Executable Model Based Design Methodology for Fast Prototyping of Mobile Network Protocol: A Case Study on MIPI LLI

Student Name: Rahul Kumar Shah

IIIT-D-MTech-VLSI and Embedded Systems 2012-14

July 3, 2014

Indraprastha Institute of Information Technology
New Delhi

Supervisors
Dr. Alexander Fell (IIIT-Delhi)
Mr. Rajkumar Nagpal (STMicroelectronics)

Submitted in partial fulfilment of the requirements for the Degree of M.Tech. in Electronics and Communication, with specialization in VLSI and Embedded Systems

This research was performed in collaboration of STMicroelectronics India Pvt Ltd.

Certificate

This is to certify that the thesis titled **"Executable Model Based Design Methodology for Fast Prototyping of Mobile Network Protocol: A Case Study on MIPI LLI"** submitted by **Rahul Kumar Shah** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Electronics & Communication Engineering* is a record of the bonafide work carried out by him under our guidance and supervision of Alexander Fell at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Alexander Fell
Rajkumar Nagpal
Indraprastha Institute of Information Technology, New Delhi
STMicroelectronics India Pvt Ltd, Greater Noida

Abstract

Future mobile communication systems incorporate more sophisticated functionalities which improve their performance and increase their complexity. In order to reduce their time to market, the RTL development as well the simulation time of the prototyping phase has to be improved. This work presents a solution for improving the prototyping time by using Model Based Design approach comprising of Simulink HDL coder, HDL Verifier and rapid FPGA prototyping by means of FPGA in loop co-simulation. As a case study, the MIPI Low Latency Interface (LLI) layer protocol is implemented.

The results presented in this report demonstrate that hardware acceleration based on reduction in prototyping time can be achieved by reducing the RTL development time and simulation time needed to validate the behavior of the design under test and enabling FPGA in loop co-simulation. In this dissertation comparison of the automatic generated HDL code from the Simulink HDL coder to that of manual hand-written code is performed. The comparison targets the time to market, area, power and timing constrains for the Data Link Layer (DLL) of MIPI LLI for both the procedures. Moreover, this dissertation discusses the limitation associated with Simulink Model Based Design methodology with a test case, modeling of single cycle latency CRC algorithm. The automatic HDL code generated from the Simulink Model Based Design using MATLAB R2013a, and the manual hand-written Verilog code for the Data Link Layer are synthesized for CMOS 45 nm standard cell ASIC technology. The comparison result shows that time to market value is reduced by more than half with significant decrease of 11% to 17% in the operating speed, the area and power consumption also increases by 25% and 29% respectively.

Keyword- Area, Data Link Layer, Executable Model Based Design, FPGA in loop, Low Latency Interface, power, Rapid prototyping, Simulink HDL coder, time to market, timing.

Acknowledgments

It is an extremely arduous job to acknowledge the invaluable help rendered by all those esteemed people who directly or indirectly contributes to the accomplishment of thesis work. I certainly feel elated and privileged to express my deep sense of gratitude to all of them.

I would like to express my deepest gratitude to my adviser and guide, Dr. Alexander Fell, for his extreme guidance during entire M.Tech thesis work. Moreover, his guidance was phenomenal and always helped me out whenever I got stuck to some problem; he showed me the right path to deal with it.

Secondly I express my acknowledgment towards Rajkumar Nagpal, my industry supervisor at STMicroelectronics India, and my mentor Tejinder Kumar for their specialized and technical support during my stay at STMicroelectronics. I am sure this work would have been extremely difficult for me without their help and guidance.

Besides I would like to thank my lab mate and subcorn Mohammad S Dohadwala at STMicroelectronics Noida Lab for helping in verifying my architecture modeling, testing implementations and reporting bugs.

This thesis would have not completed without thanking Mathworks Inc. people support, and Synopsis Design Compilers expert guidance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increased demand of new functionalities in the mobile communication systems lead to an increase of their complexity. This raises a set of challenges in their development like - energy consumption, tight timing constrains, physical constrains, reconfigurability, production cost etc. However, these functionalities extend the architecture modeling, RTL development and simulation time, consequently increasing the time to market. It may be possible to speed up the architecture modeling by using Model Based Design approach and reducing simulation time, which facilitates reduction in prototyping phase leading to a shorter time to market and enhances the possibility of obtaining better designs for complex systems.

One widely used term to describe model-based development is an Executable Model Based Design [25,26]. The term Executable implies that the model can be simulated to illustrate the functional behavior. Thus by adopting the Executable Model Based Design of Simulink HDL coder, the architecture modeling and the RTL development time is reduced considerably [11,16,22,24]. FPGA based rapid prototyping supported by Matlab environment, facilitates the improvement in simulation time of complex algorithms by taking advantage of target independent code generation for FPGA vendors [15,19,20]. Thus, different algorithms could be tested readily irrespective of target FGPA vendor devices, which accelerate the prototyping.

The hardware acceleration solution proposed in this dissertation is based on Model Based Design approach of Simulink HDL coder [12] integrated with HDL Verifier [13] and hardware in loop co-simulation. Model based feature of Simulink HDL coder allows the fast modeling of architecture or algorithms and corresponding RTL code generation while hardware in the loop co-simulation feature allows data exchange between Matlab and FPGA, and can be used as the backbone for a hardware acceleration environment. Moreover HDL Verifier enhances the process of functional verification which automatically generates test input sequences and saves time spent in writing test benches [11]. In the FPGA-in-loop approach (also known as Hardware-in-loop), the design is deployed on hardware and runs in real time. However, the surrounding components are simulated in a software environment. Such methodology allows software flexibility with real-world accuracy and hardware speed execution [10]. Further, inclusion of FPGA-in-loop co-simulation enables prototyping of an algorithm on an FPGA, which increases confidence that the algorithm will

work under more realistic conditions [15, 21].

As a case study, the MIPI Low Latency Interface (LLI) protocol layer implementation has been considered. The Low Latency Interface is a point-to-point interconnect that allows two devices on two different chips to communicate, i.e. for companion chip or the modem to share the main DDR memory located on the application processor side in the mobile phone. The Low Latency Interface technology connects two devices within a mobile terminal at the interconnect level, e.g. OCP, AMBA protocols, using memory mapped transactions. The Low Latency Interface targets low-latency cache refill transactions [9, 23].

However, it is necessary to model multiple architectures and simulate multiple configurations of the LLI in order to obtain the required functionality and satisfy the requirements of the specification, to test the theoretical requirement. In order to model and run multiple architectures, and develop RTL code for the corresponding architectures, the modeling and RTL generation time is reduced by adopting Model Based Design and simulations are accelerated by HDL Verifier along with FPGA-in-loop co-simulation. This will speed up the development time for the system under test and will ease the process of evaluating the advantages or disadvantages of the hardware acceleration as a tool for fast simulation of complex systems.

This dissertation is organized as follows: Section 1.1 defines the problem targeted by the work presented in this report. Previous work is discussed in chapter 2. Chapter 3 explains Rapid Prototyping characteristics based on Model Based Design Methodology, discusses possibility of Rapid Prototyping by adopting Model Based Design technique and introduces Model Based Design methodology flow. Chapter 4 discusses the theory of case study taken. Chapter 5 discusses implementation of Simulink Model for protocol layers of LLI. Benchmarking results for the automatic generated code to that of manual hand-written code for Data Link Layer is presented and the trade offs of the result in terms of time to market, area, power and timing has been discussed and analyzed in chapter 6. Chapter 7 describe the limitations associated with Model Based Design technique with a test case: Single cycle latency CRC algorithm, as an example. Finally, chapter 8 concludes the work.

