Tampere University

Sana Latif

CONTINUOUS INTEGRATION FOR FAST SOC ALGORITHM DEVELOPMENT

Information Technology and
Communication Sciences
Master Thesis
October 2019

ABSTRACT

Sana Latif: Continuous Integration for Fast SoC Algorithm Development
Master Thesis
Tampere University
Master of Science (Technology)
October 2019

Digital systems have become advanced, hard to design and optimize due to ever-growing technology. Integrated Circuits (ICs) have become more complicated due to complex computations in latest technologies. Communication systems such as mobile networks have evolved and become a part of our daily lives with the advancement in technology over the years. Hence, need of efficient, reusable and automated processes for System-on-a-Chip (SoC) development has been increased. Purpose of this thesis is to study and evaluate currently used SoC development processes and presents guidelines on how these processes can be streamlined.

The thesis starts by evaluating currently used SoC development flows and their advantages and disadvantages. One important aspect is to identify step which cause duplication of work and unnecessary idle times in SoC development teams. A study is conducted and input from SoC development experts is taken in order to optimize SoC flows and use of Continuous Integration (CI) system. An algorithm model is implemented that can be used in multiple stages of SoC development at adequate complexity and is “easy enough” to be used for a person not mastering the topic. The thesis outcome is proposal for CI system in SoC development for accelerating the speed and reliability of implementing algorithms to RTL code and finally into product. CI system tool is also implemented to automate and test the model design so that it also remains up to date.

Keywords: Continuous Integration, Mobile networks, Algorithm model, SoC, Communication system, DSP, Jenkins, CI, Modeling

PREFACE

This thesis is done with Nokia Solutions & Networks, Finland. I dedicate this thesis to my late father Muhammad Latif, who believed and trusted his daughter's dreams.

First and foremost, I would like to thank my supervisors Mika Kuulusa and Jukka Reunamaki for their admirable guidance and support during the whole process. I would also like to express my foremost gratitude to my examiners Prof. Mikko Valkama, Prof. Timo D. Hämäläinen and Dr. Antti Rautakoura for their valuable feedback. A special thanks goes to my line manager Sakari Patrikainen for giving me the opportunity to do this thesis with Nokia.

Additionally, my deepest respect goes to my colleague Muazam Ali and team lead Arto Palin for their valuable contributions and advice. I also want to thank all the respondents who participated voluntarily in my research study and gave their time and sincere opinions.

Furthermore, a special acknowledgment to my siblings for their encouraging support and prayers during the stressful time. Finally, I will always be indebted to my mother Jannat Bi for her prayers and pushing me to pursue my masters.

Tampere, 30 October 2019

Sana Latif

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

1G	First Generation
2G	Second Generation
3G	Third Generation
4G	Fourth Generation
ADC	Analog to Digital Converter
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
AXI	Advanced Extensible Interface
BTS	Base Transceiver Station
CDMA	Code Division Multiple Access
CI	Continuous Integration
CLB	Configurable Logic Block
DAC	Digital to Analog Converter
DFE	Digital Front End
DL	Downlink
DRC	Design Rule Violations
DSP	Digital Signal Processing
DSPs	Digital Signal Processors
DUT	Design Under Test
eNB	Evolved Node B
ERC	Electrical Rule Violations
EVM	Error Vector Magnitude
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDSII	Graphic Database System for Information Interchange
GSM	Global System for Mobile
HDL	Hardware Descriptive Language
HW	Hardware
IC	Integrated Circuit
IP	Intellectual Property
IQ	In-phase Quadrature
ISI	Inter-symbol Interference
IPXACT	An XML based standard
LTE	Long term Evolution
LSB	Least Significant Bit
LVS	Layout vs Schematic
M2M	Machine to Machine
MAC	Multiple Access Control
MSB	Most Significant Bit
OFDM	Orthogonal Frequency Division Multiplexing
OVM	Open Verification Methodology
PARP	Peak-to-Average Power Ratio
PDCP	Packet Data Control Protocol
PHY	Physical
PLL	Phase Locked Loop
RAN	Radio Access Network
RAM	Random Access Memory
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
RTL	Register Transfer Level
SC-FDMA	Single Carrier Frequency Division Multiple Access

SoC	System on a Chip
SQNR	Signal to Quantization Noise Ratio
TDD	Time Division Duplex
UE	User Equipment
UL	Uplink
UVM	Universal Verification Methodology
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1. INTRODUCTION

Communication is the process that starts from anything that needs an exchange between two entities. We use communication in our daily life from sharing ideas to communicating with people around the globe. Hence, constant advancement has become a necessity.

Communication systems have evolved and become a part of our existence with the advancement in technology over the years. Examples of modern-day communication systems include Internet, mobile networks, audio and TV broadcast systems etc.

Mobile networks play a huge role in personal as well as work life due to increasing need in effective and timely communication. Over the years, mobile networks have become advanced through number of generations with added features in each network generation. A mobile network comprises of a device and a network of base stations communicating over a wireless radio network. In Radio Access Network (RAN), Physical Layer (PHY) communication plays an important role to understand cellular mobile networks.

Digital systems are the backbone of wireless mobile networks. Mobile networks communication is mainly based on Digital Signal Processing (DSP) algorithms. Digital systems are meant to deal with DSP applications that set strong requirements such as real-time computations, power, frequency and area. Digital systems are implemented on high performance devices, such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) to meet the requirements of communication systems. In modern era, digital systems have become extremely complex which brings the need of SoC. To establish mobile RAN architecture, SoCs are used for their deployment.

It is extremely important to streamline and automate the SoC development processes because of the complex nature of SoC devices, to reduce the cost, resources and time to market. SoC industry requires to implement development models to fulfil these needs. DevOps extends the agile methodologies to swiftly deliver products in an automated and integrated process.

CI is an agile practice that enhance benefits of development models. Although, CI was developed initially for software development, but it helps in faster release and feedback mechanism as well. To implement this software development model in SoC development, it has its own benefits and complications. However, the result is a well-integrated and cohesive product. CI implementation in hardware development automate overall SoC processes and seamless working between different teams. For CI implementation

in SoC development, a literature research is carried out to identify the potential problems and solutions.

A part of the research work is to automate and streamline work between different SoC development teams. A Simulink model is designed so that it can be used in different stages of SoC design. CI system is also implemented by using Jenkins which is a CI tool for automating the build and testing of the model design.

Another research study is conducted with different SoC development experts to identify the gaps in SoC development workflows, steps to avoid duplication of work and idle time. Input from SoC experts is also considered on usability and benefits of CI in their daily workflow.

1.1 Thesis Objective

The main objective of this thesis research work is to study and evaluate currently used SoC development processes and flows. In addition, it presents guidelines on how CI can be implemented to streamline SoC workflows and avoid duplication of work. Moreover, a model is designed that can be used in different stages of SoC development with adequate complexity and is easy enough to be used for a person not mastering the topic. CI system tool is also implemented to automate and test the model design so that it remains up to date.

1.2 Thesis Organization

This thesis document is organized as follows:

Chapter 2 explains the basics of communication systems, mobile networks and its evolution. In addition, it gives an overview of RAN, PHY layer architecture and future of mobile networks.

Chapter 3 defines the basic concepts of signals, DSP, digital systems, IP (Intellectual Property) blocks, ICs and gives an overview of SoC system. It further discusses different SoC development flows, tools and processes that are being used in the industry as well as academia.

Chapter 4 discusses the different formats of data representations, data storage and data processing.

Chapter 5 presents basic concepts of CI, its benefits and tools in SoC development.

Chapter 6 discusses about the practical work, a Simulink model implementation using Jenkins CI tool.

Chapter 7 discusses conclusions and future work.

2. COMMUNICATION SYSTEMS

Communication system can be defined as an exchange of information from source to destination over a medium. The information to be sent can be analog or digital, however the medium to transmit and receive information is always analog and continuous in nature for example cables and radio waves etc [3]. In Digital Communication, analog signals are first converted into digital signals and then transmitted over a channel depending on the modulation technique. Once again at the receiver end, the signal is converted back from digital to analog. Figure 1 presents a basic communication system block diagram.

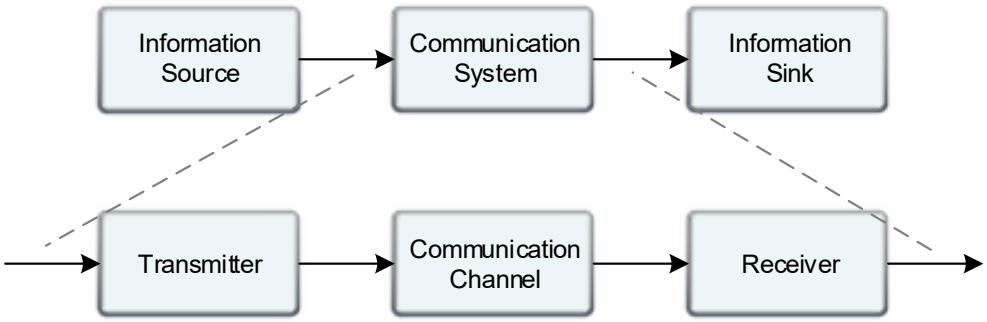

Figure 1. *Block diagram of a basic communication system*

In the last four decades, the key revolutionary technologies have been digital communication systems such as mobile cellular network, satellite communication and wireless sensor networks etc. [1].

2.1 Mobile Networks

A mobile network is a digital communication system where a targeted data communication between at least one fixed transceiver and a mobile wireless transceiver over a land area called cell is established. Each cell is served by at least one Base Transceiver Stations (BTS) that provides network coverage to mobile User Equipment (UE). There are many benefits of cellular architecture such as frequency reuse by allocating different frequencies to neighbouring cells, less power consumption and large coverage area than single high power BTS [4]. Figure 2 presents a simple mobile network.

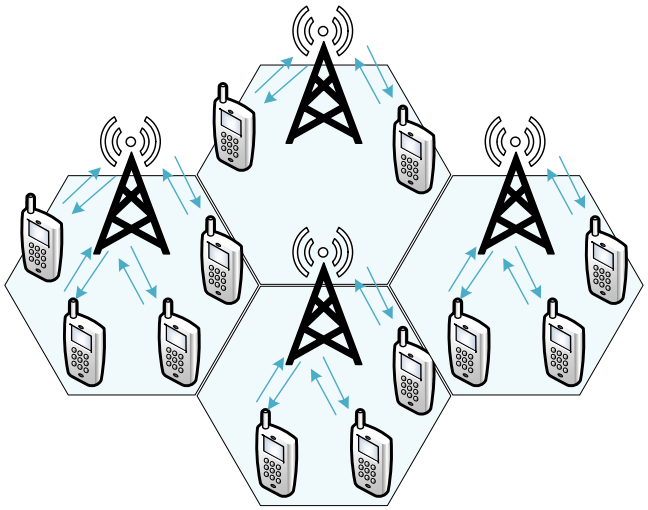

Figure 2. *Mobile network*

2.2 Evolution of Mobile Networks

Mobile networks have evolved through a series of generations, each with improved technology and features from the previous generation. The First Generation (1G) mobile network was deployed in 1980's and it used analog transmission with one call per radio channel. In 1G, voice during a call was just modulated to higher frequencies by using frequency modulation [2]. Few of the breakdowns of 1G were poor & noisy voice quality, less secure and unencrypted transmission.

Second Generation (2G) mobile network was launched in 1990's, one of the major improvements was introduction of digital transmission and Global System for Mobile (GSM) standard. In 2G, digital voice calls, high data rates, SMS and email services were offered due to Code-Division Multiple Access (CDMA) implementation [14].

The evolution of Third Generation (3G) mobile network marks the start of smart phone era. In early 2000's, 3G unities multiple technologies to deliver higher data rates on higher frequency bands for transmission. Some of the new features included video call, higher number of users per cell, increased security, location tracking and maps [3].

Fourth Generation (4G) or Long Term Evolution (LTE) mobile network, offered fundamental changes in mobile communication such as higher data rates with improved spectral efficiency. LTE mobile network offered advanced multimedia services and compatibility with previous generations that helped in easier network upgrade and deployment. New PHY layer wireless technologies were introduced to transmit voice and data at the same time [2].Orthogonal Frequency Division Multiplexing (OFDM) was first used to avoid Inter-Symbol Interference (ISI) due to higher bandwidths. Multiple Input Multiple Output (MIMO) technology is one of the differentiating aspects of LTE that helped to offer

higher data rates [4]. Figure 3 presents evolution of mobile networks through the years from 1G to LTE [21].

