An integrated Hardware/Software Verification and Validation methodology for Signal Processing Systems

Manokar Venugopal, Manju Nanda, Anand G[*1], *Hari Chandana Voora*

CSIR-National Aerospace Laboratories, Kodihalli, Bengaluru, Karnataka, India, 560017

Abstract. The testing and validation services team assesses project deliverables at various stages of development using innovative and effective verification and validation, to ensure that the deliverables are compliance with the customer specifications and requirements. Whenever new products and devices are released, completely integrated verification and validation services are delivered to accurate and complete records usability, performance, and quality assurance services. Throughout the product development and testing process, the testing and validation services team employs verification and validation techniques. Code reviews, walk - through, inspections, desk-checking, and code execution are all examples of verification and validation techniques. Services for verification and validation are used to assess whether or not the software or application provided complies with the requirements and serves the intended purpose. A procedure used to ensure that the software created is of good quality and consistently operates as expected is independent testing and validation services. Unit testing (also known as "White Box Testing"), hardware-software integration testing (HSIT), and system testing are the three primary independent verification and validation approaches (Black Box Testing). The teams responsible for the verification and validation services actively participate in each stage of the project and design the services according to the project's needs (e.g., prototype, spiral, iterative, V Model, and Agile). Our expertise in the embedded domain, tried-and-true verification and validation techniques, and a thorough methodology provide a quick turnaround and excellent results for the targeted solution.

Independent Verification and validation services covering

- Source code, design, and requirements
- White box testing, or unit testing
- Testing for hardware-software integration
- Black box testing, or system testing

❖ To reduce test cycle-time significantly on test Automation solutions.

❖ Verification and validation techniques can be used to effectively and efficiently carry out stress and performance tests, and to detect defects early in the life cycle.

* Corresponding Author: g.anand777@nal.res.in

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

- Documentation of test process.
- Liaison and Certification

Key Words: Software Process, Software Tools, Unit Testing, Integration Testing, Verification and Validation Techniques.

1. Introductory

System design teams, software design teams, and software verification teams are all included in the engineering and development organization. The manager is in charge of all aspects of integration and development of the software applications. Workers who build systems and software all accountable to the management. The System Design Team (ST) develops performance requirements along with functional requirements for hardware. Based on recommendations from the system design team, the software design team (DT) creates software design specifications and software. The Software Verification Team (VT) performs various actions for the software's verification and validation. In order to increase the quality of the software, the V&V process is carried out in a way that keeps it separate from the design process. By completing V&V activities independently and severing VT from the design group, VT may confirm compliance with the independence standards.

Communication between VT and the design group should be documented in written reports. An embedded system may be a control mechanism that features hardware and software components. Hardware can include computers, microcomputers, and/or microcontrollers. Software components ensure functional and logical control of hardware. a major think about the event time, cost, and complexity of embedded systems is software development. While hardware enables the system to try to to useful work, software controls embedded systems at every level. Base software is answerable for the low-level functional control of hardware components, ensuring that they're properly configured and integrated into the physical system during which they reside. Application software controls the info input, typically provided by external sensors, and the way that input is processed by the system Applications provide the behavior of hardware systems.

Although hardware presents its own complex challenges, this document focuses on software development to deal with the varied complexities involved in developing large-scale, real-time critical systems. The products we design are very complex. the look and quality assurance process is extremely complicated. The tools used and also the process of using those tools are generally complex. These compartmentalized considerations emphasize the necessity for theoretical models for large-scale system integration. Software application tools that provide connectivity, reduce interface and integration complexity, and ensure traceability through all phases of software development, considering design, development process, testing, verification, and validation issues is very necessary. This white book provides some possible considerations of both potential commercial software application problems and solutions, enabling a potentially integrated all-in-one process for the verification, validation, and testing phases of a design. to it end, I propose some aspects that are currently empty. Projects focused on basic and application software moreover as configuration and alter management. There could also be many possible solutions, but this report presents ideas that may help speed up the method of designing, verifying, validating, testing, and reporting on embedded software systems.