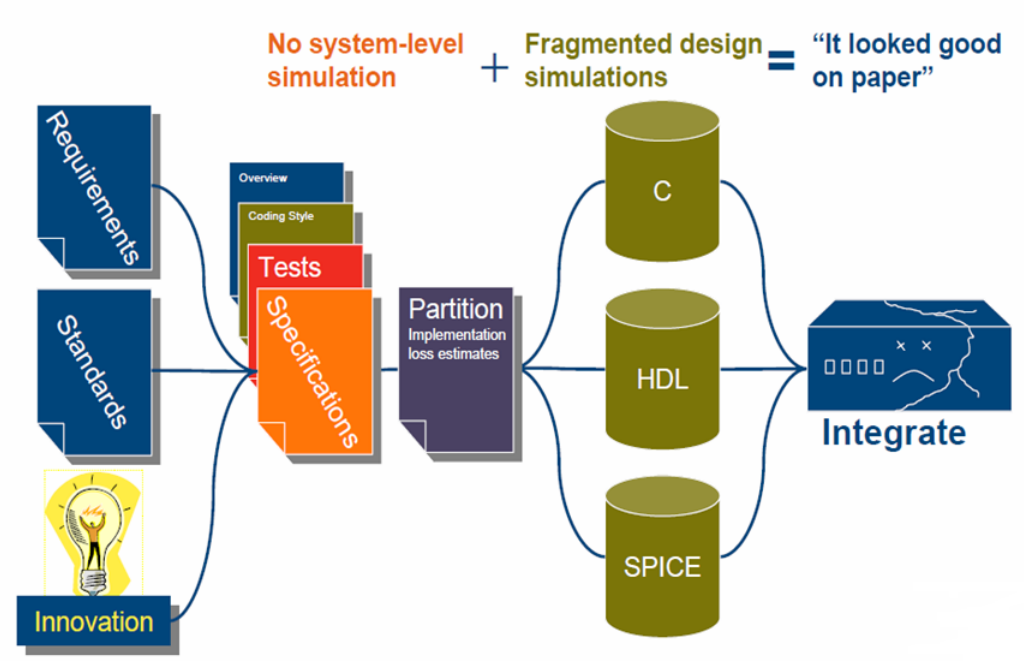

Figure 1.1: Traditional Design Flow. Source: [14]

1.1 Problem Definition

System engineering begins with requirements. In the traditional design approach, the system level designer decomposes the user level requirement into a functional subsystem. This process of functional decomposition continues to lower level of abstractions.

This traditional approach is effective but need not be the efficient mean of the product design. The inefficiency arises due to the inability to objectively verify the design at each step of the process, thus pushing the thorough verification of design to the end. The inability to reuse any design implementation from previous design steps is also inefficient. Moreover the design implementation as well as verification are not at the system-level. Rather they are partitioned as shown in figure 1.1.

To accelerate the engineering productivity of the entire design process, there is a requirement of iterative design modeling techniques. This benchmark is needed to be satisfied that too in very short span of time because the design architectures are vulnerable to very frequent changes during the specification modeling. Adoption of Simulink Model Based Design includes a graphical representation of a design architecture, this facilitates easy debugging of the design model. The use of the Model Based Design method enables the designer to gain a thorough understanding of the design details much earlier in the development process. It also allows the verification process to occur concurrently with the design process, thus improving the quality of the design. Furthermore it supports system level verification unlike the traditional design methodology, as illustrated figure 1.2.

In this work, the concept of a complete Executable Model Based Design which includes all the

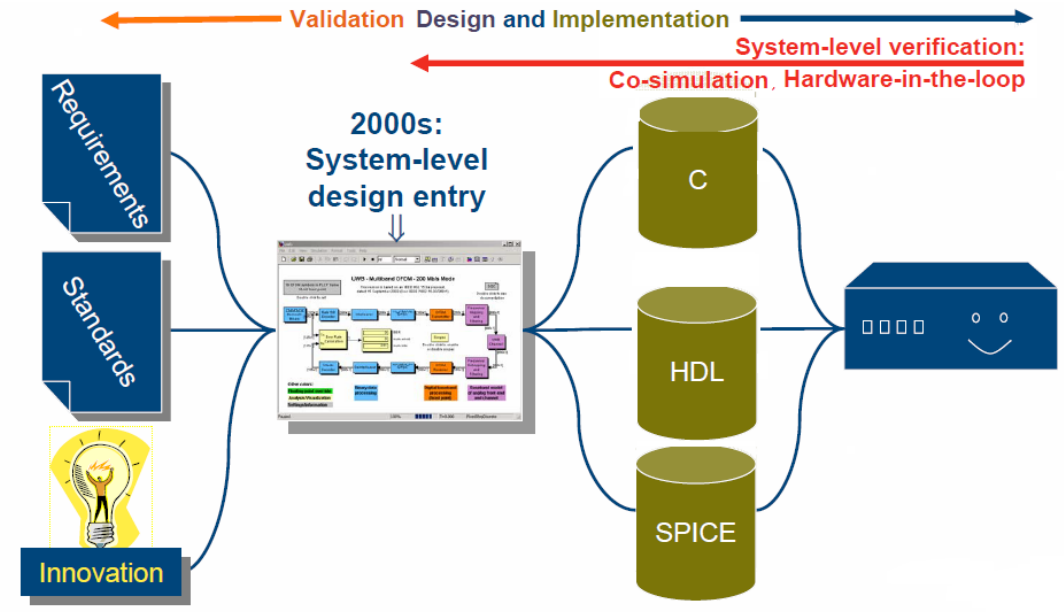

Figure 1.2: Model-Based Design Flow. Source: [14]

elements necessary to illustrate the gain in efficiency promised by the Model Based Design.

Chapter 2

Related Work

System design engineers employ various software design tools and hardware to test prototypes. However, there is no interface between the specification phase to the design phase and further to implementation, testing and validation phase. So the errors occurred in the design process require several iteration stages before the issue is resolved. Those consume large amount of development time necessitated the requirement of a tool for fast prototyping of systems. The prime objective in the development of rapid prototyping system is the availability of better design tools and user friendly highly integrated development environment. Achievement in this direction is a tool that translates an algorithm or architecture model to hardware. There are various tools that perform the conversion from model to hardware description language; few of those include System Generator for DSP and AccelDSP from Xilinx and Matlab/Simulink HDL coder from Mathworks.

2.1 System Generator for DSP

System Generator [3] is a high level design tool specifically designed for Digital Signal Processing (DSP) applications. System Generator is designed by Xilinx to be used in Model Based Design environment along with Simulink from Mathworks [2] for modeling and implementing systems in FPGAs. Simulink provides a powerful component based computing model including several different blocks to be connected together by the user for designing and simulating functional systems. System Generator provides similar blocks which are used and connected in the same way as Simulink blocks does but are target FPGA dependent. The simulation results of the designed systems are bit and cycle accurate which means simulation (through Simulink) and hardware (through System Generator) results exactly match together.