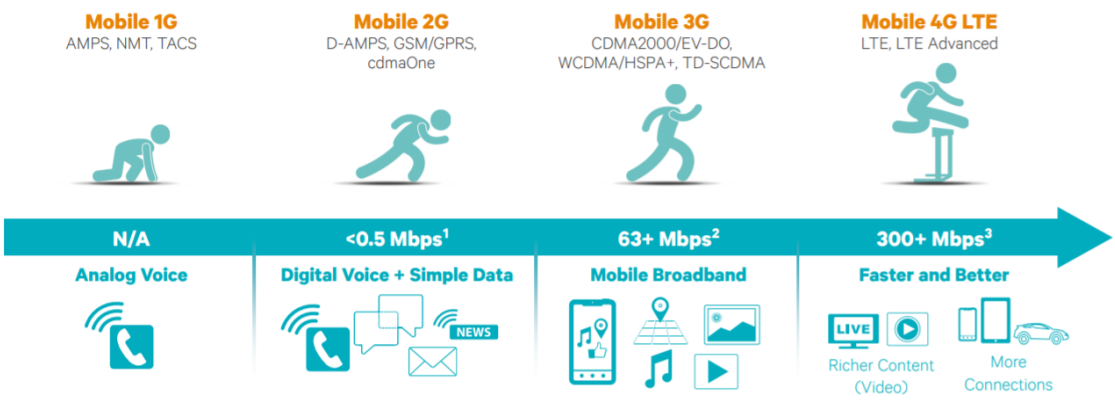

Figure 3. *Evolution of mobile networks*

2.3 Radio Access Network

A RAN helps to connect UE to its core network through radio connections, Figure 4 presents basic RAN. It is a major part of mobile network that resides between UE, BTS and antennas. Silicon chips such as SoC provides RAN functionality in both core network and UE. RAN architecture consists of PHY, Radio Link Control (RLC), Medium Access Control (MAC), Packet Data Control Protocol (PDCP) layers and Radio Resource Control (RRC) protocol. In 4G, some of the main functionalities of RAN includes, Radio Resource Management (RRM), scheduling, compression/decompression of the DL/UL user plane packet headers and HARQ error correction etc [20].

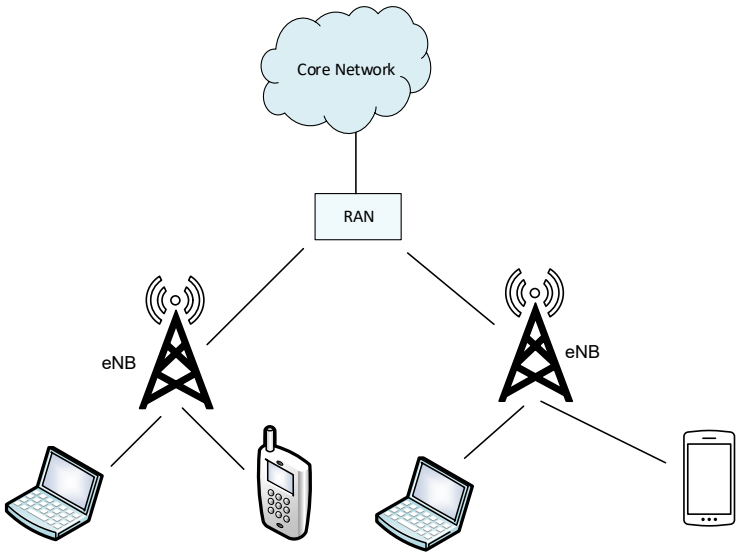

Figure 4. *Radio access network*

2.4 Physical Layer

In RAN, PHY layer functionality is the main part of the architecture. It provides data and control information transfer between BTS/Evolved Node B (eNB) and UE. Figure 5 presents general block diagram of PHY layer architecture in downlink [2].

The PHY layer is responsible for coding techniques, modulation, multiantenna processing and implements advance technologies like MIMO and OFDM. It also handles mapping of MAC transport channels to PHY channels. PHY layer implements multiple technologies to offer spectral efficiency and high data rates. OFDMA allows high throughput in the downlink and Single Carrier-FDMA (SC-FDMA) in the uplink reduces Peak-to-Average Power Ratio (PARP). Adaptive modulation and coding maximize data rates at individual and cell level. Few of the PHY layer features are dynamic bandwidth allocation to users and high cell-edge performance. PHY layer considers path losses and environmental interferences while designing control channels and reference signals [4].

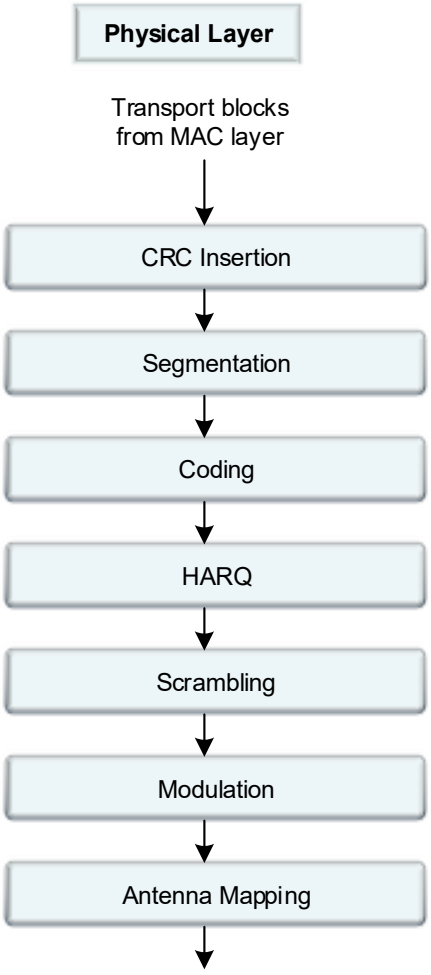

Figure 5. *Downlink physical layer architecture*

2.5 Fifth Generation: Future of Mobile Networks

Mobile technology's global trends are changing rapidly, the need of high data rates, high speed and low latency have become crucial. It is predicted that in 5th Generation (5G) mobile technology, features such as controlling home appliances from any part of the world to extreme music & video streaming will become part of our daily lives. Some of the potential global trends in future mobile network development are [14]:

- Autonomous driving
- Machine to machine communication (M2M)
- Remote robotic surgery
- Device remote controlling
- Virtual Reality
- Bandwidth throughput
- Smart cities
- Healthcare data management
- Real time gaming
- Bi-directional remote controlling
- Wireless cloud-based office
- Video streaming etc.

In order to enable these services new technologies are being designed for future mobile networks and some of the stand-out technology features are [21]:

- Scalability: Scalable numerology with dynamic TDD.
- Low latency: Mini time slots
- Forward Compatibility: Bandwidth parts with no fixed time relationship between channels
- Beamforming: Initial access / Beam sweeping with massive MIMO
- Network slicing

There are many design challenges despite the positive outlook and motivation for future technologies. In 5G at millimeter wavelength (>6GHz), large number of antennas will be required to provide coverage which in return is going to increase the deployment cost. In addition, energy efficiency and security will be high risk areas. As a result of network densification, challenges will arise in interference, energy and mobility management [20].

3. DIGITAL SYSTEMS

This chapter defines the basic concepts of signals, DSP, digital systems, IP blocks, ICs and gives an overview of SoC. This chapter also discusses different SoC development flows, tools and processes that are being used in the industry as well as academia.

3.1 Overview

A signal is anything that carries information, for example, sound, smoke or heat etc. Signals are of different types, but we generally divide them in two categories, a continuous-time signal and a discrete-time signal. A continuous-time signal can be defined and represented at any instance of time whereas a discrete-time signal is only represented at discrete intervals of time [3]. A discrete signal can be converted into a pattern of bits (1s and 0s) which is called as a digital signal. In computer systems, signals in digital format are easily represented because each sample can be represented as a sequence of bits.

Processing of digital signals defines the digital signal processing and the systems processing them are called as digital systems. In the modern era, digital systems have been constantly replacing the analog systems. Nowadays, digital systems are most important in the organizations. Digital systems are designed to store, process and communicate digital information using binary system. Digital systems can vary from a CD player to as complex as mobile networks. Digital systems bring major advantages over old analog systems such as [5]:

- Easier to design
- Less effected by noise
- More precise and accurate
- Easily programmable
- Cost effective

Digital systems are meant to deal with DSP applications that set strong requirements such as real-time computations, power, frequency and area [6]. Digital systems are implemented on ASICs and FPGAs.

The following section presents the digital system design flow in general.

3.2 Digital System Design Flow

Modern day designs of digital systems demand different design methodologies and level of abstractions in order to meet the strong requirements set by their applications. In the section below, general steps of digital system design are given [7]:

Specifications: The first step in the digital design is usually to define the system specifications. Specifications can either be defined by the customer of that product or an authoritative body that defines functional requirements for a reliable and efficient design such as power, area and performance etc.

Architecture: An architect designs and specifies how the system will be implemented by defining memories, timing budget, software/hardware, power management etc in order to achieve design goals set by specifications. Sub-blocks or IPs specifications are also derived from the top-level system architecture design.

Algorithm Model: An algorithm model is implemented mostly in higher level languages to observe the system's behaviours at the inputs and at the outputs according to the specifications. The model does not define steps on how to implement a design on physical level but provide ideal behaviour of the prospective system. Reference models for digital hardware design verification also fall under this category [24].

Hardware Description Model: Hardware Description model of the system is usually developed in Hardware Descriptive Language (HDL). Entire system is divided into the subsystems and then subsystems are further divided into the IP blocks. Two most commonly used HDL languages in the industry and in academia are VHDL that stands for Very High-Speed Integrated Circuit (VHSIC) HDL and Verilog.

Verification: HDL model verification is divided into certain subcategories such as [8]:

- **Preliminary Verification:** It is important to do some preliminary or sanity verification in parallel with Register Transfer Level (RTL) design and it is mostly done by the HDL model designer. In simple words, design is checked to confirm if it compiles correctly and did not break any of the existing features.
- **Functional Verification:** During the design stage, verification testbench environment is set by the verification engineers using hardware verification languages. A testbench is simply a piece of code which is meant to verify the functional correctness of the HDL model. VHDL, Verilog and SystemVerilog are the commonly used languages to write verification testbenches in the industry and academia.

Logic Synthesis: It is the process of translating RTL HDL model into an optimized gate level representation which is named as a netlist. The verified RTL code is synthesized into logic gates by the synthesis tools [6].

Logical Equivalence Testing: Once we have gate level netlist and verified RTL code, instead of verifying netlist both are compared on the Boolean level. This equivalence check result should be same, after that tested netlist can be used in next level of digital design.

Static Timing Analysis: It's the last step in frontend design part of the digital system. This analysis is done to verify the performance, frequency and timing of the design and it is then sent to the backend design team.

Floor Planning: It's the first step in backend design, to decide where and how the synthesized design is to be placed on the chip die. At this stage, power planning is also done on how to route power within the design.

Clock-Tree Synthesis: In this step, it is made sure that every flip-flop receives clock signal at the same time in the whole design.

Routing: This process determines the path through which cells are to be connected and what path interconnection takes.

Physical Verification: The final routed design goes through multiple checks like Design Rule Violations (DRC), Electrical Rule Violation (ERC), Layout vs Schematic (LVS) etc. in order to confirm all the requirements are met [8].

Post Layout Timing Analysis: As a last step in backend design, after all the checks design is verified one more time to confirm timing correctness.

GDSII generation & Fabrication: After the back-end design, the Graphic Database System for Information Interchange (GDSII) stream created from the layout and sent to the fabrication company for chip fabrication.

Post Fabrication Testing: After fabrication, all the chips for any manufacturing errors are tested against the required specification and expected results before sending them to the market or customers. Figure 6 presents the digital design flow.

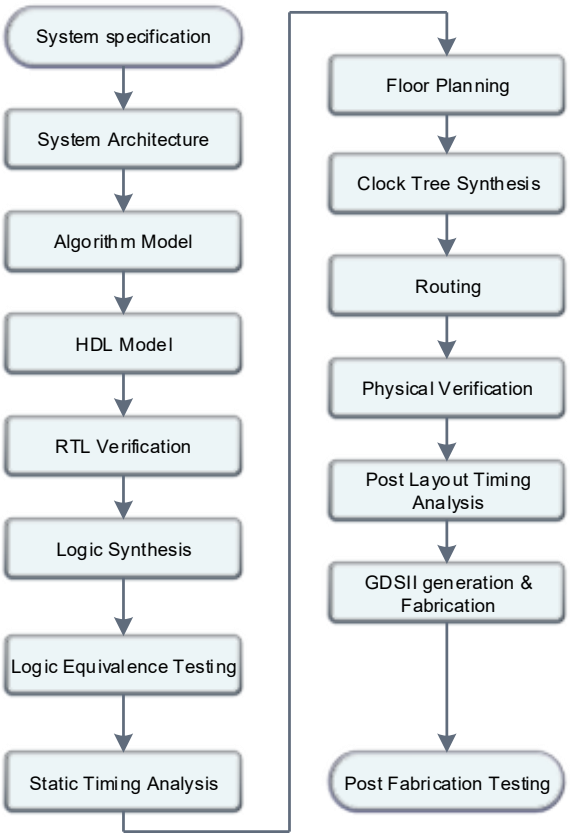

Figure 6. *Digital system design flow*

3.3 Integrated Circuits

ICs are electronic components that unite a group of electronic devices like millions of transistors, resistors and capacitors etc on a small flat area made of semi conducting material mostly silicon [5]. These small chips can perform calculations and processing of stored data in both analog or digital form. In modern days, ICs are evolving and their functionalities vary from a simple amplifier to processor or even a whole computer memory. Moore's law states that the "number of transistors per chip gets doubled after each one and a half year and at the same time decreasing in size from micro to nano-metres" [7].

3.4 IP blocks

In Digital system design, IPs are building blocks that make up the whole digital system. However due to increase in complexity and advancement of technology in SoC, IPs are reused extensively to save time and man power as well as to reduce time-to-market.

