2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Agile Approaches to ASIC Verification (A3V) – A Novel Agile Flow in Functional Verification

Adithya Rangan CK, Senior Engineer, Infineon Technologies, Bangalore, India
(AdithyaRangan.ChakravarthyKannan@infineon.com)

Vidyasagar Kantamneni, Senior Manager, Infineon Technologies, Bangalore, India
(Vidyasagar.Kantamneni@infineon.com)

Vishal Dalal, Principal Engineer, Infineon Technologies, Bangalore, India
(Vishal.Dalal@infineon.com)

***Abstract*— ASIC development is undergoing rapid growth in the current digital world scenario in terms of innovation and implementation. Specifically, there is humongous advancement in the domain of digital design, verification and validation of IPs and SOCs. As RTL designs are evolving forward in the direction of re-usability and complexity, there arises a need for versatile and robust strategies in functional verification. Various challenges related to cost, quality, resources and time posed by complex designs need effective solutions through innovative Design Verification (DV) approaches.**

Agile Approaches to ASIC Verification (A3V) is one such state-of-the-art DV flow based on agile methodologies used in software development and testing related project executions. A3V flow is a conglomeration of various agile processes that involves leveraging the benefits of each framework and its application to functional verification. This paper presents the strategy and implementation of A3V flow for a highly configurable and scalable IP used widely in Memory Subsystems. A3V is compared with traditional verification approaches like waterfall model, requirements driven flow etc. and results prove to be encouraging in improving the overall efficiency of DV execution and quality of design, thus augmenting focused efforts towards first-pass silicon.

Keywords*— *Agile; Functional Verification; Feature-Driven-Development; Extreme Programming

I. INTRODUCTION

ASIC development is undergoing fast evolution across various digital platforms involving design and implementation of IPs and SOCs. With increasing complexity and challenging time-to-market scenarios, RTL designs are progressing in the direction of re-usability and complexity. Such designs require versatile and robust verification strategies that involve improvised functional and formal approaches. Thus, complex designs pose various challenges towards verification efforts which inherently brings the need for optimization in DV processes to have higher efficiency in improving quality, and optimize cost, resources and time.

Traditional strategies such as the waterfall model, requirements-driven-flow (RDF) etc. are being implemented in both the conventional software and VLSI design/verification domains. However, these strategies are associated with many disadvantages. Few to mention are the risks associated with the changing requirements and the progress which cannot be measured within standalone execution stages. RDF has fixed priorities on project tasks which may hinder progress when struck with complex bugs, thus delaying the sign-off and adding to the cost factor [1]. In order to counter such problems, agile techniques are being employed in various industries, particularly in software development and testing. Agile methodology is a combination of various frameworks that involves generation of solutions for requirements based on project task division into small phases where task execution is sequential in nature. Each consecutive phase is associated with iterative and incremental changes resulting in continuous value addition. Continuous collaboration and periodic assessments lead to better visibility of progress that can be measured with quantitative metrics [2].

Various techniques from Agile framework exist out of which any of the chosen techniques is typically applied and implemented based on the nature of project [3]. However, each individual technique has its own list of merits and overheads which may result in trade-off among quality, time and resource-utilization. For example,

disadvantages may be in the form of tight-schedules and aggressive sprints for Scrum exclusively or need of experienced coders for Extreme-Programming [4]. In-order to overcome specific pitfalls and arrive at best possible solution where verification is vivid and robust, there is a need to actuate Agile methodology where the advantages can be utilized based on the requirements, scope and resources associated with a project. In this direction, this paper proposes a novel flow named **Agile Approaches to ASIC Verification (A3V)** that incorporates different frameworks by identifying the characteristics of design verification. Benefits of each of these Agile techniques are derived to achieve best possible results that add value to functional verification.