2.1.1 Salient Features of System Generator

1. It generates technology dependent netlists from Simulink, such as- HDL Netlist and NGC Netlist. HDL Netlist is the gate level design of the HDL file and NGC Netlist is the

generated from the corgen blocks in the System Generator library.

2. System Generator provides bitstream that is ready to run on an FPGA platform.

3. System Generator incorporates EDK Export Tool to export the design to the Xilinx Embedded Development Kit (EDK).

System Generator for DSP is also capable of using Synplify, Synplify Pro and Xilinx XST to synthesize the design. A block in System Generator library operates on boolean or fixed-point precision values which is different from Simulink which provides floating-point values. Therefore at the integration interface, Simulink and System Generator blocks are connected to each other with a gateway block which converts floating point precision to the given fixed point precision or vice versa using sampling.

2.1.2 Advantages of System Generator

1. System Generator enables a even moderately skilled user to be able to use it, who is not much experience in Xilinx FPGA and RTL design methodologies because the transition from specification to implementation, including synthesis are automatically performed and referred as push to button transition [1].

2. Xilinx blocksets are optimized architectures since they are generated by Xilinx IP core generators.

3. System Generator for DSP provides an integration platform, in which a user can combine RTL through Simulink and Matlab through MCode block, and C/C++ through Micro Blaze embedded processor components of a system, in order simulate and implement the system level model design.

4. Hardware co-simulation that is provided by System Generator is considerably faster than traditional HDL simulators [4].

2.1.3 Disadvantages of System Generator

1. There is no implicit synchronization mechanism, provided in System Generator i.e. blocks that have the capability to provide a valid signal when the sample is valid; therefore it is the users responsibility to satisfy synchronization explicitly. Thus manual synchronization requires extra effort.

2. System Generator designs often result in inefficient performance values since designed IP cores are treated as black boxes by synthesis tools and therefore synthesis tools do not apply optimization over design module. On the other hand, manual hand-written RTL design provides more independency to the synthesis tools to perform better optimizations.

3. Not all the blocks from Simulink are available in Xilinx blockset. Therefore, user needs to design their own blocks using existing basic Xilinx blocks which will delay the product development cycle and consequently time to market.

4. The System generator does not provide support for system level simulation and verification rather it supports fragmented design simulations.

5. One of the other disadvantage of System Generator is that it is Xilinx platform specific therefore not useful for other FPGA vendor devices such as Altera, and even for the Xilinx platform it is FPGA specific i.e. Vertix-4 netlist is not suited for Vertix-6.

6. High level abstraction that is provided with the MCode blocks come at a cost of efficiency in hardware realization. System Generator provides direct support for Matlab through the MCode block. The MCode block applies input values to an Matlab function for evaluation using Xilinx's fixed-point data type. The evaluation is done once for each sample period.

System Generator enables push a button transition from specification to implementation, it provide libraries to design optimized architectures faster and its ability to compare fixed point and floating point results for any part of the design, makes it one of the better tool for Model Based Design for Matlab code environment. However users need to satisfy synchronization explicitly in their designs through control unit which gets complex in larger designs. System Generator provides a solution for this problem MCode blocks, which comes at a cost of efficiency in hardware realization. Because of these reasons, Xilinx designed another tool, AccelDSP Synthesis Tool, specifically for hardware generation from a given Matlab (.m) code as introduced in the next section.

2.2 AccelDSP Synthesis Tool

AccelDSP Synthesis Tool [5] is another high level tool specifically designed for Digital Signal Processing (DSP) applications. It is a Matlab code based tool developed by Xilinx in 2006. The purpose of AccelDSP is to convert floating point design from Matlab into a hardware implementation that targets FPGAs. AccelDSP provides a user friendly Graphical User Interface (GUI) to write Matlab code and to control design tools (various industry standard HDL simulators and logic synthesizers) in an integrated environment. AccelDSP automatically generates bit-true and cycle-accurate HDL codes which are ready to synthesize, implement and map onto FPGA hardware.

2.2.1 Advantages of AccelDSP design tool

1. AccelDSP is particularly beneficial for user who prefers to use Matlab code to realize hardware.

2. AccelDSP is capable of generating a System Generator Block that can be used in a larger design.

3. AccelDSP provides built-in reference design library, AccelWare [6], that includes a library of synthesizable Matlab equivalent functions.

4. AccelDSP provides a detailed analysis for floating-point to fixed-point conversion. Therefore a user can decide if the precision is sufficient or not.

5. AccelDSP is capable of providing a high level design space exploration to determine design trade-offs by giving capabilities: unrolling a loop and inserting pipeline stages, to the user.

6. AccelDSP have an implicit synchronization mechanism unlike DSP System Generator. The flow of data into and out of the hardware ports is controlled by a handshake interface protocol.

2.2.2 Disadvantages of AccelDSP design tool

1. AccelDSP generated designs result in inefficient architectures in terms of area and timing compared to manual hand written code results. Lindberg and Nissbrandt [18] concluded the same in this work he compared AccelDSP/AccelWare and Hand-Code/Coregen implementations for various signal processing algorithms including FFT, 10×10 matrix multiplication, FIR filter, and CORDIC.

2. AccelDSP design tool requires the input Matlab file to be written in a very specific format. Thus, this code modification can take considerably large amount of time.

3. Even though AccelWare library provides some reference designs, many built in functions are not supported [7].

4. The inputs to the main program should be a scalar, a row vector or a column vector. Other argument types are not supported like matrices.

AccelDSP Synthesis Tool is recommended for generation of small and less complex building blocks, in which used algorithms employ AccelDSP supported built-in Matlab functions (or easy to rewrite) and are easy to change to streaming behavior. These small blocks can be employed in a larger system using System Generator for DSP.

2.3 HDL Coder

HDL coder by Matlab fulfills the dual purpose of generation HDL code from Model Based Design using Simulink HDL coder and that of Matlab function using Matlab HDL coder in Matlab environment. Simulink HDL coder lets users to realize hardware directly from Simulink designs and stateflow charts. Simulink HDL coder provides a detailed code generation report

for each subsystem which allow user to view the subsystems or blocks from which the code was generated in model.

Simulink HDL coder allows user to insert distributed pipeline using Embedded Matlab function blocks and Stateflow charts. Inserting pipeline permits user to gain higher clock rates using pipeline registers with the price of increasing latency. Simulink HDL coder employs Embedded Matlab function block to automatically generate HDL code from a Matlab (.m) file. Matlab coder is used to generate HDL code directly from the Matlab functions provided by Matlab environment. Later the HDL coder performs verification by co-simulation using Mentor Graphics Modelsim HDL simulator or Cadence Incisive HDL Simulator and validates the design in the real condition scenario using FPGA in loop simulation. This combination of Model Based Design feature and Matlab function usage to generate hardware description code enables user to model RTL for both. The one who has little or no knowledge of hardware description language or the other who prefers Matlab code, by using Model Based Design approach and Matlab function translation approach for hardware realization respectively.

Chapter 3

Rapid Prototyping using Model Based Design

For FPGA and ASIC designs, we can use HDL Coder and HDL Verifier to specify and explore functional behavior, generate HDL code for implementation, and continuously test and verify our design through co-simulation with HDL simulators or FPGA-in-loop simulators. HDL Verifier, in conjunction with HDL Coder accelerate our FPGA and ASIC design and verification work flow.

