 2.3) begins with Jenkins. Here it runs the regression (vManager is the actual tool that runs this regression). Second, after the regression is done then the PASS/FAIL status has to be pass back to Jenkins from the specific tool (in (8) and in the master project is VRM). This status flag in Jenkins must match the pass fail status of the regression in order to ensure trust in the results.

Once the regression is run, then a linking to the HTML results is performed by using the HTML publisher plugin of Jenkins.

This study case has a strong relation with the current master project since it also specifies the integration of a EDA specific tool into the CI practice. In section 3 the integration of VRM into the CI is presented and a particular case of integration of VRM into the CI practice (which is QME) as an automated regression systems build based on the VRM and integrated into the CI practice.

2.2 Approaches to the automated flow

In this section 2 main study cases are presented. The first one relates to the Cypress case which is a customized regression management systems that uses as basis VRM. The second study case gives practical guidelines to the deployment of the Continuous Integration in the verification environment.

2.2.1 Cypress case (2)

Cypress is a semiconductor company that has implemented a use case of an automated system in its verification flow (2). The implemented verification management infrastructure at Cypress (they named it as VMS or Verification Management System) provides an uniform front-end shell and a back end-data base. This provides a division of labor among Ruby scripts and VRM (Verification Run Management of questasim). The ruby scripts are used for front-end tasks while VRM is used for launching tasks. The VMS infrastructure includes the following: (2):

1. Design Tree and File list gathering: A configuration file is created. This has the information of all prerequisite designs and any possible extra HDL file for the general design. The VMS provides a full list of commands that support different features while creating the design list that is going to be verified.

2. Testbench Tree and File List Gathering. As in the case of design, complex testbenches contains multiple simple testbech files so that a testbench is described based on configuration files that list all the necessary files

3. Verification Management configuration: These configuration files provides information about compilation for design and tests

4. Test Tree Infrastructure and Test Lists: The test tree contains information for each leaf and branch. In this way each test is made with inheritance information from the higher levels.

5. Launching VMS. When VMS is executed, it gets the necessary information from design and testbenches. Then VRM (Verification Run Manager from Questa) launches the jobs accordingly.

6. Metrics gathering, Output generation and reporting: In simulation, the coverage data is saved in UCDBs. For each test, a UCDB and a report is generated. For regression, a merged files is generated with its respective report.

Here it is important to point out that VMS specifies a design directory that all designs has to follow. This provides that the tools can rely on finding the information (2).

On the other hand, VRM is used for launching tasks based on a generic RMDB.

A this point, this case does not have an approach for Continuous integration development as is this particular case with Jenkins in this project. However it uses the tools offered by VRM as the core for regression systems. In this sense vrm is in the core basis and a set of ruby scripts implement the verification management system.

2.2.2 Building automated Regression Systems

A regression system can be divide into (11) :

- Capture: Usually, when implementing regression system the configuration has to be captured trough scripts. Therefore, the idea here is to separate this configuration from the scripts in the sense that configuration is not performed under the scripts but rather as a sequence of commands. In this way, a system with inheritance and parametrization is a good way to do it since the same settings can be captured over and over trough different projects with a small amount of changes. For this particular case of study, inheritance can be performed trough base-runnable type for the VRM script (see section 2.3.3 for more information about runnable in VRM). However it has not been implemented since there is not a big spectrum of different projects. In this case implementation of vrm in

detailed (as separate scripts) has been done for the SPI but for the FPU it has been applied using QME (which was one of the demos)

- Control: It is related to questions such as:
  - How the action should be executed? in this case action are executed based on the type of shell (see section 2.3.3 for more information about the different types of shells that can be executed at this layer)
  - How to interact with execution resources? like for instance with grid systems. In this particular project, the implementation of the different options has been done on the local machine since the projects are not big and not require to run the jobs in the grid

- Automation: It is associated with the implemented systems that separates capture and control. In this context, there can be actions that are automatically triggered after a simulation run is performed and based on the PASS-FAIL status then it can trigger specifics jobs, such as re-run the simulation with another setting. Besides, every run, that is saved into a specific UCDB, can be pointed to a merged UCDB. Also the testplan can be automatically imported to a UCDB and merged as well.

- Visibility: the status of the regression should be available in order to check which actions have been already completed. In this way, the user interface should allow to check for the start, monitoring and analysing of the regression. In this case, regression results are also available in HTML views. It is important to note that this project has got a strong focus on reporting visibility. However there are other important aspects related to debug visibility. In this sense VRM offers options for debug visibility that are described in section 2.3.3.5

2.3 Verification Management

The main factors that force an efficient verification management [1] and planning are time to market pressure and high functionality. Therefore a successful project depends

[1] section extracted from (6)

on how to develop a high quality product in a given schedule with limited resources. (12).

Verification management is mainly related on how to measure, track and analyse the verification process. The general verification flow is depicted in the figure 2.4

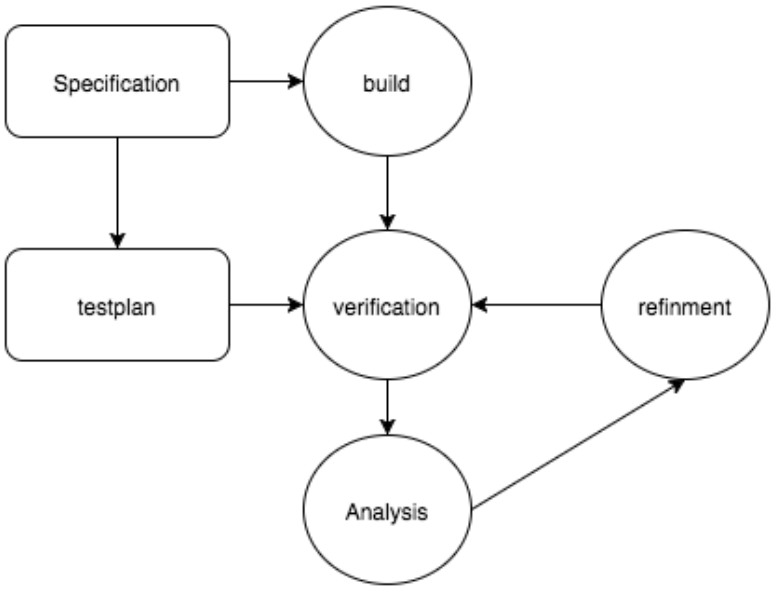


Figure 2.4: Verification Management flow (3)

The project starts with the specification, from where the verification plan is derived and the design is build. Once the test plan is defined, it allows to see how the changes in the specification are reflected along the development of the project. In this development, verification says what are the possible warnings, errors and the areas where more effort has to be put on. Besides, it is necessary to refine the test based on the data analysis which has to consider aspects related to tests, coverage, instances and resources.

Under this context, Questa offers a Verification Manager Tool for Verification Management. Its core component is the UCDB (Unified Coverage Database) as shown in figure 2.5. Its main feature is the capability of unifying coverage data like assertions, test data and test plans. On the top of the UCDB, there are the components for test plan tracking, trend analysis and Result analysis.

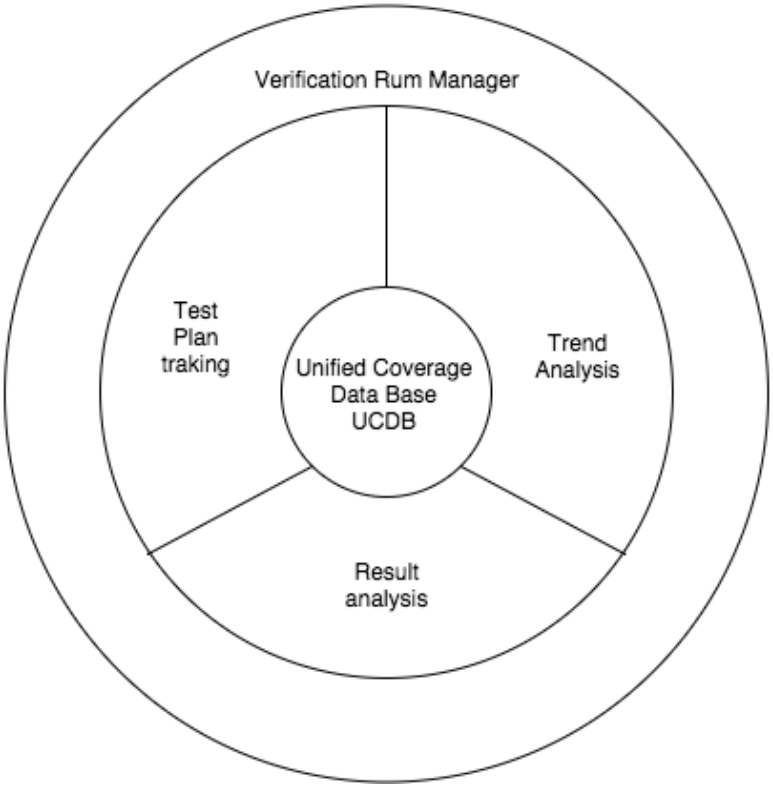


Figure 2.5: Questa Verification Management: software overview (3)

The test plan tracking allows the test plan to be the driving document for the verification process. Trend analysis allows to see the progress of data coverage along the development of the project over time. Last, in the result analysis has the capability of combine results from different regressions or runs-tests.

In the last layer there is the Verification run manager (VRM, see section 2.3.3) which gives control and visibility of the project. At this layers there are different automation capabilities that easies tedious tasks, specially those related to data coverage analysis and data organization.

2.3.1 Test-plan tracking

The verification manager[1] allows to track quality, schedule and resources from the testplan allowing automation of tedious tasks, like gathering and organizing coverage data for later analysis. This tracking is done on the basis of linking information. This

[1]section extracted from (6)

linking actually means to set common arguments between the arguments in the testplan and the metrics in the project implementation or in the different databases that save the data of the different runs.

After several run-tests have been performed, then a UCDB database is saved. After some time, there are several of this data bases which are merged in order to compress information for historical analysis (see figure 2.6).

After this merge is done, it is easy to analyse the data and check what was the test that gave the maximum coverage in certain areas of the project. With this information, it becomes very easy to verify any later change in the specification.

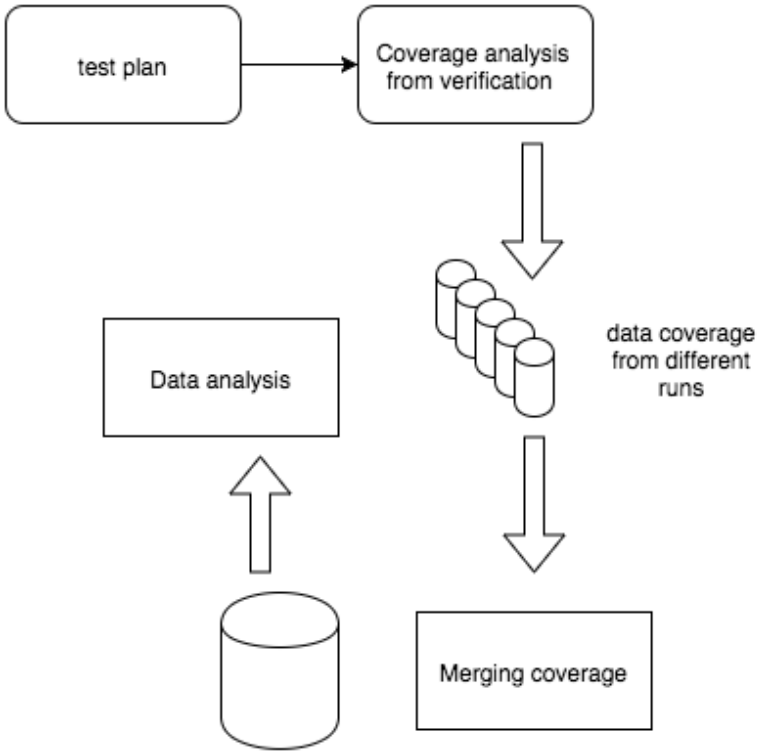


Figure 2.6: Test plan tracking flow (3)

Questa verification manager allows to implement this linking trough several formats under excel, open office, word or xml.

2.3.2 Trend analysis

In every regression[1], there is a high volume of data. Besides, along the project, the amount of data can really explode. When all this data can be viewed, can improve the decisions that have to be made in the project. At this point, the UCDB (Unified coverage database) allow to take "snapshots" of the coverage metrics along the time in a trend UCDB that has a format that allows to reduce the amount of data to save while keeping the useful data that can be reported (see figure 2.7). This data can be reported in, svc, xml, html. This trend analysis is implemented in this project (see section 5.5 for trend implementation in the FPU design) however since there is not development then the coverage is always the same.

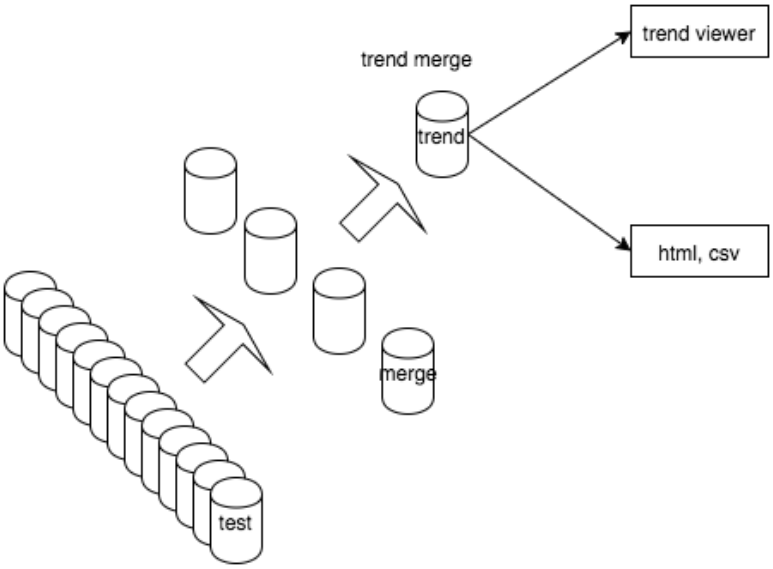


Figure 2.7: Trend analysis (3)

2.3.3 Verification Run Manager VRM

Verification Run Manager allows to execute tasks that optimize and organized regressions (3). These actions can be performed as background tasks, exported to a specific server or queued for execution in a grid system.

The database that uses VRM is named RMDB (Run manager database (3)) and is in this file where the configuration of the different tasks are set. These tasks are related

[1]section extracted from (6)

to the regression suite and are organized in a hierarchical tree topology that allows to group the tasks.

The RMDB is defined in XML. In this definition, the building blocks are named runnables which can be of three types:

- group: Here another members are defined
- task: this is the leaf level in where the actual execution is performed
- base: here it defines parameters that are going to be used or inherited among different members of a given group.

As an example, under a given top runnable of type group, there are members that are also defined as runnables of type tasks.

In a given runnable different kinds of scripts can be specified as shown below:

- preScipt: This script is executed only once. Usually executes tasks related to compilation, that are executed only once at the beginning
- postScipt: this script es executed at the end. The tasks that usually are performed in this scripts are merge tasks among different regressions. Also specifying tasks related to report generation.
- execScipt: These are the tasks that are mainly related to simulation tasks

In the example in figure 2.8, a tree topology for the RMDB is presented. Here, there is a top runnable (group 1 with a preScript runnable in it) that has three members (Member 1 / Direct Tests; Member 2 / Random tests; Member 3 / Formal tests). Finally, the last step in this tree is the postscript/ merge and report.

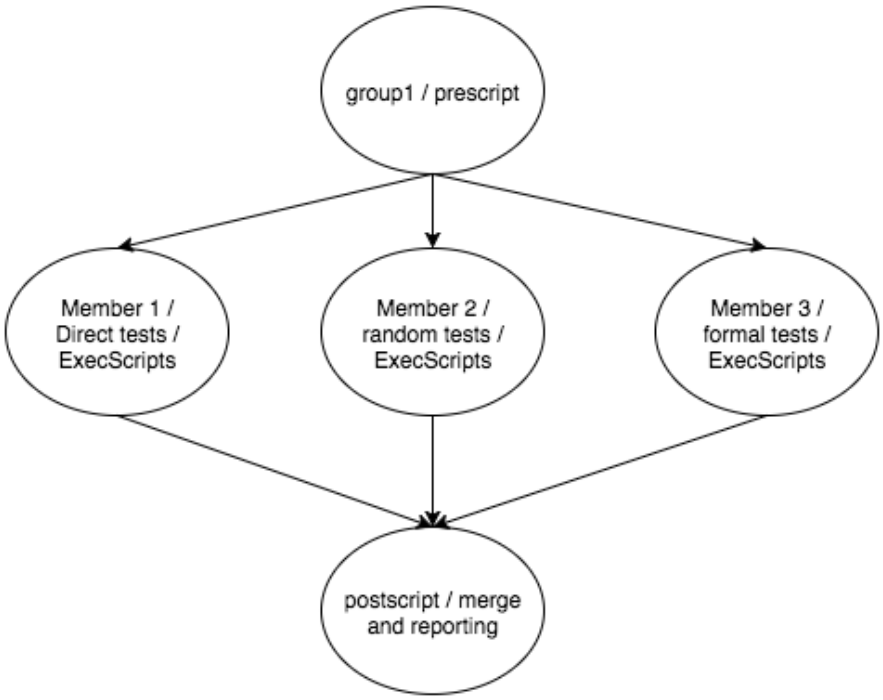


Figure 2.8: Example a RMDB topology tree

This kind of topology of the RMDB is modeled with XML, in which different setting can be described based on the specifics requirements for the automation of the regression system. In fact, for this particular case of study, a template in RMDB was developed. For more information see section 4.3

2.3.3.1 Inheritance (3)

VRM allows , through inheritance and parametrization, reduce redundancy in the configuration across RMDB databases. For example, if there is 100 tests that all share some common configuration parameters and only some few changes among them, then it would not be necessary to specify for each test a particular configuration but to use inheritance. VRM supports two kinds of inheritance: explicit and implicit inheritance:

- Explicit inheritance (base inheritance): This type of inheritance is similar to the one offered by Object Oriented Programming. In this case a base inheritance is specified as shown next ((3)):

```
1     <runnable name="A">
2         <!-- A content -->
```

```
3       </runnable>
4       <runnable name="B" base="A">
5           <!-- B content -->
6       </runnable>
```

Therefore, what is specified in A is also specified in B. Base inheritance is related with content but not with behaviour. In this way, base runnables are not involved in the execution graph themselves and it is done by specifying the type as base, instead of type group or task. In this way, the runnables of type base cannot be instantiated, that means that cannot be a member or another runnable of type group.

- Implicit inheritance (group inheritance). This type of inheritance is defined at runtime based on the tree topology of the regression suite (membership of a group). The parameters of the parent group are inherited to the child group as shown in the next code and in figure 2.9

```
1       <runnable name="parent" type="group">
2           <members>
3                   <member>child</member>
4                   <!-- other members -->
5           </members>
6           <!-- other content -->
7       </runnable>
8       <runnable name="child" type="task">
9               <!-- content -->
10      </runnable>
```

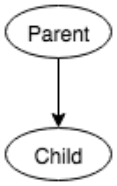


Figure 2.9: group inheritance: In this kind of inheritance, the parameter of the parent are inherited to the child. In this example, the parent runnable, which is defined as group type, inherits its parameters to its children (another runnables that are defined as its own members)

Group Inheritance is not enough by itself. For example, in the case of multiple

groups each one defining a specific configuration of a design. Besides assuming that there is also a number of tasks in which it is defined a randomization seed. If these tasks are going to be assigned to each of the groups, then it would be necessary to replicate each task in every group. However if base inheritance is used then the tasks (that defines a list of seed values) can be set in one runnable which is inherited to each of the groups that defines a specific configuration of the design. So in this way configuring the regression suite becomes easier.