Even if the two SoCs are different in functionality and design, there are always multiple blocks that are present in every SoC for example AXIs etc., so constant redesign is not

necessary instead reuse is the way to go. Different design tools act as connectivity tissues in the chip and play a role in filling the gaps between new and old IPs. There are two different types of IP cores; hard and soft. Hard core IPs usually have certain defined functionalities and cannot be modified by designers. However, soft core IPs are adjustable according to the certain design requirements [10]. IP reuse also brings certain disadvantage such as challenging to debug and optimize the design. In addition to it, bad documentation can also lead to more time in understanding the design.

To facilitate automation and reuse of IPs in SoC design, IPXACT standard format is used between most of the IP vendors and SoC development companies. IPXACT is an open standard XML-based format.

3.5 SoCs

A SoC systems can be classified into two main categories as ASICs and FPGAs. It is an integrated circuit that integrates custom logic on a single chip such as, microprocessors, coprocessors, digital signal processors (DSPs), hardware accelerators, on-chip memory, peripherals, external interfaces, analog and custom circuitry and all types of other system and processing components.

Figure 7 demonstrates some of the basic elements of a SoC system, that includes assortment of processors connected to memory elements. The SoC can also have analog circuitry for handling sensor data, analog to digital convertor, or to support wireless data transmission or reception etc [7].

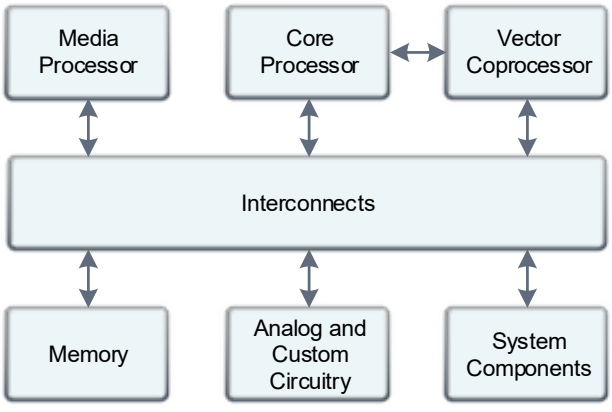

Figure 7. *A basic SoC model*

The distinguishing factor between SoC and a general-purpose IC is the specific requirement for the target design. The requirements are assumed to be known beforehand so that the foundation of the system can be laid accordingly. This emphasis on selection

and configuration of system components for tailored application distinguishes system architecture from computer architecture.

3.5.1 ASIC

As the name suggests, ASICs are customized ICs designed for a specific application. An ASIC can be categorized as a SoC if it contains one or multiple processors. ASICs are designed for a specific purpose and their functionality remains the same for their entire lifetime and their functional logic cannot be modified. ASICs are very expensive, time and resource consuming ICs to develop but they offer extremely high performance and low power consumption [22].

3.5.2 FPGA

ASICs offer extremely high performance and lower power consumption. However, we cannot modify any of its algorithmic digital functional logic because these algorithms are frozen in the silicon chip. This brings the need for FPGAs, as the name suggests FPGA ICs are field programmable to operate according to the intended applications designs. FPGA devices are made up of hundreds of thousands of Configurable Logic Blocks (CLBs) connected through hundreds of thousands of programmable interconnects. Flip-Flops, Multiplexers and Look-up Tables are the primary building blocks of CLBs. In addition to the CLBs, FGPA devices also contain PLLs, block RAMs, DSPs and external memory controllers [6]. Table 1 provides the comparison between FPGAs and ASICs.

Table 1 . Comparison between FPGAs and ASICs

Comparison between FPGA and ASIC	
FPGA	**ASIC**
Reconfigurable logic	Fixed logic
Faster time to market	Takes longer time
More power consumption	Power efficient
Not suited for high volume production as large FPGAs are expensive and not justifiable for high volume production	Suited for high volume production due to less cost/unit but expensive to develop in comparison to FPGAs
Limited in operating frequency due to routing and configurable logic	Higher operating frequency since its optimized for specific function
Analog designs are not possible	Analog circuitry is possible
Well suited for applications where the current design might need to be upgraded so the high cost is not the deciding factor but the reprogrammability of FPGAs	Not suitable for applications where the design needs to be upgraded
Designers do not need to worry about back end design	Designers need to care for every step from RTL down to testing constraints

After the overview of digital systems, the following section discusses the algorithm modeling in the SoC development.

3.6 SoC Algorithm Modeling

A model is conceptualization and simplification of the reality. Modeling can be said as a proactive approach to highlight aspects of a system and to help in order to solve the problems [7].

Complexity in SoC design is increasing at a rapid pace and faster time to market is also strongly required. To manage these complexities, the earliest steps that are taken to overcome these problems are to analyse and model the high-level abstract design before implementation.

In general, there are certain goals for system modeling, such as [24]:

- Performance evaluation of the design
- Resource estimation for the design
- Power consumption
- Design-space estimation
- Verification of functional correctness
- Testing under multiple real-life scenarios

This modeling flow and process enables engineers to evaluate design trade-offs, test design under multiple real-life scenarios. This approach helps to save huge amount of time at the stage of verifying the design.

At the time of designing a system model, irrelevant details are not considered but the abstract of the design. One of the approaches that leads to an efficient design, is to divide and conquer while forming hierarchies.

Some other qualities in addition to the ones mentioned earlier of a good model are [6]:

- Easy to understand
- Accurate representation of the specifications
- Affordable so that it is easier to develop to study actual design
- Answers all the analytical behaviour of the actual system

3.6.1 Modeling Flow

Figure 8 presents the workflow of a model-based design [25].

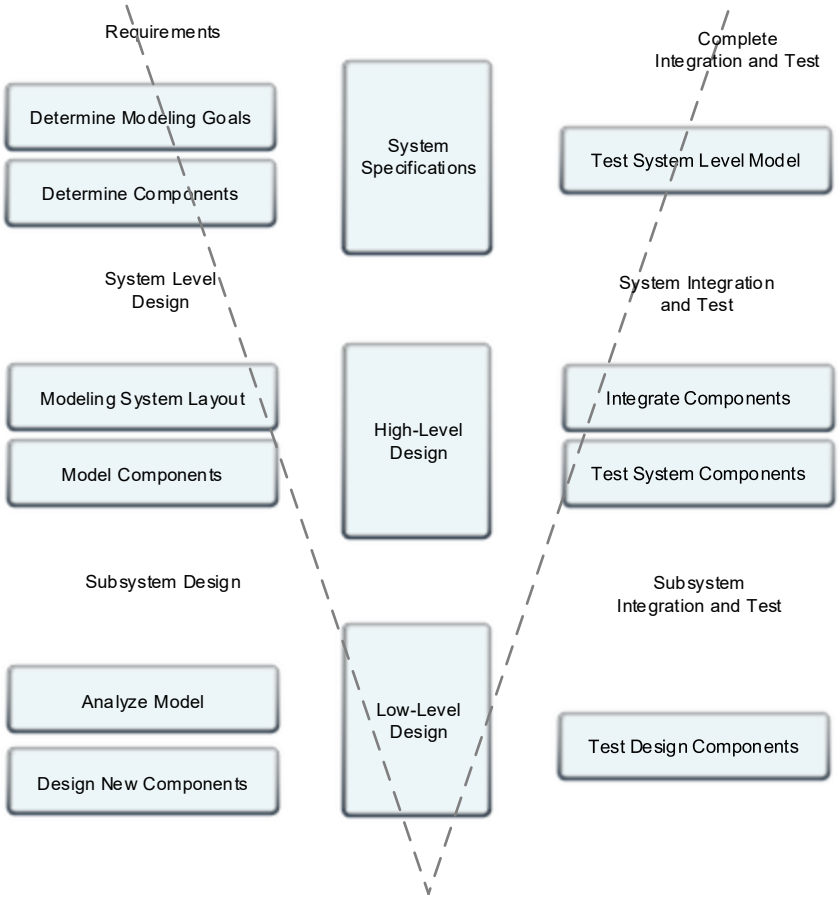

Figure 8. *Model based design workflow*

Below are the few methods a model can be represented, such as [24]:

- **Graphical Design:** This method uses block diagrams and graphs in designing a model. Different design tools are used, such as MATLAB and Simulink allow engineers to comprehend by swiftly prototyping their algorithms into models.
- **Textual Design:** This method uses mathematical expressions and programming languages such as C, C++, System C, Python and MATLAB tool to design a system model in the industry as well as academia. Multiple tools are used in designing such system models that support and compile one or all the above-mentioned programming as well as system modeling languages. In this method even a simple text editor can work to develop a system model.
- **Hybrid Design:** This method uses both graphical and hybrid methods together to design a system model.

3.7 SoC Hardware Design Process

The HDL model design process encompasses specifications from the start to a plan that consists of all the information required for physical construction at the end. The HDL model design process begins with designing the SoC system architecture that is usually designed by a SoC architect. System architecture defines the overall hierarchy of the design, such as system architecture, subsystems architecture, IP blocks and interconnect blocks. Breaking down functionality of the entire SoC system into smaller subsystems and IP blocks makes the SoC development process easier. It brings many benefits, such as project management, algorithm modeling, HDL model development and verification etc [22].

Digital logic design, computer architecture and HDL model design languages along with concepts of mathematics help the HDL model design engineers in development of digital system design or a SoC system development. Some important logical functional blocks that are written in HDL languages, help in development of complex HDL model design. Combinational logic components, sequential logic elements, registers, Finite State Machines (FSMs), Arithmetic Logic Units (ALUs), decoders and multiplexers etc. are some of the examples of logical functional blocks or elements.

In the SoC development, many analog logic blocks or components are also designed that help us to interact with real world, such as Digital to Analog Converters (DAC), Analog to Digital Converters (ADC) and clock generating units such as Phase Locked Loops (PLLs) [5].

3.7.1 Design Flow

HDL model of an IP block or any logical functional block or component goes through the different design flow steps until the block is ready for integration at subsystem or system level. Different design and verification methodologies are used to the block that is ready for integration at a higher level as depicted in the Figure 9 [23].

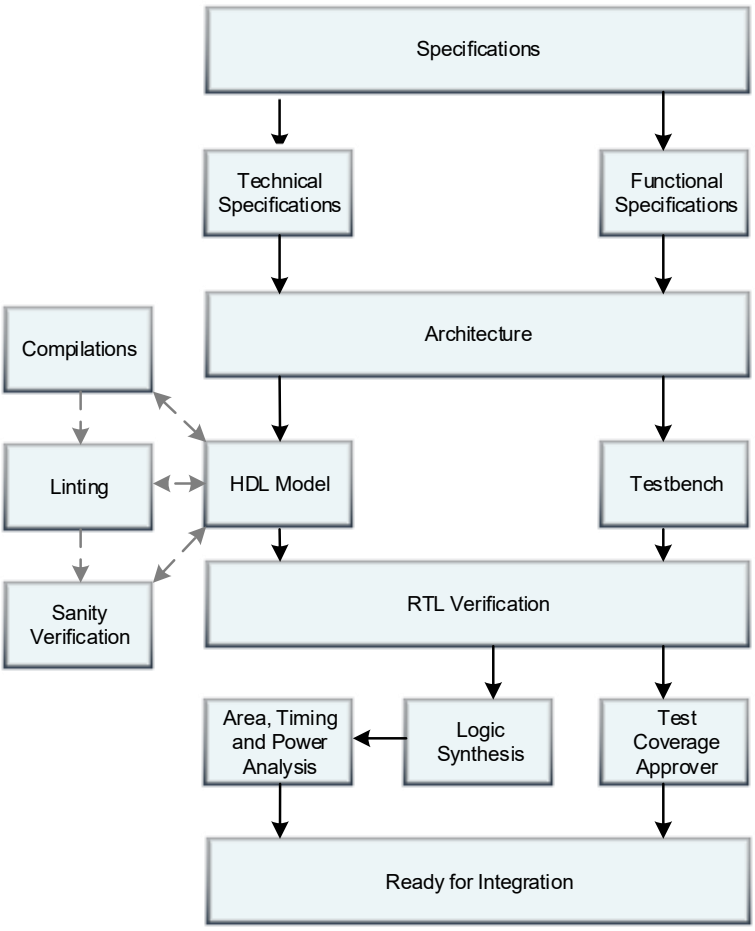

Figure 9. *HDL model design flow*

3.7.2 Design Tools and Languages

In HDL model design process of the SoC development, different HDL model design tool and HDL languages are used in the industry as well as in academia. It involves different HDL languages editors, compilers, linting, testbench development, synthesis and power analysis tools. HDL model design engineers use different languages, such as, VHDL, Verilog, SystemVerilog and System C etc for HDL modeling of the digital hardware [6].

3.7.3 Design Reusability

Modern day SoCs are extremely complex, it makes the development of a SoC system extremely difficult to start from scratch. This is where the need to reuse the existing IPs arises. There are different vendors that provide the pre-verified reusable IPs in IPXACT format which is an open XML-based standard [6]. Section 3.4 discusses more about IP reuse.

3.8 SoC Hardware Verification

This section discusses the digital hardware or HDL model verification in general as well as from SoC viewpoint. Verification is the most important part in the process of any system development, and it cannot be omitted from the process. A system can only be claimed to behave functionally correct after the proper verification of all the system features is completed and the system has passed all the verification tests [10].

In a SoC system development, approximately 60-80% system design efforts are dedicated to the system verification. Therefore, reusability of the pre-verified IP blocks from inhouse and other IP vendors helps in saving cost, time, efforts and other resources [8].

3.8.1 Verification Flow

The digital hardware verification process in a SoC development or in any other digital hardware system design starts at the same time with HDL model design.

