Novel Approach to Automate Design Verification Cycle Using Continuous Integration-Continuous Deployment Platform

Juilly Sunilrao Videkar, Sri Harsha Reddy Kaliki, Shaik Babjan Sohail, Kannusamy Mariappan
Samsung Semiconductor India Research, Bangalore, India
juilly.v@samsung.com, sriharsha.k@samsung.com, sohail.s@samsung.com, kanns.m@samsung.com

***Abstract*-During the development phase of an IP or a SoC, design and verification teams continuously update the RTL design and Testbenches respectively to add new features. Hence, it becomes essential to keep continuous track on sanity of the latest design and verification databases. Manual verification of an IP or SoC requires loads of effort from the engineers due to the increasing complexity of the designs. Additionally, continuous manual runs and analysis demands lot of time to go through each log and report, hence decreasing the efficiency of an engineer. To address this requirement, we have established a comprehensive Automated Regression Framework (ARF) using Atlassian Bamboo Continuous Integration and Continuous Deployment (CI-CD) platform for verifying an IP or SoC. This is a web based framework with multiple automated verification steps, triggers the flow at scheduled time and extracts vital key performance indicators (KPIs) for each step in a single summary file, thereby cutting down high amount of manual effort and increasing the productivity of the project design cycle. Along with this, we have developed an innovative robust monitoring system, which keeps track of each flow executed by the user and displays the status of each run in a web-based dashboard, thereby improving the tracking of the project status and streamlining the traceability of the project.**

Keywords- Continuous Integration and Continuous Deployment (CI-CD), Automated Regression Framework (ARF), Key Performance Indicator (KPI), System on Chip (SoC), Intellectual Property (IP), Register Transfer Logic (RTL), Application Specific Integrated Circuit (ASIC).

I. INTRODUCTION

In the modern day environment, ensuring the sanity of RTL or Testbenches in the early design cycle has become an essential part of the ASIC flow due to multiple engineers working simultaneously on different blocks in the design and the changes are committed to a common repository. Any change to the repository need to be verified constantly, so that any flagged issues due to recent commits can be fixed during early release cycle. Also performing all the verification checks manually is time consuming and repetitive. To address this, we have developed ARF to automate the verification cycle along with a robust monitoring system, which helps in reducing the time, cost associated with fixing issues, and helps in tracking and managing the project status. The automated framework proposed in this paper uses CI-CD platform. CI-CD is a development methodology based on the principles of continuous integration and continuous delivery, which ensures that changes are integrated into the codebase frequently and continuously, and also ensures that changes are delivered to the end-users in a timely and automated manner [1]. We used Atlassian Bamboo as a CI-CD platform in this paper over other solutions like Jenkins due to its seamless integration with Bitbucket and Jira, extensive inbuilt tools for building application, intuitive user interface and ease of use.

II. ATLASSIAN BAMBOO CI-CD FRAMEWORK

Atlassian Bamboo is a CI-CD tool that can be used to automate the build, test, and deployment of projects. It can be integrated with other tools and processes used in chip design to automate certain tasks. As shown in Fig. 1, a Bamboo project contains 4 main entities namely task, job, stage and plan [3].

1. Task: It is a small discrete unit of work, such as source code checkout, running a script, or parsing test results. This is the place where all automation scripts for each of the flow are programmed. Tasks present in the same job gets executed sequentially.
2. Job: A Bamboo job is a single build unit within a plan. It processes a series of one or more tasks that are run sequentially. Jobs present in the same stage are executed in parallel.
3. Stage: A Bamboo stage is a group of jobs which processes the jobs in parallel. A stage must successfully complete all its jobs before processing the next stage in the plan.
4. Plan: A plan defines everything about continuous integration build process in Bamboo.