Table 2.1 shows which elements can be inherited from the base or group approach.

Element Base	Data Item Group	Inheritance from Base	Inheritance from Group
runnable	parameters	yes	yes
runnable	execScript	yes	yes
runnable	preScript	yes	No
runnable	postScript	yes	No
runnable	members list	yes	No
method	method command	yes	No
usertcl	TCL content	yes	No

Table 2.1: Table to test captions and labels

At this point in the implementation section the inheritance was not used because the projects that were implemented does not share common settings since they are from different sources (Nordic SPI and Mentor graphics FPU) , however this feature allows to implement a reusable framework that easies the setting of common regression systems that share common settings.

2.3.3.2 Merging in VRM (3)

VRM has three ways of automation for merging porpoises.

1. Do-it-yourself DIY merge: ExecScripts has to generate UCDB and the user calls the command vcover merge with the propers arguments. In this way the merge can be done in a exeScript when used only for one task or in a postScript when used for a group. In this automation level, the path of the UCDB have to be passed to the respective arguments of the vcover merge command.

The algorithm defines that each execScript will generate a UCDB to a specific location and then the vcover merge is call to perform the merge operation into a merge file . Besides postScripts can merge intermediate merge files into higher levels of merge files based on the hierarchy of the regression suite.

Next example (3) shows how the merge process is implemented in a postScript action script.

```
1 <rmdb>
2 <runnable name="nightly" type="group">
3 <parameters>
4 ...
5 <parameter name="mfile">(\%DATADIR\%)/nightly/merge.ucdb</
    parameter>
6 </parameters>
7 ...
8 <execScript launch="vsim">
9 <command>vsim -c top -lib (\%DATADIR\%)/nightly/work
10 -sv_seed (\%seed\%)</command>
11 <command>run -all</command>
12 <command>coverage attribute -name TESTNAME -value
13 (\%testname\%)</command>
14 <command>coverage save (\%ucdbfile\%)</command>
15 </execScript>
16 <postScript launch="vsim">
17 <command>vcover merge (\%mfile\%) test*/*.ucdb</command>
18 </execScript>
19 </runnable>
20 ...
21 </rmdb>
```

In this case, the postscript merges all the test level UCDBs that generated and saved in the specified path.

2. list based merging: As in the case of DIY merge, each simulation generates a UCDB which is later on merged in an upper level merge file. However the difference is that the paths of the UCDB files that are going to be merged are specified in a vrun list trough a mergelist parameter. The values in this list are used as inputs to the command vcover merge

```
1
2 <rmdb>
```

```
 3 <runnable name="nightly" type="group">
 4 <parameters>
 5 ...
 6 <parameter name="mergelist">(\%DATADIR\%)/nightly/
       mergelist
 7 </parameter>
 8 </parameters>
 9 ...
10 <execScript launch="vsim">
11 <command>vsim -c top -lib (\%DATADIR\%)/nightly/work -
       sv_seed
12             (\%seed\%)</command>
13 <command>run -all</command>
14 <command>coverage attribute -name TESTNAME -value
15  (\%testname\%)</command>
16 <command>coverage save (\%ucdbfile\%)</command>
17 </execScript>
18 <postScript launch="vsim">
19 <command>vcover merge -out (\%DATADIR\%)/nightly/merge.
       ucdb
20 -inputs (\%mergelist\%)</command>
21 </execScript>
22 </runnable>
23 ...
24 </rmdb>
```

Every time a test pass, its path is written to the mergelist. In this case, the vcover merge command uses the merge file as input. This file does not has to be deleted every time since vrun clears the file.

3. Incremental merging: In this automation level, every time a execScript ends (in this execScript the UCDB is generated and saved to the specified path) then the command vcover merge is automatically call therefore the target merge file is incrementally merged with the UCDBs that are being generated in ongoing basis

   This merge works only when the execScript is done. On the other hand, when a postScript is done (as a short reminder, postscript is executed as part of groups) then no merge is done (for preventing automated merging at the group level).

   Next code (3) shows the example for incremental merging

```
1 <rmdb>
```

```
 2 <runnable name="nightly" type="group">
 3 <parameters>
 4 ...
 5 <parameter name="mergefile">(\%DATADIR\%)/nightly/merge.
       ucdb
 6 </parameter>
 7 </parameters>
 8 ...
 9 </runnable>
10 ...
11 </rmdb>
```

In this case it is just enough with specifying the merge-file parameter

4. Queued merging (see figure 2.10 for detail in this merging): It is a variation of incremental merging. For queued merging, a list of the UCDB are listed and then a vcover merge command is launched but only when the regression test finished under a small time window. In this way this approach improves performance since merge can be executed at the same time

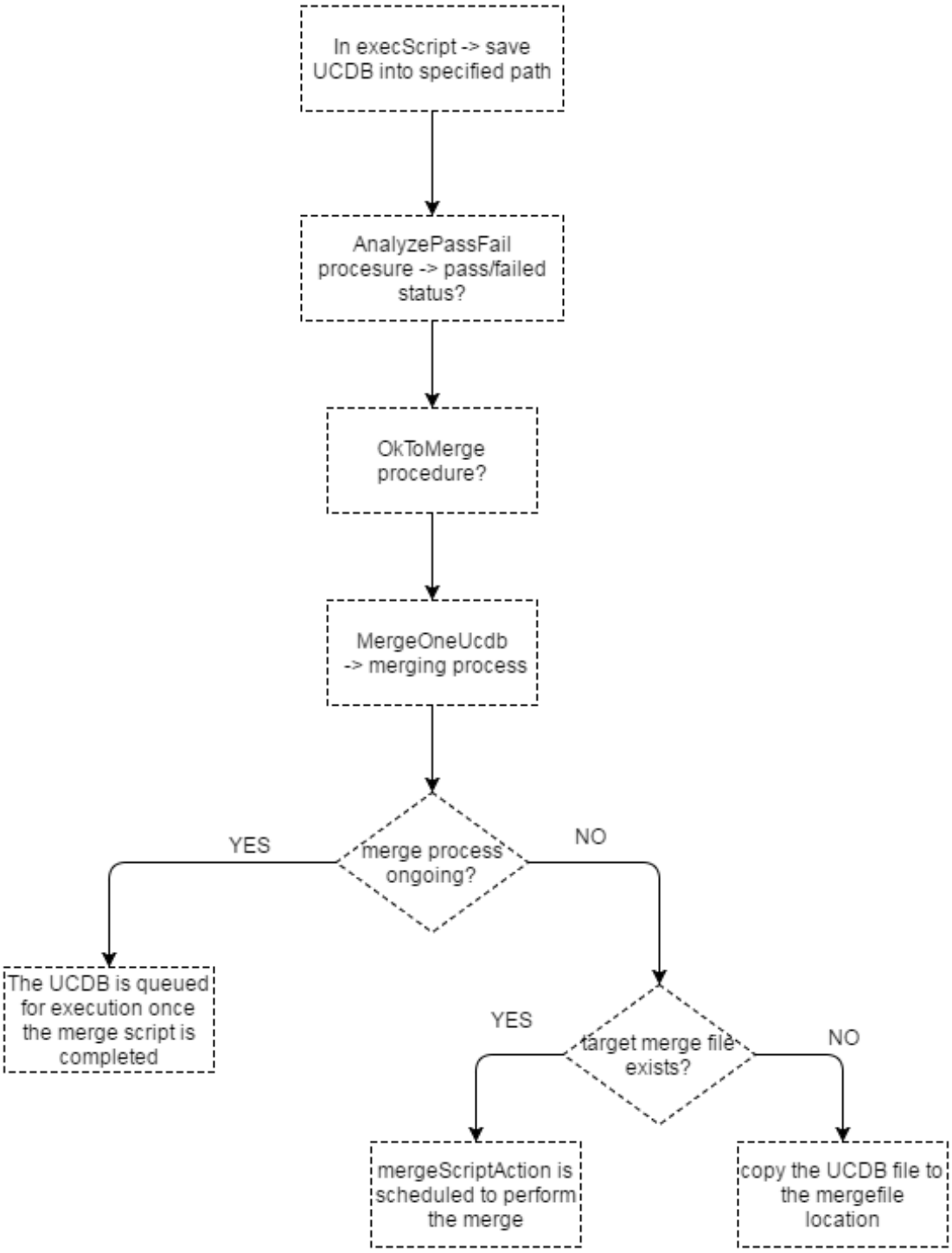


Figure 2.10: queued merging process:flow diagram for the queued process performed in Questa (3)

The main goal is to have only one vcover process writing to a target file one at a time.

Table 2.2 shows the merge flow summary

UCDB?	mergelist	mergefile	noqueuemerge	merge type	Merge command
no	–	–	–	unspecified	no Specified
yes	no	no	–	DIY	User postScript
yes	yes	no	–	list merge	User postScript
yes	–	yes	yes	incremental merge	vrun
yes	–	yes	no	queued merge	vrun

Table 2.2: Merge flow summary (3)

2.3.3.3 Triage reporting in VRM (Automated result analysis) (3)

One of the capabilities of VRM (which is very relevant compared to the flow given by Jenkins) is that it allows the implementation of automated results analysis. Triage means the collection of messages from multiple verification tasks. This messages can be filtered and organized in such a way that it will be easy to determine failures and patterns from regression runs. Therefore failures that are similar across regression tests can be identified. The main advantage on this Automated result analysis is the reduction in debug time since similar failures can be identified (saving time and resources). Figure 2.11 shows the flow for triage reporting. The first step in this flow is to run the simulation in order to get the results in a UCDB data base. In this run the messages are written in log files or in WLF files. Once this ucdbs are obteined, they have to be merged (in case there are multiple ucdbs). The next step is to create the tdb file (triage data base file) that uses as inputs the UCDB, the WLF and the LOG files. Finally is the report view

Here, after the UCDB,LOG and WLF files has been generated, they go to a tdb (triage data base) from which a triage report will be made.

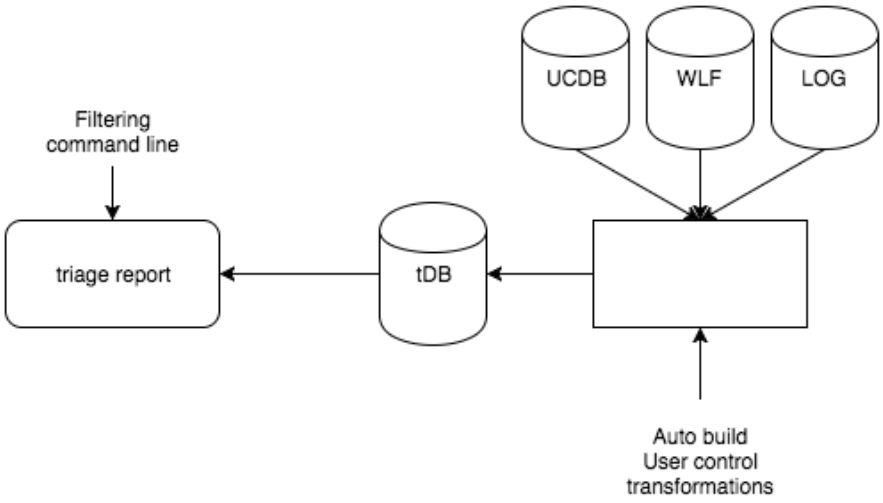


Figure 2.11: Automated result analysis (triage reporting) in VRM (3)

The triage reporting can be implemented in the same way as the merge reporting. Table 2.3 shows triage flow summary options for triage actions. This result analysis has two modes that are predominant. A regression suite or as individual simulations. For regression the analysis is made based on multiple runs from where the needed data is extracted for a post-filtering in order to identify root failure causes or perform debug based on the chosen resources and priorities. On the other hand, for individual simulations, the result analysis gets the failures in an automated approach.

ucdb?	triagefile	noqueuetriage	triage type	triage command
no	–	–	unspecified	no ucdb Specified
yes	no	–	DIY	User postScript
yes	yes	yes	incremental autotriage	vrun
yes	yes	no	queued autotriaged	vrun

Table 2.3: Triage flow summary

2.3.3.4 Combination flow for merge and triage

There are some options that VRM specifies in order to implement in the verification flow the merge and triage actions.

In the first option, the action is checked in order to identify if it fails or pass. If

pass then the auto merge process will make the merge process to point to a *passed.ucdb* otherwise it will be point to a *failed.ucdb*. Therefore the postscript only need to pass the failed UCDB to the *triage* command. Since this actions are done on the fly (3) then the passed and failed ucdb's can be done concurrently which end up in an improvement of efficiency (3). Figure 2.12 shows this concurrently option as a diagram flow.

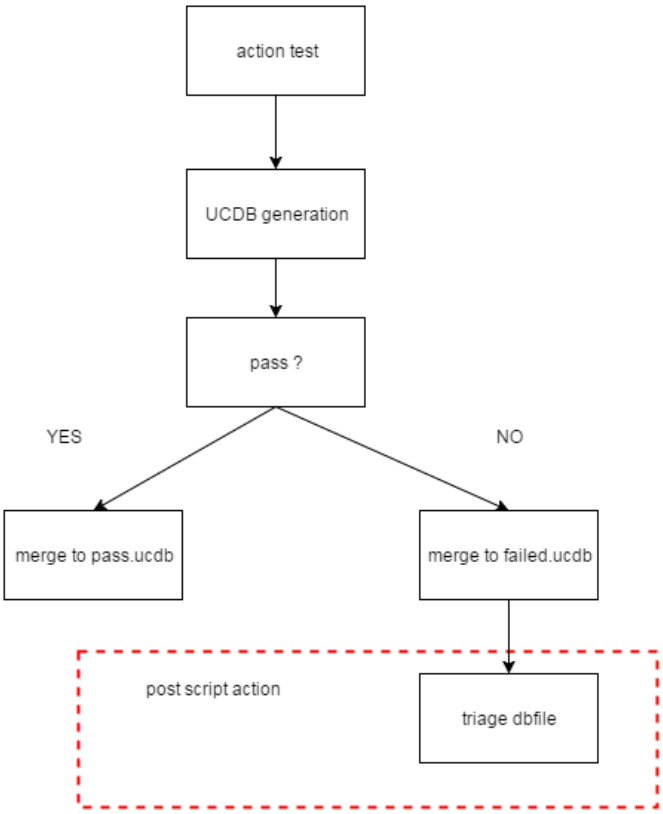


Figure 2.12: Flow for a concurrent merging files for pass and failed actions. When failed, a postscript takes the *failed.ucdb* to the *triage* command. This flow is the most efficient when trying to generate concurrent ucdb for pass and failed actions.

Another alternative is by using file lists in which the pass and failed ucdbs (from the respective tests) are named. For this case the *merge* and *triage* commands are implemented only at the end of the simulations (therefore it is not as efficient as the first option).

2.3.3.5 VRM Debug Capabilities

One important aspect of regression systems is that the tests should be able to be run in debug mode either in the design process or in the nightly run. Besides the ability to run failed tests is also another option that a regression system has to support for it.

In this sense VRM allows to have several methods for providing the regression system with this option.

- Local rerun method: In this method the failing actions can be re-executed. In this way, every time there is a failure when running the regression suite, then this actions are scheduled for a posterior re-execution. Besides, the using custom TCL scripts, another settings such as how many times the action should be re-executed, or which parameters should be overridden. Figure 2.13 shows the flow in local debug flow

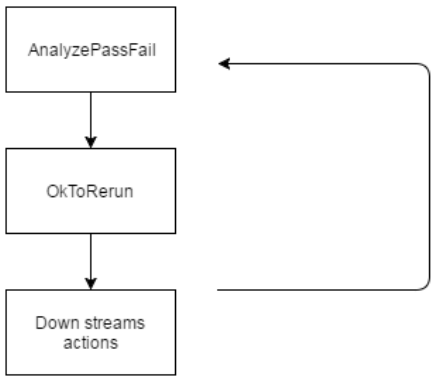


Figure 2.13: Local debug flow. Here, once the action is performed it enters to the local debug flow from where first an action for analyze if pass or failed is performed. if so, then the process re-run the action again. Based on the configurations, this process can be scheduled several times.

- Global method: When there are failed tests , then this global methods allows to compile again the list of failed tests based on the parameters that are given in the RMDB. It is important to see that these global methods are not related to the local one, which is set by default

- Semi-automatic mode: once a regression suite has been already run then VRM looks for those tests that have failed or for the tests that have not been yet

run. In this case a macro runnable is set for this purpose. Therefore, inside this runnable , VRUN is called recursively as many times as needed in order to run the regression. For instance, it could be the case that once a regression is run, then the failed test are selected and run again in debug mode so these tests are ready to debug. This particular case is relevant compared to the given flow that is managed now.

When implementing this debug capabilities, it can be done either in the same working directory of the non-debug tests or they can be implemented in a different working directory. The recommended way of doing it is to set a different directory. In this case a top runnable will handle the non-debug. Therefore after the simulation is run then the failed tests are set in the debug-top runnable.

Among the debug options (typical RTL debug features) that Questa has are the ones related to verilog deltas, process debugging, tracing, comparing results.

2.3.3.6 Study case of triage and debug - Vennsa technologies (4)

A debug process is made of two steps:

- Triage: this step is applied only when bugs are discovered trough checking the regression runs. Here the main idea is to filter and group failures in order to determine common patterns and in order to identify who is the engineer that is going to deal with the bugs
- Root analysis: Once the bug has been identify, then the next step is to identify the root cause and fix it.

After a regression suite has been already run, then the next step is to go through all the failure and warning messages in order to identify where are the bugs. In this process there are several aspects that have to be faced, which can be summarized on the next questions: (extracted from (4))

- Which failures are caused from the same and/or distinct source?
- Which failures are new?
- Which failures are in fact bugs but have not been fixed yet?

- who is the engineer that should be assigned the corresponding bug?

These are difficult questions, specially from the fact that in the beginning there is limited visibility. In this sense the only way in which these questions can be answered (like if a two failures are caused by the same bug) only when the cause has been identify and the bug has been fixed. In this context, triage specifies the failures but the way to actually fix the problem is still long. In this sense there are three mayor problems that are faced:

1. After triage, the failure is given to the engineer who has to get the error cause and fix it. However after the root has been identified it is possible that the error is not in this block but in another verification engineer's block. This can consume time and resources until the right owner of the error is identified.

2. It can be the case that there are two different bugs in the design that actually fires the same failure. In this sense the first error source can be identified and then fixed. However it can take time and resources until the other error source is identify and fixed.

3. It can be also the case in which the same bug fires different failures. In the triage step it is difficult actually to identify that the failures are from the same source. In this case every failure will have the same root cause analysis and at the end they will be identified as from the same bug.

Therefore, under the context of the three last cases, messages can differ strongly from the real cause of error Form this particular study case (see (4)), the messages (failure messages such as wlf, vcd, fsdb) that are obtained from Questa, later they are given as inputs to an automated triage engine which in this case is Vennsa's OnPint. This tool is an automatic debugger tool that analyses a simulation failures and identifies the root cause. In fact it does not identify the exact cause of error but rather provides a list of possible cause of error that are the basis for the engineer to identify the bug and fix it. These error causes are related to the rtl sources and the test-benches.

In this study case, the important facts are that Questa uses an external tool for implementing an automated debug which start from a triage analysis. In this master project there is no any external debug analysis, since the triage analysis that can be

implemented is the one Questa uses and is integrated into the flow (although was not implemented since there was not any development process) in order to analyse the possibilities that vrm can add to an automated flow. However from this study case (Vennsa teachnologies) it is clear that the triage analysis can only help in the filtering and grouping of failures that in some sense can make an improvement in the debug process, but there is not any tool that actually helps in the error cause analysis (finding potential errors) in the debugging phase of a verification flow. This analysis is done in the analysis part (see section 6)

2.3.4 VRM Flow Diagram

The Flow diagram of VRM is the one shown in figure 2.14.