As the modern FPGAs and ASICs have became more complex, system designer have discovered that verification of system-level design efficiently and timely using HDL simulators is not enough. Engineers therefore deploy FPGAs for rapid algorithm prototyping and hardware acceleration. Usage of FPGAs to process large test data vectors allow engineers to rapidly assess algorithm and architecture trade-offs and test designs under real environment conditions without suffering heavy time penalty associated with HDL simulator. System level design and verification tools such as- Matlab and Simulink, allow system designers to rapidly prototype their algorithms on FPGAs.

This work discusses Model Based Design practices for accelerating FPGA prototyping with Matlab and Simulink. The prime applications are indexed below and highlighted in figure 3.1.

1. Employing fixed-point quantization in the design process and optimizing the word length enables us to achieve low area and more power efficient implementations.
2. Enables automatic code generation to accelerate FPGA prototyping.
3. Utilization of system level test benches with HDL co-simulation.
4. Enhances verification rate via FPGA-in-loop co-simulation.

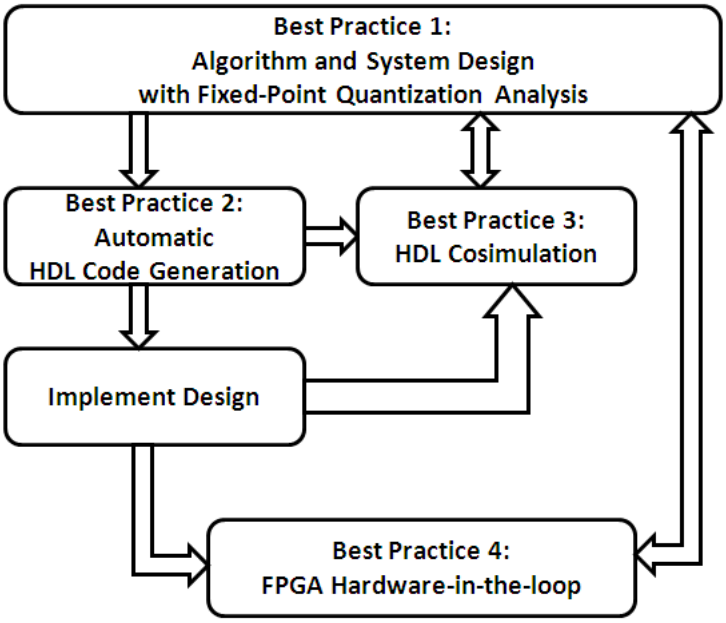

Figure 3.1: Model Based Design practice for fast architecture prototyping. Source: [15]

3.1 Rapid Prototyping How and Why?

Fast Prototyping is achieved in Communication Network Protocol Design due to following reason:

- HDL Coder helps to generate automatically the HDL Code for the Model Based Design thus saving the time compared to hand written RTL code.
- HDL Verifier enables the rapid prototyping of the design by generating the input test vectors which saves time in writing the test benches.
- A prototyped algorithm on FPGA increases belief that the algorithm will work properly in more realistic conditions. The generated code is target independent (of FPGA devices). Moreover, because of the fast data processing capability of FPGA prototypes large number of test data sets can be used for the validation process resulting in exposing errors that would have not been discovered by a simulation model.

Usage of Model Based Design facilitates automatic HDL code generation as well as automatic functional verification using HDL Verifier and enables to produce the prototype faster than a manually coded workflow [15]. In addition the advantage associated with this approach lets engineers accelerate hardware iterations by changing algorithm at the system level, unlike implementation level modification.

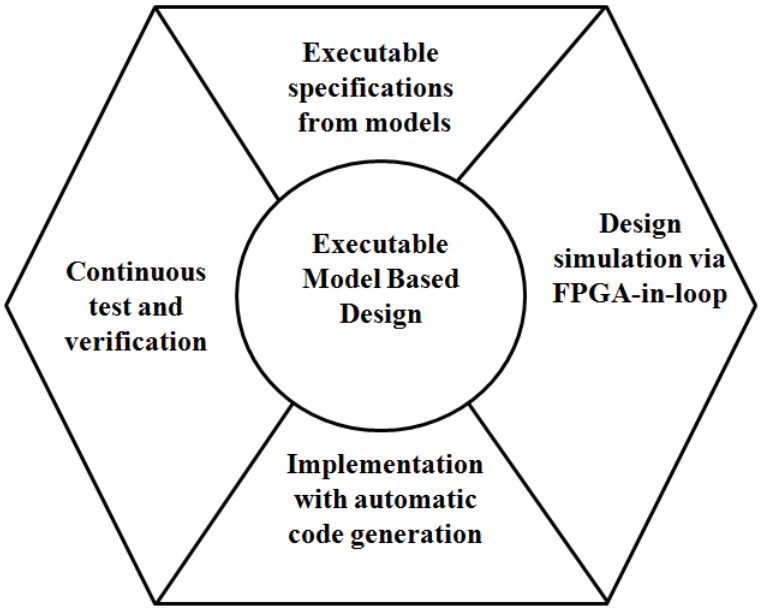

Figure 3.2: Graphical Representation of complete Executable Model Based Design

3.2 Model Based Design Definition

Executable Model Based Designs capture the functional behavior of the system. The complete Executable Model Based Design must first include design documentation. There must be an executable model and there must be a verification environment within which model can be run. In Model Based Design, the development process centers around a system model from requirements capture and design to implementation and test, as illustrated figure 3.2. One of the major advantages of executable Model Based Design is that even a moderately skilled user can readily, and with the little help, execute the model and be able to assess the output quality.

3.3 Model-Based Design flow using Matlab/Simulink

Model Based Design is used as an executable specification throughout the development cycle unlike the traditional manual approach which relies on physical prototypes and textual specifications. It supports both component level as well as system level design and simulation, automatic code generation, continuous test and verification and prototyping algorithm on an FPGA via FPGA-in-loop co-simulation; as illustrated figure 3.3.

The flowchart of Model Based Design approach from automatic generation of HDL code to the functional verification of design under test on real hardware via FPGA-in-loop co-simulation is shown in figure 3.4. The HDL coder generates Verilog or VHDL code that implements the design incorporated in the model together with a test bench. Further, to ease the verification process, scripts are generated.