Variables used inside a plan can be configured using 2 ways –

1. Using configuration file – This is a text file where multiple variables can be configured using <variable name>=<value> syntax. This file needs to be injected once for each job so that, values of variables configured inside this file become accessible to all tasks of that job.
2. Using 'Variables' section of Bamboo – Variables and its value can be provided under "variable name" and value columns present in 'Variables' section of a Bamboo plan. This section can be also used to input password for various handshake applications. Path to the configuration file has to be provided in this section.

Bamboo also has 'Triggers' section where user can configure schedule time when plan should run automatically.

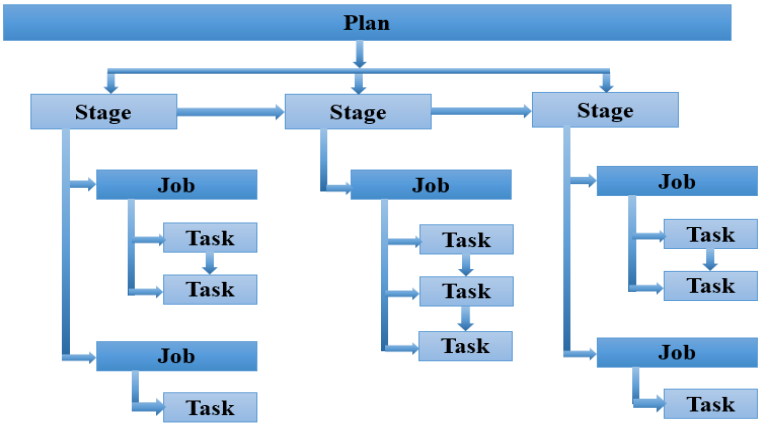

Figure 1. Atlassian Bamboo Framework

III. ARF WORKFLOW

As shown in Fig. 2, multiple engineers commit the updates into a common Git repository. In order to ensure sanity of design verification database, an engineer (referred as sanity runner) triggers ARF using manual or scheduled trigger to verify the latest changes made to IP or SoC repository. On reception of trigger from sanity runner, ARF clones latest design verification database from Git repository and executes various design verification steps on it. To ensure sanity, ARF executes various verification steps such as Testbench compilation, execution of regression, generation of coverage database for current sessions along with coverage merged with initial model. In addition to these, ARF periodically provides regression statistics and creates checker-wise Jira tickets (i.e. one ticket per unique checker) to report information of all testcases failed against that checker. After completion of an ARF run, a consolidated text report containing KPIs will be generated for quick analysis of each verification step result. These results will also be displayed on to the dashboard as soon as they are available.