The flow that VRM implements, from its general view, it stars with the configuration of the rmdb from where grouping, parametrization, inheritance of the different runnables that contains the different jobs of the regression system. From here then the regression is executed (Vrun). When the execution starts, this starts basically with the extraction of the tree topology of the runnables that compose the regression system. Then there can be third parties, like scripts that can create more facilities to the regression system (maybe coverage extraction to another server (like in the case of QME and jenkins (see section 2.4 for more information about that))). After the regression has been run, then there are several actions that are taken. If the tests pass o fail what to do? there should be any auto merge script, any failure analysis. All this post actions are integrated in the monitoring part of the flow from where the decision of what to do are considered. This decisions are in fact related mainly to the pass or fail analysis of the tests.

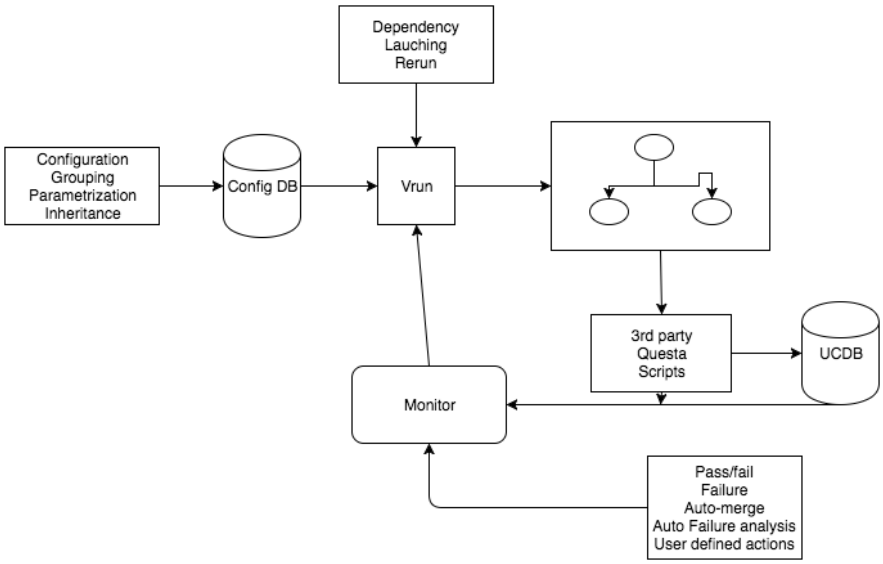


Figure 2.14: VRM flow diagram from QuestaSim (3)

From a general view, VRM allows the end user to apply the next features to the regression system:

- Automate the regression process. In this project the automation has been highly related to the merge capabilities and the report generation.
- Efficient Regression result management, improving debug capabilities. This is another aspect that this project has focused since this is an important feature that any regression system in the verification flows has to have.
- Allows fast bug identification. In this case, this relates to the trend analysis
- Manage different computing jobs. This is out of the scope of this project since the given IPs are relatively small and the integration with the grid is not tackled.

VRM executes jobs (or actions like execScripts, preScripts, postScripts) that will end up in the background (local machine), grid systems or specific server. Once the jobs have been executed then its status is passed to another activities. From the figure 2.14, at the beginning of the flow, first is the configuration of the rmdb database, then the script is run. Later Post actions are going to be specified, like the merge capabilities for reporting of coverage results, or Auto failure analysis.

In this sense VRM can be used in several modes such as:

- Nighly regression: In this mode, there is a top level test-suite, that contains subgroups of another tests. The it is run in batch mode unattended.
- Desktop run of user tests. A rmdb defines a set of groups of tests that are run manually by the user in background or in foreground (batch mode). At the end the user waits for the results. This mode has been the one implemented in the options improvement (see section 4.3)
- Re-run of failing tests: After a regression has been run, then there is a collection of failed tests that are listed. Then a temporary suite is build and run. At the end the user waits for the results.
- Automatic run in debug mode: when there is a regression suite that is being executed. If suddenly there is a failed test, then these tests can be run in debug mode in order o get more information of it.
- smoke tests: A short broad suite is created based on short test suites for maximum short/broad coverage. This suite is run in background.
- High coverage random suite: Random tests are run and then they are ranked based on the coverage. Therefore the random seeds that get most coverage are saved in a database for later use.

2.4 QME Questa Makefile Environment

QME (Questa Makefile Environment) is an environment that takes away the need for creating scripts in the regression management system. In this sense QME provides an infrastructure that easies the utilization of scripts across the project for the configuration of the regression system. The relevant aspect of QME is that this uses VRM and Jenkins in the development flow.

Figure 2.15 shows the general architecture of QME. This platform has the next inputs ((5)):

- RTL file list (DUT file). This is the list of all source code for the design.
- Test benches file list (TB filelist)

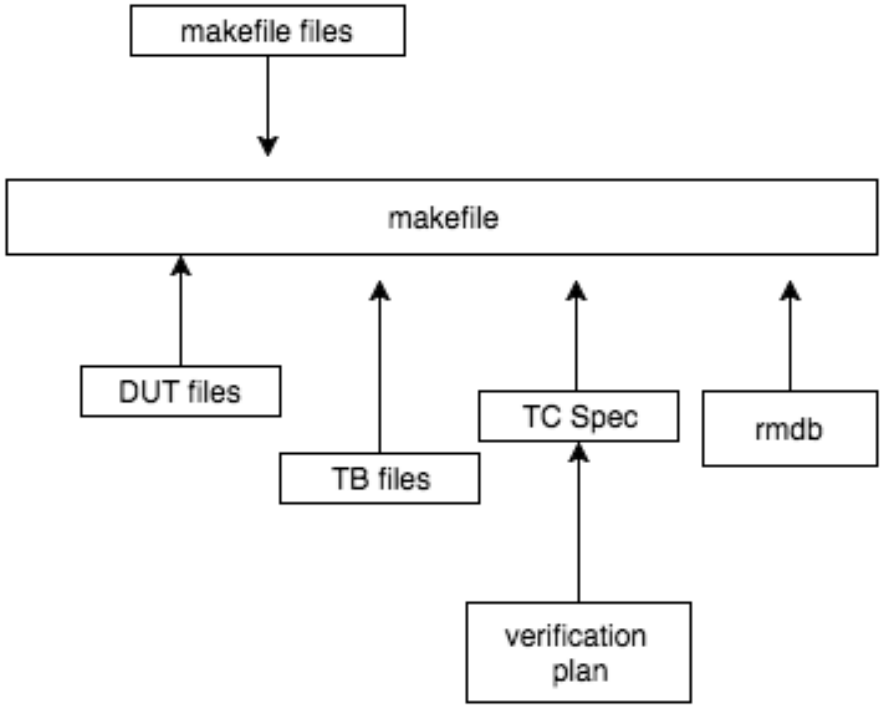


Figure 2.15: qme architecture (5)

- verification plan in xml file (Verification plan).
- Testcase specification file: This file is related to the specification for the configuration of the regression suite.

In figure 2.15 the makefile files are used for configuration of environment variables. On the other hand, RMDB are implemented for using the VRM capabilities such as merging, coverage reporting, triage reporting.

Figure 2.16 shows the different verification levels that QME embeds.

- The first level is the code in the rtl source and testbench source.
- The second level is the RTL file list, the TB file list and the TC specification file (which is in where the regression suite is configured).
- The 3rd and 4th level is actually the Makefile of QME as specified in figure 2.15.
  - Here it defines a Day to Day work (3rd level) for compilation and simulation. One added feature is that here it only compiles those files that were changed.

In fact this feature changes turn around time, however a good coding style has to be defined.

- In the regression Verification closure it makes the regression suite.

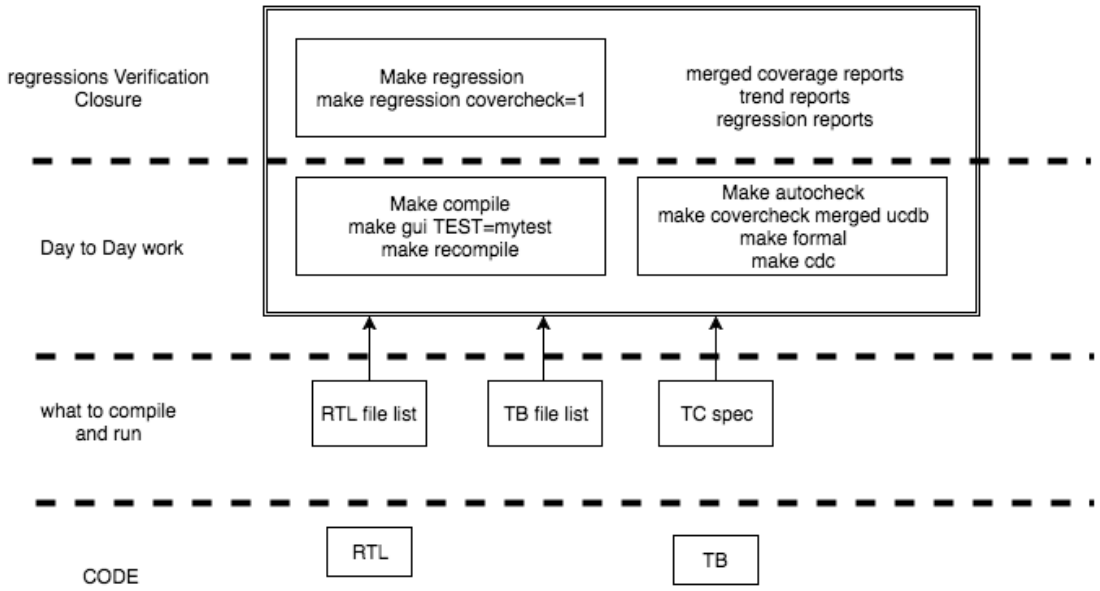


Figure 2.16: qme architecture (5)

QME also as a management tool allows to perform Merged UCDB and trend UCDB files with its HTML report files as shown in figure 2.17

2.4.1 Setups

QME uses what is called setup which is the setting of parameters for the DUT. If there are two simulations of the same DUT but with different setup then each setup manages the DUT as a separate DUT with separate coverage.

An important feature is that QME adjust the visibility and performance improvement when the verification is done in block level or in chip level. If the simulation is done in chip level then less visibility to improve performance is set. This affects the debug and coverage collection.

2.4.2 QME flow

The general flow for QME is shown in figure 2.18

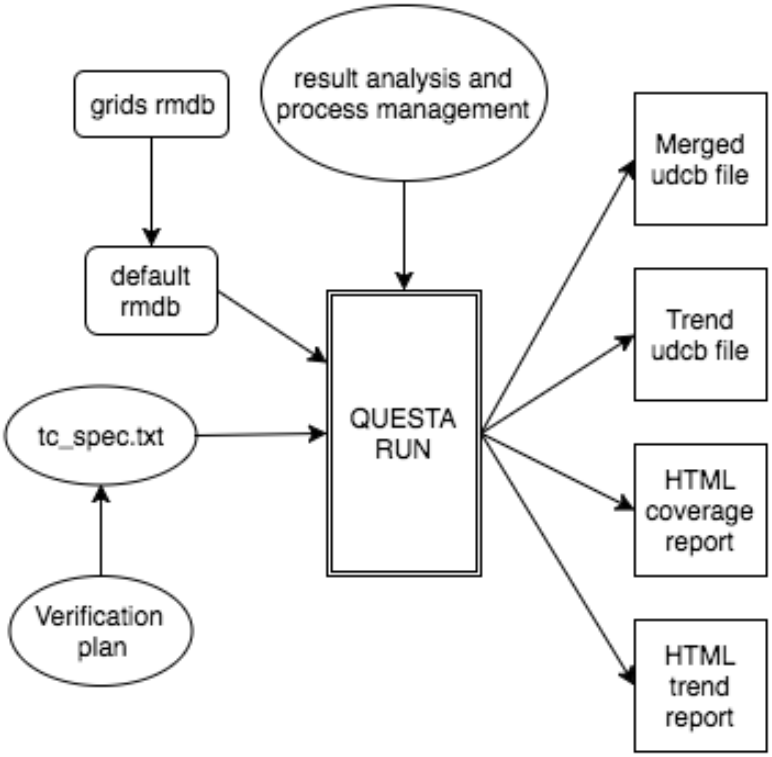


Figure 2.17: qme run management (5)

The variables that are managed are the ones shown in section 5 (extracted from (5)) .

For the re-compilation options, QME allows to compile only those affected libraries that have been affected either in the rtl or tb files.

When the simulation is run, QME besides allows debug levels. They are shown in table 2.4

Debug	Visibility	Class	Schematic	UVM	Constraints	Postsim	Assertion	FSM
LOW	ARGS	no	no	limited	no	no	no	no
MEDIUM	ARGS	yes	no	yes	no	no	yes	yes
FULL	ARGS	yes	yes	limited	yes	yes	yes	no

Table 2.4: Debug levels on QME

There are some setting that are called "hooks" that allow to customize some parts of the flow. These "hooks" are shown in the figure 2.17 and are explained below:

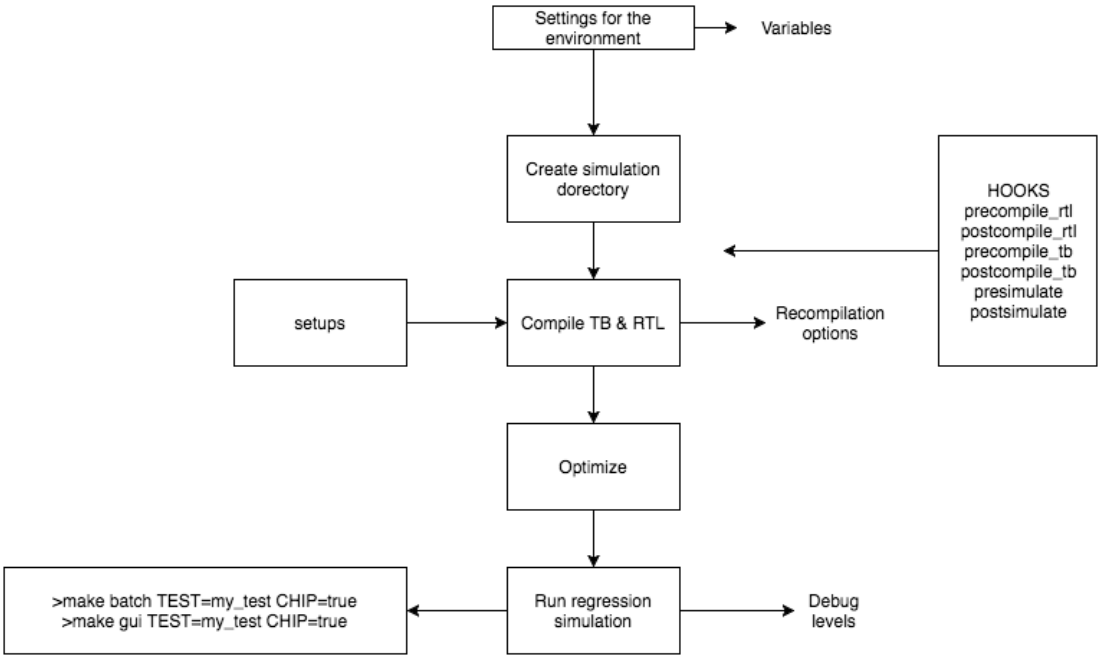


Figure 2.18: General QME flow (5)

- recompile_rtl: The target is executed before compiling the RTL code. (auto-generation of code)
- postcompile_rtl: This target is executed after compiling the RTL code.
- precompile_tb: This target is executed before compiling the TB code. (auto-generation of registers)
- postcompile_tb: This target is executed after compiling the TB code.
- presimulate: This target is executed before starting a simulation.
- postsimulate:This target is executed after simulations.

2.4.3 Block level and Chip level

Block level is the default configuration for QME. A block means the DUT that is considered unit. Therefore chip level is the composition of different blocks (Se figure 2.19 for a description of this concept)

One important difference is that in chip level there is not code coverage since this is performed in block level

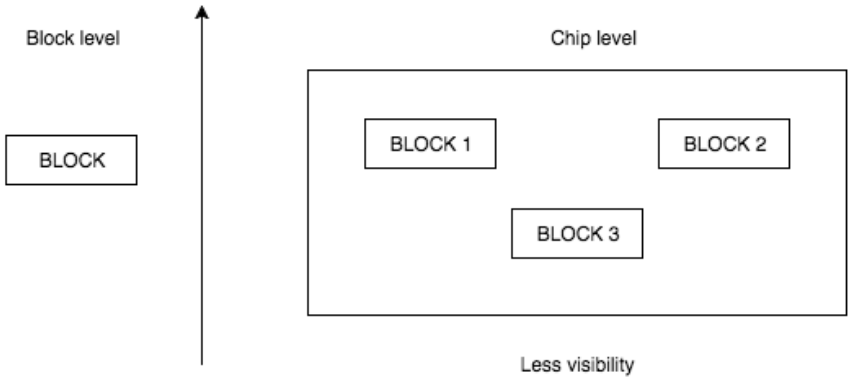


Figure 2.19: Block and chip level as setting for QME (5)

In this way, for instance when working with big blocks a pre-compile version of a FPU can be loaded and the turn around time can be improved.

2.4.4 QME - plugin

When doing research in the possible aspect regarding the integration of jenkins and VRM, it was observed that jenkins uses plugins to present the coverage data, However this data is only about the code coverage regarding software process development, but not hardware. Among the plugins that jenkins uses for this purpose are Cobertura and Emma (for java development), or clover. However since these are plugins that work only with software, then cannot be applied to the hardware development (as far as this research has gone). However since there are the html reports already given by vrm, then a html plugin could be implemented in order to set this reporting directly into jenkins. Therefore some options for it were HTML PUBLISHER PLUGIN or JUnit that supports report based on html reports or xml format.

Another important fact here is that QME has done an important contribution in this aspect since it has already developed a plugin that contributes to the integration of verification coverage data into the CI development process.

Figure 2.20 shows the report scheme made in the Jenkins console. This information is extracted directly form QME.

Figure 2.20: Basic report offered by QME to the jenkins capabilities

The main characteristics of this report are in the general coverage (that in this case is code coverage), number of tests that have passed and the test result trend.

3

Verification Flow

This sections presents designs for the flow based on the VRM capabilities and in the possible potential integrations of jenkins and VRM as a tool for regression management systems in the verifiaction-development phase.

There are some questions that are going to be described and are shown below:

- Should it be a good idea to transform the current development process into a VRM based approach? which are the positive and negative aspects?
- Which features of VRM can be implemented into CI so that the process can be improved?
- Does VRM really fit into a CI approach? How good is VRM while working in a group?
- How flexible is the system (Jenkins) for configuring different regression runs?. In the case of VRM it can be implemented through inheritance among runnables in the RMDB.
- Are Jenkins and VRM functionally equivalent technologies?

How good is VRM while working in group? With jenkins, the system is made on the basis that each member of the team will commit to the repository from which Jenkins performs in a regular basis the building of the entire system and runs the regression system. On the other hand, VRM builds also the entire system since this is a management tool for regression systems, however it does not have capabilities for building systems based on a subversion system.

3.1 Current development process - Continuous Integration

As it has been already discussed, Jenkins (CI build server) is a powerful tool that allows to build and to do regression to the systems. In this study case, the flow for the software development is as shown in the figure 3.1.