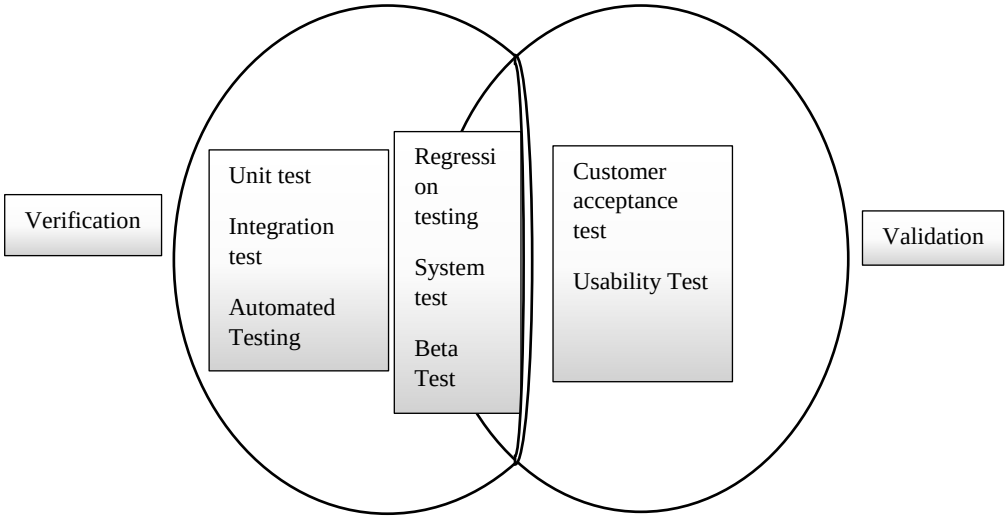

Fig. 1. Verification Vs. Validation test graph

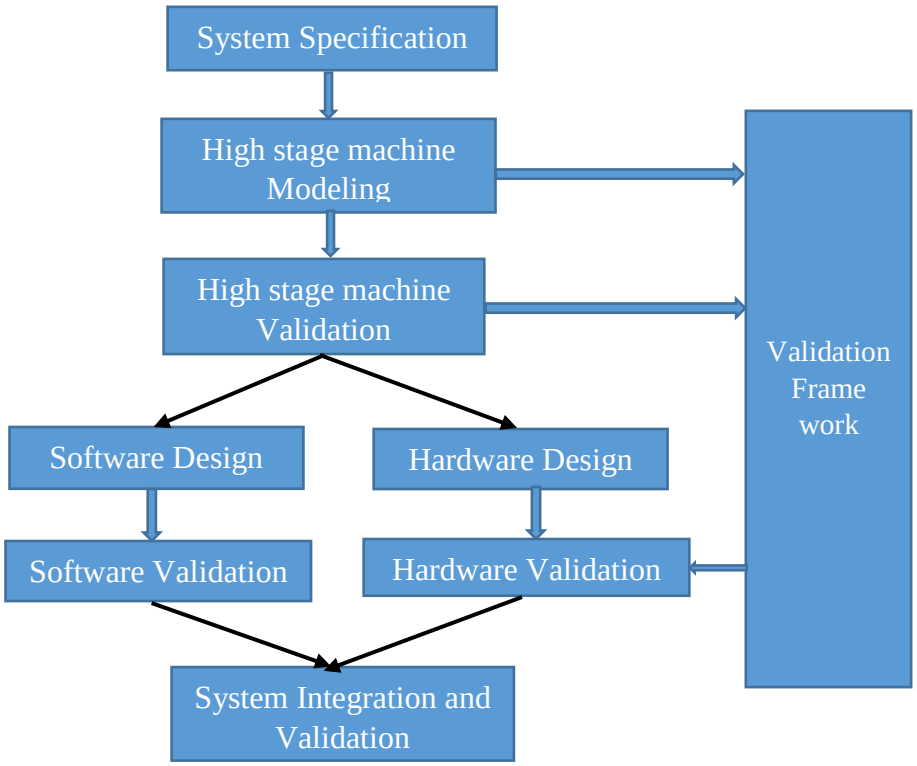

Fig. 2. Hardware-Software Design Validation Flow

2. Overview of a software development process

Hardware/software co-verification is intended to verify that the embedded system software works correctly on the hardware design representation. Validation means accuracy, validation shows the value of the final product or the final product. Software validation, testing, and maintenance is essential to reduce the risks associated with the use of technology. It is best to do validation and testing before using the system. After successful testing, proper maintenance is required to confirm that the software is functioning effectively. The 2 accompanying sections divide hardware, software, and communications into three subjects, whose operations are often interdependent, and therefore the following verification, testing, and maintenance procedures are divided into three Sometimes it's necessary to mix all elements.

2.1. Hardware/Software Testing and Maintenance

- Software testing to confirm that the appropriate standards are met and that the software performs the intended functions, under actual conditions, including storage, transportation, operation and maintenance environments.
- verifying that the code is logically correct.
- Ensure equipment meets local environmental requirements, including housing, space, furniture and accessories, power supply, and extremes of temperature, humidity, and pollution.
- applying functional tests to determine whether the test criteria have been met

A V-model software development process that has to be modified to incorporate a concurrent test design phase derived from the fundamental phase of the look process. Extensions to the V-model, like the V-model, provide further parallelism by allowing the phases to be layered within the time domain to the extent that the stages themselves are orthogonal designs tangent to the first V. This leads to a linear reduction in development time. A key feature of this diagram is that the test design is associated with both the design arm and the (V&V) verification and validation arm phases. This process suggests that test suites can be easily linked throughout the design process for complete verification and verification, but this is not the case. A web survey of validation and validation test process software application tools quickly reveals a lack of integrated tool support for linking the final stages with pre- and post-design documentation support widely used in the design branch of the process. will be Various commercial software tools are available that provide process steps, some of which approach the complete solution individually or step by step.