A3V flow mainly focuses on fine-tuning the DV flow alongside utilization of existing tools and resources. The flow adheres to Agile principles and core-values with a combination of merits aggregated from frameworks such as Scrum, Extreme-Programming (XP), Feature-Driven-Development (FDD), Adaptive-Software-Development (ASD), Behavioral-Driven-Development (BDD) and Kanban. A3V flow involves division and categorization of features or sub-features in sync with the DV requirements and mapped onto the verification plan. It consists of sequential implementation of prioritized tasks and sub-tasks dynamically in line with the prioritized features. Common tasks are grouped together and executed in a peer manner so that there is an incremental value addition in terms of test-bench changes, assertions, test-cases, code or functional coverage. Every block or sub-block related to tasks or sub-tasks is verified and reviewed for 100% coverage which ensures successful implementation of consecutive blocks. A status tracker is maintained with a list of pending tasks and backlogs which are regularly addressed through continuous reviews/retrospective meetings between all the teams involved in the DV project. Each iteration provides incremental improvement and all the chunks are aggregated at the end for comprehensive sign-off satisfying the requirements and specifications.

A3V flow is applied on a highly configurable and scalable IP which is widely used in Memory Subsystems; henceforth termed as MIP in this paper. A list of features involving verification of memories through testing algorithms on the MIP is prepared. Requirements consist of developing a User Interface (UI) with configurability options such as clock frequency variation, choice of memories (ROM/RAM), choice of algorithms, fault detection features etc. Features are fragmented and mapped onto blocks or tasks based on their dependencies. Execution is planned in sprints of two weeks. Features are also categorized as basic, intermediate and advanced based on their complexities and prioritized based on the well-known MoSCoW(Must Have, Should Have, Could Have, Would Have but not now) model of agile frameworks [5]. Development and verification of each feature is carried out in parallel through scripting for automation and verification tests. Any blockage or failure is addressed through daily retrospective meetings and tracked through status tracker which is reviewed periodically till task completion and coverage closure. Thus, this paper elaborately discusses A3V flow strategy with block diagram and implementation for a use-case example in a live design with associated results supported by advantages in comparison with traditional strategies.

II. A3V FLOW

A. Background

Conventional approaches for ASIC DV are usually plan-driven and exercised for large systems and teams. Some of these approaches include waterfall model, requirements driven flow, rapid application development, prototype modelling, etc. These might be suitable for critical systems which are monotonous and functional-specific in nature. However, current trends demand continuous upgradation in response to changes in requirements that cannot be incorporated in the aforementioned approaches. Some of these demerits are as follows [6] –

- Applicable only to stable environment with limited alterations to design or verification
- Require experienced personnel right from the planning stage till sign-off
- Limited or lack of user involvement at various stages of development or verification cycle
- Increased cost, time and resources effort to incorporate even small changes
- Progress is completed only at the end of cycle which sometimes can be invalidated and discarded

The above list contributes towards additional overheads on quality of deliverables and can cause adverse impact on unit cost, cost-to-market and time-to-market.

B. *Agile Frameworks used in A3V Flow*

In-order to mitigate the problems faced by conventional approaches, different agile frameworks are incorporated based on their advantages in improving the efficiency of functional verification of complex designs. They are listed as follows –

- *Scrum* – task/block division based on dependency and execution in well-defined schedule of sprints classified into demo, intermediate and final versions based on priorities. Consecutive tasks/blocks are executed only after 100% completion of previous tasks/blocks. Each sprint consists of reviews and retrospective meetings to ensure proper adherence of status with respect to planned schedule of task completion.
- *Feature-Driven-Development (FDD)* – classification and fragmentation of features/sub-features derived from the DV requirements. Priorities are assigned to each feature/sub-feature based on the MoSCoW model. Execution follows the steps of plan, design, verify and close.
- *Extreme-Programming* – sharing/substitution of resources for design, verification of blocks and fixing bugs in parallel. It also includes peer reviewing and code optimization for testbench and testcase development, assertions, coverage closure, etc. with incremental progress.
- *Adaptive-Software-Development (ASD)* – supports to addition or enhancement of features based on changes in requirements. This also includes verification of all possible scenarios such as those classified as basic, intermediate and corner cases.
- *Behavioral-Driven-Development (BDD)* – generates prototype testcase to fail a scenario followed by testbench implementation of the feature till it finally concludes into testcase pass and coverage closure.
- *Kanban* – a billboard that acts as a status tracker to provide information on completed, pending and yet-to-start tasks/blocks mapped onto the features.