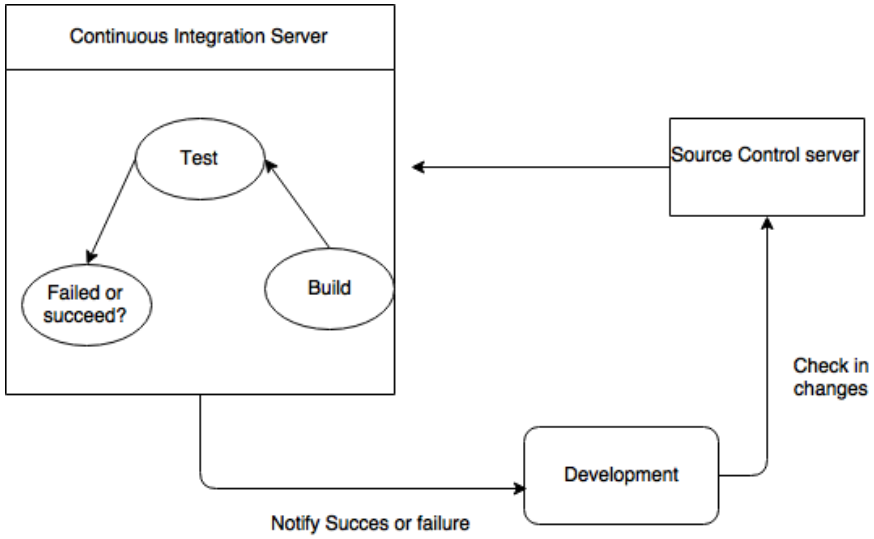


Figure 3.1: Continuous Integration flow

In the development face, every time there is a new change and it is committed to the source control server, which in this case is SVN (Subversion Control Server), then the Continuous integration server automatically build the design files with the specified testbenches and after it has finished, checks the status of the UCDB for checking if the test has passed or not. Based on the results, the development process is notify about the last integration of the system. If it does not works if the test did not pass, then the debugging process start. In this contex, the building, running and testing are performed in a regular basis and the main goal is to check if any of the new commits has broken down the system.

3.2 Proposed verification flow for process improvement

Based on the capabilities of VRM and in the current verification flow of Jenkins, 4 options for process improvement have been created and implemented for the given IP

under verification.

1. Option 1 - Implementation -VRM based: Implements a platform based ONLY on VRM, In here this process will take as core base the VRM capabilities. However, If a system is based only on VRM then a Continuous integration approach cannot be implemented since VRM does not have repository-monitoring capabilities.

2. Option 2 - Implementation - Jenkins and vrm (not QME): This option uses Jenkins as a tool that triggers regressions runs that have been already implemented in VRM. Figure 6.1 shows this approach. Here there are several advantages that could be added to the flow, among them several features of VRM (see section 2.3.3) such as merging, reporting, triage report generation, flexibility while defining different regression runs configurations through inheritance capabilities. However for this approach, the entire project, in terms of regression suite, has to be defined under RMDB databases.

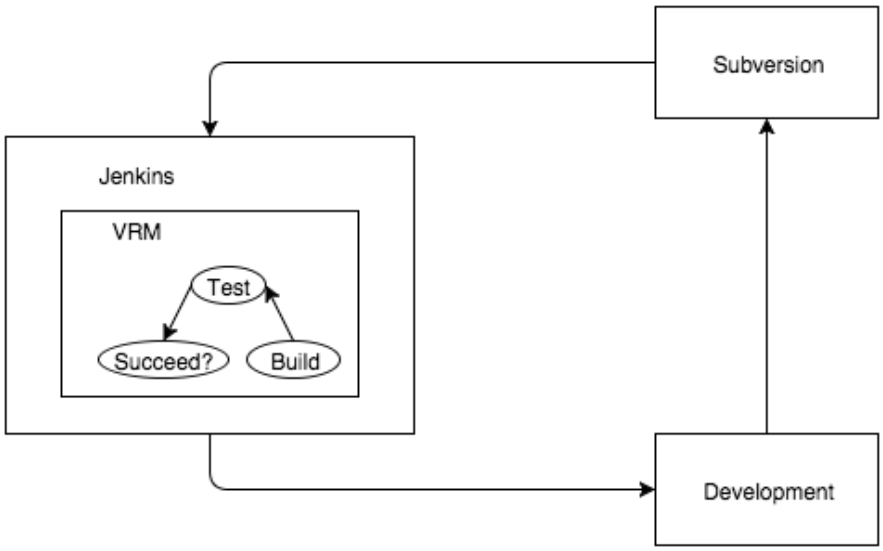


Figure 3.2: Process improvement - Option 2. In this approach, Jenkins triggers the already implemented regression suite defined in the RMDB.

At the beginning it seems that these 2 technologies (Jenkins and VRM) where competitive in the context that they do almost the same things that is build, run and report on the regression systems. Actually each of these ones allow to manage regression systems. The main big difference is that jenkins does not know

anything about functional verification metrics (but VRM has great capabilities on it) and VRM does not allow a Continuous Integration Approach for hardware development (Jenkins has been specially created for this purpose)

Therefore the question is, can these 2 technologies be applied in the same flow? Since Jenkins allows to run almost any program, therefore it can also run VRM. Therefore, at first glance it seems that both technologies, if mixed in an appropriate way, can be complementary and create a strong verification-development flow. Regarding this point of mixing these two technologies, it is important to highlight the plugin that Mentor Graphics releases on March of 2016 in which VRM is integrated in Jenkins. Here Mentor Graphics point out that both technologies are in fact complementary and if both technologies are applied, powerful solutions can be given to the testing and building process (In this case the QME is applied. see section 5). Actually this is the option 4 that uses QME and jenkins in the verification process.

In this option, Jenkins will run VRM scripts (rmdb data bases). VRM will run the regression systems, getting coverage data and possible error or failure data. At the end of the regression run, the user access a report folder in which the report coverage is presented in html format.

In this context, the verification and development flow conserve their Continuous Integration nature and besides it is improved by the capabilities of VRM, especially those regarding verification metrics.

This approach was implemented. Please refer to the section 4.3.2 for more information. However this option only implements the regression system under the created rmdb database. In this case it is a custom rmdb for this particular project and does not present any general template for which other projects could be specified.

3. Option 3 - Implementation - VRM post layer: The third options includes the CI approach that is currently being using (see figure 3.3), Since in this case there is no definition of RMDB database for configuring the regression system, the advantages that could be applied to the current system are limited. Therefore, the capabilities of merging and reporting are added as an independent component

that works based on the already given UCDB database that comes from the regression runs triggered by jenkins. This implementation will potentially improve the verification flow for every project that is currently being performed under the CI approach regarding the presentation and coverage generation. However the capabilities of vrm that can be applied are limited to merging and report generation.

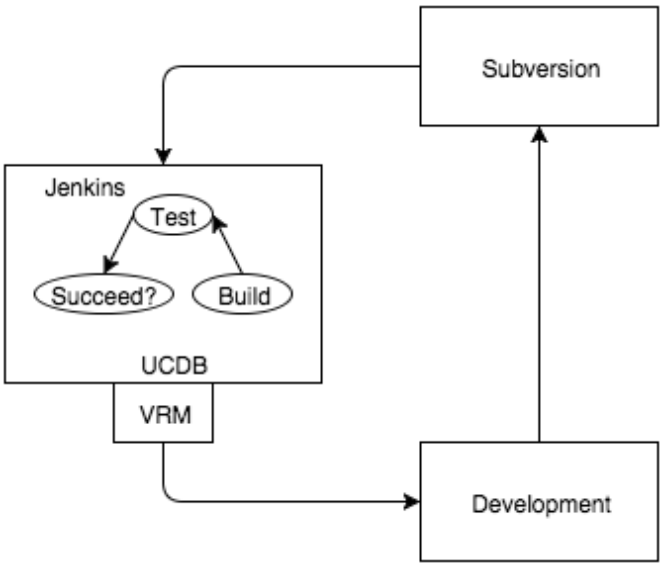


Figure 3.3: Process improvement - Option 3. In this approach, Jenkins triggers the regression suite without RMDB. At the end of the simulation, the UCDB generated will be the input to an upper level layer in which merging process and reporting generation will take place

The proposed approach is the one that is shown in the figure 3.4. This implementation can be understood like another layer that starts from the UCDB that are saved at the end of the integration face. From here, the custom rmdb will take these UCDB from the regression and the testplan as parameters and will generate the merge process and the html report.

This action will be executed as a post-build action in the Jenkins build server.

4. Option 4 - Implementation QME - jenkins plugin (3): in March of 2016 Mentor graphics presented a potential combination tool (QME) based on VRM that can be combined with Jenkins in order to manage regression systems in a continuous integration approach. The main idea here is that Jenkins gets the information

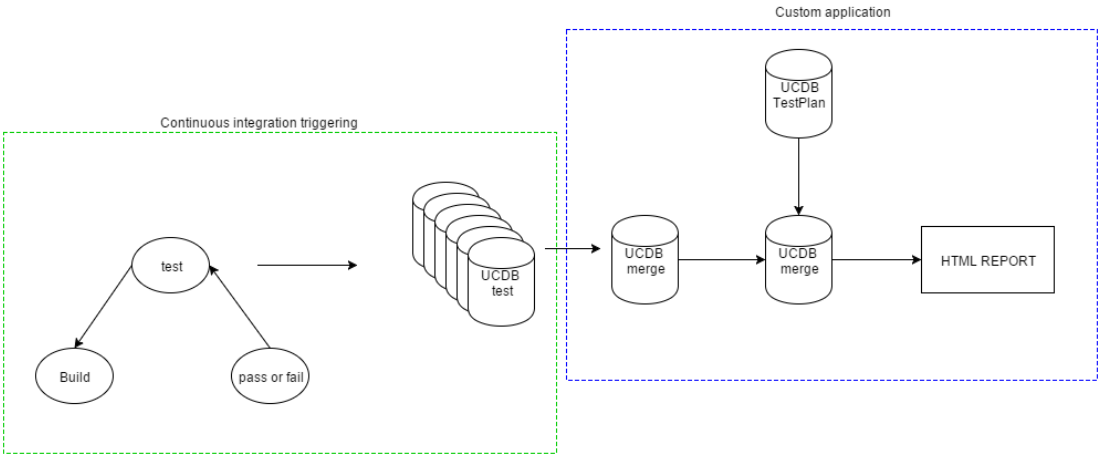


Figure 3.4: Proposed implementation for the automatic coverage extraction using post build action in Jenkins

that VRM has obtained from the regression system and then Jenkins publish this information. The implementation of this option can be checked in section 4.3.4

4

Implementation (exploratory approach) -SPI for the given options

This chapter shows the implementation of the options described in section 3 for the given SPI[1]. Here the description of the implementation is presented, showing the results from the report generation that vrm offers. Besides an introductory analysis is presented. This analysis is presented in more detailed in section 6. Figure 4.1 shows the implementation of the flow that is taken in this implementation. The flows starts with the given testplan. From here, and once the simulations are done, the UCDBS (coverage data base) are taken into the flow for the merging process and report generation (trend, coverage, triage). However in this implementation no triage report is generated since there is not development process (the IP is already developed and verified) therefore there is no bug fixing. However it is stated here as a tool that can be implemented in the flow.

4.1 IP under verification

The given IP is a serial peripheral interface (spi) as depicted in figure 4.2. This project was provided with the design files and the respective testbenches (from previous semester project[2]). Based on the documentation and the test use cases, the

[1] section extracted from (6)
[2] section extracted from (6)

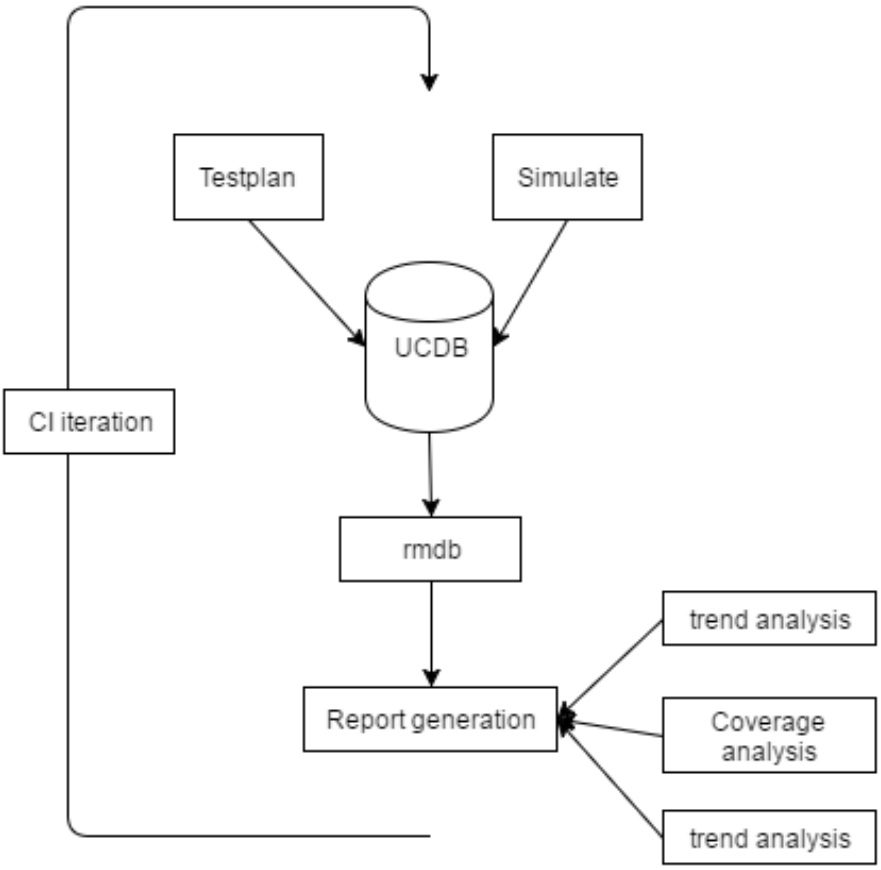


Figure 4.1: General implementation flow for the options described in section 3. This scheme shows the flow under which these tasks are performed. However for each option some variations are inserted

linking between the elements in the IP and the entries in the testplan was performed (from previous semester project). At this point, the documentation summarises a set of tasks that are implemented in verilog in the given IP.

4.2 Test plan

The test plan[1] describes the top level and SPI level

Once the testplan is made, it is exported to a XML format that will be used in the automatic merging and report generation.

[1]section extracted from (6). This testplan has the same format as the one shown in appendix A.8

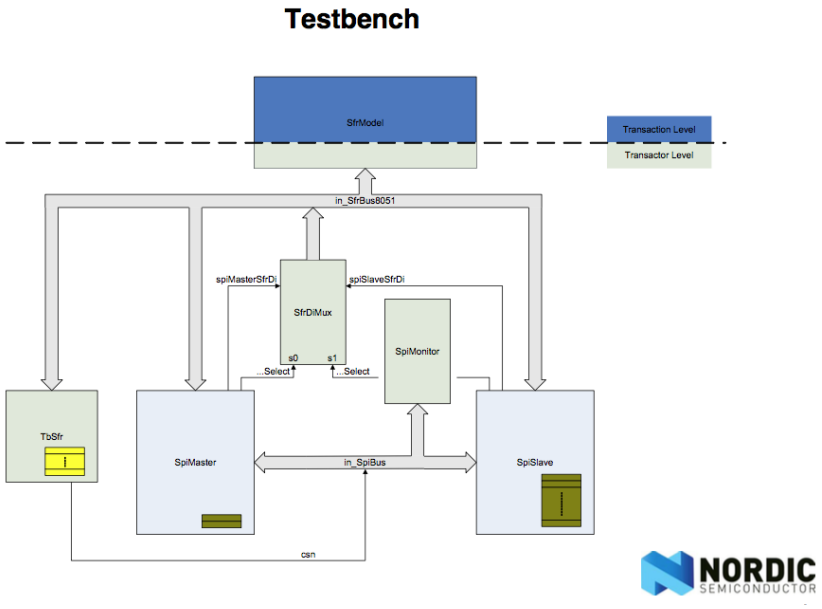


Figure 4.2: IP: Serial Peripheral Interface SPI (This IP was already given by Nordic Semiconductor)

4.2.1 Testplan SPI

For implementing the test plan for the SPI level[1], the next items have been taken into consideration:

- SPI-slave: cover the slave verification use cases specified in the documentation of the SPI given by Nordic.
- SPI-master: cover the master verification use cases specified in the documentation of the SPI given by Nordic.
- All-Assertions-that-no-match-use-case: This section details the assertions that do not match any of the verification use cases and are explicitly defined in the test plan.
- All-code-overage: This section takes into account only the code coverage (Branch, condition, toggle, Expression coverage).

[1] section extracted from (6)

- All-DU-Design-unit-coverage-type: Design unit coverage (DU) are coverage that manage the average coverage of code and functional coverage of the design units.
- All-instances: this section details the average coverage of code and functional coverage of the given instances.

4.3 Process Improvement Implementation

Once the testplan has been already defined (see appendix of (6) for more information about the testplan that was implemented for the SPI), then it is one of the inputs in the flow that are going to be implemented.

From section 3.2 there were 3 options that were implemented:

1. Implementing a platform based only on VRM.
2. The second one uses Jenkins as a tool that triggers regression runs that have been already implemented in VRM.
3. The third one uses the actual CI approach that the flow is currently using and add at the top a new layer that uses the capabilities of VRM for merging and report generation.

4.3.1 Option 1 - Implementation -VRM based

This option implements a platform based only on VRM, This process will take as core-base the VRM capabilities and will exclude the Jenkins from the flow. Therefore, the created rmdb was executed from the command promt with vrun command.

The coverage information that every option shows is the same (since every option differs only in the way in which the flow is performed as already shown in section 3). The summary of the coverage information is presented in figure 4.3.

4.3.2 Option 2 - Implementation - Jenkins and vrm (not QME)

The second option uses Jenkins as a tool that triggers regressions runs that have been already implemented in VRM. In this case, Jenkins launch a rmdb database that builds and executes the regression system

Coverage Summary by Testplan Section:

Scope	Coverage	% of Goal	Bins	Hits	% Hit	Description	Link Status	Weight	Goal
0 testplan	60.41%	60.41%	38369	12442	32.42%		Partial	1	--
1 SPI level	61.77%	61.77%	9819	7852	79.96%	this is the testplan for the SPI verification	Partial	1	100%
2 toggle	71.31%	71.31%	26406	3494	13.23%	communication between the CPU and the SPI	Clean	1	100%
3 inst power manager	39.41%	39.41%	1438	588	40.89%		Clean	1	100%
4 inst SpiMaster	69.13%	69.13%	706	508	71.95%		Clean	1	100%

Figure 4.3: Summary of the report coverage presented in each of the three options for process improvement

The tree topology of the rmdb is shown in figure 4.4. In this rmdb, there are three runnables. One of type group which is called as "general" that has inside two another members called "spi" and "top". Besides this "general" runnable has one postscript in which the merge operations are done and also the UCDB testplan generation is performed. Both the "spi" and "top" runnables perform operations for compilation and simulation. The implementation for this rmdb is in the appendix A.1

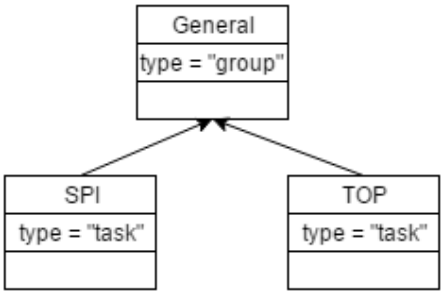


Figure 4.4: Tree topology of the rmdb for option 2 - process improvement

Every time Jenkins launches this rmdb database, then VRM build and run the simulation. At the end, it performs the merging and report generation. The directory topology under which this building works is shown in figure 4.5

Inside the rtl folder is located the rmdb that builds and run the regression run. The top runnable is of type group and is called general. Inside this folder are located the "spi", "top", and "report" folders. In the spi folder is located the files regarding the spi level for building and testing. In the top folder are located the files of the top level testbenches. Finally in the report folder are located the html files regarding the regression run report and the coverage report.

The report that is generated for the rmdb execution is shown in figure 4.6 and 4.7.