2.2. Verification and validation methods

Model-based design and development has become a standard practice in the engineering industry. There are several tools that have proven powerful and reliable for graphical modeling and simulation of common engineering systems such as manufacturing, electrical, medical, computing, mechanical, and communications. Commercial software simulation tools are currently in a very advanced stage of development and have long proven their usefulness and reliability in many technical areas of the global marketplace. The concept of graphical modeling is a simple representation of a physical system with inputs and contains functional logical outputs. This approach can be top-down or bottom-up hierarchical, where each black box can contain multiple subsystems, with the lowest

level containing basic logic and arithmetic operations and the required It can even lead to bit-level control depending on the architecture of a given physical system is laid out graphically or designed to make the underlying logic easiest to conceptualize and understand.

Validation and validation as an integral part of control theory describes the "optimal control problem formulation". Physical limit description and specification are the first two phases, and performance measurement is considered the last of the three main phases of problem formulation. Here's the problem: Mathematical modeling, such as creating state diagrams, and specification of physical boundary conditions correspond to system requirements. These phases form the leftmost branch of the V-Design paradigm for designs that need to undergo verification. Each hardware and software component of the system may have its own requirements document, down to the lowest level. The utility of this approach is that in an exceedingly well-modeled system using the proper software tools, the software controlling the model may be optimized because the source control software for the microprocessors and microcontrollers of the important physical system. . This approach tests custom hand-written or software computer-generated code at multiple levels, employing a multitude of interfaces that progressively move closer to integration into the ultimate physical system, either through iterative verification or into the verification approach itself. there's also a bent towards This iterative approach is widely known in modern design engineering as a series of in-the-loop processes for MIL/SIL/PIL/HIL testing.