Table I provides a mapping between the verification activity and the agile framework incorporated in the A3V flow.

Table I. Mapping Agile Framework to Method Description

Sl. No	Description	Agile Framework
1.	Categorization and prioritization of features/sub-features with Demo, Retrospective and Final versions associated with well-defined time schedule (sprints) and iterations	FDD, Scrum
2.	Fragmentation of blocks/sub-blocks and dynamic execution or adaptation of tasks based on basic, intermediate and final versions	FDD, ASD
3.	Continuous review and parallel execution of design-verification tasks within bound time-frame including basic and corner case scenarios	XP, BDD, Scrum
4.	Track priorities, status & action items for each sub-task/task corresponding to sub-block/block and converge towards feature/sub-feature implementation and closure	Kanban, Scrum

A3V flow is a derivative of all the above listed agile frameworks considering their merits and tailoring them based on the nature and scope of functional verification for defined features and requirements [7][8][9].

C. *Flow Diagram*

Figure 1 shows the A3V flow diagram as applicable to functional verification of ASIC design. The flow begins with the Project Start that is aggregated by RTL specifications and requirements, supported by the concept and architecture, and executed through RTL design implementation. The next step is to define the DV environment that consists of verification requirements with detailed description and documentation of the same. Testbench and testcase development constitute the implementation part of verification while functional/code coverage and assertion-based coverage correspond to verification metrics.

The next stage involves fragmentation of features F1, F2…Fn into sub-features SF11, SF12…SF1n; SF21, SF22… SF2n, etc. based on complexity and inter-dependency. The features are prioritized based on the MoSCoW model and their feasibility for basic, intermediate and advanced versions satisfying the requirements. This is followed by the verification planning stage where the features and sub-features are mapped onto the requirements and verification specification with priorities labelled as P1, P2… Pn, thus creating the verification plan database and consequently the verification specification format. With changing requirements, there is ample scope for the priorities to change for features thereby providing flexibility and adaptability to upgradation. The test-bench development and implementation stage involve actual realization of the features through various tasks/sub-tasks T1, T2, T3…Tn associated with blocks B1, B2…Bn and sub-blocks SB11, SB12…SB1n; SB21, SB22… SB2n, etc. Each feature/sub-feature is implemented sequentially through serial execution of tasks on blocks/sub-blocks.

The next consecutive part is executed only after 100% closure through implementation, analysis and metrics of previous part is achieved. While the division of features follows a top-down approach, execution of tasks follows a bottom-up approach. Every closure of individual feature is denoted by an individual sign-off SO1, SO2… SOn which concludes to final comprehensive sign-off which deems the project complete.