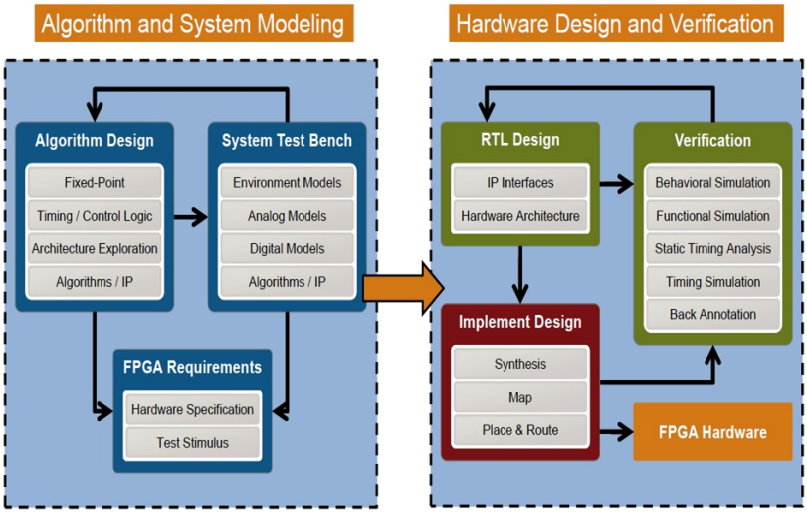

Figure 3.3: Model Based Design flow from Algorithm to FPGA Implementation. Source: [8]

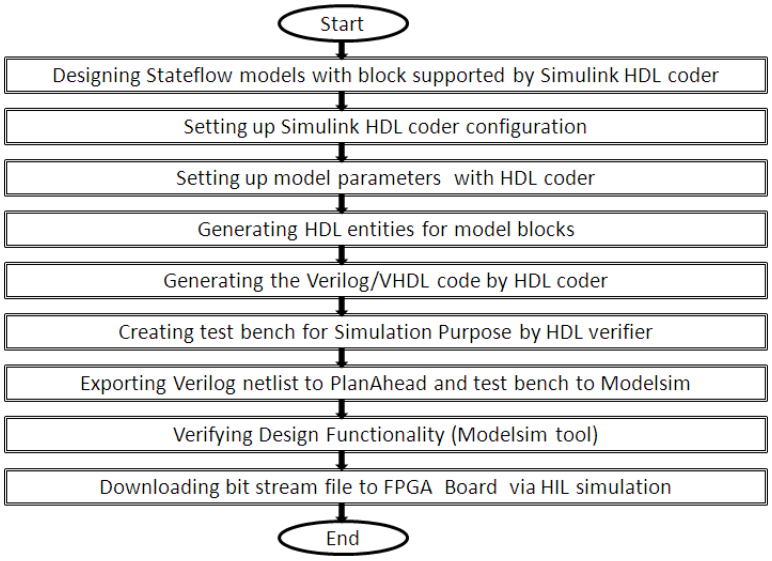

Figure 3.4: Design Methodology Flowchart for Simulink Model Based Design

It takes several iterations in the design process to reach an implementation that is functionally accurate and fulfills the performance criteria; same applies for the testing phase as well. Since the design process is composed of collection of design alternatives and in order to choose the optimal design alternative it is required that designer should have flexibility to return back to original model, make the required changes and regenerate the corresponding code whenever required. After the successful accomplishment of design and testing phase, the generated code is exported for synthesis and later to the FPGA or ASIC for hardware realization.

Chapter 4

Case Study: Low Latency Interface (LLI)

4.1 Introduction

In today's smart mobile phones and tablets industry, one of the biggest challenges is the board level communication process between the device components because variety of interfaces connecting these components are required such as : LLI, SSIC, UniPort, UFS, CSI, DSI, SLIMbus, RFFE, SPMI, DigRF etc [23].

One of the major challenges in mobile phone is to reduce the electronic bill of material (e-BOM). With phone peripheral becoming more and more complex, as most of them have their own CPU-DDR (Double Data Rate RAM) subsystem, reducing BoM is not a simple task. To resolve this issue Mobile Industry Processor Interface (MIPI) Alliance developed the LLI (Low Latency Interface) which is a serial interface that enables peripherals, like modems for example, to share the system main DDR located on the application processor side, which thereby, enables mobile phones manufacturers to remove the modem DDR and reduce the total phone cost. LLI is also used for Inter-processor applications (IPC) [9].

A case study demonstrates the design approach, discussing the high speed serial link protocol LLI design for M-PHY being the physical interface to interact with outside world [9].

4.2 LLI Architecture Overview

1. System-Level Overview

   LLI is a Low Latency point to point mapped interconnect that enables two devices on different chips to transfer data, i.e. for companion chip or the modem to share the main DDR memory located on the application processor side in the mobile phone [23]. LLI is also used in IPC in which host SoC is attached to an external memory device, and the

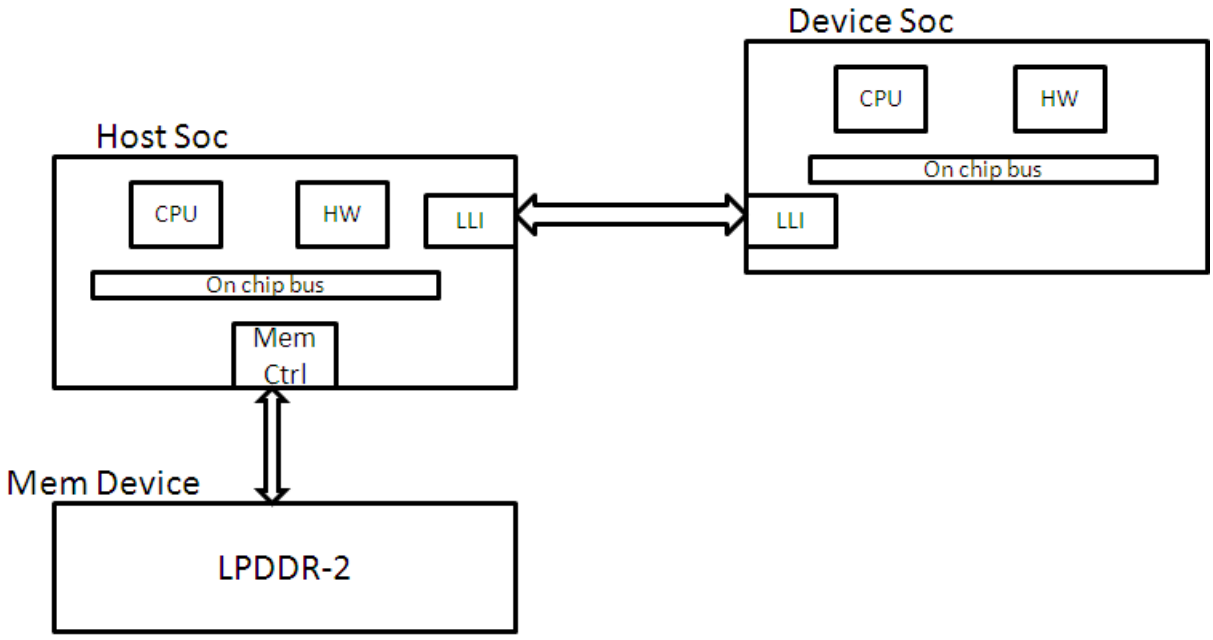

Figure 4.1: LLI system Level model. Source [27]

device memory is required to share its memory with the host device. The device SoC has access to the memory via the LLI link while the host SoC has access via its memory controller [9], as shown in figure 4.1.

Therefore LLI could be considered as a tunnel between two chips on the interconnect level, using memory mapped transaction i.e. memory device is shared between two devices.

2. LLI Layering Model

   LLI is a protocol layer and its stack constitutes of three layers- The Transaction layer (TL), Data Link Layer (DLL) and PHY Adapter (PA) Layer. The LLI protocol layer is implemented on top of physical layer. The TL is the uppermost layer of the LLI stack. It provides an interface between the LLI stack and the chip local to it. The prime duty of TL is to generate header and data packets. The DLL in the LLI stack adds credit -based flow control on each packet on the transmit side, and manages flow control on receive side. The PA layer adds CRC and sequence number for each transmit packet and does the CRC and sequence number checking on the receive side. The figure 4.2 illustrates the functionality of all layers of LLI.