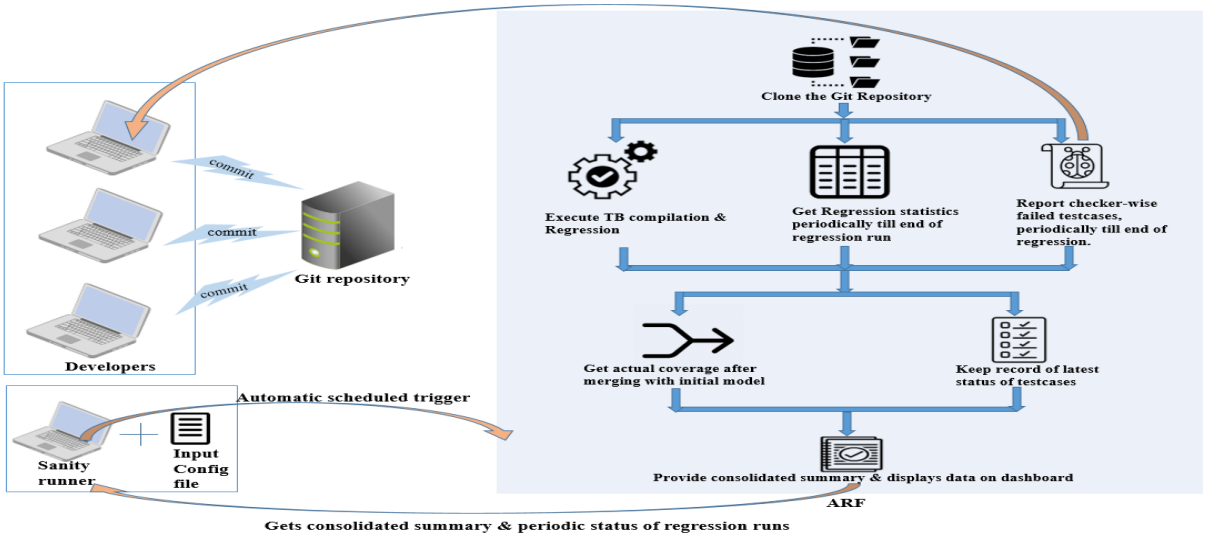

Figure 2. ARF workflow

IV. ARF DETAILS

In proposed framework, all verification steps are automated using a set of scripts inside CI-CD Platform. Automation scripts are programmed in languages like Python, TCL, Bash and Expect. These are present inside each 'Job' as 'Tasks' and are the brain of the platform. ARF plan for design verification is made of five stages as shown in Fig. 3 namely "Default", "Git Actions", "TB Sanity", "Coverage and Test Execution" and "Summary". The details of these stages are as explained below -

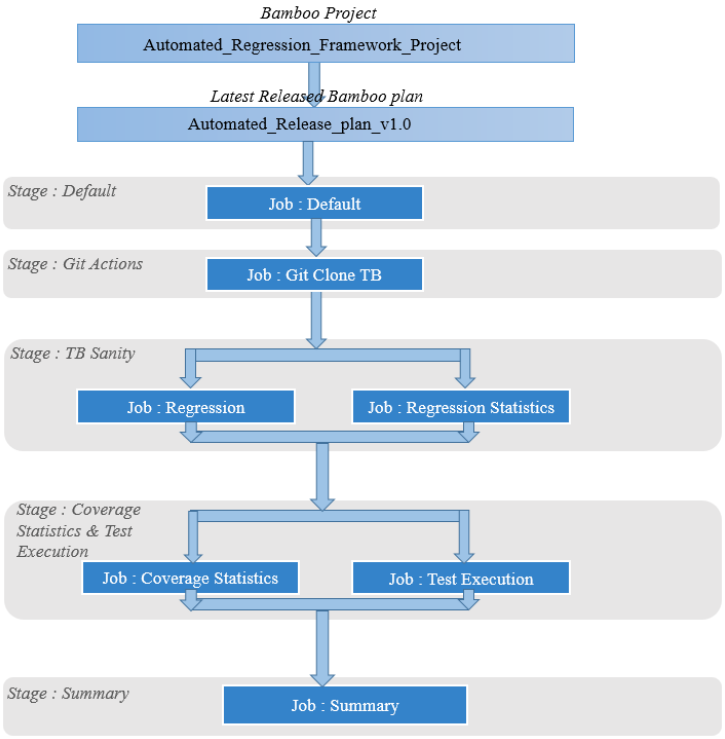

Figure 3. Structure of ARF

1. "Default" stage has a job called "Default", which loads the input configuration variables into the environment and also configures the environment.
2. "Git actions" stage has a job called "Git clone TB" which clones latest verification database from Git repository with user provided branch and tag.
3. "TB sanity" stage contains two jobs which run in parallel, namely "Regression" and "Regression and Statistics". The scripts present inside Regression Statistics job gets called after every "n" hours to provide periodic statistics until Regression is completed.
4. "Coverage Statistics and Test Execution" stage contains two jobs which run in parallel namely "Coverage Statistics" and "Test Execution", to generate coverage details (functional coverage, code coverage, etc.) and to update the latest execution status (passed/failed) of each unique testcase in Jira project management system.
5. "Summary" stage contains a job "Summary" which creates a common summary file listing execution status (Successful/Unsuccessful) along with the needed KPI's for each job.