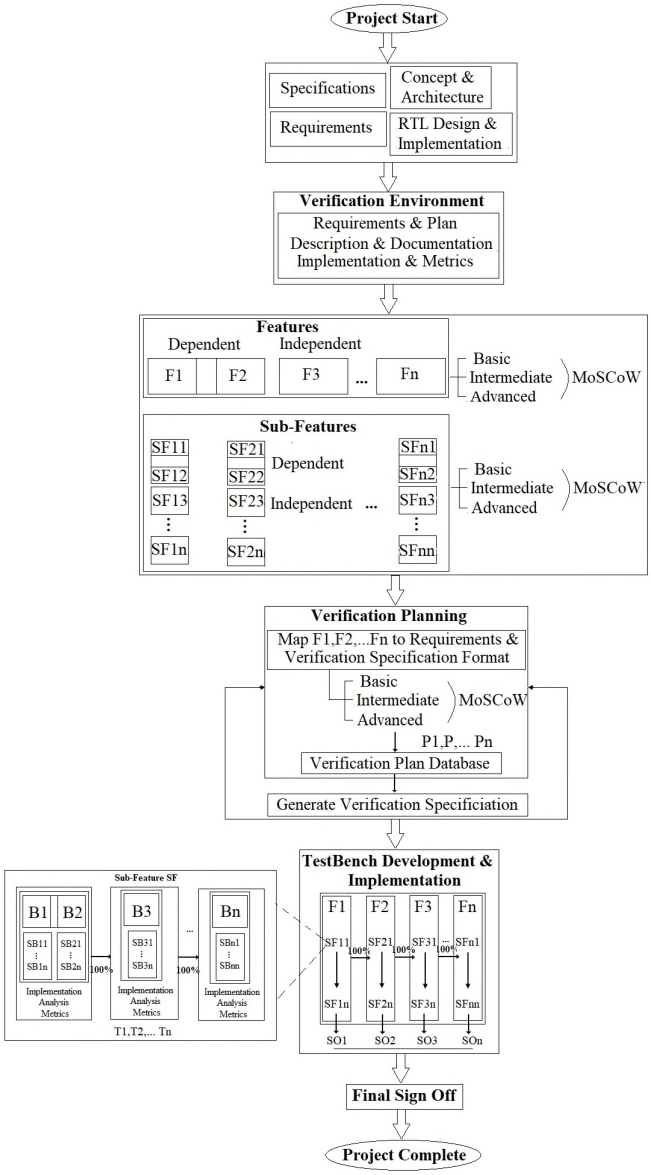

Figure 1. A3V Flow Diagram

Table II depicts the detailed comparison between A3V and conventional verification flow [10][11][12].

Table II. Comparison between Conventional Verification Flow and A3V Flow

Sl. No.	Conventional Verification Flow	A3V Flow
1.	Plan-driven and requirements-oriented	Agile and features-oriented mapped onto requirements
2.	Generalized or fixed priorities for tasks/sub-tasks throughout the entire execution cycle	Specific or ad-hoc priorities for tasks or sub-tasks at every stage in the execution cycle
3.	Concurrent execution of blocks/sub-blocks with specific features of interest (multi-block basis)	Sequential execution of blocks/sub-blocks covering all features consecutively (block-by-block basis)
4.	Ensures coverage metrics (ex: 50%, 75%, 100%) in parallel for multiple stages till final sign-off	Ensures overall coverage (100%) and individual sign-off for each stage serially till final sign-off of all stages
5.	Limited level of visibility and clarity of blocks/sub-blocks due to isolation of tasks/sub-tasks implementation	Better visibility and clarity of blocks/sub-blocks due to sharing and continuous review of tasks/sub-tasks implementation
6.	High probability of repetition/duplication of tasks/sub-tasks due to less scope for adaptability to dynamic changes	Low probability of repetition/duplication of tasks/sub-tasks due to high scope for adaptability to dynamic changes
7.	Frequency of status alignment meetings is low initially and increases towards the sign-off	Frequency of status alignment meetings is very high initially and decreases to constant value towards sign-off
8.	Non-iterative and non-linear progress which can be concluded only at the final stage of project	Iterative and incremental progress at every stage ensures continuous assessment of project status

III. USE CASE EXAMPLE

A3V flow is applied to a use-case example that involves development and verification of a user-interface that automates feature-implementation for the MIP. The requirements correspond to verifying combinations of algorithm execution for different memory units. Some of the features are listed as follows –

- Clock frequency variation
- Initialization of memory address access
- Selection of memories – RAM / ROM
- Selection of algorithms – SCAN, MARCH, READONLY etc.
- Selection of memory accesses – Sequential / Concurrent
- Selection of interfaces – Serial / Parallel