Figure 10 present the generic digital hardware verification process [8][10].

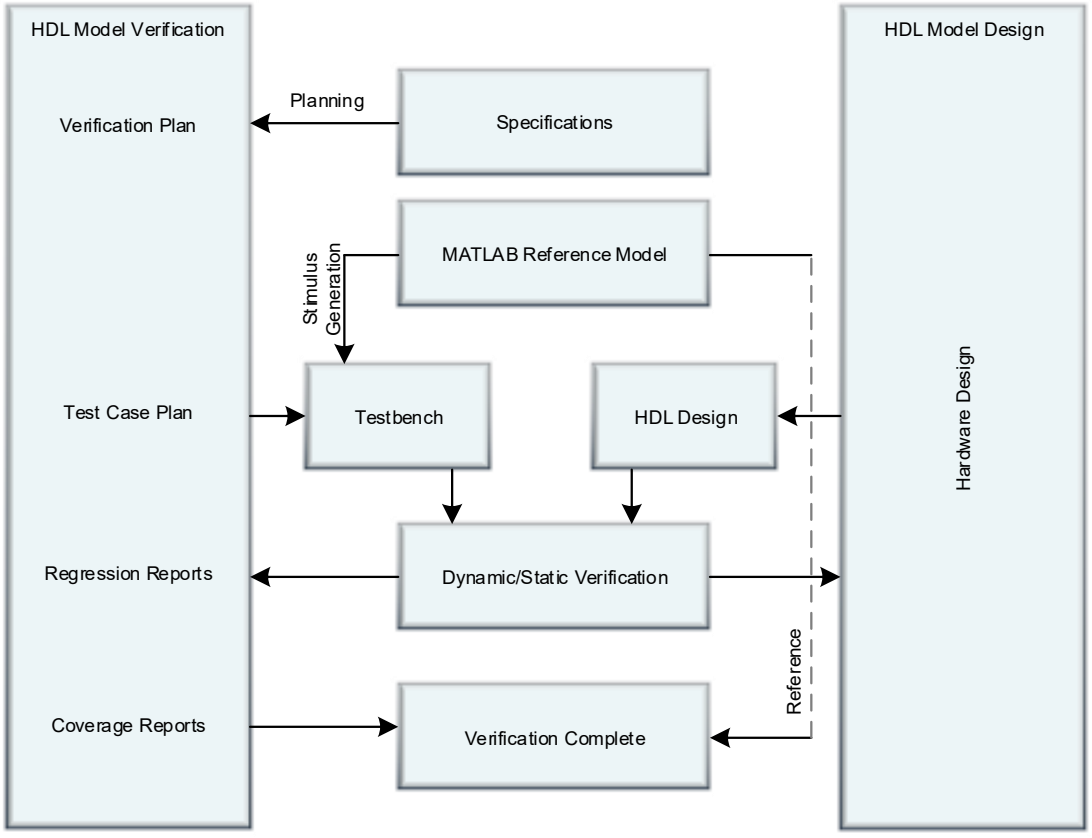

Figure 10. *HDL model verification flow*

It is extremely important to properly verify the system so that it meets the performance, power, security and safety requirements of the design. The verification process is an extremely critical part of the system design cycle, as any bug in the design not found and debugged before tape-out can make the whole chip useless or cause millions of dollars in lose.

Digital hardware system design and verification projects are developed in parallel. Specifications decide the features to be designed and verified in the system. Reference models are developed to verify the functional behaviour of the system. Testbenches are developed to test the actual design against the reference model, where a testbench is a simple piece of code that helps to verify the functional correctness of the design. Reference models are meant for providing a set of reference input/output vectors to the testbench so that it can verify the behaviour of the actual design. Verification completes once the system output behaviour is matched against the reference model data [7].

3.8.2 Verification Methodologies

There are many different digital hardware verification methodologies being used in the FPGAs and ASIC SoCs. The most common method in the academia is writing HDL testbenches to verify the digital systems but sometimes advance verification methodologies are also used.

However, in industries advance methods of verification in addition to HDL testbenches are being followed worldwide such as Universal Verification Methodology (UVM) and Open Verification Methodology (OVM) etc which depends on the need and requirements of a system [8].

There are multiple approaches that are used in the testbench development as well as verification processes and flows such as top-down, bottom-up, platform and system interface based etc.

Top-down and bottom-up are the most common methods to verify a system. In the top-down approach, system level verification is done before the block or IP level verification. However, bottom-up approach is the most common one which does the block level verification before the system level verification [10].

3.8.3 Verification Tests

There are different verification tests that are performed in verification process, such as [8]:

- Random testing to verify the design against random input data
- Functional testing to verify the functional correctness of the specified features
- Regression testing to make sure exiting functionality does not break after each bug fix

3.8.4 Verification Languages

There are many languages and tools that are being used in the industry and academia to develop the testbenches for verification of a digital system. VHDL, Verilog and SystemVerilog etc. are the most common ones among all [7].

4. DATA REPRESENTATION

This chapter discusses the different formats of data representation, storage and processing. In computer systems and SoC system development we always deal with numeric data which makes it very important to understand how this numeric data is represented. The following section discusses the binary representation of data.

4.1 Binary Representation

In computer systems, binary representation of data is very important because digital electronic systems and devices works on the concept of two known states, on and off state. Numeric data is represented using only two levels such as true or false, on or off and high or low. This representation of data in 0s and 1s is called the binary representation [5].

To convert a decimal number to a binary number, we find all the powers in 2 and then add them to calculate the required binary number. Each 0 and 1 digit in the binary number representation defines a bit and a group of 8 bits is termed as a byte. For example, a decimal number 89 is represented in the binary format as follows:

$89_{10} = (1011001)_2$

$89_{10} = 1\times64 + 0\times32 + 1\times16 + 1\times8 + 0\times4 + 0\times2 + 1\times1$

$89_{10} = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$89_{10} = (1011001)_2$

4.1.1 2's Complement

Two's complement is a convention that represents signed binary integers. To represent a negative number through this convention, we first invert all the bits from 0 to 1 and 1 to 0 then add 1 to the reversed number as follows:

- To represent -89, first we will write binary form of $89_2 = (1011001)_2$
- Reverse all the bits to 0100110
- Now add 1 to the reversed bits 0100110 + 1 = 0100111

The representation of -89 in 2's complement is $(0100111)_2$

Below sections presents Fixed-point and Floating-point binary representations. They are the most important representations of data in the binary format among others in the field of digital system design [3].

4.2 Fixed-Point Representation

Fixed-point data representation is characterized by the fixed position of radix point, where the Least Significant Bit (LSB) or the bits before the radix point represent integer part and Most Significant Bit (MSB) or the bits after the radix point represents fractional part of the real number. Decision about selection of bits always depends on the requirements, whether MSB and LSB bits represent fraction and integer part of the number respectively or vice versa. For signed numbers, one bit is reserved for sign representation in data [7].

Figure 11 presents the generalized fixed-point representation.

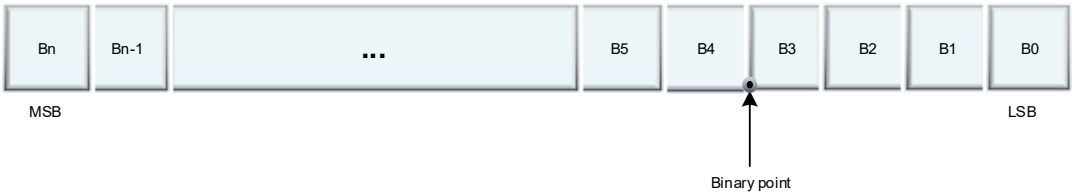

Figure 11. *Fixed-point number representation*

In the fixed-point representation, the radix point position is predefined based on system requirements, more precision is achieved with more bits after the radix point. It is important to pay attention that in the memory radix point is not kept but it is a way to interpret stored bits of data [13].

4.2.1 Integer Representation

In the case of integer data representation, we consider the radix point at the end. To represent numeric range of *n* numbers of bits, we have [30]:

$$0 \leq x \leq 2^n - 1$$

The above representation can only show numbers in the positive range, hence called unsigned binary representation. If we take 4 bits, the represented range of numbers is 0-15.

Signed binary representation is defined to represent negative numbers which is:

$$-2^{n-1} \leq x \leq 2^{n-1} - 1$$

With the above formula the represented number range stays the same, but the highest positive number that can be represented is reduced to half. First bit is used to signify sign of the number so for 4 bits, now the numbers that can be represented are from -8 to 7 [30].

4.2.2 Fractional Representation

For fractional data representation place of radix point is interchangeable according to the number of fractional bits assigned. In signed fractional representation, one bit is reserved for sign and rest of the bits are reserved to represent fractional and integer part of the number. Figure 12 represents fixed point signed and unsigned fractional representation [31].

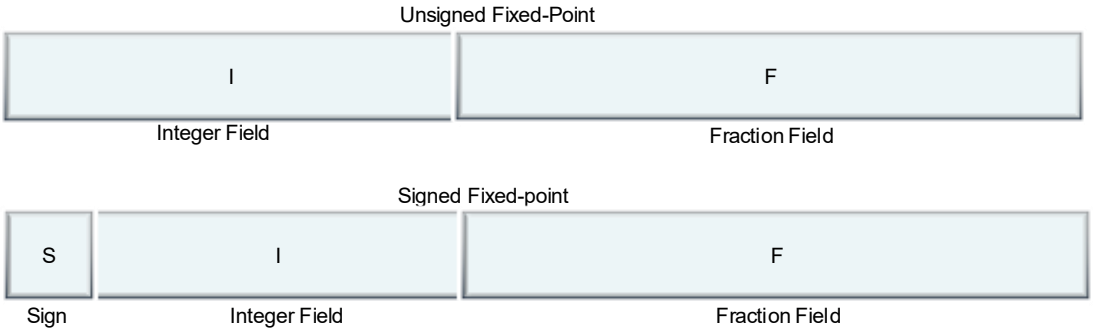

Figure 12. *Fixed-point signed and unsigned number representation*

4.3 Floating-Point Representation

In floating-point data representation, as the name suggests radix point can float or placed anywhere comparative to a number's significant digit. The floating-point data representation is very important in DSP computations where numbers are represented with large dynamic range. To represent numbers with large dynamic range, more bits are required to represent the smallest and largest number in the range. It results in decreased processing speed, more surplus memory and high cost [13].

Floating-point arithmetic with scientific notation representation was introduced to overcome this issue. This representation with its exponent increases the dynamic range representation with a smaller number of bits. Scientific notation of a number consists of a mantissa, a radix and an exponent, it is expressed as follows:

Mantissa x Radix $^{\text{exponent}}$

In binary floating-point number system, the represented number has a signed binary mantissa and an exponent with a radix of 2. As per above representation, fractional part is the mantissa that represents accuracy in the computations. However, exponent represents the dynamic range of a number. For the representation of larger dynamic ranges, number of bits increases to represent the exponent. For higher accuracies in the computations, number of bits increases to represent the mantissa [30].

4.3.1 IEEE-754 Standard

To represent floating-point data, different standards have been introduced but IEEE-754 standard is the most widely used with a few exceptions. This data representation is divided into three fields to represent a floating-point number; one-bit sign, exponent bits and fraction bits as presented in Figure 13. The sign of the fraction field indicates the sign of the whole number whereas the sign of the exponent indicates the magnitude of the entire number [31].

Figure 13. *Basic floating-point number representation*

IEEE-754 standard has different floating-point formats of which ’32-bit single precision’ and ’64-bit double precision’ are more commonly used. Table 2 represents different formats with exponent ‘E’ and fractional bits ‘F’ division [31].

Table 2 IEEE-754 Floating-Point Standard

Common Name	No. of Bits	Sign bit	Expo-nent	Fraction	Max. Expo-nent	Min. Exponent
Half Precision	16	1	5	10	15	-14
Single Precision	32	1	8	23	127	-126
Double Precision	64	1	11	52	1023	-1022
Quadruple Precision	128	1	15	112	16383	-16382

4.4 Quantization Effects

Quantization is the process of converting analog signal into digital domain, due to the nature of analog signal having infinite values. The received digital signal have to be quantized to a certain set of values. The quantization of an analog signal from infinite samples to a finite number of samples results in errors and affects on signal quality. These errors and degradation of signal quality is termed as quantization errors. There are two main groups of quantization techniques, truncation and rounding [13]. Figure 14 presents the different quantization techniques [29].