The details of each job present in ARF are as explained below -

1. Git Clone TB –
   This job clones the verification database with user provided branch and tag.

2. Regression –
   This job runs after successful completion of "Git Clone TB" job. It performs Testbench compilation and triggers regression using Cadence vManager tool for the specified VSIF files, if there are no compilation errors. If ARF run (i.e. build) is stopped manually before completion of regression, then ARF will automatically stop the execution of remaining testcases. If regression runtime exceeds user provided maximum regression

runtime, then Regression job stops remaining testcases and proceeds to execute subsequent jobs. All such testcases are reported as stopped testcases in the regression statistics. If runtime of a particular testcase exceeds user provided maximum possible testcase runtime, then the job reports the testcases as hanged/looping testcase in regression statistics. After successful or partial completion of this job, it updates build summary file of this job with session paths and number of testcases passed, failed, running, hanged, stopped against each session.

3. Regression Statistics –
   This job runs in parallel with "Regression" job until the end of regression. The purpose of this job is to generate regression statistics periodically after every "n" hours from the start of regression to the end in a Jira ticket. Regression statistics is a table containing a list of sessions and the number of testcases passed, failed, running, hanged and stopped against each session. It also has a table containing checker to total failed testcases count against each checker. This Jira ticket also contains run timestamp, session folder path, CI-CD plan name and RTL tag. These can be used for debugging purposes. This job also periodically generates checker-wise Jira tickets for reporting list of unique failed testcases and assigns the ticket to appropriate owner. It creates one ticket for each checker under the user provided Jira component and if checker ticket is already present then it appends details of new failed unique testcases to the same checker ticket. After successful completion of this job, it updates build summary file of this job with Regression and Coverage statistics ticket number, list of checker tickets created to report unique failed testcases and list of already existing checker tickets which are updated with new failed testcases.

4. Coverage Statistics –
   This job runs after successful completion of regression. It uses Cadence Integrated Metrics Center (IMC) tool commands, to merge coverage database of current regression with initial/previous models to get actual coverage values. In case of Top module, it merges coverage databases of all submodules with top level coverage (current and initial model), to get coverage at top level. Using IMC commands, it also creates a coverage report containing coverage data (functional coverage, code coverage, etc.) for each module present in design. This job captures the coverage data of user provided top module, in the form of a table and appends it to regression statistics ticket. After successful completion of Coverage Statistics, it updates build summary file of this job with path of merged coverage database, coverage data for user provided top module and path to the detailed coverage report.

5. Test Execution flow –
   This job runs after successful completion of regression and it gets executed in parallel to the Coverage Statistics job. It keeps track of latest status (passed/failed) of each unique testcase by creating a 'Test' type Jira ticket under given project component. This job also creates one 'Test Execution' type Jira ticket, which contains consolidated view of all 'Test' type Jira tickets.

6. Summary –
   The purpose of this job is to consolidate build summary files for all jobs present in ARF and create a consolidated build summary file as shown in Fig 4.

```
time_stamp: 2023-08-17 17:44:58.116385

Git Clone TB Summary
====================
Execution Status: Successful

Regression Summary
====================
Execution Status: Successful_TestcasesFailed
Output Directory:
/user/xyz/ARF_clone/DB_2023_08_17__17_45_37/top_design/vm/sessions/presto_dm_qd1h_m1.xyz.23_08_17_1
7_54_02_0022/chain_0
Total Testcases: 168
Passed Testcases Count: 147
Failed Testcases Count: 21
Running Testcases Count: 0
Looping Testcases Count: 0
Stopped Testcases Count: 0

Output Directory:
/user/xyz/ARF_clone/DB_2023_08_17__17_45_37/top_design/vm/sessions/presto_dm_trp_m2.xyz.23_08_17_17
_54_25_1138/chain_0
Total Testcases: 7
Passed Testcases Count: 7
Failed Testcases Count: 0
Running Testcases Count: 0
Looping Testcases Count: 0
Stopped Testcases Count: 0

Regression Statistics Summary
============================
Execution Status: Successful
Regression and Coverage Statistics Ticket No.: Jira-6733
List of New Checker Tickets: Jira-6734
List of Updated Checker Tickets: Jira-6731

XTAP Flow Summary
====================
Execution Status: Successful
XTAP Flow Logfile: /user/xyz/common_folder/ARF_Release_v1.0_plan/xtap_flow_execution.log
Number of Test type tickets created/updated: 4

Coverage Statistics Summary
====================
Execution Status: Successful
Merged Coverage Database: /user/xyz /coverage_db/overall_merged_db/merged_cov_data_17:08:23_18:26:10
Detailed Coverage Report: /user/xyz /coverage_db/reports/cov_report.txt
Top level (I_NVMe_dm_top) Coverages:
 -Functional Coverage: 99.3
 -FSM Coverage: n/a
 -Block Coverage: 100.00
 -Expression Coverage: 99.22
 -Overall Coverage: 99.70
```