Each feature requires to be automated through the user-interface and thereby reflect the changes on the test-bench environment. Various levels of fragmentation are done by identifying the dependent and independent features with assigned priorities for each as depicted in Figure 2. For example, memory type (ROM/RAM), clock frequency variation and memory address initialization are identified as independent features F1, F2 and F3 respectively in the hierarchy. Level 1 lists different ROMs and RAMs such as ROM1, ROM2 and RAM1, RAM2 as sub-features respectively. At Level 2, testing algorithms such as SCAN, MARCH and READONLY are identified as dependent features on Level 1. Sequential/concurrent access of memories are identified for each at Level 3 (SEQ/CON) which is then applied to access via serial/parallel interface (SI/PI) for each at Level 4. Selection of memory address initialization (Mem Init) and clock frequency (Clock Freq) variation are dependent features which are applied to the hierarchy.

Blocks/sub-blocks definition and tasks/sub-tasks implementation correspond to levels of hierarchy as depicted in Figure 2. Each task/sub-task is executed in bottom-up approach from Level 4 to Level 1 sequentially until each

ladder is completed with analysis and coverage metrics. F2 and F3 are implemented in parallel and upon closure of F1, they are super-imposed on each other to cover all possible scenarios. Metrics in the form of test regression results (pass status), code and functional coverage, assertions-based coverage act as measurables to ensure completion of features which are then annotated onto the requirements and specifications.

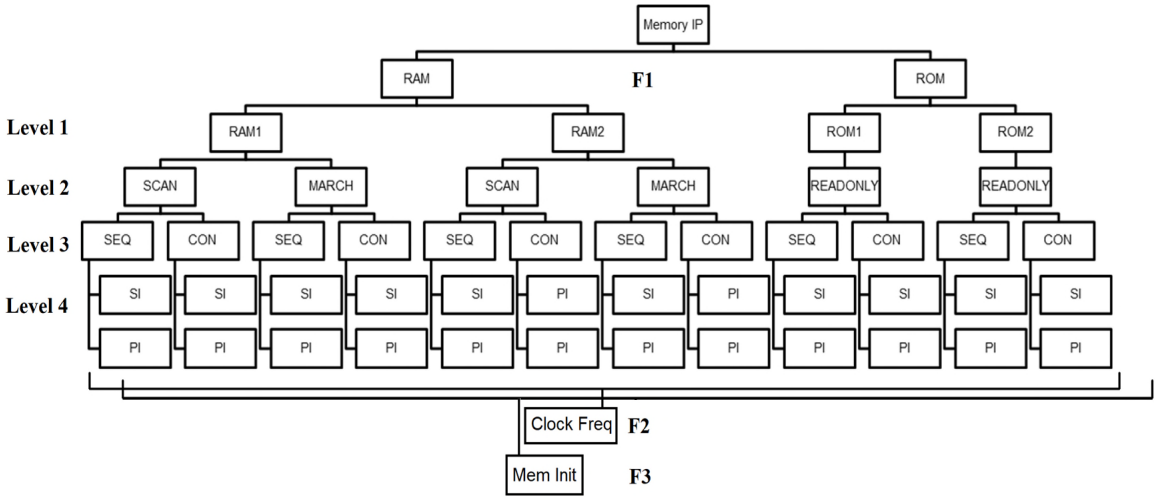

Figure 2. Feature fragmentation hierarchy for MIP

The whole process was driven by a small team of four personnel, with existing tools and resources. The target duration for completion was around 20 work weeks which was divided into a sprint of two weeks each for individual feature/sub-feature closure. Peer sharing and substitution of tasks/sub-tasks execution based on XP was followed with rigorous retrospective and review meetings on every alternate day based on Scrum for status alignment and backlog refinement with status tracker as per Kanban method. Continuous sync up meetings between the design and verification team ensured adaptability of test environment with respect to dynamic changes in the design.