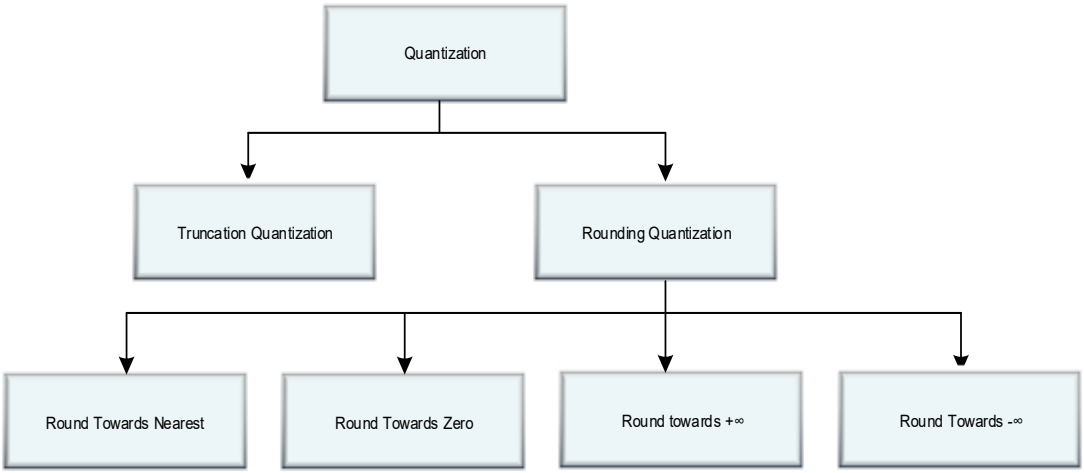

Figure 14. *Quantization techniques*

Truncation is defined as a process where bits exceeding from the allowed number range regardless of the number sign are dropped-off from the LSB side. However, in the rounding-off technique, extra bits are dropped-off while rounding the number to the next nearest number in allowed number of bits. Table 3 presents affects of fixed-point and floating-point quantization [29].

Table 3 Quantization affects on Fixed-point vs floating-point data representation

Fixed-point representation	Floating-point representation
Quantization error is additive	Quantization error is multiplicative in nature
The power of the error remains constant	Due to continuous change in quantization step, error power also changes constantly
Noise level is represented by number of bits	Noise level is represented by bits in mantissa
Removing quantization error is easy	Removing quantization error is complex
Quantization error is constant irrespective of signal value	Quantization error increases with the increase in signal value

4.5 Data Representation in Digital Electronics Design

In digital electronics, data or instructions are not entered and processed by directly using human language. The data such as numbers, symbols or pictures etc are first converted into machine readable format that is binary form before processing. Digital electronic components like processors all are made of millions of electronic circuits. In electronic circuits, high voltage is interpreted as ‘1’ and low voltage is interpreted as ‘0’ similar to switching an electronic circuit ‘on’ and ‘off’. This forms the base for using binary number system in digital electronics. The following paragraph discusses the reasons why we use binary system in digital electronics.

Developing devices that can understand human languages directly is proven difficult because of its complex nature. However, constructing electronic circuits based on ‘off & on’ logic is simple in comparison. All forms of data are possible to represent in binary number representation. Few of the other benefits are more reliable digital devices, small in size and use less energy when compared to analog devices. The basic unit of binary representation in digital electronics is bit.

In digital design flow, specifications are usually written in textual representation to define system behaviour and performance. The next step is the design of algorithm model with the specifications defined. Most of the time system modeling is done by using tools such as MATLAB, Simulink, C/C++ etc. to simulate DSP algorithms. At this design stage, MATLAB model uses floating-point arithmetic due to the need of accurate computations and ideal system performance analysis. MATLAB model is used as reference and it helps to generate input data for bit-exact C++ model testing and provide information of the signal quality of the C++ model. Although MATLAB model uses floating-point arithmetic, C++ model provides fixed-point bit exact reference stimulus and response vectors to be used later in verification of the HDL design.

After the algorithm modeling, the design is implemented in fixed-point binary arithmetic at RTL using HDL. At verification stage of digital design flow, reference stimulus vector is fed to the Design Under Test (DUT) as represented in the Figure 15. The simulation output data of DUT is dumped and compared with reference response vectors to verify the functional correctness of the HDL design. This process is iterated until expected results are achieved or the HDL model is fully verified.

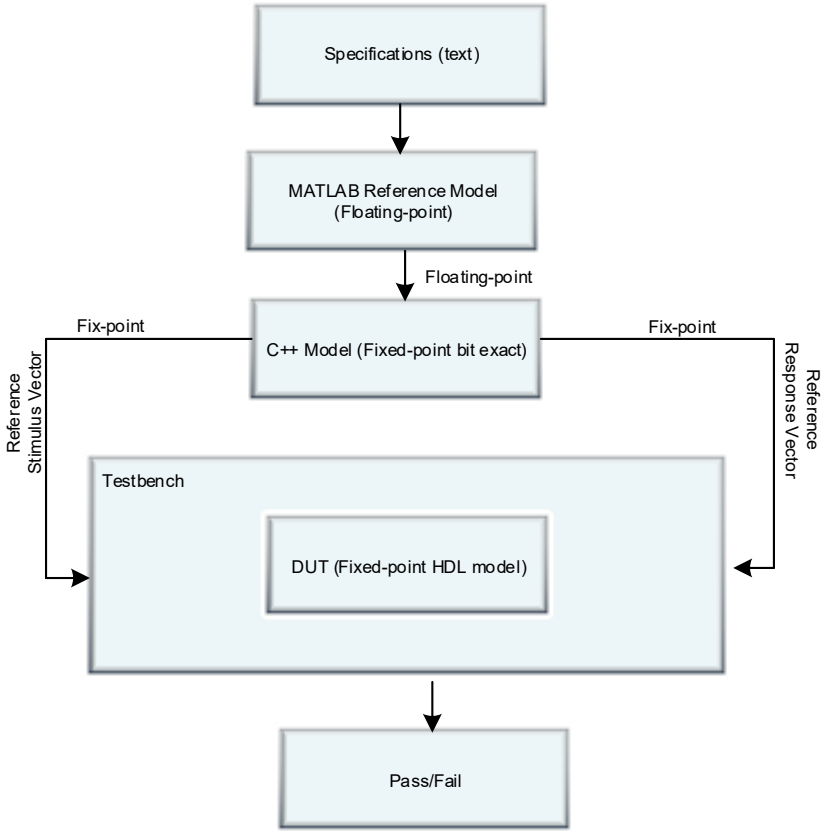

Figure 15. *Data Representation in Digital Design Flow*

The selection of arithmetic data representation is very crucial in digital electronics as it has the highest impact on cost/performance trade-off [30]. One of the two data types from fixed-point or floating-point are selected due to system requirements.

Fixed-point data representation is selected due to low power, small silicon area, low latency and simple circuitry. However, floating-point offers higher accuracy, greater dynamic range and Signal to Quantization Noise Ratio (SQNR) [32].

Even though floating-point offers higher dynamic range for data greater than 16-bits but computations become complex with varying exponent for different data. One of the key requirements in embedded systems is achieving low power. Based on IEEE-754 standard, floating-point arithmetic needs minimum of 32-bits while fixed-point mostly requires 6 to 16 bits. The use of more bits leads to larger bus sizes, memories and complex computations that results in more power consumption and high cost architecture [31].

Hence in digital electronic design, fixed-point arithmetic is preferred over floating-point unless it cannot be avoided.

5. CONTINUOUS INTEGRATION SYSTEM FOR SOC

In this chapter, we will discuss about CI due to advancement in technology and the need to release faster time-to-market products. It became crucial for organizations to deliver innovative products in a consistent and reliable manner.

In this scenario, companies have to better respond to dynamic market requirements by following latest development models that will help to improve performance and quality control. The answer to an efficient and reliable system is Agile methodologies and DevOps culture. DevOps extends the agile methodologies to swiftly deliver products in an automated and integrated process. CI is an agile practice that helps to verify and integrate code from the early stages of the design cycle [9].

5.1 Continuous Integration System

CI was first developed for improving software development cycle. One of the reasons was to automate integration process that takes a big part of the development process in traditional software development cycle [16].

CI goes hand in hand with the evolution of development models. This methodology emerged at the same time and hence why is defined as:

"*A software development practice where each team member integrates their work regularly, usually each developer integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect errors as quickly as possible [19].*"

As a result of following this approach, a cohesive software with reduced integration issues is developed. This technique also helps to save large amount of time and human resources. Some of the mandatory requirements to implement a CI system that is presented in the Figure 16 are [9]:

- Version Control Repository
- Build script
- CI server
- Feedback mechanism of choice

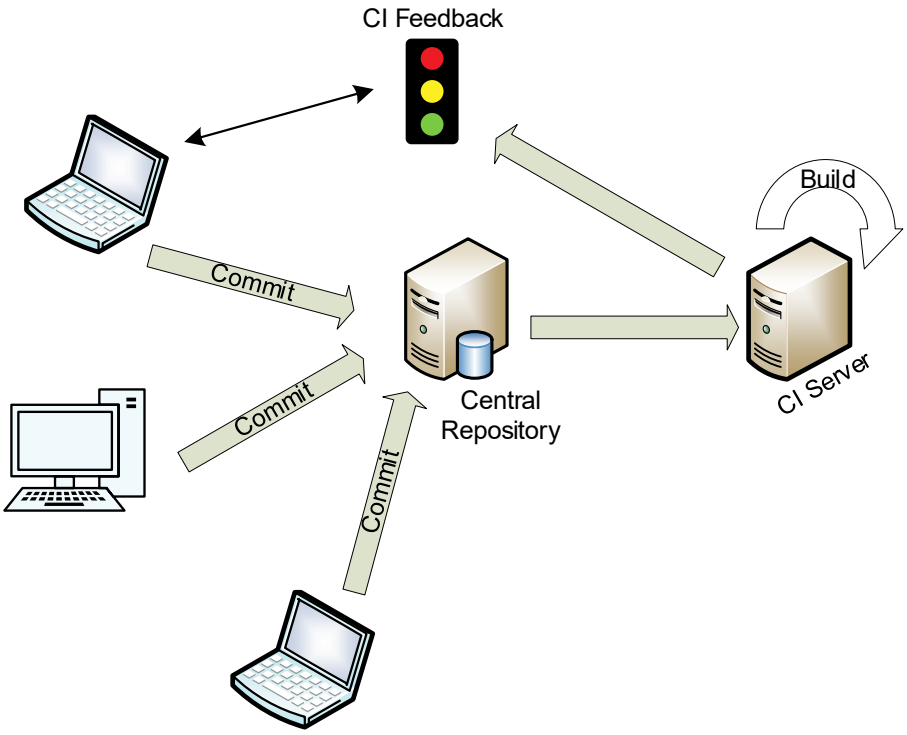

Figure 16. *Continuous integration design flow*

In order to design a CI system, below are the summarized main rules to implement [19]:

1. Design automated tests to run on local machine before committing.
2. After local test pass, submit to the version control repository.
3. After certain time intervals, CI server checks for any possible change.
4. In case of CI server detects any change in central repository, it fetches the latest code from the version control repository. Main purpose of CI server is to perform build process and creates scripts to integrate code centrally.
5. CI server sends build results to the team members via email or SMS etc. notifications.
6. In case of any change detected in central repository, CI system goes to step 4 and continues till the end.

5.1.1 CI System Benefits

CI was first designed and implemented for software development project, even than the benefits of this methodology are applicable for both software and hardware development. Some of the benefits of CI system are given below [16][19]:

- CI helps to automate the usual repetitive manual processes like code compilation, testing and integration.
- Higher code quality is achieved due to the fact that the code which is in version control works.
- CI increases confidence in the work done, helps in risk reduction and leaves no room for the fear of undelivered project.
- Complete overview of the system at all times with the knowledge of what works, work done and any bugs in the system.
- Due to frequent commits, a small bit of the system is changed. Designers don't have to look far in order to detect bugs, hence easier to fix because of less bugs at each stage.
- CI allows better project visibility and makes collaboration easier between the teams.
- Better division of work responsibilities among teams due to impartial CI system.
- CI helps in better project planning and time-to-market estimates because of current status visibility.

5.1.2 CI System Tools

There are numerous CI tools that are available in the market. However, selection of the tool depends upon multiple factors but the main feature targeted is automating the design development processes. There are many CI tools such as Jenkins, Bamboo, CircleCI and Codeship etc. All of the CI tools offer almost similar features with few exceptions from each other [16]. Although there are frequent releases of new tools, Jenkins is the most used and readily available open-source tool. Despite being open-source low cost tool, Jenkins has the largest library of available plugins. Rest of the Jenkins features are given below and presented in Figure 17 that demonstrates its convenience of use [15]:

- Multilanguage support
- Easy configuration
- Compatible with all version control systems
- Supports extension to customise a project
- Largest number of plugins
- RSS/E-mail/IM easy feedback integration

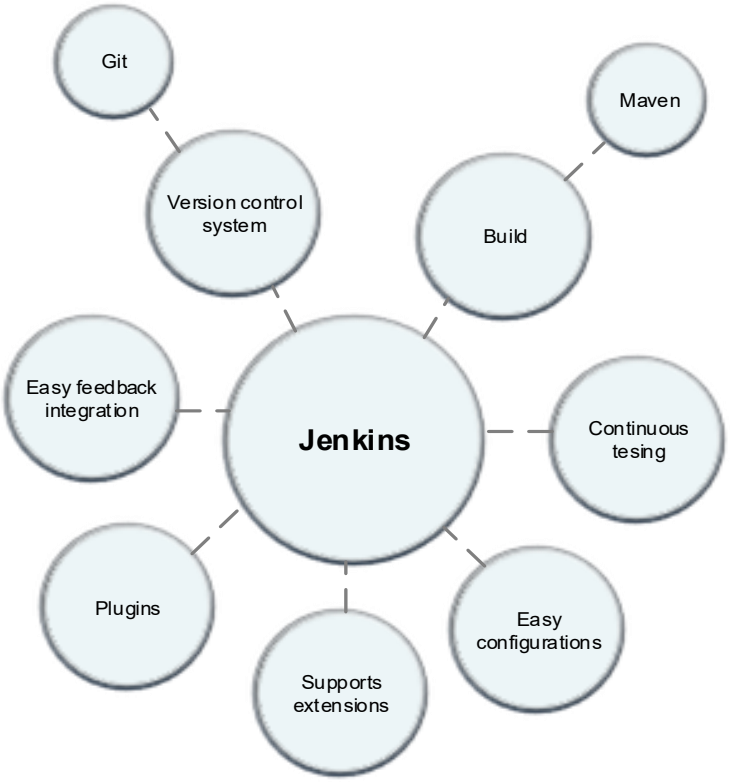

Figure 17. *Jenkins-CI tool features*

5.2 CI System for SoC

Implementation of Agile methodologies in design and verification processes of hardware development and need for CI implementation was realized early on. During different stages of SoC development, concepts of CI helps in building test cases, designing regression/non-regression tests and coverage driven verification [18].