Figure 4. Sample build summary file

To use ARF, user has to follow the below four step process –

1. User has to clone the ARF plan into their Bamboo Project.
2. Configuration file needs to be filled with project specific values and path of filled configuration file needs to be provided in ‘Variables’ tab of Bamboo.
3. Run the plan using ‘Run plan’ option. Check the logs and Artifacts after plan completion.
4. User can optionally schedule their plan runs based on their requirement.

V. Dashboard Monitoring System

There is a need for streamlined mechanism to assist Engineers and the project leads to track the status of each project at IP or SoC level. The current manual method of checking results and reports is cumbersome and error prone. To overcome this challenge, we have developed a comprehensive web-based monitoring dashboard to display the ARF regression results (as shown in Fig. 5) using Bamboo REST API's and Jira plugins [4]. Dashboard can store up to twelve weeks' data and using this it's possible to retrieve the old data with the help of multiple configuration options. The dashboard page also supports filtering of the data based on sub project, week and RTL tag.

Project Details			Regression Details					Test Cases				
S.No	Project	Sub Project	First Reported (IST)	Last Updated (IST)	Seed	Status	Remark	Pass Rate(%)	TOTAL TESTS	PASSED	FAILED	RUNNING TESTS
1	Project1	Sub Project 1	2023-09-05 05:42	2023-09-05 06:09	DEFAULT	COMPLETE		83.54	243	203	40	0
			2023-09-04 05:42	2023-09-04 06:10	DEFAULT	COMPLETE		83.54	243	203	40	0
			2023-09-03 05:41	2023-09-03 06:04	DEFAULT	COMPLETE		83.54	243	203	40	0
			2023-09-02 05:41	2023-09-02 06:09	DEFAULT	COMPLETE		83.54	243	203	40	0
			2023-09-01 05:41	2023-09-01 06:09	DEFAULT	COMPLETE		83.54	243	203	40	0
			2023-08-31 05:41	2023-08-31 06:14	DEFAULT	COMPLETE		84.30	242	204	38	0
			2023-08-30 05:41	2023-08-30 06:14	DEFAULT	COMPLETE		84.30	242	204	38	0
		Sub Project 2	2023-09-05 01:10	2023-09-05 03:33	RANDOM	COMPLETE		99.60	1760	1753	7	0
			2023-09-04 01:08	2023-09-04 01:08	RANDOM	COMPLETE		99.60	1760	1753	7	0
			2023-09-03 01:10	2023-09-03 01:10	RANDOM	COMPLETE		99.72	1760	1755	5	0
			2023-09-02 01:13	2023-09-02 01:25	RANDOM	COMPLETE		99.55	1760	1752	8	0
			2023-09-01 01:12	2023-09-01 04:15	RANDOM	COMPLETE		99.72	1760	1755	5	0
			2023-08-31 01:08	2023-08-31 01:08	RANDOM	COMPLETE		99.66	1760	1754	6	0
			2023-08-30 01:09	2023-08-30 01:09	RANDOM	COMPLETE		99.66	1760	1754	6	0