3. Data Exchange Protocol

   This subsection focus on the data exchange protocol in burst mode. The LLI transaction begins with a header packet containing the 36-bit address and other necessary information for the propagations like transaction length, burst transaction type and an Ordering Identifier (OrdId) etc, specific for each transaction.

   The data packets are then sent with a 64-bit payload each and an 8-bit Bytes Enable

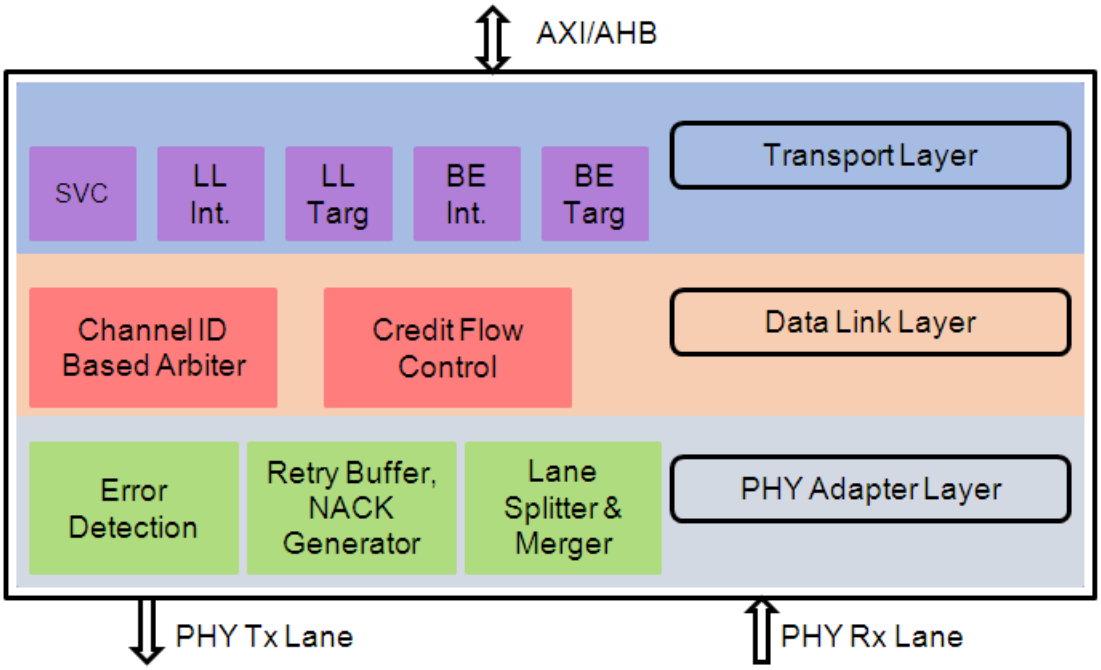

Figure 4.2: LLI Layer model

(BE) plus 1-bit Last indicator (La). The Data Link Layer then adds 7-bit flow control information for the packets (header packet and every data packet) and the PHY Adapter Layer adds 6-bit Sequence Number (SEQ) and a CRC-8 for the packets (header packet and every data packet) [9,23]. The figure 4.3 shows data exchange protocol in burst mode.

4. Error Management

   For the LLI, a CRC-8 is sent with each data packet. The receiver does not acknowledge each data packet; it only sends Non-Acknowledge to the transmitter when an error is detected. The transmitter does not resend all the transaction again but only the latest data packets contained in its retry buffer, which is guaranteed to contain the corrupted data packet because the round-trip latency is bounded. The receiver identifies the corrupted packet thanks to its specific sequence number.

   The sequence number is also used for error detection; if the sequence number of a coming packet does not match with the expected one (latest packet's SEQ + 1), an error will be signaled to the transmitter. However sending CRC and sequence number information with each data packet will have a negative impact on the throughput. LLI can't have a throughput ratio greater 0.67 (i.e. throughput cannot exceed 67%) and it is because CRC, Sequence Number and flow control information are sent with each data packet which is responsible for decreased data throughput.

5. Latency

   The main latency parameter is the Round Trip Time (RTT), which is the time that it

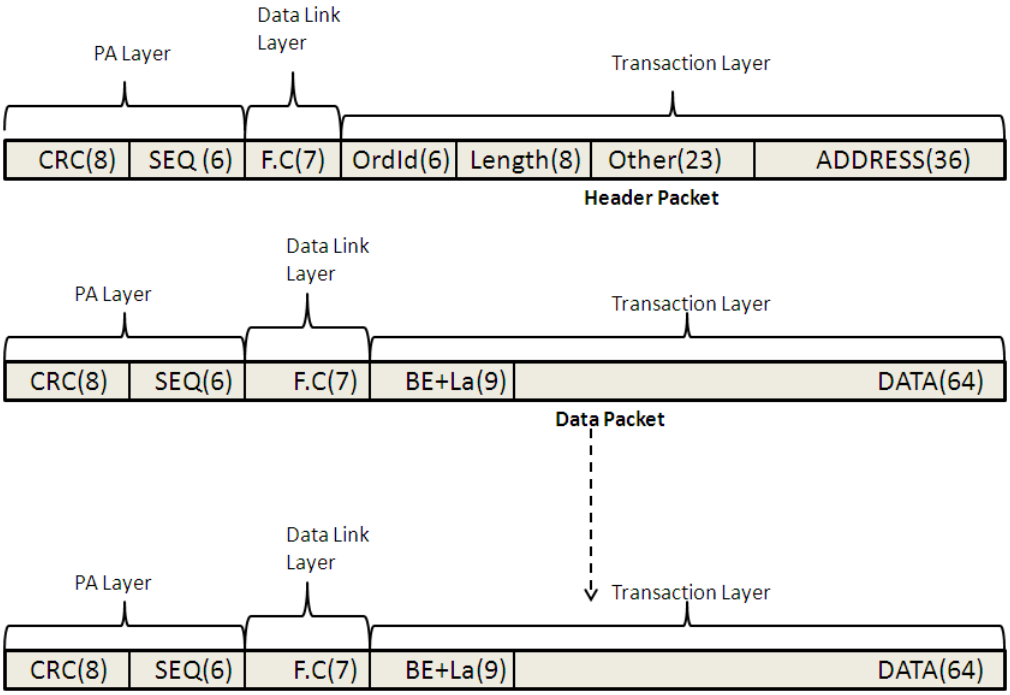

Figure 4.3: LLI Data Exchange Protocol in Burst mode. Source: [23]

takes for a receiver to send Non-Acknowledge to the transmitter and to receive the resent data. LLI's RTT is independent of payload size because the transmitter resends only the latest data packets. LLI guarantees a maximum of 100ns RTT [9,23]. Figure 4.4 illustrates the latency offered by the LLI stack during cache refill transaction.

6. Bit Error Rate (BER)

   A BER of 10^{-20} is desired to meet the requirements of cache refill operations [23], thanks to the low latency error management mechanism in the PA layer of LLI. The error management scheme of LL interface adopts the dual error detection strategy of providing CRC-8 checksum value as well as 6 bit Sequence Number (SEQ) for each transmitting data packet of 80 bit unlike the other high speed link serial protocols such as PCIe 2.0 and USB 3.0, which calculate the CRC checksum value for the data in a payload. This makes it more reliable and less prone to errors, helping in achieving the required BER of 10^{-20}. However, these extra bits added for the procurement of low BER decreases the throughput to 67% [9, 23].