2.3. "V" Development Process

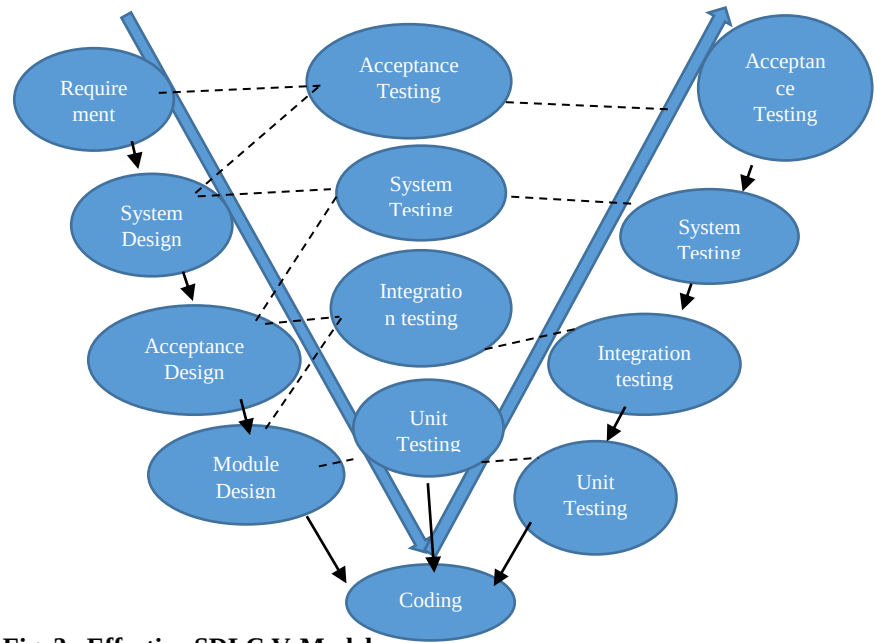

Fig. 3. Effective SDLC V-Model

2.3.1. Avionics Development Method:

The design process consists of 6 parts activities, followed by the objectives for each step as it is shown in Figure.

a) Requirement capture and analysis
- Aircraft Level Requirement
- System Level Requirement

b) Functional Hazard Analysis
- Aircraft Level FHA
- System Level FHA

c) Preliminary design of the related system
- Aircraft Level Architecture
- Functional Level Architecture

d) 4. Detailed design of hardware and software.
e) 5. Verification and Validation using simulation
f) 6. Obtain performance of the system
g) 7. Integration design for the overall system.

3. Integrating design process with validation/verification process:

Verification and validation procedures are specific measures to improve product quality and customer satisfaction. By applying these techniques to the earliest stages of development, significant cost savings can be achieved by iterating improvements in the earliest development stages possible. Even a typical engineering project for a system of low to medium complexity can become too complex when multiple tools are needed to accomplish different aspects of the overall system task. Modern software development projects often have multiple resource databases for specifications, requirements, project files, design and testing tools, change management, and report formats. a fully integrated process as a V-Gap bridge solves the problem of configuring multiple tools and databases to meet project requirements. Furthermore, such an approach can simplify the method in line with recent development trends that increase the use of model-driven design.

3.1. Potential benefits of an improved approach include

- Excessive levels of traceability lead to easy navigation through the business at all technical/management stages.
- Excessive level of concurrent innovation leads to reduced time/time of normal business innovation to market.
- Early/all-stage testing improves advanced product capabilities and reduces debugging costs.

4. Hardware/Software Integration

The time spent on an electronic systems development project can typically be divided into three basic phases: system, hardware/software design, integration, and testing. Interviews with 18 clients showed considerable consistency in the relative duration of each period. they all have roughly the same duration, each representing a third of the project's duration.

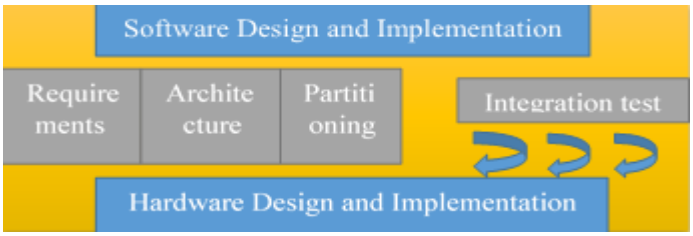

Fig. 4. Hardware and Software integration process

5. Embedded system design process overview

- **System phase**: In this phase, the entity to be designed is considered as a whole system, not as separate hardware and software components. This phase, usually performed by system engineers, ends with the specifications of how the system will function. Other distributions include system architecture and functional specifications. In addition, functional and budgetary requirements for the system's hardware and software components are created. These include cost limitations, size, performance, and physical attributes. At the end of this phase, the system is partitioned into software and hardware subsystems.