Hardware development consists of many parallel processes despite the contrary belief of delivering one final chip at the end. It requires development of functional block (IPs) mainly from modeling to designing in RTL while integrating and updating the blocks. Until the first silicon sample which too needs integration and packaging, each step can utilize benefits that a CI system offers [17].

From SoCs modeling perspective, different functionalities and feature are continuously added in models which requires multiple builds and tests to confirm their status. At RTL level, designing modules and then integrating them to higher level systems can utilize integration and build methodology. Functional and regression tests are done for the verification of IPs for both system and subsystem level. All of these indications confirm the benefits of CI in hardware development.

One of the benefits of CI is the confirmed availability of working code in the repository. This allows better collaboration between different teams; one scenario is the timely setup of test vectors and verification environment from design model to check RTL design before its completion. One more example is that layout teams can use the CI code from repository anytime and start with synthesis because of no likely issues due to already passed pre-checks [16].

In hardware, CI helps to better analyse bug-fixes and need of new features because of the frequent sanity checks. Other benefits of CI in SoC includes, preventing designers to keep massive code changes at their desk because of frequent commits which plays a huge role in faster time-to-market expectation of the customers.

5.3 Potential Problems in CI for SoC

Implementation of CI systems for hardware provides a lot of benefits but there are many problem areas which hardware designers face while using CI systems for their work. Some of the potential problems which SoC hardware teams face are [9][18]:

- One of the problem occurs due to unavailability of test cases to RTL design team in advance by verification team. In this case, committed code should be considered as only a primary release.
- Due to large number of features, integrating an IP can take a long time. Hence, designers cannot commit several times in a day.
- In CI systems one of the most undesired practise is to commit broken code, but in hardware this sometimes can be a possibility. Broken code is committed for backend teams to analyse and do floor planning.
- One of the CI practise that gets violated is to fix broken builds immediately. For example, when verification team tests the integrated IP and provides a fix. Then once again design team integrates the IP. This whole process takes time then recommended immediate actions.
- The recommended debug time for a broken build is 10-20 minutes or else revert the commit to previous stable state. However, in hardware simulations can take hours ultimately debugging also takes much more time.
- In CI systems, it is recommended to keep the build time around 10 minutes which cannot be applied for hardware implementation as the build time can lead up to hours.
- For successful CI implementation, it is not recommended to go home with a broken build. In hardware design, it is not possible as it takes a long time to fix a build.

5.4 Proposed Solutions for CI for SoC

In order to make CI system work seamlessly and efficiently for SoC flows, there are few adaptations made to make it work best for hardware development [17][18]:

- In CI it is suggested to commit multiple times a day in version control repository. However, in hardware it can be as soon as a new functionality is added so that it can be verified regularly.
- One solution to the issue of committing broken code for backend teams in hardware is to implement a workaround to artificially keep the status passing.
- In order to speed up the whole process, isolate sanity tests that take less time and create a dedicated test suite.
- In CI for hardware, it is better to design separate checks to test different features. This will help to isolate the problem in less time with few tests given the design does not work.
- In hardware, fixing bugs can take a lot of time due to complex design. A proposed solution is to revert the design to an older commit and fix the problem on a local machine before committing it back to the central repository.
- To save time, separate more complex/full test suits for overnight run.

6. MODEL DESIGN WITH CI IMPLEMENTATION

This chapter discusses about the Simulink model and its implementation. Jenkins CI tool is used to build and run the test case in order to confirm the correct model functionality. Section 6.5 discusses the results of research study conducted and possible actions to improve SoC development.

6.1 System Architecture

The base model is a frequency domain channel filter that is filtering one carrier [26][27]. The whole modeling design has three phases, first is the MATLAB floating point reference model, second phase is modeling C++ bit exact model and then designing Simulink model.

The MATLAB model act as a test bench for bit exact C++ model and to also quickly test new features. C++ bit exact model is developed so that it can be used in actual testing/verification process of SoC development. Simulink model uses C++ bit exact model so that it can be further used in MATLAB to provide bit exact model to digital front end (DFE). Figure 18 presents relativity between MATLAB and C++ models.

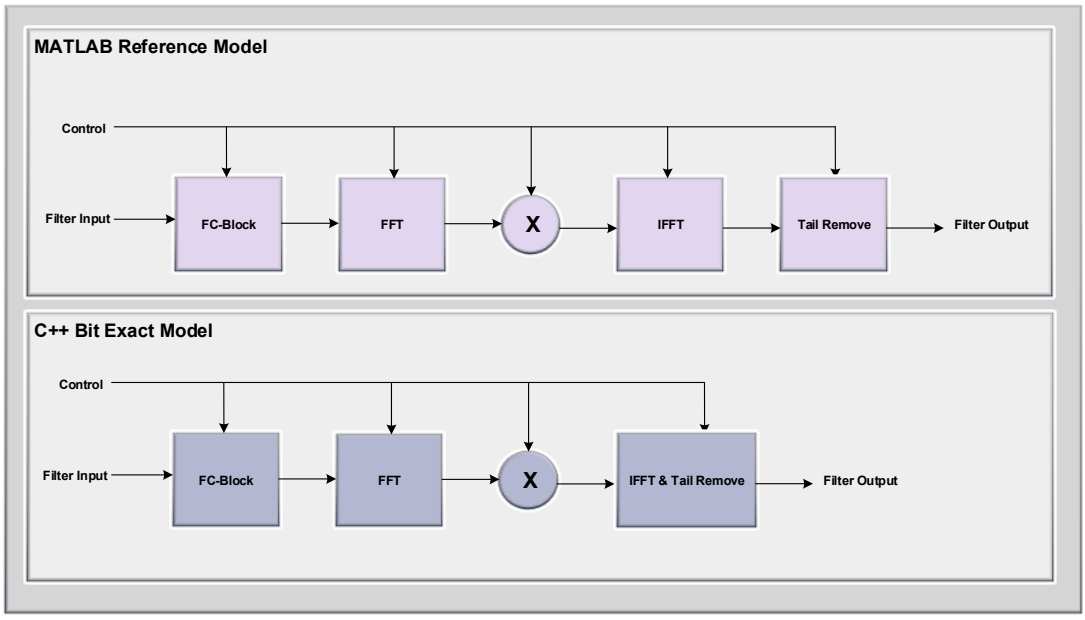

Figure 18. *Channel filter model block diagram*

6.2 Simulink Model

To design this model, first we need to run existing C/C++ model in Simulink. The C++ model takes input signal generated by using MATLAB. For actual simulation there are two input signals; data and configuration. The data is vector of I/Q samples and the configuration is a control vector that defines the information related to current symbol. Figure 19 presents block diagram of the Simulink model.

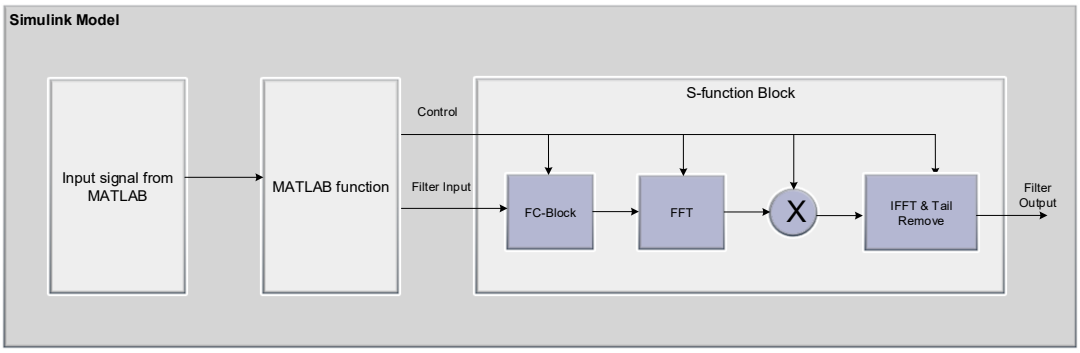

Figure 19. *Simulink model block diagram*

Input data that is generated by using MATLAB, in Simulink model ‘Signal from Workspace’ block is used to feed the input data to C++ model. Both I/Q samples and control have header, which defines whether input is valid or not. 'MATLAB function’ block is used to define a function that checks input and feed to C++ model. To examine the symbol id that checks data validity, read the configuration values and reset the counter for next samples accordingly.

There is a S-function method in Simulink to run the existing C++ model, which is a computer language description of a block that is written in C, C++, MATLAB or Fortran. S-functions are compiled as MEX (MATLAB executables) files, providing an interface between MATLAB and functions written in C/C++. When complied, it allows to automatically invoke external function as if they are built-in to execute Simulink model. The signal generated are then fed into S-function block as input. Multiple set of S-function callback methods are implemented to perform tasks required at each simulation stage. Following paragraphs in this section presents a few of these methods.

Figure 20 presents the snippet of mdlInitializeSizes callback method. It defines the size of several parameters in the SimStruct, for example number of inputs, output, states and parameters of the block.

```
// mdlInitializeSizes
// specifying number of Inputs, Outputs, parameters
static void mdlInitializeSizes(SimStruct *S) {
   int nInputPorts = 2;   // number of input ports
   int nOutputPorts = 1;  // number of output ports

   ssSetNumSFcnParams(S, NUMBER_OF_DIALOG_PAR);  // Number of expected parameters

   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return;  // Parameter mismatch reported by the Simulink engine
   }

   // Configure the Input ports (Data,configuration and register signals)
   if (!ssSetNumInputPorts(S, nInputPorts)) return;

   // Set input port dimensions for each input port index starting at 0
   // CONFIGURATION
   ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
   ssSetInputPortDataType(S, 0, CONTROL_DATA_TYPE);
   ssSetInputPortDirectFeedThrough(S, 0, 1);

   // DATA
   ssSetInputPortMatrixDimensions(S, 1, DYNAMICALLY_SIZED, DYNAMICALLY_SIZED);
   ssSetInputPortDataType(S, 1, INPUT_DATA_TYPE);
   ssSetInputPortComplexSignal(S, 1, INPUT_DATA_COMPLEX);
   ssSetInputPortDirectFeedThrough(S, 1, 1);
   ssSetInputPortRequiredContiguous(S, 1, 1);
```

Figure 20. *Input parameters of the S-block*

Figure 21 presents the mdlInitializeSampleTimes that specifies the sample rates at which this S-function works.

```
// mdlInitializeSampleTimes
// Specify sample rates at which this S-function operates

static void mdlInitializeSampleTimes(SimStruct *S) {
   ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);  // period of a sample time
   ssSetOffsetTime(S, 0, 0.0);                    // Set the offset time of a block
   ssSetModelReferenceSampleTimeDefaultInheritance(
       S);  // referenced model containing this S-function can inherit its sample time from its parent model
}
```

Figure 21. *Sample rate setup*

In mdlStart callback method, it initializes the state vectors of the S-function. Figure 22 presents how the register values from text file are fed that contains current symbol information to call C++ function.

```
   // Read filter coefficients from file
   int regValue;
   std::ifstream inFile;
   inFile.open("../filterCoefficients.txt");
   if (inFile) {
      while (inFile >> regValue) {
         regValues.push_back(regValue);
      }
   }
   inFile.close();

   fftfilt->registerWrite(regValues);
}
```

Figure 22. *Register values to call C++ function*

One of the most important callback methods is mdlOutputs, it computes the signals that the S-function block emits. Figure 23 presents, valid output complex sample data is transmitted to output port by calling C++ function to the output port and separating the real and imaginary parts.

```
// Emit data to output
int stiOut = ssGetOutputPortSampleTimeIndex(S, 0);

if (ssIsSampleHit(S, stiOut, tid)) {
   creal_T *out = (creal_T *)ssGetOutputPortRealSignal(S, 0);
   int_T no = ssGetOutputPortWidth(S, 0);

   std::vector<L1Low_fftfilt::IqData> outData;
   outData = sm->samplesOut(header, no - 1);
   out[0].re = header.tvalid;
   out[0].im = header.tid;

   for (int i = 0; i < (no - 1); ++i) {
      out[i + 1].re = 0;
      out[i + 1].im = 0;
   }

   for (int i = 0; i < outData.size(); ++i) {
      out[i + 1].re = (outData.at(i).real());
      out[i + 1].im = (outData.at(i).imag());
   }
}
```

Figure 23. *Signal at output port*

mdlTerminate – performs any actions essential for the termination of simulation. If no actions are needed, this function can be executed as a stub.

Finally, Figure 24 presents frequency domain channel filter's input and output signal spectrum [26][28].