Figure 5. Snapshot of Dashboard Monitoring System to display ARF results

VI. Results

ARF provides an end to end automated solution for complete design verification cycle to reduce the manual effort and repetitive tasks while verifying an IP or SoC. It's proved to be an effective solution over traditional approaches like running regression before every code commit due to multi flow support in a single tightly integrated framework. Through ARF features and scheduled nightly builds, engineers are able to save approximately 80% of their time for one regression run, effectively translates to 2.5 man-days per week, thereby improving their productivity. Table 1 compares the time taken between prior manual approach and current ARF approach for each verification task in one regression run. Streaming the project information in a web-based dashboard has helped the leads to track the status of each project and capture highly critical bugs earlier in the development cycle. Using the history of dashboard data, grouped across different RTL releases, failure analysis has become lot simpler and this helped leads to have an additional focus on priority modules and features during project execution, thereby improving productivity, saving compute resources and cost. Table 2 compares the resource usage data trend for a project across different RTL releases in the course of project execution.

TABLE 1

Time comparison between manual and ARF approach for each verification step in one regression run

Verification Task/Comment	Manual Approach (in hours)	ARF Approach (in hours)
Regression setup	0.5	0.25
Regression results analysis	1	0.3
Coverage merging	0.5	0.2
Update test execution status in project tracking system	3	0.25
Comment	Total time taken by an engineer-5hr	Total time taken by ARF-1hr
	Time saved with ARF compared to manual approach-4hr(80%)	

TABLE 2

RESOURCE USAGE COMPARISON BETWEEN MANUAL AND ARF RUNS FOR A PROJECT ACROSS DIFFERENT RTL RELEASES

RTL Releases	Week Number	Regression Cycle	Number of simulations		Number of LSF compute resources		Number of Tool Licenses	
			Manual Runs	ARF	Manual Runs	ARF	Manual Runs	ARF
R1	W1	C1	2000	2000	500	500	2000	2000
		C2	2000	2000	500	500	2000	2000
		C3	2000	2000	500	500	2000	2000
	W2(CP)	C1	2000	1500	500	375	2000	1500
		C2	2000	1500	500	375	2000	1500
		C3	2000	1500	500	375	2000	1500
R2	W3	C1	2000	2000	500	500	2000	2000
		C2(CP)	2000	1000	500	250	2000	1000
		C3	2000	1000	500	250	2000	1000
	W4(CP)	C1	2000	500	500	125	2000	500
		C2	2000	500	500	125	2000	500
		C3	2000	500	500	125	2000	500
R3	W5	C1	2000	2000	500	500	2000	2000
		C2(CP)	2000	1500	500	375	2000	1500
		C3	2000	1500	500	375	2000	1500
	W6	C1	2000	1000	500	250	2000	1000
		C2	2000	1000	500	250	2000	1000
		C3	2000	1000	500	250	2000	1000
Comment	Significant reduction in resource usage for ARF runs by leveraging the history saved on the dashboard. **Check Point (CP)**: After every one week or RTL release, the number of simulations that need to be run were decided based on the previous week data or run. In the above table, the simulations belong to pass modules are removed from the regression cycle after every Check Point, effectively translates to reduction in compute resources and tool licenses.							

REFERENCES

[1] (Arachchi, S. A. I. B. S., Perera, I., 2018) "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management", Moratuwa Engineering Research Conference (MERCon), doi:10.1109/mercon.2018.8421965.

[2] (M. Fowler, 2017) "Continuous Integration", martinfowler.com, [Online]. Available: http://martinfowler.com/articles/continuousIntegration.html.

[3] Bamboo documentation. [Online]. Available: https://confluence.atlassian.com/alldoc/bamboo-documentation-directory-23855144.html

[4] Bamboo REST API documentation. [Online]. Available: https://docs.atlassian.com/atlassian-bamboo/REST/