- **Hardware/Software Design phase**: In this phase, separate organizations (except in the case of small projects) deal with their respective problems. Hardware and software design and implementation efforts often begin at the same time and end at the same time. However, the work proceeds independently between the beginning and the end. Software engineers bridge the gap between the two design teams. Firmware engineers typically develop low-level software that communicates with hardware, providing a software foundation for higher-level software. the top level, or application software, is where the sole functionality of the commodity is typically implemented (e.g., call forwarding, engine control decisions.)

- **Integration and Test phase**: In theory, Integration and Testing is just the last round of testing before shipping the system. In fact, this is the first time that "complete" hardware and independently developed software have closed as a system. At this point, many problems arise, which are: consequences of misinterpreting the interface definition, outdated specifications, poorly communicated changes, and inefficient performance modeling, etc Thus, a third or more of the total development time is spent in this phase.

Faced with cost and planning delays, developers were forced to reorganize and/or reduce product goals. Due to the long production times and costs involved in redesigning an ASIC, the redesign is often done in software, which is not always the easiest solution to the best product. Integration and testing become redesign and re-implementation, and take the same amount of time as the original design and implementation. Software change costs also tend to be less visible than the cost of ASIC towers, and thus the end product can be compromised. Additionally, in some cases, a major product release will not contain all of the expected software features because it is not possible to initiate the incorporation effort earlier in the design. Two things are needed before integration and virtual testing is completed. the main one is the ability to emulate hardware at a speed sufficient to make software execution a reality. In most cases this implies that the hardware emulation performance should be increased by at least 1000 elements above the current execution speed. Second, they need to bring the debug and development environment closer to the hardware and software. No applied scientist would be happy to see waveforms when their development has occurred in the high-level language process.

5.1. Hardware/Software Co-Simulation Tool:

Based on an in-depth review of the wide selection of methodologies used in embedded system design, it appears that a good hardware/software co-simulation tool can have a profound impact on a wide range of factors. important success. Such a tool would provide an infrastructure for virtual integration by supporting a wide range of modeling techniques.

In addition, substantial production margins over traditional hardware emulation tools are expected to be realized.

5.2. Integration and Testing Process:

Electronic systems integration can be a methodical process with predefined goals and test scenarios. The avionics kit consists of a set of LRUs or systems for different functions. Systems are tested at various levels ranging from case or system to full integration testing, avionics integration and test patterns designed with the following objectives in mind:

1. Integrate and test claims about intended functionality
2. In accordance with FAR 25/23/121, FMET program and requirements.
3. To find out about interface and functionality issues, if any
4. To accept crew input and combine with other derived information
5. To activate the controls in the plane and simulator
6. external environment according to system task requirements
7. Research interface, interference and interoperability issues with the entire suite of avionics in integrated mode
8. Operational study based on failure scenarios of various types of display systems with a complete set.