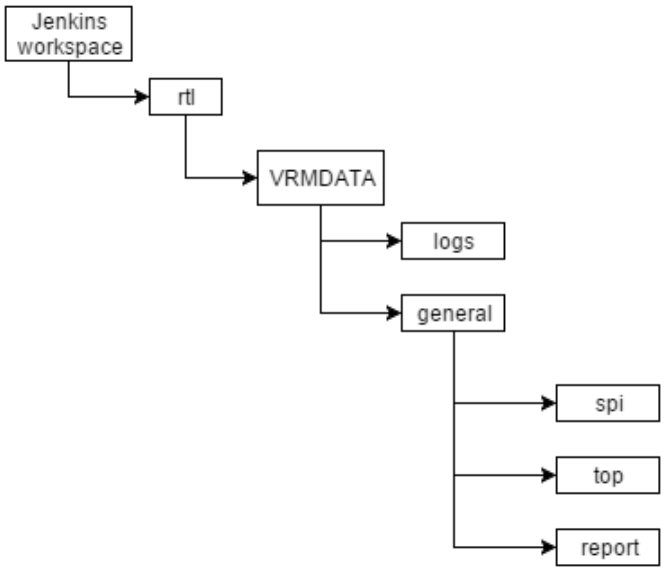


Figure 4.5: Directory topology in Jenkins for the building of option 2

Figure 4.8 shows a more detailed status report for the postscript (information regarding execution time, identification name and executed commands are presented)

Pass/fail status summary:

Status	Count
Passed	3
Failed	0
Incomplete	0
TOTAL	3

UCDB status summary:

Status	Count
Ok	0
Warning	0
Error	0
Fatal	0
Missing	0
Merge Error	0
Unknown Error	0
UCDB Error	0
TOTAL	0

Non-UCDB status summary:

Status	Count
Passed	3
Failed	0
Timeout	0
TOTAL	3

Pending status summary:

Status	Count
Pending	0
Running	0
Suspended	0
TOTAL	0

Action completion summary:

Status	Empty	Passed
execScript	0	2
postScript	0	1
preScript	1	0

Figure 4.6: Option 2. HTML report for the rmdb execution - part1: it shows the general status information for the execution of the rmdb. In this case, there are 3 scripts that passed with no errors. 2 are execScripts and 1 is a postScript

At this point there are some questions that have arisen. These questions are in fact

Action status:

Show All | Show Passed | Show Failed

Runnable	Script	Testname	Coverage	Date/Time	Status
general	preScript	--	--	Tue Mar 15 11:16:46 CET 2016	Empty
general/spi	execScript	--	--	Tue Mar 15 11:17:01 CET 2016	Passed
general/top	execScript	--	--	Tue Mar 15 11:19:09 CET 2016	Passed
general	postScript	--	--	Tue Mar 15 11:19:15 CET 2016	Passed

Figure 4.7: Option 2. HTML report for the rmdb execution - part2: Each runnable status that conforms the rmdb database is presented.

Verification Run Manager Status Report

Action	general/postScript
Status	Passed
Log File	/work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/rtl/VRMDATA/general/postScript.log

Per-test attributes:

Attribute	Value
Action context (Runnable)	general
Script name	postScript
Date/Time	Tue Mar 15 11:19:15 CET 2016
UCDB File	--
Queued	02.00
Elapsed	03.00
CPU Time	--
Hostname	jenkinsworker.nordicsemi.no
Username	--
Seed	--

Script (defined in Runnable 'general'):

Figure 4.8: Option 2. HTML report for the rmdb execution - part3: A detailed status report for the postscript. A detailed report for each script (execScript, postScript, preScript) is generated.

analyzed in the section 6

1. Which benefits does this approach have?

2. Does it really makes sense this kind of integration between VRM ad Jenkins?

3. Could this approach still be applied to a CI process for team working?

4.3.3 Option 3 - Implementation - VRM post layer

This option uses the current flow of Continuous Integration and add at the end of the flow additional VRM capabilities for merging and reporting (see figure 3.3)

Regarding this implementation there are several questions that are faced:

- Which capabilities can be implemented here? Since the regression run is already implemented in Jenkins, then the VRM capabilities that are applied are limited to merging and report generation. Another features like debug mode or rerun of failed test can not be implemented.
- How is it going to be integrated to the current verification flow? Jenkins has the option to add post-build actions. In this case, the post-build actions are perform by VRM scripts that run the merging and report generation actions
- Which approach for the merge is the best that fits the requirements? In this case, since the build and running of the test are not made by VRM, then automated merging or incremental merging cannot be applied. Therefore the merging types that suits are DIY merge and List Based Merging. However the one that is used is DIY since it has a more complete documentation and examples.

In this use case, the UCDBs come from the SPI level testbench, top level testbench and from the testplan. The generated VRM script performs the merging and the generation of the testplan UCDB from the XML excel tesplan. The report generation is performed out of the VRM, in a sh-type shell. Figure 4.9 shows the operations performed by the VRM script in which first it is performed the merging from the testbenches. Later, the testplan UCDB is generated from the XML-testplan format. Finally the testplan UCDB database is merged in the HTML report.

The VRM script has 1 runnable with parameters and 1 execution script as described below:

1. Parameters: the parameters that are defined here are as shown below:
   (a) mergefile: The merge file name output

(b) tplanfile: This is the path specification for the XML testpan file that will be transformed in a UCDB file format.

(c) tplanoptions: This specifies characteristics of the XML testplan file, which in this case it comes from an excel format.

(d) tplanucdb: specifies the name of the UCDB testplan output

2. ExecScript: The steps that this script executes are the ones depicted in the figure 4.9. The main procedures are merging and report generation. Appendix A.2 shows the rmdb database implemented for this process improvement option

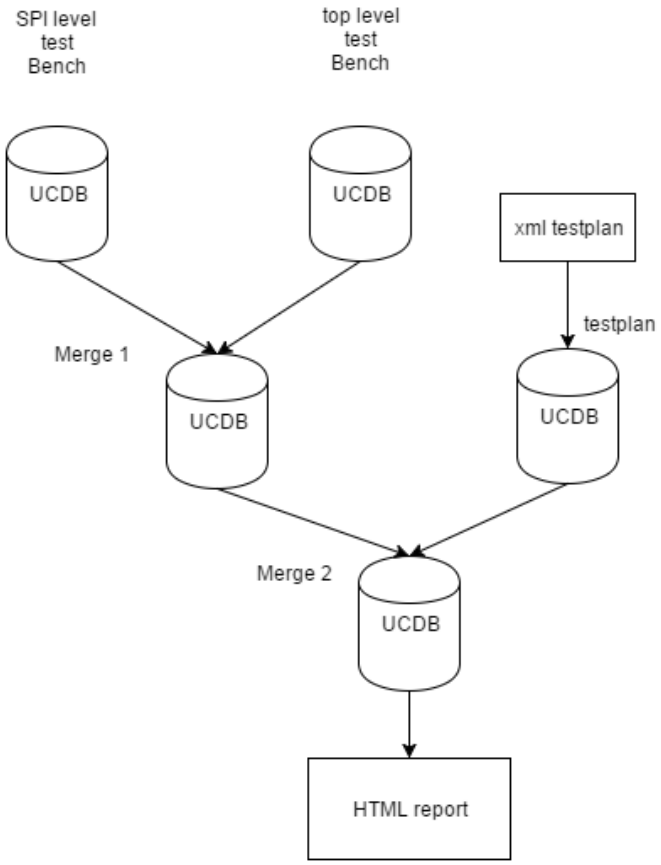


Figure 4.9: Option 3 - process improvement: operations done in the execScript- Merging and report generation

After the process was run, the report generation was extracted and located in a folder inside the VRMDATA in the jenkins workspace. Figure 4.10 shows the directory

tree topology that this option performs. In the rtl folder is located the simulation and compilation for the top level testbench. In the spi folder is located the spi level simulation and compilation. In the merge folder is located the rmdb that was created. Once this rmdb is run (As a post-build execution in jenkins build server), then it creates the working directory VRMDATA under which the merging and reporting is performed. Once this post-build execution is performed, then the report can be accessed trough the directory path of the report folder.

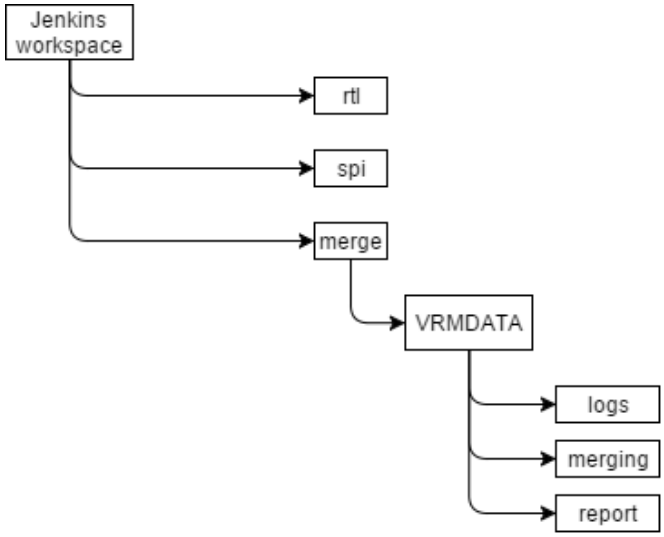
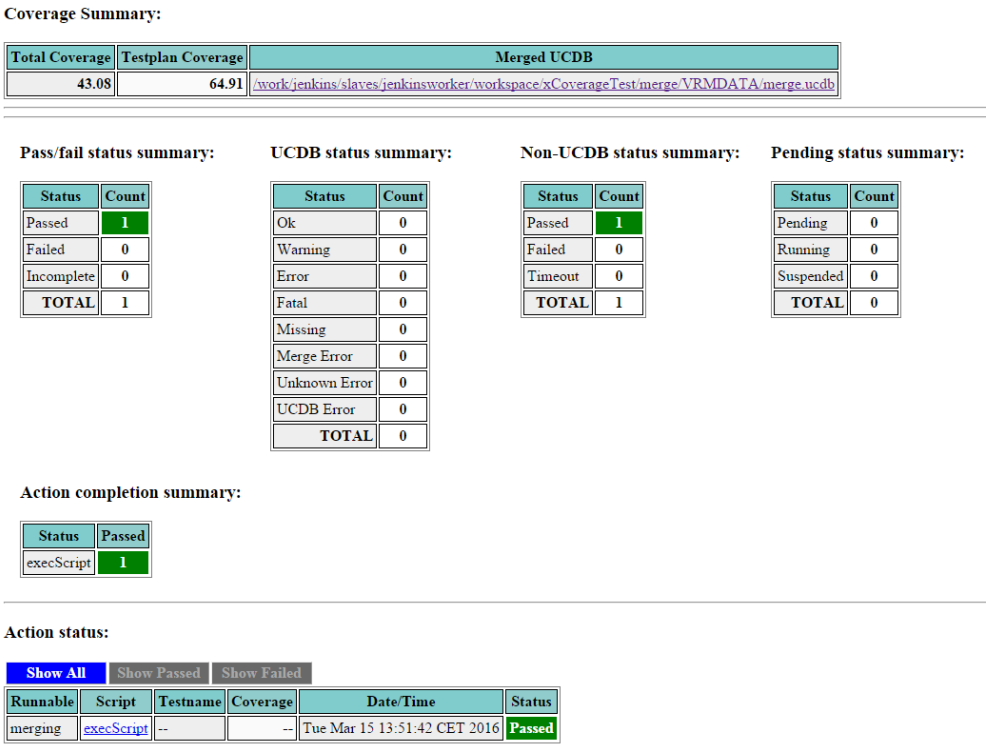


Figure 4.10: Directory tree topology performed in the option 3 for process improvement.

Figure 4.11 shows the status report for the rmdb performed. It differs from the one in option 2 mainly in the number of runnables, since here no compilation or simulation is performed under the rmdb. This runnable has been cheked as passed. The detailed status report is shown in figure 4.12 in which the execution script (which its commands) are also presented.

4.3.4 Option 4 - Implementation QME - jenkins plugin (5)

Up to this point, it seems that among the 3 options mentioned before, and after the released of QME, it is a good idea (and the path this process improvement has to take) to mix these two technologies (VRM and jenkins) in the same flow. Therefore a further and detailed implementation of QME is done in section 5. The implementation is done based on the demo offered by Mentor Graphics with a FPU and the already given design



Coverage Summary:

Total Coverage	Testplan Coverage	Merged UCDB
43.08	64.91	/work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/merge/VRMDATA/merge.ucdb

Pass/fail status summary:

Status	Count
Passed	1
Failed	0
Incomplete	0
TOTAL	1

UCDB status summary:

Status	Count
Ok	0
Warning	0
Error	0
Fatal	0
Missing	0
Merge Error	0
Unknown Error	0
UCDB Error	0
TOTAL	0

Non-UCDB status summary:

Status	Count
Passed	1
Failed	0
Timeout	0
TOTAL	1

Pending status summary:

Status	Count
Pending	0
Running	0
Suspended	0
TOTAL	0

Action completion summary:

Status	Passed
execScript	1

Action status:

Show All | Show Passed | Show Failed

Runnable	Script	Testname	Coverage	Date/Time	Status
merging	execScript	--	--	Tue Mar 15 13:51:42 CET 2016	Passed

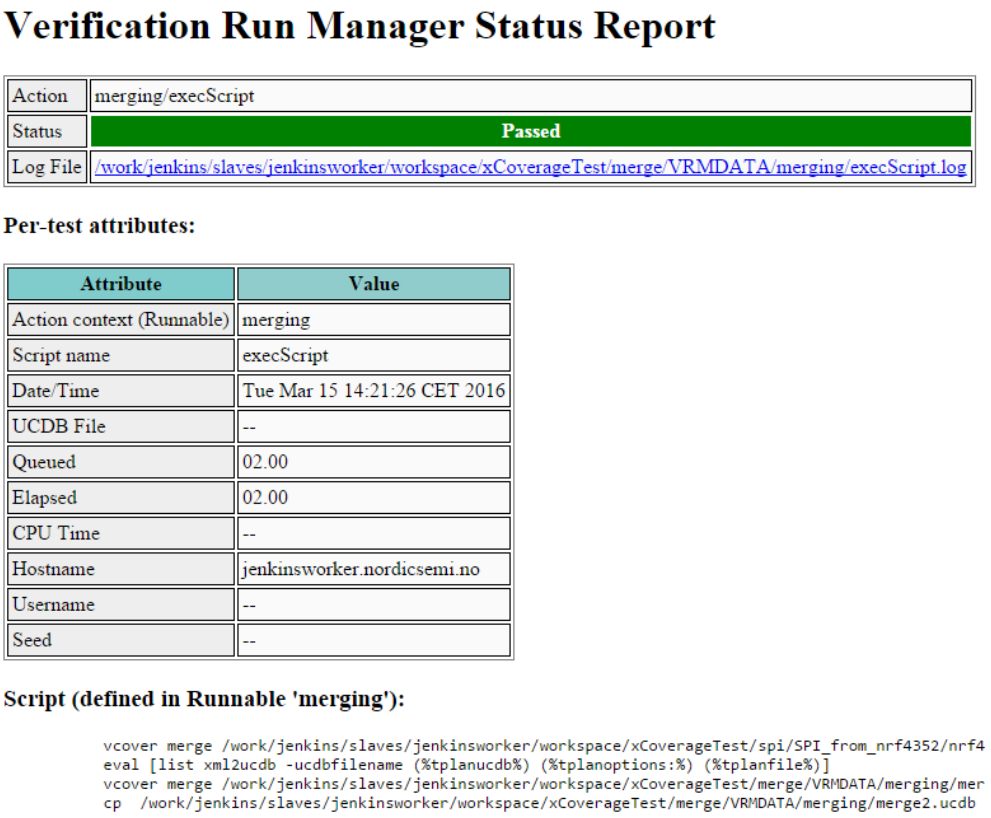
Figure 4.11: Process Improvement option 3. HTML report for the rmdb execution:

of the SPI. The results are for the reporting (coverage, trend and regression status report) the analysis is related to the potential improvements that actually QME-Jenkins (with plugin) gives to a coverage driven methodology in the development-verification process in the context of a Continuous Integration practice (See section 6.3 for more details on this analysis)

This SPI is also implemented in the flow presented by QME as described in section 5.2.1 in order to do a further analysis.

4.4 Qualitative summary of results

This chapter has shown the exploratory implementation of the SPI for the proposed implementation options regarding the implementation of VRM (and its possible integration with the CI practice). The main purpose of this exploratory implementation is to see how the VRM performs for a given design and how VRM can be integrated into CI.



Verification Run Manager Status Report

Action	merging/execScript
Status	**Passed**
Log File	/work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/merge/VRMDATA/merging/execScript.log

Per-test attributes:

Attribute	Value
Action context (Runnable)	merging
Script name	execScript
Date/Time	Tue Mar 15 14:21:26 CET 2016
UCDB File	--
Queued	02.00
Elapsed	02.00
CPU Time	--
Hostname	jenkinsworker.nordicsemi.no
Username	--
Seed	--

Script (defined in Runnable 'merging'):

```
vcover merge /work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/spi/SPI_from_nrf4352/nrf4
eval [list xml2ucdb -ucdbfilename (%tplanucdb%) (%tplanoptions:%) (%tplanfile%)]
vcover merge /work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/merge/VRMDATA/merging/mer
cp  /work/jenkins/slaves/jenkinsworker/workspace/xCoverageTest/merge/VRMDATA/merging/merge2.ucdb
```

Figure 4.12: Process Improvement. HTML report for the rmdb execution:

This exploratory section shows important findings that are summarized as shown below:

- The most important finding up to this point is that VRM can be integrated to the CI practice either in a full approach (option s) or partial approach. However the partial approach can only implements the reporting of metrics and status report. Therefore the benefits that VRM (under the scope of automation regarding coverage closure evaluation, debug capabilities, rerun of failed tests) can be potentially applied to the flow of verification under a CI practice

- As expected, VRM has strong capabilities for metric coverage closure evaluation. Besides the report for the regression system (status regression report) with its respective scripts execution report.

it is important to point out that in this exploratory section, there was not imple-

mentation of the triage reporting and the debug capabilities of VRM (as specified in the background chapter, see section 2.3.3.3 and 2.3.3.5) since there was not developing process, therefore a triage analysis, for the given design, is not helpful in the analysis of this feature of VRM.

Further analysis of these results are given in section 6

5

QME implementation (exploratory approach) with Floating Point Unit demo

QME as a makefile environment creates the necessary scripts for building regression systems. Therefore this chapter presents the implementation of this system as a study case in which the VRM capabilities (that were described in section 2.3.3) are implemented with the given configuration settings.

As a reminder from background section, figure 5.1 shows the flow that QME implements. This section therefore goes trough every step from the configuration, getting the testplan and until the report generation and integration in Jenkins as a CI tool.

In this case, the qme is applied to the FPU and also to the SPI that was already taken in section 4

The main purpose of this section is therefore the exploratory implementation of QME for the SPI and FPU designs. The aim is to explore the capabilities of VRM that are implemented in QME and how it integrates in the CI practice.

5.1 QME setting and implementation

This section presents the settings that were made in QME and the results of coverage and the flow that qme implements.