7. Quality of Service (QoS)

   LLI uses 3 service access points (SAP) based on priority: The Best Effort SAP (BE SAP) is used for normal Read/Write transactions, it has the lowest priority. The Low Latency SAP (LL SAP) has a higher priority than BE SAP; it is generally used for cache refill. The Service SAP (SVC SAP) or signaling is used for interrupt events, DMA request and configuration purpose. It has the highest priority [9]. Figure 4.5 shows the traffic flow mechanism of LLI.

Bit clock (MHz)	BC	5800	
Byte clock (MHz)	ByC	580	
Byte clock/2 (MHz)	ByC2	290	
Interface clock (MHz)	I	400	
LLI MPHY Lanes	#	4	
Delay per Stack traversal (RX = TX)	Cyc	Clock	Delay(ns)
LLI IP IF Wrapper/Encapsulation (=TL)	1.0	I	2.50
Arbitration/DeMUX/Credit MGMT(=DL)	1.0	I	2.50
Upper half of I<->BC2 Synchronizer	1.0	I	2.50
Lower half I<->BC2 Synchronizer	1.5	ByC2	5.17
CRC/ARQ/Serialization (=PA)	1.0	ByC2	3.45
PHY I/O delay (=PHY)	0.5	ByC	0.86
Stack traversal delay (= 1/4 of LLI CWF)			16.98
PHIT size calculation (in bits)			
TL Packet Payload (per REQ or RSP)	64		
TL Packet Header (max. overhead)	16		
LL FLIT Header	8		
PA PHIT Trailer (CRC-8)	8		
PHIT size	96		
PHIT one-way Transmission Delay		BC	5.17
Total Critical Word First Delay (*Roundtrip*)			78.28

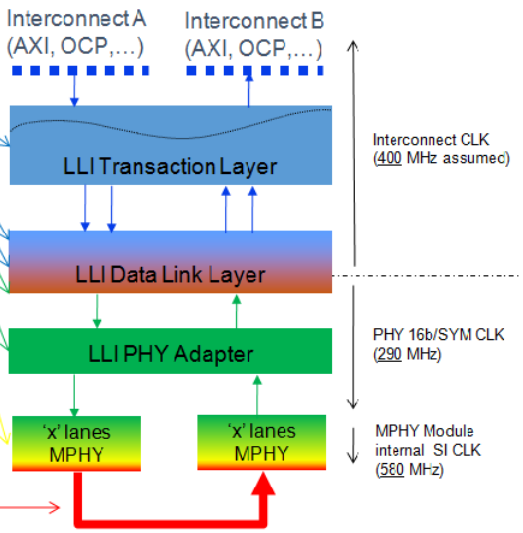

Figure 4.4: LLI Stack Delay. Source: [9]

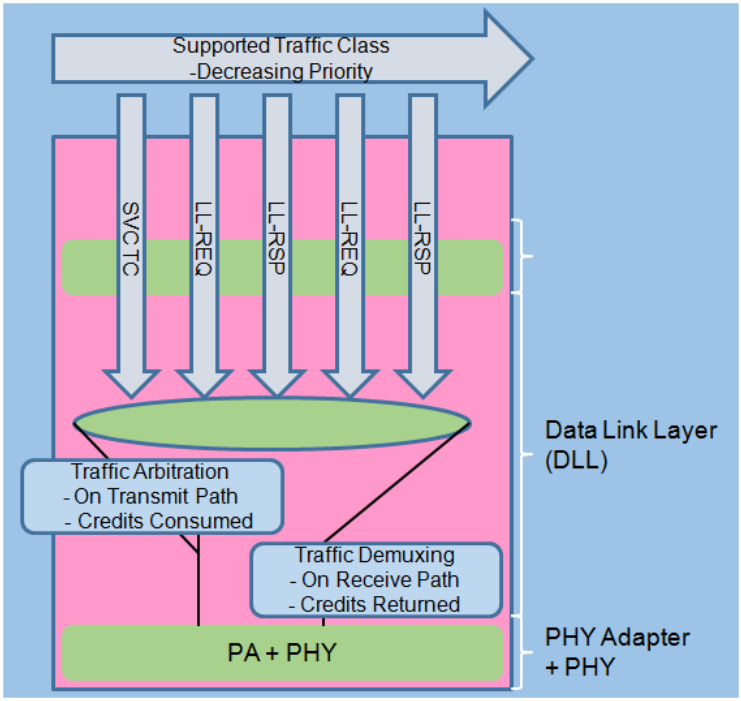

Figure 4.5: LLI Traffic Flow Mechanism. Source: [17]

8. Discussion

To allow DDR chip-to-chip sharing and cache refill operations in mobile phones, and in order to enable manufacturers to remove the modem's DDR and thereby reduce the BoM, MIPI Alliance created the LLI, featuring a low BER, low latency and low power consumption physical layer (the M-PHY), but at the cost of lower efficiency. The summary of the LLI characteristic feature is shown by Table 4.1.

Parameter	Advantage	Consequence
Memory Mapping	Direct access to data (Memory Sharing possibility)	Occupying the CPU bus
Low latency Error Retry Time	Cache refill operation possibility	Low data efficiency (Throughput)
Priority Based QoS	Low latency for interrupt and high priority operations	other operation or devices high priority operations

Table 4.1: LLI Feature Summary

Chapter 5

Implementation Of LLI using Simulink HDL Coder

The Low Latency Interface (LLI) is a point to point mapped interconnect that enables two devices or SoC on two different chips to transfer data between devices. LLI is a low-latency interface technology designed for connecting two devices within a mobile terminal at the interconnect level e.g., OCP, AMBA protocols, using memory-mapped transactions.

The Low Latency Interface primarily targets low-latency cache refill transaction. The LLI specification is a layered protocol, where target access port and initiator access port on two connected chips exchange transactions without software intervention. This configuration thus reduces the latency. The low latency case is assisted by the dedicated Low Latency traffic class (LL TC). Moreover, in order to increase the LLI's throughput, a best effort traffic class (BE TC) is also defined that enables access to memory mapped remote devices without imparting any decline in the performance of low-latency traffic. LLI also provides a set of transaction for transmitting sideband signal between two chips, improving overall communication. LLI leverages the MIPI Alliance M-PHY physical layer [9],it interfaces the PHY adapter layer with the physical medium. The physical layer is not the part of LLI specification. LLI stack layers being layered protocol include three layer's, a PHY adapter layer, data link layer and transaction layer which are discussed in the following:

5.1 PHY Adapter Layer

The PHY Adapter (PA) Layer is responsible for transmitting and receiving frames to and from the remote Data Link Layer using PHY symbols. It implements error protection features and mechanism to save power of the PHY Link, as shown in figure 5.1. It comprises of two main blocks.