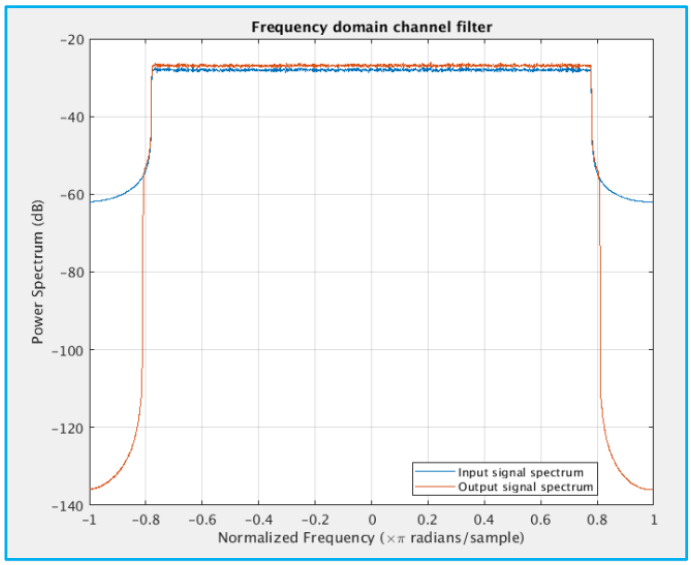

Figure 24. *Signal before and after channel filtering*

6.3 CI System for Model Implementation

Chapter 5, section 5.2 discusses the benefits of CI implementation in hardware development. There are few other reasons due to which CI system is recommended in model design such as multiple people using or modifying the same model. CI is used to avoid using broken models and only the working model stays in repository. One more practise that modeling team takes advantage is in case of new tool releases, CI can test all the used cases with new release and verify existing models.

6.3.1 Jenkins Testcase

Continuous integration tool Jenkins is used to test build results and working of Simulink model. To check, input signal is filtered according to the control information. A testcase was implemented in Jenkins to check received signal quality. The error vector magnitude (EVM) is a measurement for signal quality and it defines the error and actual signal relation [26]. It is confirmed that the build is good after the testcase passes in Jenkins. Jenkins is also used to automate release notes with other SoC teams, for promoting results to higher level instead of carrying out manual testing and creating releases to inform relevant teams.

This section explains, how to setup a project with CI tool for testing and building different targets. First step is to create a new item in Jenkins and then write the name of the project and select ‘free style project’ as presented in Figure 25.

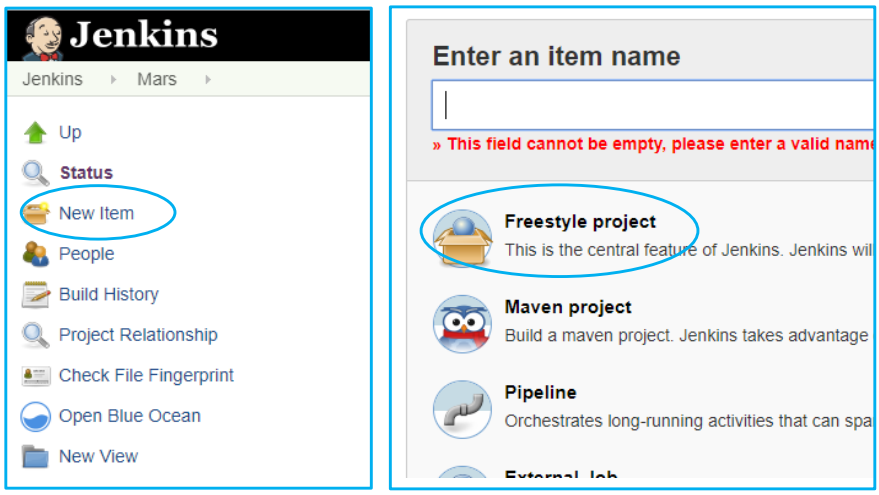

Figure 25. *Jenkins web interface to setup a new project*

Next step is to configure the project, copy the repository URL from GitHub. If the project is built at this phase, it will clone the repository to workspace. After the initial build, add URL and secret token from the Jenkins to GitLab project. As presented in Figure 26, select ‘incoming’ branch to trigger the build and ‘Merge Results’ for every time code is pushed to the incoming branch. Recipients and email alerts are also setup at this phase.

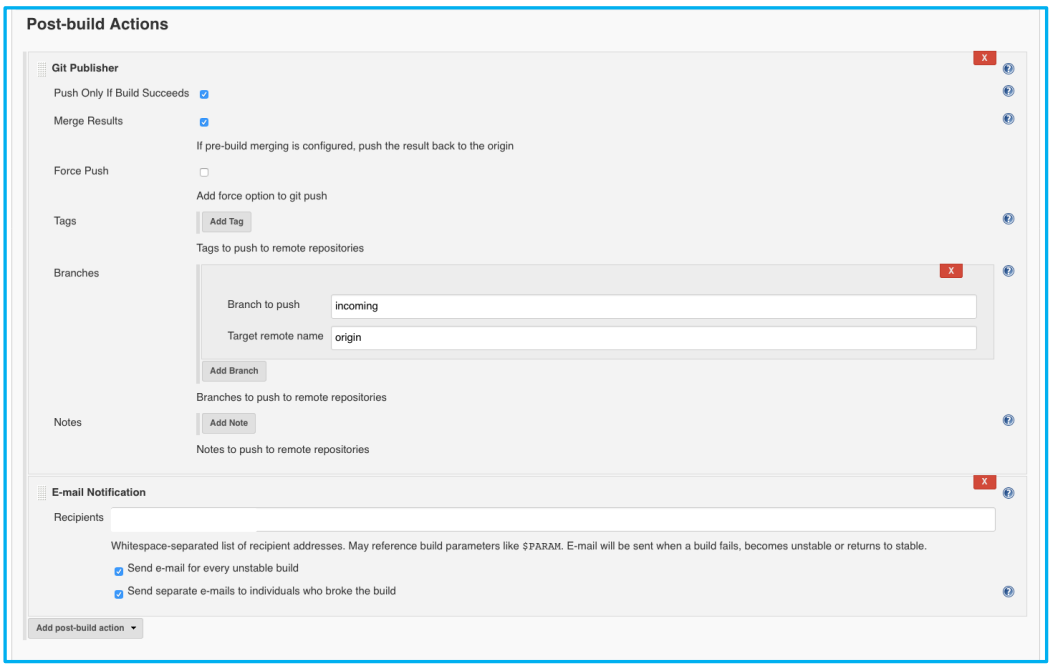

Figure 26. *Jenkins Post-build actions*

Jenkins job is triggered at every commit to the incoming branch. For Jenkins tool, bash script is in the root of the repository to build and test for both desktop and Unix. This script also merges master and incoming code to check updated master's functional correctness. In the bash script, setup for the different environment is done to load correct compile environment and MATLAB in grid or desktop. Figure 27 presents bash script's snippet for environment setup.

```
# Environment based setups

MATLABRUN=""
if [ "$ENVGRID" -eq 0 ]; then
    echo "Grid machine"
    checkexitcode source /projects/ddm/util/.bashrc.project
    checkexitcode module load cmake/3_6_2
    checkexitcode module load gcc/4_9_2
    checkexitcode module load lsf
    checkexitcode module load matlab/2017B
    checkexitcode module load python/2_7_12
    MATLABRUN="bsub -Is -q i_soc -R "rusage[mem=8000]" matlab -nodisplay -nosplash -nodesktop -r"
else
    # desktop setup
    echo "Desktop"
    if [[ "$OSTYPE" == "darwin"* ]]; then
        MATLABRUN="/Applications/MATLAB_R2019a.app/bin/matlab -nodisplay -nosplash -nodesktop -r"
    fi
fi
```

```
# check current environment, if grid/unix is used
echo $HOSTNAME | grep -q "server.net"
ENVGRID=$?
# if Jenkins is used
if [ -z "$JENKINS_HOME" ]; then
    echo "No Jenkins"
    JENKINSEXISTS=1
else
    echo "Jenkins run"
    JENKINSEXISTS=0
fi
```

Figure 27. *(top) Check if grid or Jenkins environment, (bottom) Environment based setup for MATLAB*

To track the changes in the selected directories, paths are given to compare latest and previous versions by running compilations and tests as presented in Figure 28.

```
# SIMULINK changes check
checkexitcode cd $SIMWRAP_PATH
#memorizing test directory
ctd=$(pwd)
checkdiff $SIMULINK_PATH
SIMULINKDIFF=$?
if [ "$SIMULINKDIFF" -eq 1 ] || [ "$TESTTYPESIMULINK" -eq 1 ]|| [ "$TESTTYPEMATLAB" -eq 1 ]; then
    if [ "$TESTTYPESIMULINK" -eq 1 ]|| [ "$TESTTYPEMATLAB" -eq 1 ] ; then

        echo "Running Simulink tests"
    else
        echo "Simulink code has changed, perform testing..."
    fi
    echo $(pwd)

    cd test/tb_simulink
    $MATLABRUN test
    cd $ctd
fi
```

Figure 28. *Bash script snippet to track changes and run test*

This bash script also checks the code in Unix or desktop to start Jenkins job. It is important to provide correct exit codes so the Jenkins tool can differentiate between a successful and unsuccessful test. Figure 29 presents Jenkins exit code in case of incorrect test results.

```
    disp('Jenkins run')
    evm = fftfilt_testrun_release('simulink_test','../test/tb_simulink/simulink_out.mat');
    try


    if evm > 43
        disp("Simulink test fail")
        exit(1)
    else
        disp("Simulink part OK")
        exit(0)
    end

    catch

        disp("Simulink test code error")
        exit(1)

    end
end
```

Figure 29. *Jenkins exit code*

Figure 30 presents grid results in case of changes in Simulink model and a successful testcase result.

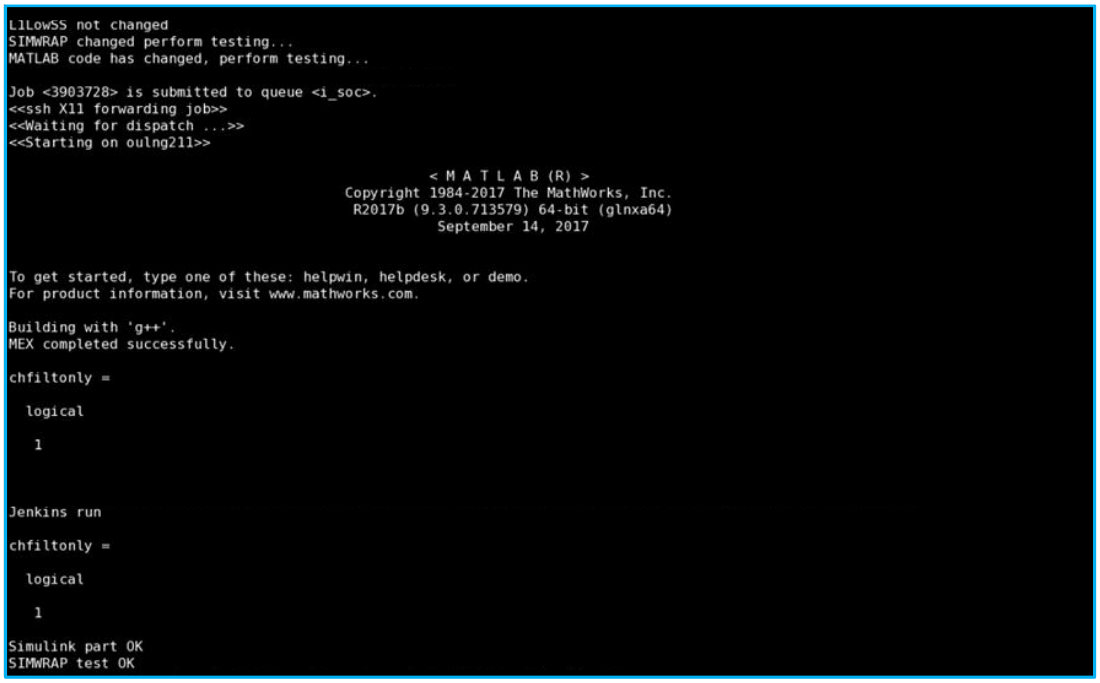

Figure 30. *Jenkins script runs and test Simulink model*

6.4 Result Analysis

The reason behind designing a C++ model of the same exact MATLAB refence model is that it can be used for RTL verification of this particular system in later stages of SoC design. The benefit of designing a Simulink model is that the C++ bit exact model can be used further in other models to provide bit exact filtered data especially to DFE model. Due to use of CI system, the aim of designing a model that is always up to date, at an adequate complexity and is easy enough to be used for a person not mastering the topic is achieved.

6.5 SoC Practices Research and Findings

This section describes about the study conducted as part of the research work done for the thesis work. The first section describes about the interview group and selection criteria. In the next section, results and proposed actions are discussed.

In order to conduct research study, multiple experts from different SoC development teams were selected such as few individuals from modeling team, RTL design and verification teams. All the members volunteered for the study. No particular study style is

followed but a relax discussion sessions with each expert so that they can speak without feeling conscious or pressurised. The research group includes, people from different nationalities with varying work experience of 5 years up to 25 years. Although no particular format was followed but to start the conversation below question was asked:

What are the potential problems you face w.r.t your work and other SoC teams? Some ideas to improve overall SoC development process and impact of CI in your work.