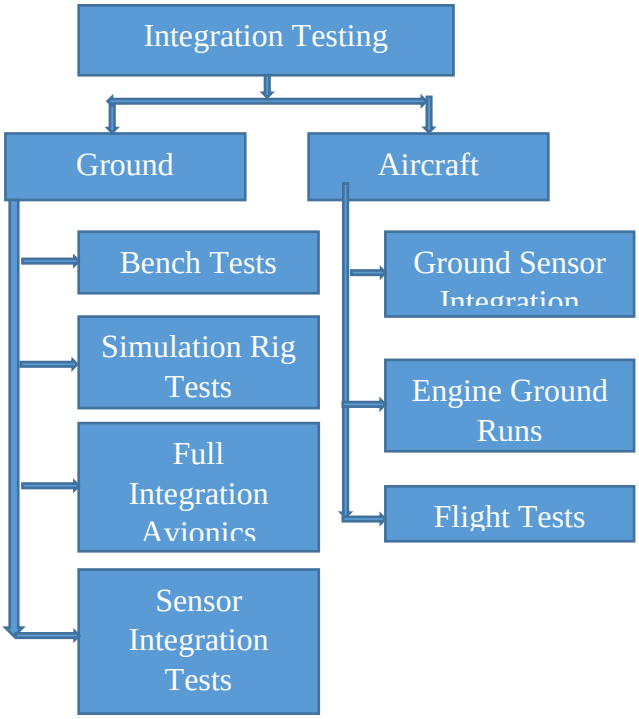

Fig. 5. Avionics system integration test mechanism

5.3. Hardware and Software Verification:

To test the hardware (HW) and debugging software (SW) running in the highly integrated system System on chip (SOC) poses technical problems. The processor cores embedded in

the SOC are no longer visible, as there are no more pins to connect the in-circuit emulator (ICE) and logic analyzer (LA) for debugging and analysis. ICE and LA require the address, data, and control bus for debugging, but these signals are hidden in the SOC. In addition to verifying hardware functionality, the methodology must take into account the growing amount of software used in consumer electronic products. This chapter covers the following topics:

- ❖ HW/SW verification environment and method
- ❖ Soft Prototype
- ❖ Verification and authentication
- ❖ Rapid Prototyping System
- ❖ FPGA based design
- ❖ Development of computer printed circuits
- ❖ Software Testing

The software prototype and HW/SW verification method are illustrated with a debugging example of the Universal Asynchronous Transmitter and Receiver (UART) utility used in the planning.

5.4. Hardware-assisted
5.4.1. Validation and Verification Platforms:

Hardware emulators and FPGA prototypes are not new technologies and have been around for decades. Broadly speaking, emulators after design capacity, speed of execution and debug visibility necessary to debug the system-on-chip (SoC) hardware, including software drivers and operating systems. FPGA prototyping complements emulation by providing the additional speed required for processing large software workloads and for running long regression suites.

5.4.2. At a closer inspection, five main differences separate emulators from FPGA prototypes:

- Design capacity/scalability
- Compilation speed
- Execution speed
- Design debug capabilities and
- Use models.

Best-in-class hardware emulators boast design capacity in the multi-billion ASIC-equivalent gates range with extensive scalability to support multiple concurrent users. They compile the design under test (DUT) at orders of magnitude faster than FPGA-based prototyping platforms. Their execution speed of a couple of megahertz coupled with massive input/output throughput support processing real-world workloads, including industry-specific frameworks and benchmarks. They provide design visibility, as well as bug tracing capabilities for quick and effective hardware debug. They can be deployed in in-circuit emulation (ICE) mode and in virtual mode. In ICE mode, the DUT is driven by the physical target system where it ultimately will reside driven by real-world traffic subject to random behaviour. In virtual mode, the DUT is exercise via software models leading to a deterministic and repeatable environment. Repeatability is critical to perform low-power design analysis, evaluate power estimation by keeping track of DUT internal

activity, and assess overall design performance before silicon availability. Deployment in virtual mode supports remote access 24/7 from anywhere in the world.

Traditional FPGA prototypes are resources deployed in ICE mode. With design capacity of less than one-billion gates, they trade off usage flexibility, hardware debug capabilities and quick compilation for two orders of magnitude faster execution speed than hardware emulation on the same design size at a fraction of its cost.
In that past couple of years, enterprise prototyping has been bridging the gap between emulation and traditional FPGA prototyping. In combination with emulation, the enterprise prototype increases the productivity of verification teams. By sharing several characteristics with the emulator, such as large capacity, extensive scalability, multi-users, and virtual deployment, the enterprise FPGA prototype can replace the emulator on-the-fly and provide the best attributes at each stage of the verification flow.