IV. RESULTS

Results are measured in terms of metrics related to time, resources, effort and cost. It was observed that there was no extra cost factor since existing tools and resources were sufficient to drive the verification. In contrast to traditional strategies where effort was multiplied for each stage of implementation, optimal effort was required in case of A3V flow where every stage involved all the resource personal working on the same block till completion and closure. Taking timeline into consideration, the planned period for completion and closure of UI features was 20 work weeks. Each sprint consisted of 2 work weeks directed towards completing each feature or sub-feature. Relatively less complex feature implementation such as clock frequency variation and memory address access initialization required less than each sprint to complete thereby providing opportunity to implement other complex features. Complex features such as algorithm implementation on different memories with combination of serial and parallel interfaces for sequential and concurrent memory access consumed previously available time; thereby feature closure concluded within the expected timeframe.

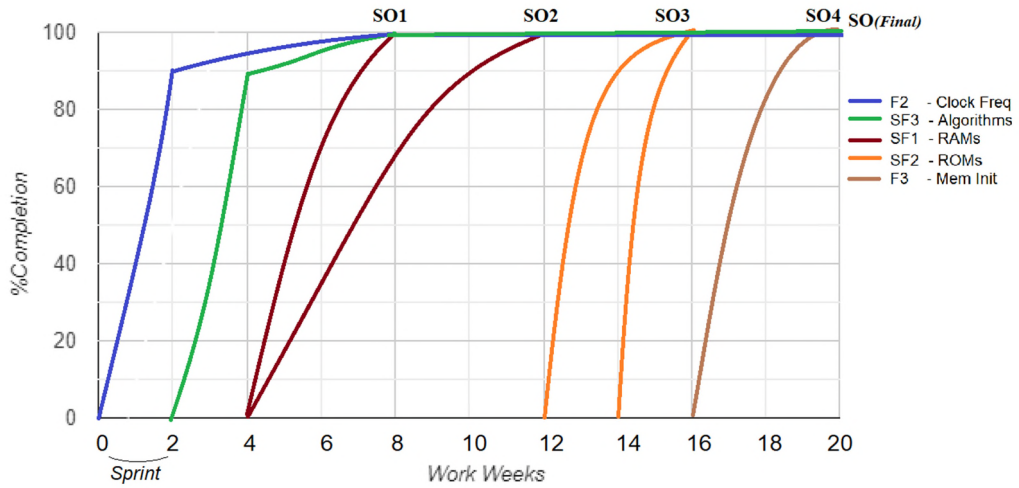

Figure 3. Graph of Percentage Completion versus Work Weeks

Figure 3 depicts the graphical representation of percentage task completion versus the timeframe in terms of work weeks for overall feature closure. As observed from the graph, clock frequency variation F2 is implemented first due to its non-dependency with pending verification till at-least one algorithm for memory is available till the next available sprint. Algorithms SF3 are implemented in the second sprint and verified once the RAMs SF1 are available at the third sprint. SF1 RAMs (RAM1, RAM2) and SF2 ROMs (ROM1, ROM2) are depicted by two lines in the graph. ROMs are implemented relatively faster due to already improved verification environment in previous sprints. The last stage involves application of F3 Memory Address Access Initialization feature for all the available algorithms and memories. Each stage of completion at 100% corresponds to individual sign-off SO1, SO2, SO3 and SO4 corresponding to Level 1, 2, 3 and 4 till the final comprehensive sign-off SO (Final) at the end of 20 work weeks.

V. OPPORTUNITIES VERSUS CHALLENGES

A3V flow provides a novel approach for functional verification of complex designs especially those involving large number of functionalities to be verified within stringent time duration.
Some of the opportunities of the A3V flow are as follows –

- Better visibility and clear distinction of features, blocks and tasks ensures easy approach to verify in terms of fixing bugs, adapting to future changes etc.
- Iterative and incremental progress of tasks with constant retrospection ensures tangible deliverables.
- Peer reviews and work-sharing ensures higher rate of success with code optimization.
- Sequential execution and closure of blocks ensures better quality and high-level of confidence with minimal rework.
- Status tracking ensures proper alignment of tasks ensures optimum utilization of time, resources, effort and cost.