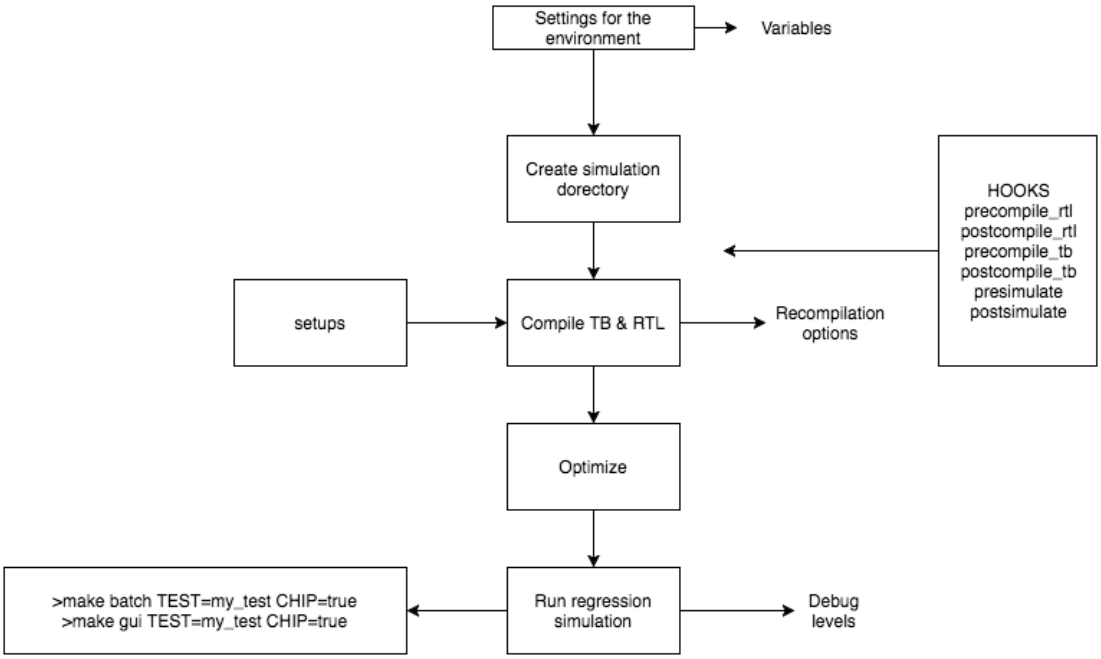


Figure 5.1: General QME flow

5.1.1 QME test plan

The testplan that supports the QME-Jenkins flow is the one presented in the appendix A.8. This verification plan is part of the demo provided by QME. Among the most important aspects of this test plan are the ones that are described below:

1. Code coverage such as Branch, Condition, FMS, Expression
2. Core Functionality such as Exeption signals, Invalid operations, NaN1, division by zero, Round operations, Operations on floating point numbers
3. Pipelines - operation delays
4. Pin interface - Valid opcode on pins

5.1.2 QME and Jenkins setting

After the project run successfully in the local machine, the next step was to set it up in Jenkins. The setting is shown in the appendix A.3.

The source_me.bsh defines the general variables ((5))

```
1 setenv QME_HOME $PWD
2 setenv QME_PROJECT_DEFAULTS $PWD/.../Makefile.project.defaults
3 setenv QME_PROJECT_HOME $PWD/examples
4 setenv QME_SIM_SETTINGS_DIR sim
5 setenv QME_SITE_SETTINGS path_to_your_sites_qme_settings_dir
6 setenv QME_SCRATCH_HOME $PWD/examples/scratch_home
7 setenv PYTHONPATH $PWD/uvmf/templates/python
8 setenv PATH $PATH":$PWD/scripts"
```

The description of each of this variables is as shown below:

- QME_HOME: It is where the location of the QME is located
- QME_PROJECT_HOME: Here it is located the top level directory where the projects are stored
- QME_SITE_SETTINGS: Here the site specific settings are located
- QME_SCRATCH_HOME: site for the simulation directories
- QUESTA_HOME: home site of QuestaSim

The next line describes the creation of the fpu block

```
1 #create a simulation directory using the
2 # create_questa_simdir.pl command.
3 create_questa_simdir.pl -simdir=simdir -block=fpu
4 #-l=${WORKSPACE} -f
```

In this case, simdir is the name of the simulation and fpu is the name of the block that is defined.

The next line defines the commands for doing the compilation for the regression system. Here it specifies to make it as a regression batch, do not notify to email and the location of the TC_SPEC file.

```
1 make regression_batch regression_dvt_junit NOCOLOR=1
2 SEND\EMAIL=0
3 TC_SPEC=${QME_HOME}/examples/fpu/sim/tc_spec_qa.txt
```

A very important setting in this flow is the file tc_spec_qa.txt that defines the information necessary for the regression.

```
1 #<SETUP> <TESTNAME> <No of runs> <list of seeds>
```

```
1 DEFAULT:VPLAN_EXCEL=$QME_PROJECT_HOME/fpu/vplan/
2 fpu_vplan_excel.xls
3 # This one is used if USE_EXCEL=0
4 DEFAULT:VPLAN_XML=$ ... /fpu/vplan/fpu_vplan_excel.xml
5 DEFAULT fpu_test_patternset 1 0
6 DEFAULT fpu_test_neg_sqr_sequence 5 random
7 DEFAULT fpu_test_simple_sanity 5 random
8 #DEFAULT fpu_test_failing 15 random
9 #DEFAULT fpu_test_failing 4 0 1 3 4
10 DEFAULT fpu_test_simple_sanity 5 12
11 DEFAULT fpu_test_random_sequence 15 random
12 DEFAULT fpu_test_sequence_fair 15 random
```

Here this file defines the location of the verification plan for the FPU. (see section A.8 for details in this verification plan) and other settings as defined below:

1. Name of parameter set-up
2. Name of test
3. Number of simulations
4. List of seeds

Up to this point these are all the setting that have been managed in QME. Among them the more relevant are the listfiles that specifies the list for the rtl and tb sources. Also the tc_spec file defines the test that are going to be run and the number of times it is going to be run.

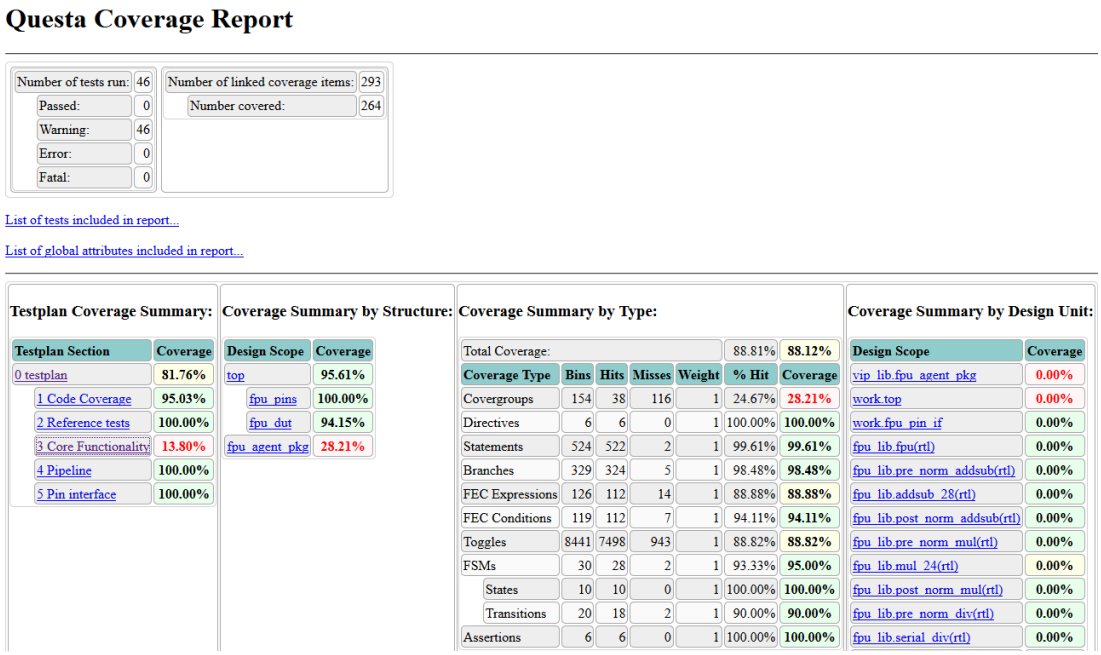
5.2 QME results

This section present the information for the report generation, regression status, trend analysis.

When using QME in Jenkins under a FPU, Jenkins shows the report for coverage, trend analysis and regression system for the default setting and for the setup 1.

Figure 5.2 shows the results for the coverage under the default setting. At this point this figure defines the coverage based on the given testplan. However this is the setting by default.

The results are highly different from the ones presented in the setup 1 (see figure 5.3). As it was already specified, if for a given block there are different setups, the coverage differs from both (because of the parametrization that each setup specifies)



Questa Coverage Report

Number of tests run:	46	Number of linked coverage items:	293
Passed:	0	Number covered:	264
Warning:	46		
Error:	0		
Fatal:	0		

List of tests included in report...

List of global attributes included in report...

Testplan Coverage Summary:

Testplan Section	Coverage
0 testplan	81.76%
1 Code Coverage	95.03%
2 Reference tests	100.00%
3 Core Functionality	13.80%
4 Pipeline	100.00%
5 Pin interface	100.00%

Coverage Summary by Structure:

Design Scope	Coverage
top	95.61%
fpu_pins	100.00%
fpu_dut	94.15%
fpu_agent_pkg	28.21%

Coverage Summary by Type:

Total Coverage:					88.81%	88.12%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Covergroups	154	38	116	1	24.67%	28.21%
Directives	6	6	0	1	100.00%	100.00%
Statements	524	522	2	1	99.61%	99.61%
Branches	329	324	5	1	98.48%	98.48%
FEC Expressions	126	112	14	1	88.88%	88.88%
FEC Conditions	119	112	7	1	94.11%	94.11%
Toggles	8441	7498	943	1	88.82%	88.82%
FSMs	30	28	2	1	93.33%	95.00%
States	10	10	0	1	100.00%	100.00%
Transitions	20	18	2	1	90.00%	90.00%
Assertions	6	6	0	1	100.00%	100.00%

Coverage Summary by Design Unit:

Design Scope	Coverage
vip_lib.fpu_agent_pkg	0.00%
work.top	0.00%
work.fpu_pin_if	0.00%
fpu_lib.fpu(rtl)	0.00%
fpu_lib.pre_norm_addsub(rtl)	0.00%
fpu_lib.addsub_28(rtl)	0.00%
fpu_lib.post_norm_addsub(rtl)	0.00%
fpu_lib.pre_norm_mul(rtl)	0.00%
fpu_lib.mul_24(rtl)	0.00%
fpu_lib.post_norm_mul(rtl)	0.00%
fpu_lib.pre_norm_div(rtl)	0.00%
fpu_lib.serial_div(rtl)	0.00%

Figure 5.2: Coverage report for FPU as default setting

The status report shown in figure 5.4 is explained as shown below:

- The regression is made of 55 scripts under the rmdb database. From here 53 are excecScript, 1 is postScript and 1 is preScript.
- In total there were 46 tests. No one has failed but there are warnings on it.

A very important improvements that QME-JENKINS shows is that it presents trend reports. Figure 5.5 shows the one for the FPU. However since this is not a development of a FPU and the code is already given, the trend shows the same coverage for all the times the regression has been performed. Here it shows 3 reports.

5. QME IMPLEMENTATION (EXPLORATORY APPROACH) WITH FLOATING POINT UNIT DEMO

Questa Coverage Report

Number of tests run:	3	Number of linked coverage items:	293
Passed:	0	Number covered:	123
Warning:	2		
Error:	0		
Fatal:	0		
Not run:	1		

List of tests included in report...

List of global attributes included in report...

Testplan Coverage Summary:

Testplan Section	Coverage
0 testplan	37.44%
1 Code Coverage	57.72%
2 Reference tests	0.00%
3 Core Functionality	9.51%
4 Pipeline	20.00%
5 Pin interface	100.00%

Coverage Summary by Structure:

Design Scope	Coverage
top	53.53%
fpu_pins	33.33%
fpu_dut	60.26%
fpu_agent_pkg	15.79%

Coverage Summary by Type:

Total Coverage:					57.44%	49.33%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	154	20	134	1	12.98%	15.79%
Directives	6	2	4	1	33.33%	33.33%
Statements	524	460	64	1	87.78%	87.78%
Branches	329	236	93	1	71.73%	71.73%
FEC Expressions	126	36	90	1	28.57%	28.57%
FEC Conditions	119	32	87	1	26.89%	26.89%
Toggles	8441	4778	3663	1	56.60%	56.60%
FSMs	30	26	4	1	86.66%	90.00%
States	10	10	0	1	100.00%	100.00%
Transitions	20	16	4	1	80.00%	80.00%
Assertions	6	2	4	1	33.33%	33.33%

Coverage Summary by Design Unit:

Design Scope	Coverage
vip_lib.fpu_agent_pkg	0.00%
work.top	0.00%
work.fpu_pin_if	0.00%
fpu_lib.fpu(rtl)	0.00%
fpu_lib.pre_norm_addsub(rtl)	0.00%
fpu_lib.addsub_28(rtl)	0.00%
fpu_lib.post_norm_addsub(rtl)	0.00%
fpu_lib.pre_norm_mul(rtl)	0.00%
fpu_lib.mul_24(rtl)	0.00%
fpu_lib.post_norm_mul(rtl)	0.00%
fpu_lib.pre_norm_div(rtl)	0.00%
fpu_lib.serial_div(rtl)	0.00%
fpu_lib.post_norm_div(rtl)	0.00%
fpu_lib.pre_norm_sqrt(rtl)	0.00%
fpu_lib.sqrt(rtl)	0.00%
fpu_lib.post_norm_sqrt(rtl)	0.00%
fpu_lib.serial_mul(rtl)	87.85%

Figure 5.3: Coverage report for FPU as setup 1

- Test summary: it shows the status of the tests along the time. For this case all the tests has passed.
- Linked Bins summary. In this case it shows the number of bins that have been linked to the report and the number of links that have been already covered. This is an important measure since it provides specific information regarding the functions coverage of the FPU. At this point it seems there are almost 150 bins that have not been covered.
- Coverage summary by type: covergroup coverage, testplan coverage and weighted coverage.

5.2.1 SPI implementation in QME

In order to go further in the implementation and analysis of QME, the SPI was also set in the QME environment. For doing it first the three files are set : tb file list; rtl

Verification Run Manager Status Report

Regression started:	Fri Apr 15 09:14:47 CEST 2016
Regression finished:	(unknown)
Configuration (RMDB) file:	/work/jupa/jenkins/slave0/workspace/qme_1/simdir/default.rmdb
VRMDATA directory:	/work/jupa/jenkins/slave0/workspace/qme_1/simdir/regression_data
Command:	vrun -clean -nolocalrerun -j 2 -rmdb /work/jupa/jenkins/slave0/workspace/qme_1/simdir/default.rmdb

Pass/fail status summary:

Status	Count
Passed	55
Failed	0
Incomplete	0
TOTAL	55

UCDB status summary:

Status	Count
Ok	0
Warning	46
Error	0
Fatal	0
Missing	0
Merge Error	0
Unknown Error	0
UCDB Error	0
TOTAL	46

Non-UCDB status summary:

Status	Count
Passed	9
Failed	0
Timeout	0
TOTAL	9

Pending status summary:

Status	Count
Pending	0
Running	1
Suspended	0
TOTAL	1

Action completion summary:

Status	Empty	Passed
execScript	0	53
postScript	14	1
preScript	15	1

Figure 5.4: Status regression report for FPU as setup 1

file list; tc_spec file.

- tb file list. this is the filelist that specifies the list of files for the testbench that is going to be specified:

```
1       @library work
2       @vlogargs:+incdir+/ ... //Include/hdl
3       @vlogargs:+incdir+/... /sim/tb
4       @vlogargs:-v /... /ip_testbenches/... /Hinst.sv
5       $nrf4352_HOME/... /hdl/in_SfrBus8051.sv
6       $nrf4352_HOME/... /hdl/in_SpiBus.sv
7       $nrf4352_HOME/design/digital/Spi/sim/tb/nVip_SfrDiMux.
            sv
```

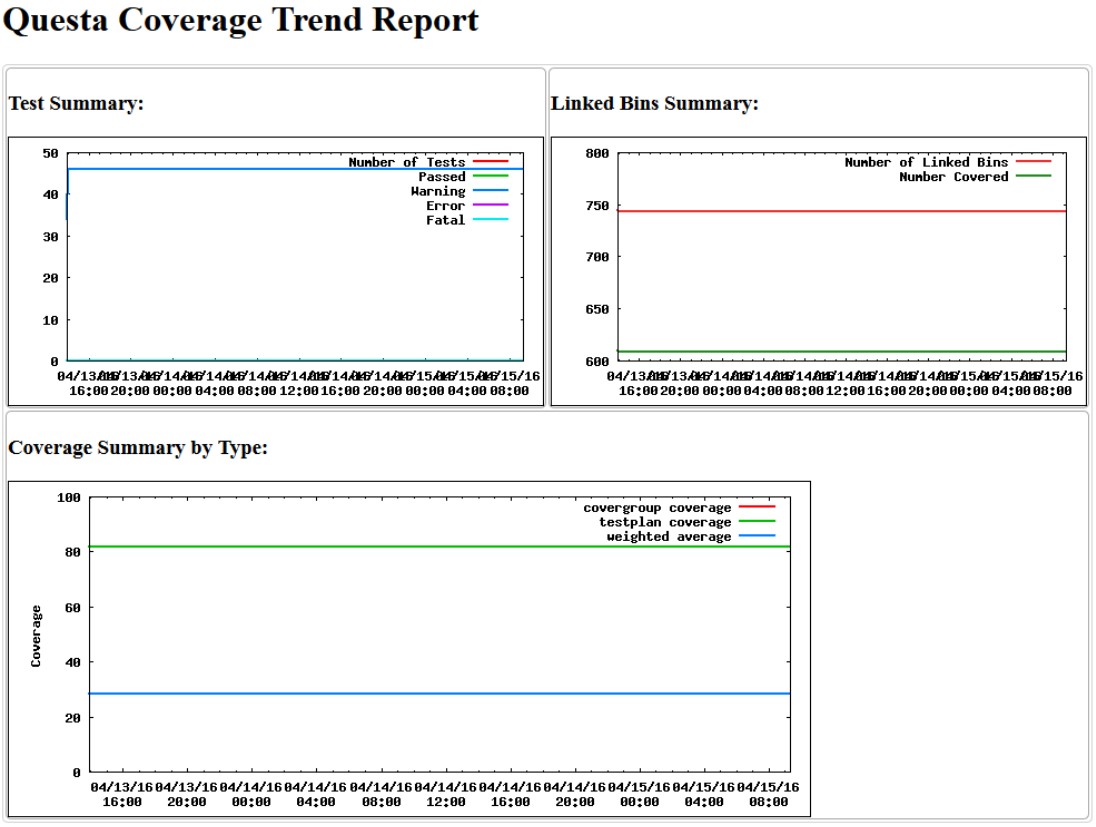


Figure 5.5: Trend analysis report for FPU as default setting

```
8       $nrf4352_HOME/design/digital/Spi/sim/tb/
            nVip_SpiMonitor.sv
9       $nrf4352_HOME/design/digital/Spi/sim/tb/nVip_TbSfr.sv
10      $nrf4352_HOME/design/digital/Spi/sim/tb/test_Spi.sv
```

- rtl file list: this are the file list for the rtl code

```
1       @library spi_lib
2       @vlogargs:+incdir+/pri/jupa/project/.../hdl
3       @vlogargs:-v /pri/jupa/project/ip_testbenches/.../
            Hinst.sv
4       $nrf4352_HOME.../Spi/SpiSlave/hdl/SpiSlaveCapture.sv
5       $nrf4352_HOME/.../Spi/SpiSlave/hdl/SpiSlaveCore.sv
6       $nrf4352_HOME/.../Spi/SpiSlave/hdl/SpiSlaveSfr.sv
7       $nrf4352_HOME/...l/Spi/SpiSlave/hdl/SpiSlave.sv
8       $nrf4352_HOME/.../Spi/SpiMaster/hdl/SpiMaster.sv
9       $nrf4352_HOME/.../Spi/SpiMaster/hdl/SpiMasterCapture.
            sv
```

```
10      $nrf4352_HOME/.../Spi/SpiMaster/hdl/SpiMasterCore.sv
11      $nrf4352_HOME/.../Spi/SpiMaster/hdl/SpiMasterSckCtrl.
            sv
12      $nrf4352_HOME/.../Spi/SpiMaster/hdl/SpiMasterSfr.sv
13      $nrf4352_HOME/.../in_SfrBus8051/hdl/in_SfrBus8051.sv
14      $nrf4352_HOME/.../in_SpiBus/hdl/in_SpiBus.sv
```

- tc_spec: this is the file that defines the tetplan that is going to be specified. Besides the setting (that in this case is default since there is just one specification for the design), the number of times that the test is going to be run (in this case 1) and the number of seeds (in this case is 0).

```
1       DEFAULT:VPLAN_EXCEL=/pri/jupa/thesis/test9_MGC3.xls
2       DEFAULT test_Spi 1 0
```

Besides a pre-simulation setting was implemented in the flow[1] in where the target is executed immediately before the simulation is started. In this case the functional coverage is also set up. See appendix A.6 for information about the implemented pre-simulation setting under QME for the FPU.