5.5. HW/SW verification Environment:

During the SOC system design cycle, an abstract model of the appearance is created and simulated. This abstract function is then mapped to an architecture-intensive system architecture and architectural performance modeling is performed. The architecture map divides the planning into hardware and software components, and so the specifications are passed on to the hardware and software team for implementation. The hardware team implements the hardware of the plan in Verilog or VHDL, using a hardware emulator to verify. The software team codes software modules in assembly language, C, or C++ and uses processor or ICE models to verify the software. Traditionally, the software team would then wait for a hardware prototype for final system integration. Many problems can arise during system integration. problems are caused by issues like misinterpreted specifications, incorrect interface definitions, and late design changes. Errors can be eliminated with an alternative in software, which can affect system performance, or with hardware modifications, which can be very expensive and time consuming, especially if it involves IC recycling. Moving systems integration forward in the design cycle helps catch these integration issues earlier. this will be achieved by creating a HW/SW verification environment early in the design cycle.

Some important areas for HW/SW verification environment are:

- Accuracy: Models used in the environment must be cycle or pin-accurate and be mapped to the SOC function.
- Performance: The environment must be fast enough to run real-time software (RTOS) containers and applications.
- Usability: The hardware and software team must be able to use the environment to verify functionality and performance.
- Availability: To meet time-to-market goals and enable HW/SW design and verification, the environment must be ready early in the design cycle.
- Cost: Depends on the method the environment is also considered due to accuracy, performance, number of users and system requirements.

6. Conclusion

- Usually, software verification ensures that particular software components or subsystems meet their design requirements, while the goal of validation is to

demonstrate that the overall system meets the requirements. customer requirements under the actual conditions.

- Validation is the technique of verifying whether the specification meets the customer requirements while validation is the technique of verifying that the software meets the specification.

- Hardware/software co-verification aims to verify that the embedded system software runs exactly on the representation of the hardware design. It does an early integration of software with hardware, before chips or boards. Here, the main focus is on System-on-Chip (SoC) verification techniques.

- In software project management, software testing and software engineering, verification and validation (V&V) is the process of verifying that a software system meets the specifications and requirements for it to fulfill its purpose.

- It helps to maximize the value of the software system as well as the users who use it and saves time and money by detecting defects at an early stage of the software development process. Reduce risk and improve software reliability and security.

References

1. Adnan Shaout, Dennis Breton "Validation and Verification for Embedded System Design – An Integrated Testing Process Approach" IEEE Trans, no. pp 1-3, ISSN: 2249-2593 http://www.ijcotjournal.org.
2. Syed Muslim Shah, Muhammad Irfan "Embedded Hardware/Software Verification and Validation using Hardware-In-the-Loop Simulation" IEEE --- 2005 International Conference on Emerging Technologies September 17-18.
3. Gitanjali R. Solanke, "Hardware Software Partitioning For A Digital System & Its Validation Using FPGA IEEE, International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 3, March – 2013 ISSN: 2278-0181.
4. An integrated hardware/software design methodology for signal processing systems journal homepage: www.elsevier.com/locate/sysarc
5. R.P.G.Collision, Introduction to Avionics Systems, Third Edition, Springer.
6. Skolnik, Introduction to Radar Systems, Third Edition, McGraw Hill Education.
7. Civil Avionics Systems, Second Edition by Wiley Ian Moir, Allan Seabridge and Malcolm Jukes
8. Software Verification and Validation Methodology for advanced Digital Reactor Protection System by Ki Chang Son, Hyun Kook Shin.
9. Specification,Synthesis and Validation of Hardware/Software Interfaces by Electronic System Design, Department of Electronics Electrum 229, Isafjordsgatan 22-26 S-164 40 Kista, Sweden.
10. Hardware Softawre Partitioning for a Digital System & its Validation Using FPGA by Gitanjali R. Solanke MITAOE Alandi International Journal of Engineering Research & Technology (IJERT) ISSN:2278-0181 Vol. 2 Issue 3, March – 2013.
11. Aircraft Design, Asystems Engineerimng Approach by Mohammand H Sadraey.