Some of the challenges that might be faced towards implementing A3V flow are as follows –

- The verification environment has a dependency on the design to be agile in nature.
- The inconsistencies between intra-dependencies and inter-dependencies have to be resolved during the verification planning phase.
- Verification plan should be precise and accurate with respect to classification of features and prioritization of tasks. They should also be correctly mapped on to the requirements.

VI. CONCLUSION

A novel approach of using agile methodology in functional verification named A3V Flow is presented in this paper. It is a derivative of various agile methodologies considering the advantages and applying them to suit the requirements of the project. A use-case example of a highly configurable and scalable Memory Subsystem IP is driven with A3V flow and results are observed to be beneficial as compared to the existing

conventional verification flows. Opportunities and advantages of A3V flow are enumerated with certain challenges faced during the implementation. Thus, A3V flow has proven to be successful in bringing out an efficient verification strategy with optimum utilization of resources, efforts, time and cost. Consequently, the flow has made a significant contribution in enhancing the overall quality of verification that can augment focused efforts towards first pass silicon.

VII. ACKNOWLEDGEMENT

The authors would like to extend sincere thanks to Manu Baby, Johannes Grinschgl, Alexandru Ghica, Arlin John and all other team members for their valuable support.

VIII. REFERENCES

[1] Shinobu Komai, Hiroshi Nakanishi and Hamdani Saidi, “Guidelines for selecting agile development method in system requirements definition”, 7th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), November 2017, ISBN:978-1-5386-3898-9

[2] Marco Kuhrmann, Paolo Tell, and Regina Hebig, “What makes agile software development agile,” IEEE Transactions on Software Engineering, vol. 1, pp. 1-1, July 2021, ISSN: 1939-3520

[3] Shi Zhong, Chen Liping, and Chen Tian-en, “Agile planning and development methods”, IEEE 3rd International Conference on Computer Research and Development, March 2011, ISBN:978-1-61284-840-2

[4] Wrike, “Agile Methodology Basics - Project Management Guide”, https://www.wrike.com/project-management-guide/agile-methodology-basics, 2009

[5] Mike Beedle, Arie van Bennekum, et. al, "Agile Manifesto and Principles", https://agilemanifesto.org, February 2001

[6] F. Paetsch, A. Eberlein, and F. Maurer, “Requirements engineering and agile software development,” WET ICE Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June 2003,ISBN:0-7695-1963-6

[7] Yunsup Lee, Andrew Waterman, Henry Cook et. al, “An Agile Approach to building RISC-V Microprocessors IEEE Micro, Vol. 36, Issue.2, March 2016, ISSN: 1937-4143

[8] Luke Collins, “Applying agile techniques to IC design,” https://www.techdesignforums.com/practice/technique/agile-ic-methodology/, June 2015

[9] Neil Johnson and Byran Morris, “A Giant, Baby Step Forward: Agile Techniques for Hardware Design,” http://www.synopsys.com/news/pubs/snug/boston2009/mc3_johnson_paper.pdf, 2009

[10] Paul Cunningham, "Agile Approach to SoC Design Verification", Cadence Design Systems, https://www.eetasia.com/agile-approach-to-soc-design-verification, June 2021

[11] Md Shamsur Rahim, AZM Ehtesham Chowdhury, Dip Nandi, Mashiour Rahman, "ScrumFall: A Hybrid Software Process Model", I.J. Information Technology and Computer Science, vol 12, pp 41-48, December 2018

[12] Sergio Marchese, "Formal verification enables Agile RTL development", https://www.techdesignforums.com/practice/technique, January 2014