Meanwhile, during the discussion wrote down things that came up and sometimes asked additional questions to elaborate the point made. Each session in duration lasted around 20 minutes to about 45 minutes. Below are the findings of the study conducted:

- One of the top suggestions given by multiple experts is to put more efforts in better planning and resource allocation from the beginning of the project. Taking away engineers or adding more resources later in project often results in turbulence among the teams.
- CI system is utilized in modeling work and it helps to keep the code clean and avoids breaking the models. It helps to test commits to central repository and model testing.
- Most of the modeling engineers don't have hardware design knowledge which leads them to not optimally design fixed point models. One of the key suggestions for modeling engineers is to have trainings to better understand hardware side of the design.
- Using tools to auto generate code such as from MATLAB to C/C++ is not convenient. It contains a lot more redundancy and is hard to explain other teams/members utilizing that model.
- One more suggestion for better project planning is to decide resource allocation before the start of the project and keep them updating, for example number of licenses and server space estimation etc.
- Hardware designers find use of Jenkins tool very helpful in their daily tasks as it helps them to not forget about testing and building their code.
- One of the issues highlighted was, HDL designers don't put needed efforts in verification of their design. By making very basic testbenches and leave the work on verification teams which leads to wastage of time. Proposed solution was to put more efforts in verification so that verification team does not have to initiate bug fixes too many times.

- It was also recommended by one of the designers to use hardware emulators to run verification tools instead of normal serves. This helps to avoid long time queues, as hardware emulators supports range of verification objectives - from hardware verification, hardware/software integration to operating system testing.
- One of the problems verification team encounter is due to dynamic design architecture and specifications. In this case it becomes difficult to develop testbenches. However, it is suggested to have architecture finalised before proceeding with RTL design and verification.
- It is recommended to develop reusable testbench components, so that they can be used in other projects. This will save a lot of time and helps in early delivery.
- One other area highlighted is to have company specific libraries on top of UVM to develop reusable testbench components and a common infrastructure rather than every team working on their own to develop testbenches.
- It is highly recommended to have adequate infrastructure such as EDA licenses, CPUs and servers so that engineers don't wait for the availability of licenses instead utilise their time efficiently.
- As part of SoC design methodology, unified development (emulation+software+verification) should be followed for development to achieve better results.
- One of the suggestion given by a verification engineer is to have better coordination between universities and SoC companies. A better curriculum to train students for verification work will eventually help companies to have well trained employees.
- System Verilog is recommended to be adopted in order to design complex SoCs. In case of one language for both design and verification, results in more optimised EDA tools and it will help to shuffle resources between the two teams.
- For better IP reuse and verification teams, more efforts in clear documentation of design specifications and user guidelines should be spent.
- One expert highlighted the importance of having one designated team for analysing the areas that needs improvement throughout SoC teams. This team should constantly evaluate and present idea to improve overall working of the teams.
- Key users for each tool should be allocated so that they can provide support and small trainings now and then for better utilization of the tool and its new features.

7. CONCLUSIONS AND FUTURE WORK

SoC development is a complex process that needs multistage development flows and tools. In this thesis, currently used SoC processes are evaluated to identify the areas that can be improved. A literature study is done to present ideas on how to implement CI for hardware development. Even though implementation of CI in hardware development has its own limitations; faster turn around and better integration among different teams makes it highly suitable. Better project management, clear status of design and ease of integration with version control systems are few of the benefits which allows organization to keep on using CI for SoC development.

As part of the thesis hands-on work to automate and streamline work between different teams in SoC development. A model is designed in Simulink so that it can be used in different stages of SoC development. CI system is also implemented by using Jenkins; a CI tool for automating design build and testing reliability of the model design. The benefit of designing a Simulink model is that the C++ bit exact model can be used further in other models to provide bit exact filtered data especially to DFE.

A research study was conducted with different SoC development experts to identify areas in SoC development cycle where improvements are needed, usability and benefits of CI in their work.

The purpose of the thesis is achieved by learning about SoC development flows, presenting guidelines and recommending actions to improve existing flows and implementation of CI system in SoC. The outcome of the thesis is attained by evaluating benefits of CI system in hardware and practically implementing CI in model design.

The model designed in this thesis sets a very strong base for the implementation of bit exact model for other devices as an extension to this work in future. One other future extension of this work can be to design a system level algorithm model. Additionally, research studies should be organized sporadically to track improvements in SoC flows and to understand new arising issues.

REFERENCES

[1] A. B. Carlson, P. B. Crilly, J. C. Rutledge, “COMMUNICATION SYSTEMS - An Introduction to Signals and Noise in Electrical Communication”, Published by The McGraw-Hill, 4th International edition, 2002.

[2] E. Dahlman, S. Parkvall, J. Sköld, “4G LTE/LTE-Advanced for Mobile Broadband”, Published by Elsevier, 1st edition, 2011.

[3] M. S. Alencar, V. C. daRocha, “Communication Systems”, Published by Springer, 2005.

[4] B. Clerckx, C. Oestges, “MIMO Wireless Networks”, Published by Elsevier, 2nd edition, January 23, 2013.

[5] B. Razavi, “Design of analog CMOS integrated circuits” Published by The McGraw-Hill, 2001.

[6] H. Kaeslin, “Top-Down Digital VLSI Design”, Published by Morgan Kaufmann, 1st edition, December 18, 2014.

[7] M. J. Flynn, W. Luk, “Computer System Design: System-on-Chip”, Published by Wiley, 1st edition, October 11, 2011.

[8] J. Bergeron, “Writing Testbenches: Functional Verification of HDL Models”, Published by Springer, 2nd edition, February 28th, 2003.

[9] A. Glover, S. Matyas, P. M. Duvall, “Continuous integration: improving software quality and reducing risk”, Published by Addison-Wesley, 1st edition, 2007.

[10] P. Rashinkar, P. Paterson, L. Singh, “System-on-a-Chip Verification Methodology and Techniques”, Published by Kluwer Academic Publishers, 2001.

[11] D. Lee, G. Y. Li and S. Tang, "Inter-cell interference coordination for LTE systems”, in proc. IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, 2012, pp. 4828-4833.

[12] F. Boccardi et al., "Multiple-antenna techniques in LTE-advanced," in IEEE Communications Magazine, vol. 50, no. 3, pp. 114-121, March 2012.

[13] K. Kalliojarvi and J. Astola, "Roundoff errors in block-floating-point systems," in IEEE Transactions on Signal Processing, vol. 44, no. 4, pp. 783-790, April 1996.

[14] J. A. del Peral-Rosado, R. Raulefs, J. A. López-Salcedo and G. Seco-Granados, "Survey of Cellular Mobile Radio Localization Methods: From 1G to 5G," in IEEE Communications Surveys & Tutorials, vol. 20, no. 2, pp. 1124-1148, Second quarter 2018.

[15] J. F. Smart, “Jenkins: The Definitive Guide”, Published by O’Reilly media, 1st edition, July 2011.

[16] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, pp. 3909-3943, 2017.

[17] J. Engblom, "Virtual to the (near) end - Using virtual platforms for continuous integration," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2015, pp. 1-6.

[18] “A Continuous Integration System for ASIC Development” Access date: 14 Sep 2019. Available online: http://agilesoc.com/articles/a-continuous-integration-system-for-asic-development/

[19] “Continuous Integration” Access date: 14 Sep 2019. Available online: http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/2013-14 /lecturas/10_Fowler_Continuous_Integration.pdf

[20] “Huawei 5G Wireless Network Planning Solution” White paper. Huawei technologies, 2018, https://www-file.huawei.com/-/media/corporate/pdf/white%20paper/2018/5g_wireless_network_planing_solution_en_v2.pdf?la=en

[21] “The evolution of Mobile technologies” White paper. Qualcomm, June, 2014, https://www.qualcomm.com/documents/evolution-mobile-technologies-1g-2g-3g-4g-lte

[22] Jinhyun Cho, Soonwoo Choi and SooIk-Chae, "RTL generation of channel architecture templates for a template-based SoC design flow," 2008 Forum on Specification, Verification and Design Languages, Stuttgart, 2008, pp. 251-252.

[23] V. B. Kleeberger, S. Rutkowski and R. Coppens, "Design & verification of automotive SoC firmware," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2015, pp. 1-6.

[24] Emilliano, C. K. Chakrabarty, A. K. Ramasamy and A. B. A. Ghani, "Matlab and VHDL model of real time partial discharge detection using FPGA technology," 2011 IEEE Conference on Open Systems, Langkawi, 2011, pp. 389-394.

[25] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim and R. Uribe, "Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design," 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003., Napa, CA, USA, 2003, pp. 263-264.

[26] T. Levanen, J. Pirskanen, K. Pajukoski, M. Renfors and M. Valkama, "Transparent Tx and Rx Waveform Processing for 5G New Radio Mobile Communications," in IEEE Wireless Communications, vol. 26, no. 1, pp. 128-136, February 2019.

[27] J. Yli-Kaakinen, T. Levanen, M. Renfors, M. Valkama and K. Pajukoski, "FFT-Domain Signal Processing for Spectrally-Enhanced CP-OFDM Waveforms in 5G New Radio," 2018 52nd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2018, pp. 1049-1056.

[28] J. Yli-Kaakinen, T. Levanen, M. Renfors and M. Valkama, "Optimized fast convolution based filtered-OFDM processing for 5G," 2017 European Conference on Networks and Communications (EuCNC), Oulu, 2017, pp. 1-6.

[29] Changchun Shi and R. W. Brodersen, "Floating-point to fixed-point conversion with decision errors due to quantization," 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Montreal, Que., 2004, pp. V-41.

[30] L. S. A. Hamid, K. Shehata, H. El-Ghitani and M. ElSaid, "Design of Generic Floating Point Multiplier and Adder/Subtractor Units," 2010 12th International Conference on Computer Modelling and Simulation, Cambridge, 2010, pp. 615-618.

[31] J. H. Min and E. E. Swartzlander, "Fused floating-point magnitude unit," 2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), Columbus, OH, 2013, pp. 1383-1386.

[32] L. Wang, M. J. Schulte, J. D. Thompson and N. Jairam, "Hardware Designs for Decimal Floating-Point Addition and Related Operations," in *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 322-335, March 2009.

APPENDIX A:

```
// Initialize the state vectors of s-function
static void mdlStart(SimStruct *S) {
   L1Low_fftfilt::Fftfilt *fftfilt = new L1Low_fftfilt::Fftfilt(0, 0, logger, "fftfilter");

   // store new C++ object in the pointers vector
   ssGetPWork(S)[0] = (void *)fftfilt;
   ssGetPWork(S)[1] = (void *)logger;

   // Feed register values
   std::vector<L1Low_fftfilt::RegisterData> regValues;

   // Set default values for Filter delay
   regValues.push_back(1024);
   regValues.push_back(0);

   // Read filter coefficients from file
   int regValue;
   std::ifstream inFile;
   inFile.open("../filterCoefficients.txt");
   if (inFile) {
      while (inFile >> regValue) {
         regValues.push_back(regValue);
      }
   }
   inFile.close();

   fftfilt->registerWrite(regValues);
}
```

Figure 31. *S-function mdlStart method to initialise state vectors*

```
// InputData at input port 1
int stiIn = ssGetInputPortSampleTimeIndex(S, 1);

if (ssIsSampleHit(S, stiIn, tid)) {
   const creal_T *u0 = (creal_T *)ssGetInputPortRealSignal(S, 1);

   int_T nu = ssGetInputPortWidth(S, 1);

   if (static_cast<bool>(u0[0].re)) {

      header.tvalid = u0[0].re;
      header.tid = u0[1].re;
      header.tlast = u0[2].re;

      std::vector<L1Low_fftfilt::IqData> data;
      for (int i = 3; i < nu; i++) {
         data.push_back(L1Low_fftfilt::IqData(static_cast<int>(u0[i].re), static_cast<int>(u0[i].im)));
      }

      sm->samplesIn(header, data);
   }
}
```

Figure 32. *Input data at port 1*

```
function [ctrlOut, dataOut] = fcn(dataIn)
coder.extrinsic('evalin')

persistent id;
persistent downCounter;
persistent ctrl_pos;
persistent ctrl_vec;

if isempty(downCounter)
    id = 0;
    downCounter = 0;
    ctrl_pos = 0;
    ctrl_length = 0;
    ctrl_length = evalin('base', 'length(simulink_ctrl_vec)');
    ctrl_vec = zeros(ctrl_length,1);
    ctrl_vec = evalin('base', 'simulink_ctrl_vec');
end

% If all data send
if ctrl_pos == -1 && downCounter == -1
    ctrlOut = zeros(33 + 1,1);
    dataOut = [0+0i 0 0 0 0 0 0];
    return;
end

if downCounter == 0
    id = ctrl_vec(1 + ctrl_pos);
    downCounter = ctrl_vec(10 + ctrl_pos) + 2^(ctrl_vec(14 + ctrl_pos));

    ctrlOut = [1; ctrl_vec(1+ctrl_pos:33+ctrl_pos)];

    ctrl_pos = ctrl_pos + 33;

    if ctrl_pos >= length(ctrl_vec)
        ctrl_pos = -1;
    end

else
    ctrlOut = zeros(33 + 1,1);
end

downCounter = downCounter - length(dataIn);

if downCounter == 0
    dataOut = [1 id 1 dataIn.'];
    if ctrl_pos == -1
        downCounter = -1;
    end
else
    dataOut = [1 id 0 dataIn.'];
end
```

Figure 33. *MATLAB function to check input validity*