After this local setting was done, then it was integrated in the Jenkins server. The configuration is for the questasim license, the source file (which is the one for general variables for QME). In this case the block is name as nordic_spi and the simulation directory uses a generic directory simdir. Finally the regression is done in regression batch. See appendix A.7 for the implementation of the integration for the FPU-QME-Jenkins.

When the simulation is done in jenkins, it also gives the final reporting related to the coverage, trend analysis. One important feature that this reporting approach of QME is that it defines also trend analysis for different design units. In this case, for instance the trend analysis for the u_Sfr_slave design unit is shown in figure 5.6:

In this particular case, although there is not development (since the design is already done) then a coverage driven methodology can be applied to every unit design and check coverage closure

Up to this point, the implementation relates to the given options (from section 3) and the QME. It is important to see that this is an exploratory research for understating

[1]For this particular "hook"-configuration help from Mentor Graphics was required

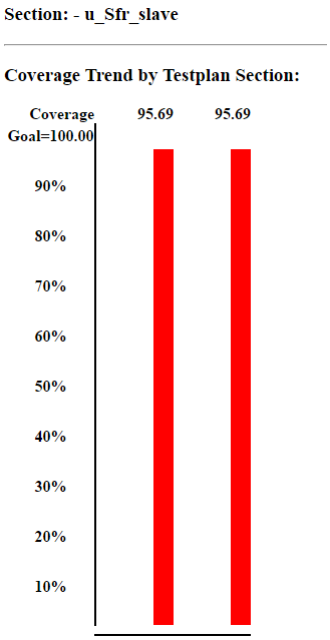


Figure 5.6: Trend analysis report for u_Sfr_slave design unit in SPI

the advantages that VRM, with their facilities gives to the flow and as a particular case the QME tool that integrates VRM in Jenkins (for a CI approach) as an open source software for automated regression systems based on VRM (as described in section 2.2.2). Along this implementation-exploratory section, there has been some insights that based on the goals of the project, will be analyzed in the next chapter.

5.3 Qualitative summary of results

The findings from this exploratory section is as described below:

- Although not a finding but a relevant observation, is that QME provides a interface(a list of names) for the definition of the different inputs to the flow (source files, tb files, regression system definition (tc spec file) and the testplan as the leading document for the coverage closure evaluation). This feature in fact provides the capture section while building automation regression systems (see section 2.2.2). Besides QME also integrates the automation (see section 2.2.2), which in this particular exploratory section, refers to the merging capabilities (for report generation) and for the post actions regarding the PASS FAIL status of the

UCDBs. With respect to the visibility, QME provides report for coverage, trend, regression status and a general coverage summary can be uploaded to the Jenkins console.

- QME has implemented more advanced report coverage for analysis, like in the case of trend analysis that shows detailed information regarding the number of test (as pass, failed,... and fatal), the bin summary - specially useful for functional coverage - (it shows the bins that were covered and the ones that were not covered). Finally the coverage summary by type (cover-groups, from testplan, and the weighted). Besides this trend analysis not only goes to the general overview, but also a detailed reporting of design units can be presented.

- The process that has been explored (implemented) always start with a given testplan. In fact this leads to having a coverage closure evaluation on regular basis (since it is implemented in the CI practice). The process without this testplan lacks effectiveness when trying to evaluate coverage closure and trend analysis. Therefore the insertion into the process of a testplan leading document is one of the main advantages that the integration of VRM and CI practice brings (respect to the process that is already implemented in the verification process).

Further analysis of these results are given in section 6

5. QME IMPLEMENTATION (EXPLORATORY APPROACH) WITH FLOATING POINT UNIT DEMO

6

Discussion

This section presents the analysis of the exploratory implementation of vrm and jenkins over the SPI and FPU.

This chapter is therefore sub-divided based on section that refers to the mayor findings and problems that have been faced as shown below:

- VRM Jenkins Integration - Process improvement options: In fact this is one important topic that has got high relevance in this project. Since this analysis is performed constructively from the process improvement options from sections 3 and 4 , The respective analysis is performed.
- VRM advantages to the flow: This section is specific to the advantages that the features of VRM gives to the flow. This features are mainly related to inheritance and parametrization, debug capabilities, report generation, rerun of failed tests
- QME - jenkins analysis. This sections analyses the QME flow, its advantages and still the disadvantages that QME has in its flow.

6.1 VRM-Jenkins integration - Process improvement options

One of the main questions that arose at the beginning of the project was how good is VRM for team working? Does it really allows to have an approach to work in a Continuous Integration approach as in the case of Jenkins? Actually, VRM is a powerful tool for regression systems management, it easies the configuration, verification report

and has strong merging capabilities and debug modes. However VRM does not have a monitoring system for version control (like in the case of Jenkins) which allows to have a Continuous Integration flow for development and verification . Therefore it was one of the most important disadvantages of VRM with respect to Jenkins. However based on the implementation of option 2 (see section 3.2), it seems that these 2 tools can be integrated in the same flow. Figure 6.1 shows the development-verification flow of option 2 in its general view.

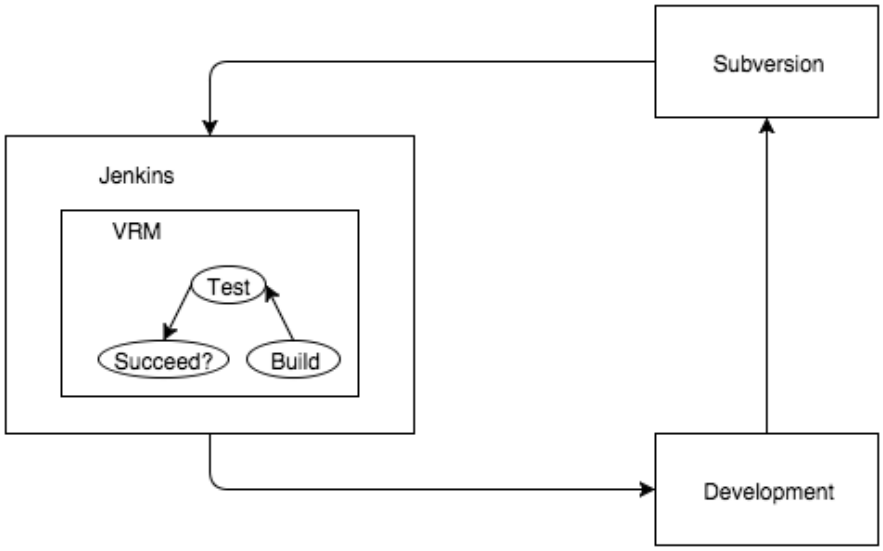


Figure 6.1: Process improvement - Option 2. In this approach, Jenkins triggers the already implemented regression suite defined in the RMDB.

However for it, the project has to be set on rmdb databases. One important fact that enforced this option as process improvement was that in march 2016 Mentor graphics releases a plugin that integrates Jenkins and VRM (13) . In fact although VRM and jenkins are technologies for regression systems management, they are complementary technologies. If integrated in the same flow, several facilities can be added. Actually, all the capabilities of VRM (specially the ones regarding coverage reporting and UCDB merging) and jenkins as a Continuous Integration approach can be applied to the development-verification flow. This partial conclution is in fact taken from the implementation and results of VRM in Jenkins and confirmed with the integration based on QME.

A relevant advantage that the integration of VRM and Jenkins Continuous Integration gives to the verification flow (compared when only a CI is implemented) is that verification metrics are supported into the continuous integration flow. This feature allows to have a continuous evaluation of coverage closure of the project while it is currently being developed. This process is implemented in an automated flow and allows to have a clear traceability of verification requirements. This affects deeply in the fact that this approach avoid wasting time since there is not a manual process for tracking the coverage. In this sense it is more easy to analyze results, take the required actions and rerun tests (maybe in debug mode).

Which benefits does this approach have? the benefits are around the capabilities of VRM that can be added to the flow. From chapter 4, based on the implemented options and the execution of each of these options, the integration of VRM and Jenkins works well in the sense that both technologies can be integrated as complementary technologies. Second is the fact that the capabilities that VRM offers can be added to the flow. From chapter 4 only the report generation capabilities and the merging capabilities have been applied based on a test plan leading document in the development process.

Besides another important factor is that for getting a full advantage of vrm when implementing it in the flow is to implement the system totally based on rmdb and not partially. From option vrm post layer (see section 4.3.3), a partial implementation of VRM to the flow was done. However this kind of approach only offers capabilities for merging and report generation, but another capabilities like debug, triage reporting, rerun of failed tests that are important in regression systems, cannot be applied. Therefore it is advisable to implement the flow completely in the VRM and not partially.

6.2 VRM advantages to the flow

When trying to compare the 2 flows (the VRM flow and the current flow that uses Jenkins as a CI tool in the development process), actually this comparison ends in the analysis of the features that VRM has and that the current flow does not have. The features add capabilities to the flow under the scope of an automation approach. Therefore in this subsection, the summarizing of the capabilities of VRM are under

analysis, showing the general flow of it and which parts were analyzed in the exploratory implementation.

Figure 6.2 shows again the VRM flow from its general view. The flow, starts from the configuration. Here potential inheritance and parametrization can be applied in order to set common settings among different projects in the rmdb. Besides grouping is implemented, based on the needs like direct tests, random tests, formal verification. Once this configuration is performed, another features, specially regarding to merging (report generation), debug and triage analysis can be implemented.

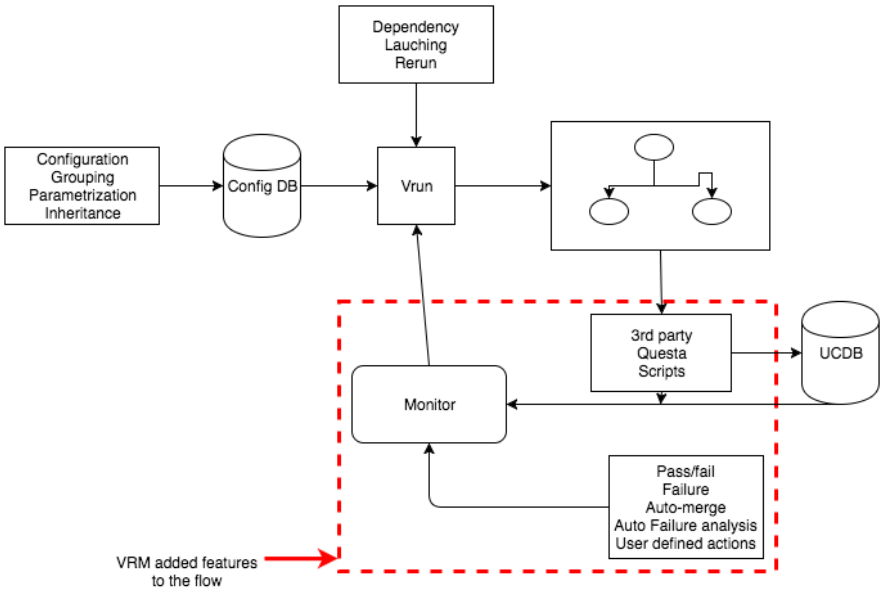


Figure 6.2: VRM flow. Here it highlights the part of the flow in where the capabilities of VRM are added. These features (merging, reporting, triage reporting, debug modes) add automation and capabilities to the flow that end up efficiency for coverage closure evaluation, traceability of requirements, debug capabilities and reporting

The first feature that has been strongly under analysis (also from exploration results) is automatic report generation from a testplan for coverage closure evaluation. In fact this feature is not implemented in the current flow that has been implemented in jenkins. This is the most important feature that VRM has in comparison with the current flow. First of all, this feature allows to have a clear traceability of requirements. Second, it easies the evaluation of coverage closure since an updated coverage reporting (from every requirement and from every design unit) has been achieved in an automatic

fashion. Besides the trend reporting that helps to visualize over time what has been the progress and the rate of coverage.

Second is the triage and debug analysis. After a regression suite is run then all the messages from failed tests are collected and filtered in order to address the failures as shown in figure 6.3. In fact this triage actions are specified in the respective runnables that configures the regression suite.

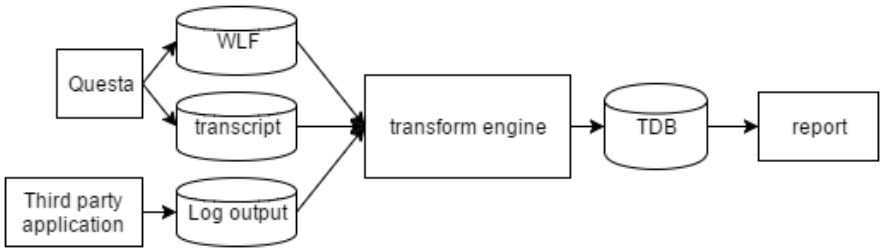


Figure 6.3: Triage analysis flow in detailed. Here the tdb (triage data base) is generated based on the data from Questa (as UCDBs WLF and log files) (3)

The relevant aspect that triage analysis (or automated result analysis) is that it collects the information of failure across all regression runs and allows to have a filtering of the causes of the failures. This kind of analysis allows the user to identify bugs more easily. In fact when this triage analysis is integrated into the CI practice, the verification process will have benefits since triage analysis allows to capture errors as early as possible in the process (8). However the analysis of the cause of the errors (the bugs) is not automated as in the case of Venssa technologies and Questa (see 2.3.3.6) In this sense it is clear that although VRM offers the automation and integration in the flow of triage analysis, that helps until some extend in the identification of common failures by filtering and grouping the error messages from multiple regression runs. However the process of error cause identification and bug fixing can not be improved since there is not any tool that offers automation in the debug identification (or at least a process that offers a list of potential error causes either in the DUT or in the verification environment) as in the case of venssa technologies. Therefore the triage reporting helps in the process of identification of common failures but not in the identification of root causes.

The next aspect is regarding with the debug capabilities that VRM supports and how these debug modes when integrated in the general verification flow then they

provide facilities and valuable options to the flow. Therefore the question is how debug capabilities can be integrated in the system using vrm and what advantages does it have when is automated, specially in a continuous integration approach?

First, VRM allows to have parametrization of the regression suite based on the cmpopts and simopts which allows to specify regression suits with debug mode active. In this sense, and based on the automated debug capabilities of VRM (see section 2.3.3.5), automated re-run of failed test can be applied to the flow either after a nightly run, or in the development process. For instance, as it was specified in the background chapter (see section 2.3.3.5), in a semi-automatic mode are selected and are re-run in debug mode. In fact this approach can be integrated in the CI approach for nighlty runs that can be run on a regular basis. Figure 6.4 shows the automated debug capabilities integrated in the flow under a CI approach.

As it was already set in the background chapter, the failed tests can be re-run in order to get a debug view. Therefore after a regression is run, the Results Analysis Database is open for showing the failing tests. In this sense, a potential way of using it is that once a regression is run then the failed tests are run again in debug mode as already shown in figure 6.4. Here the process starts with the definition of the test plan (in a UCDB testplan). After the simulation is done, then there is an evaluation of PASS/FAIL-status. In the case that it fails, then the tests that failed are rerun in debug mode.

6.3 QME - jenkins analysis

From the implemented options from section 3.2 it is clear that integrating VRM and JENKINS allows to have a coverage driven verification process under a Continuous Integration methodology for hardware development. Now, when implementing QME (see section 5) there has been several aspects that are necessary to take under consideration:

- QME-JENKINS is still a beta version and there is not support for it. In a full industrial environment it does not give a strong reliability. However in a more mature stage of QME, it could be fully adapted to the development process.
- One important fact that at the beginning was check is that QME only supports UVM test-benches. However in the implementation of the SPI, that has one di-

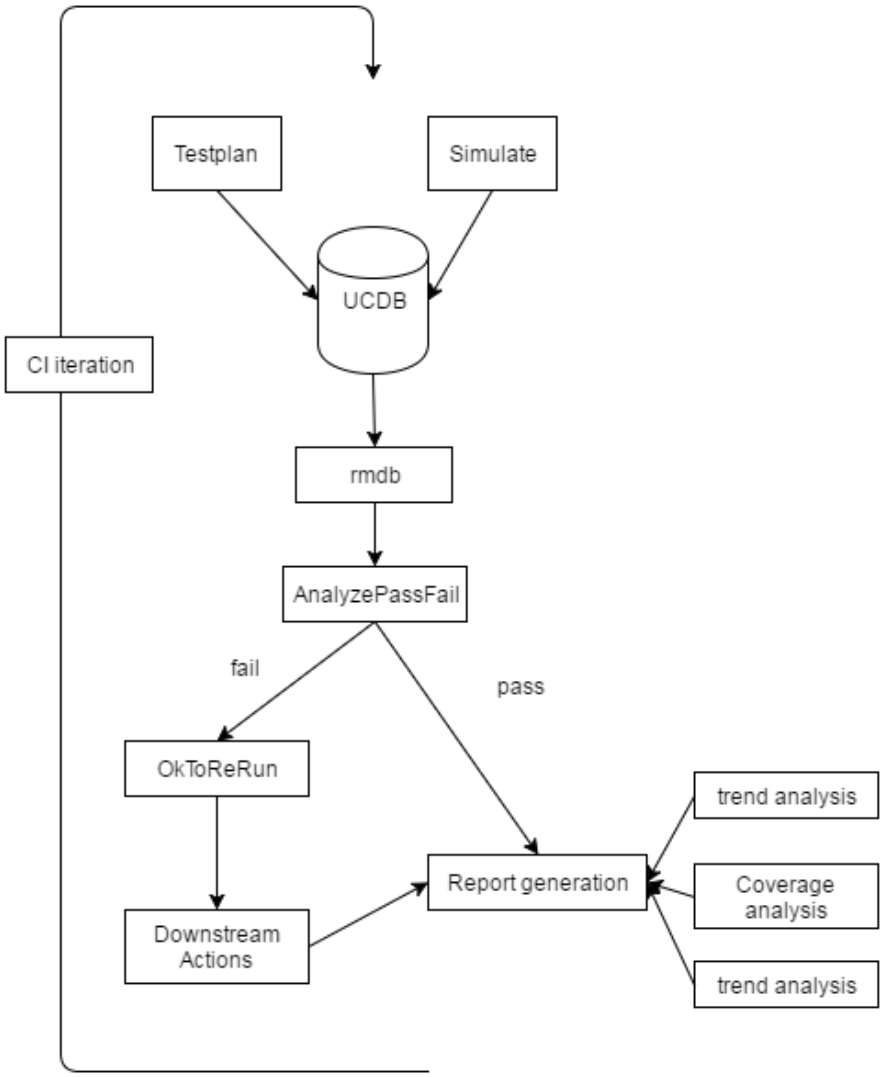


Figure 6.4: Verification flow under CI approach for debug capabilities in which re-run is performed failed tests under debug mode

rected test, this can be integrated into the environment. Therefore any type of test bench can be integrated in the regression system, but some previous configuration of the environment has to be set.

- QME specifies that if a chip level is used then code coverage is not used. The reason is that the code coverage is fully exercise at block level and therefore at chip level is not needed. Although it can also be set for chip level, it depends on

the purpose of the configuration.

Another important aspect is that a trend analysis can be observed for every design unit. It means that the coverage for every design unit can be obtained and a detailed verification analysis in terms of coverage closure can be done.

QME was implemented in order to evaluate in a further way the approach of the VRM capabilities in the flow when it is done in a continuous integration methodology. From section 5 it is clear that the 2 tools are in fact complementary. The advantages that VRM offers can be fully implemented in CI. Among these are report generation for coverage closure evaluation. Debug capabilities, triage reporting, rerun of failed test.

Besides of this particular features that VRM gives to the flow, QME offers some additional advantages that are mainly related to the next topics:

- Given interface for setting the capture of information (file list and test plan) in the flow (as already specified in the section 2.2.2 for building automated regression systems).

- Direct report generation of coverage and trend analysis in Jenkins console. Although this feature allows to have a direct view for coverage closure in the jenkin's console, the report generation of VRM (in their native HTML) are good enough for reviewing in detailed the coverage closure

- Specifies all the options for report generation, from general test plan until design unit coverage and trend reporting. These are features of VRM that are implemented directly in the flow that allows to have a coverage closure evaluation. There are some features of VRM that have been implemented in QME (and were not implemented in the flow options as described in section 3) that actually add important features-advantages to the flow. The trend reporting has detailed features that gives to the flow coverage closure evaluation: test summary, linked bins summary and coverage summary by type.

- Detailed coverage information regarding functional coverage in terms of bins, as in the case of the FPU

- Allowing to specify different set-ups based on the different parametrization for the design. As it was shown in section 5.2 under the coverage results shown in pictures 5.2 and 5.3
- From the implementation of the SPI with only the improvement options (as described in section 4) and in contrast with the implementation in QME, it is observable how the QME flow allows to have a more clear specification of the design files (source and test-benched) and the specifications of the regression suite (the name of test, number of tests, number of simulations, List of seeds). Besides the trend reporting for design units also gives a more in detailed coverage evaluation.

7

Conclusions and recommendations

The aim of this master project is to experiment in an exploratory methodology the features of VRM, the possible integration of VRM into the CI practice, all under the concept of automation of the verification flow. Therefore under this context, the analysis of the features that VRM brings to the flow have been observed (trough an implementation-exploratory approach) and analyzed.

Based on the implemented options for process improvement regarding the merge capabilities for report generation and the QME beta version that Mentor Graphics is managing, it is clear that a full integration of VRM and Jenkins can be done. It means that all the capabilities of VRM can be indeed implemented in the verification flow under a Continuous Integration practice. Besides it is to be noted that the integration of VRM in CI has to be fully implemented and not partially implemented in order to get all the advantages that VRM brings to the flow. Otherwise only the merge capabilities for report generation can be exploited and therefore another capabilities like debug modes, rerun of failed tests, triage analysis could not be exploited.

From the implementation of QME and the process improvement options, there are several advantages that VRM brings to the flow (Under a continuous integration practice) which are listed below

- A coverage driven methodology (Metric Driven Verification): Since the process is lead from a verification test plan and the coverage report is obtained at any stage of the verification development process, then time saving is obtained (no

manual steps are done) and coverage closure evaluation is performed under a driven metric verification

- Debug capabilities and trend analysis can be potentially integrated to the verification flow. However although the triage analysis easies the grouping of common failures, it does not gives actually a possible list of potential root error causes (as in the case of Venssa technologies). Therefore it is recommended to use triage analysis since it is helpful in the pre-process of finding common failures, however a detailed analysis for finding the error causes are strongly related to the debug process

- Inheritance and parametrization of rmdb: These are characteristics of the configuration of the regression suites based on the rmdb. The advantage of it is the re-usability of different settings among different projects.

The flow that has been implemented in the exploratory section (refers to the CI integration and the VRM capabilities regarding to coverage, triage, and trend analysis, and the debug capabilities in an automated fashion) actually leads to have a Metric driven verification process. In fact the many metrics that are presented (verification metrics regarding coverage (functional and and code coverage)) leads to have an analysis with detailed coverage closure

References

[1] Bonnie Kaplan and Joseph A. Maxwell. **Qualitative Research Methods for Evaluating Computer Information Systems**. *Springer*, **1**(2). iii

[2] Thom Ellis David Crutchfield. **Bringing Regression Systems into the 21st Century**. *Mentor Graphics Corporation*, **1**(2):22–25, 2007. iii, 14, 15

[3] Mentor Graphics Corporation. *Questa Verification Run Manager User Guide*. Mentor Graphics, 2011. iii, vii, viii, ix, xi, 17, 18, 19, 20, 22, 24, 25, 26, 28, 29, 30, 31, 36, 81

[4] Evean Qin Sean Safarpour and Mustafa Abbas. *Efficient Failure Triage with Automated Debug: a Case Study*. Vennsa Technologies Inc, 2015. iii, 33, 34

[5] Mikael Andersson. *QME User Guide v 0.2*. Kaiserslautern Germany, 2016. iv, v, viii, 37, 38, 39, 40, 41, 42, 60, 66, 98

[6] Juan Camilo Parra Diaz. *Automated Coverage and Verifcation Plan Management*. NTNU, 2015. vii, 1, 2, 9, 16, 18, 20, 51, 52, 53, 54

[7] Prof. Dr. Dr. h.c. Dieter Rombach. *Fundamentals of software engineering*. TU-KL, Kaiserslautern Germany, 2015. vii, 9, 10

[8] Andre Winkelmann Jason Sprott. **A GUIDE TO USING CONTINUOUS INTEGRATION WITHIN THE VERIFICATION ENVIRONMENT**. *Verilab ltd*, **1**(2):22–25, 2014. vii, 11, 12, 13, 81

[9] John Ferguson Smart. *Continuous Integration for the Masses*. Creative Commons Edition, New Zeland, 2011. 9

[10] Jonathan McAllister. *Mastering Jenkins*. Mentor Graphics, 2015. 10

[11] Fritz Ferstl Darron May. **Are you really confident that you are getting the very best from your verification resources?** *Mentor Graphics Corporation*, **1**(2):22–25, 2007. 15

[12] Darron May. **Mentor Graphics: Verification management and planning**. 17

[13] Thomas Ellis. *Increased Efficiency with Questa VRM and Jenkins Continuous Integration*. Mentor Graphics, 2016. 78

Appendix A

Process improvement

A.1

RMDB for the option 2 - process improvement

Next rmdb defines the database created for managing the regression system in the process improvement option 2.

```
<?xml version="1.0" ?>
<rmdb version="1.0" toprunnables="general">
<runnable name="general" type="group" >
 <parameters>
  <parameter name="mergefile">merge.ucdb</parameter>
  <parameter name="tplanfile">.../testplan.xml</parameter>
  <parameter name="tplanoptions">-format Excel</parameter>
  <parameter name="tplanucdb">testplan.ucdb</parameter>
 </parameters>
 <members>
  <member>spi</member>
  <member>top</member>
 </members>
 <postScript launch="vsim">
  <command> merge SPI and toplevel testbenches. Output merge
      1</command>
  <command> generate UCDB testplan from XML testplan</command>
  <command> merge merge1 UCDB with testplan UCDB </command>
 </postScript>
</runnable>

<runnable name="spi" type="task">
```

```
22 <execScript launch="exec">
23  <command> set necessary variables </command>
24  <command> copy necessary compilation files </command>
25  <command> copy necessary SV files </command>
26  <command> compilation commands</command>
27  <command> simulation commands </command>
28  </execScript>
29 </runnable>
30
31 <runnable name="top" type="task">
32 <execScript launch="exec">
33  <command> set necessary variables </command>
34  <command> copy necessary compilation files </command>
35  <command> copy necessary SV files </command>
36  <command> compilation commands</command>
37  <command> simulation commands </command>
38 </execScript>
39 </runnable>
40 </rmdb>
```

A.2 RMDB for the option 3 - process improvement

Next code shows the runnable that was implemented for the Option 3 in process improvement. (see section 4.3.3)

```
1  <?xml version="1.0" ?>
2  <rmdb version="1.0" toprunnables="merging">
3   <runnable name="merging" type="task">
4    <parameters>
5     <parameter name="mergefile">merge.ucdb</parameter>
6     <parameter name="tplanfile">...</parameter>
7     <parameter name="tplanoptions">-format Excel</parameter>
8     <parameter name="tplanucdb">testplan.ucdb</parameter>
9     <parameter name="tplanucdb">testplan.ucdb</parameter>
10   </parameters>
11   <execScript launch="vsim">
12    <command> vcover merge SPI Top -out merge1</command>
13    <command> eval [list xml2ucdb -ucdbfilename... </command>
14    <command> vcover merge merge1 testplan -out merge2.ucdb</
          command>
15   </execScript>
```

```
16  </runnable>
17 </rmdb>
```

A.3 Settings for QME - setup in jenkins

```
1  #!/bin/bash -x
2  pwd
3  echo "Running in: ${WORKSPACE}"
4
5  # adding Questasim to the path and making
6  # sure the license is up.
7  source /cad/gnu/modules/modules-tcl/init/bash
8  module load questasim
9  module load gnutools/grid-engine
10 module list
11 export QUESTA_HOME=/cad/mentor/questasim/v10.5/questasim/
12
13 cd /pri/jupa/ ... /qme_QME_1.30.beta5
14 source source_me.bsh
15 cd -
16
17
18 #create a simulation directory using the
19 #create_questa_simdir.pl command.
20 create_questa_simdir.pl -simdir=simdir -block=fpu
21 -l=\${WORKSPACE} -f
22 export TERM=vt100
23 cd simdir
24 make regression_batch regression_dvt_junit NOCOLOR=1
25 SEND_EMAIL=0  TC_SPEC=${QME_HOME}/.../fpu/sim/tc_spec_qa.txt
```

A.4 Settings for QME - makefile blocks

The Makefile.block.defaults file that defines the settings for the fpu regression is shown below. Here ARCH=64 was set for running it on a 64 bit machine.

```
1       ARCH=64
```

```
2  UVMC=1
3  SEND_EMAIL=1
4  SYSTEMC_VERSION=
5  XML2UCDB_DATAFIELDS="Section, Title, Description,
6  Link, Type, Weight, Goal, xyz"
7
8  #FIXME:No access to qsub commandsd
9  #GRID_ENGINE=SGE
10 #QUEUEARGS = "-cwd -V -j y -b y -S /bin/sh"
11
12 ifeq (${EMAIL_RECIPIENT},hakanp)
13  EMAIL_RECIPIENT=Hakan_Pettersson@mentor.com
14 endif
15 VISIBILITY_VISUALIZER=+acc
16
17 VISUALIZER=0
18
19 EXTRA_VSIM_ARGS+=-suppress 3069
20 EXTRA_VSIM_ARGS+=-suppress 6667
21 EXTRA_VSIM_ARGS+=-suppress 6610
22 EXTRA_VSIM_ARGS+=-suppress 4025
23
24
25
26 ifeq (${TEST},fpu_test_patternset)
27 EXTRA_VSIM_ARGS+=+PATTERNSET_FILENAME=${QME_PROJECT_HOME}/fpu/
       tb/
28 golden/pattern_set_ultra_small.pat +PATTERNSET_MAXCOUNT=-1
29 endif
30
31 #RERUN_FAILING_TESTS=0
32 MAX_PARALLEL_JOBS=2
33
34 # Added some stuff for compiling the SC model
35 SCCOM_INCLUDE_LIBS += -I${UVMC_HOME}/examples/fpu/tb/c
36 -I${QME_HOME}/examples/tlm_models/fpu/ -I${QME_HOME}/
37 examples/tlm_models/common/
38 -I${QME_HOME}/examples/fpu/uvmc/converters
39
40
41
42 compile_sc::compile_sc_wrapper link_sc_wrapper
43
```

```
44 compile_sc_wrapper::
45 ifeq (${NOCOLOR},0)
46         @tput setaf ${HELP_COLOR}
47         @tput bold
48 endif
49         @echo "########"
50         @echo " Compiling SC reference model from ${
               BLOCK_OVERRIDES}"
51         @echo "####################"
52 ifeq (${NOCOLOR},0)
53         @tput setaf ${RESET_COLOR}
54         @tput sgr0
55         @echo -n ""
56 endif
57         @test -d ${QUESTALIBS_DIR} || mkdir ${QUESTALIBS_DIR}
58         @test -d ${QUESTALIBS_DIR}/${SC_WORK_LIB} || vlib
           ${QUESTALIBS_DIR}/${SC_WORK_LIB}
60         vmap ${SC_WORK_LIB} ${PWD}/${QUESTALIBS_DIR}/${
               SC_WORK_LIB}
61         @echo "compiling SystemC TLM model"
62         ${SCCOM} ${SCCOM_ARGS} -verbose \
63         ${UVMC_SC_EXTRA_ARGS} \
64         ${QME_HOME}/examples/fpu/tb/c/fpu_wrapper.cpp \
65         ${QME_HOME}/examples/tlm_models/fpu/fpu_top.cpp \
66         ${QME_HOME}/examples/tlm_models/fpu/fpu_core.cpp
67
68
69 link_sc_wrapper::
70 ifeq (${NOCOLOR},0)
71         @tput setaf ${HELP_COLOR}
72         @tput bold
73 endif
74         @echo "######"
75         @echo "Linking Wrapper from ${BLOCK_OVERRIDES}"
76         @echo "############"
77 ifeq (${NOCOLOR},0)
78         @tput setaf ${RESET_COLOR}
79         @tput sgr0
80         @echo -n ""
81 endif
82         ${SCCOM_LINK}
83
84
```

```
85 PROJECT_NAME=${BLOCKNAME}
```

A.5 Settings for jenkins for a QME project

```
1  #!/bin/bash -x
2  pwd
3  echo "Running in: ${WORKSPACE}"
4
5  # adding Questasim to the path and making sure the license
       environment is up.
6  source /cad/gnu/modules/modules-tcl/init/bash
7  module load questasim
8  module load gnutools/grid-engine
9  module list
10 export QUESTA_HOME=/cad/mentor/questasim/v10.5/questasim/
11
12 # coping the files for qme
13 cd /pri/jupa/thesis/bibliography/mentor_graphics/qme_QME_1.30.
       beta5
14 source source_me.bsh
15 cd -
16
17 echo "QME_HOME in: ${QME_HOME}"
18 echo "QME_PROJECT_HOME in: ${QME_PROJECT_HOME}"
19 echo "QME_PROJECT_DEFAULTS in: ${QME_PROJECT_DEFAULTS}"
20
21 #create a simulation directory using the create_questa_simdir.
       pl command.
22 create_questa_simdir.pl -simdir=simdir -block=fpu -l=${
       WORKSPACE} -f
23 export TERM=vt100
24 cd simdir
25 make regression_batch regression_dvt_junit NOCOLOR=1
       SEND_EMAIL=0  TC_SPEC=${QME_HOME}/examples/fpu/sim/
       tc_spec_qa.txt
```

A.6 Pre-simulation setting for the FPU in QME

```
1  RCH=64
2  EXTRA_VLOG_ARGS="+delay_mode_unit +define+RTL
3  +define+ASSERT_ON +define+FUNCTIONAL_COVERAGE_ON
4  +define+ENABLE_SFR_RE_DEACTIVATION"
5
6  TB_TOP_NAME=test_Spi
7  #REQUIRE_VPLAN=0
8  SIMULATION_DUT=SpiSlave
9  EXTRA_VOPT_ARGS=+cover=fbecst+SpiMaster.
10 NOCOVER=+nocover+Hinst*
11
12
13
14 ifeq (${REGRESSION},1)
15 presim_script::
16         cp -rf /.../digital/Spi/sim/hdl/stimuli .
17 else
18 presim_script::
19         cp -rf /.../digital/Spi/sim/hdl/stimuli ${RUNDIR}
20 endif
```

A.7 Integration FPU-QME in the jenkins server

```
1  #!/bin/bash -x
2
3  echo "Running in: ${WORKSPACE}"
4
5  # adding Questasim to the path and making sure the license
       environment is up.
6  source /cad/gnu/modules/modules-tcl/init/bash
7  module load questasim
8  module load gnutools/grid-engine
9  module list
10 export QUESTA_HOME=/cad/mentor/questasim/v10.5/questasim/
11 export nrf4352_HOME=/pri/jupa/project/ip_testbenches/v4/
12          SPI_from_nrf4352/nrf4352/
13
14
15 # coping the files for qme
16 cd /pri/jupa/thesis/bibliography/mentor_graphics/qme_QME_1.30.
       beta5;
```

```
17 source source_me.bsh
18 cd -
19
20 echo "QME_HOME in: ${QME_HOME}"
21 echo "QME_PROJECT_HOME in: ${QME_PROJECT_HOME}"
22 echo "QME_PROJECT_DEFAULTS in: ${QME_PROJECT_DEFAULTS}"
23
24 #create a simulation directory using the
25 # create_questa_simdir.pl command.
26 create_questa_simdir.pl -simdir=simdir -block=nordic_spi
27     -l=${WORKSPACE} -f
28 export TERM=vt100
29 cd simdir
30 make regression_batch regression_dvt_junit NOCOLOR=1
31     SEND_EMAIL=0
```

A.8 FPU verification plan (5)

This appendix shows the verification plan used for the FPU verification process in QME as a process integrated in the Continuous integration approach used in Jenkins. This verification plan is part of the demo provided by QME.

Testplan

Section	Title	Description	Link	Type	Weight	Goal
1	**Code Coverage**					
1.1	Branch		/top/fpu_dut	Branch	1	100
1.2	Condition		/top/fpu_dut	Condition	1	100
1.3	FSM		/top/fpu_dut	FSM	1	100
1.4	Expression		/top/fpu_dut	Expression	1	100
2	Reference tests	Reference pattern			1	100
2.1	Test pattern set		DEFAULT:fpu_test_p	Test	1	100
3	**Core Functionality**					
3.1	Exceptions					
3.2	Exception Signals	Verify that all exceptions are operating	fcoverage:exception;	CoverPoint	1	100
3.2.0	Invalid Operation	Verify that results of every invalid operation are a QNaN string with a QNaN or SNaN exception	fcoverage:add_subXinf; fcoverage:mul_Xinf; fcoverage:divXinf_zero;	Cross	1	100
3.2.1	NaN1	Verify that the SNaN string can never be the result of any operation.			1	100
3.2.2	Division by Zero	Verify that division of any number by zero other than zero itself gives infinity as a result. And verify that a divide-by-zero exception is created.	fcoverage:divXexc;	Cross	1	100
3.3	Rounding Modes				1	100
3.3.0	Rounding bits	Verify that the			1	100
3.3.1	Round to nearest even	Verify that the If the result value is exactly halfway between two infinitely precise results, it is rounded up to the nearest infinitely precise even.	fcoverage:round;	CoverPoint	1	100

3.3.2	Round-to-Zero	Verify that the excess bits will simply get truncated	fcoverage:round;	CoverPoint	1	100
3.3.3	Round-Up	Verify that the number will be rounded up towards +	fcoverage:round;	CoverPoint	1	100
3.3.4	Round-Down	Verify that the opposite of round-up, the number will be rounded	fcoverage:round;	CoverPoint	1	100
3.4	Operations on floating point numbers	Verify the operations supported Add Subtract Multiply Divide Square Root	fcoverage:operation;	CoverPoint	1	100
3.5	Operations followed by a	op followed by op	fcoverage:N2N_operations	CoverPoint	1	100
4	**Pipeline**					
4.1	Operation delay	"Verify the pipelined spec: Addition, 7 clks Subtraction, 7 clks Multiplication, 12 clks Division, 35 clks Square-root, 35 clks hardware, page 16"	covered_*_delay	Directive	1	100
5	**Pin interface**					
5.1	Valid opcode on pins	"Verify the pin if: fpu_op_i always have legal values x0 -> x4 hardware, page 17"	covered_opcode	Directive	1	100