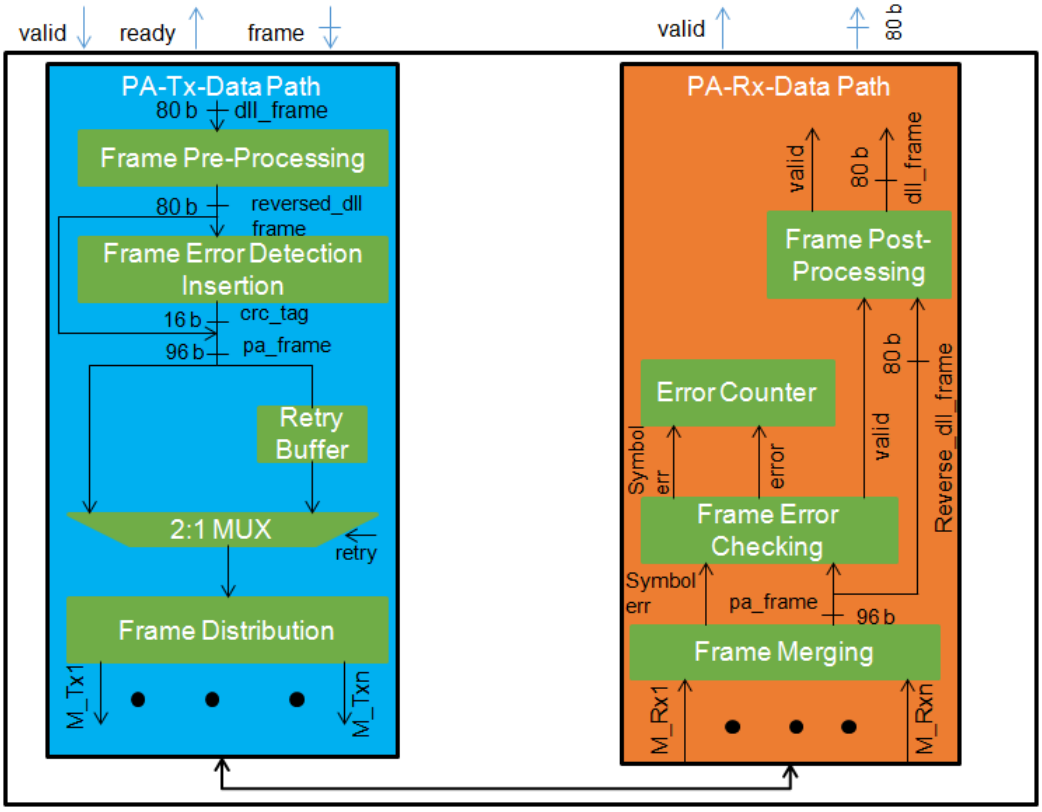

Figure 5.1: PHY Adapter Layer Architecture Model

5.1.1 PA-Tx Data Path

The main function of this block is to process the frames received from DL Layer and transmit them to the remote device through the M-PHY link. This also comprises of the error detector block CRC and Sequence Number (SEQ). This block contains a buffer named Retry buffer which stores the frames sent to the remote device in order to be re-transmitted when the Retry mechanism is started (i.e. error is detected at the remote Rx and NACK is sent for the Retry attempt by the remote side to the local side) on the local transmitter as illustrated by the Simulink based model design in figure 5.2.

1. Frame Pre-processing
   The function of this sub-block is to perform a reversion on frame received from the DL layer as stated by the MIPI LLI standard.

2. Frame Error Detector Insertion
   The function of this sub-block is to add the CRC and the frame Tag to the 10 bytes frame received from the Frame pre-processing sub-block. it incorporates dual strategy of error detector insertion, viz- SEQ frame tag and CRC tag. The SEQ frame tag is an identifier number attached to each PA frame and is used by the PA layer as frame identifier during Retry mechanism. A simple 6-bit counter is used to calculate this Tag. When a new frame is received from the DL layer, the counter is incremented by one. The counter wraps to

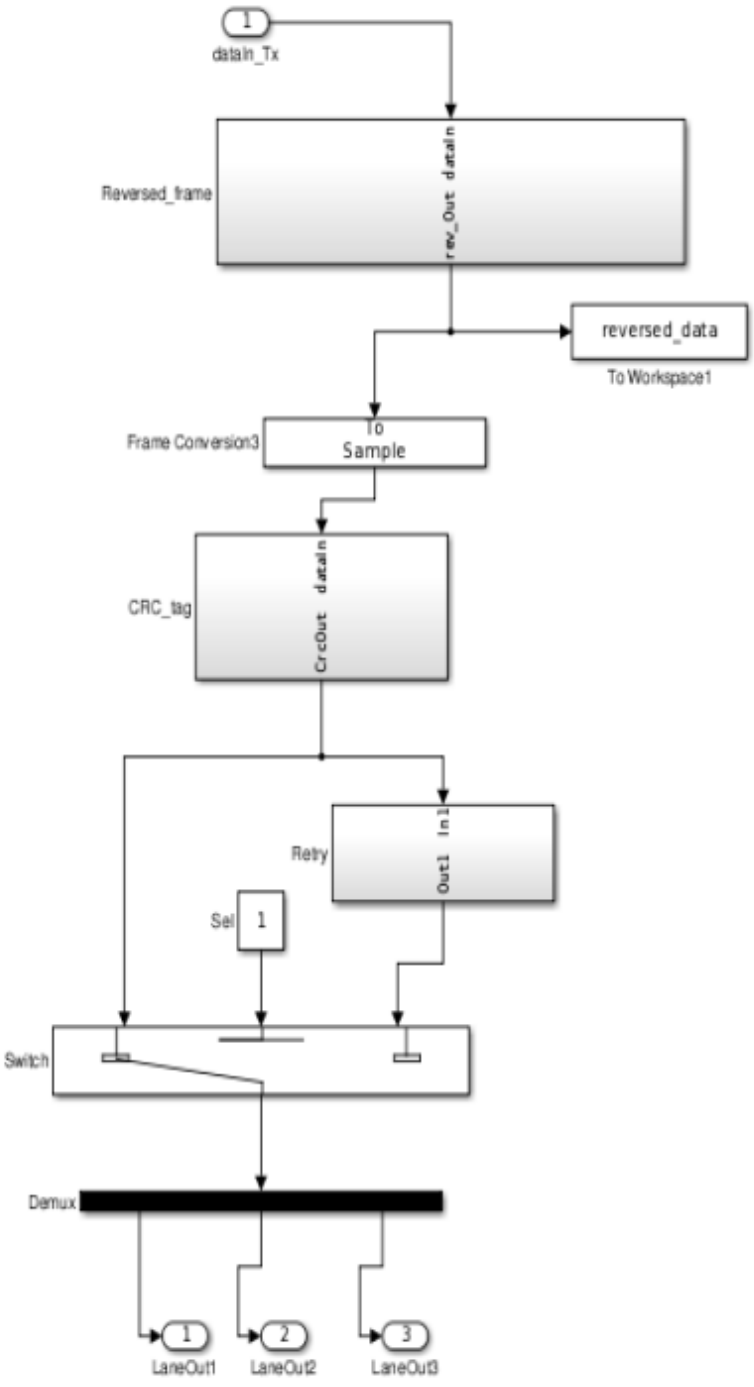

Figure 5.2: Simulink Model of PHY Adapter Transmitter Path

zero when it reaches the maximum value 63. Moreover, The CRC tag is calculated on the payload composed of the 10 bytes of the reversed DL frame and the byte 00 and SEQ. The CRC field that shall be added to the PA frame shall be the inversion (one's complement) version of the computed CRC. The calculation of this CRC shall not exceed one clock cycle.

3. Frame Distribution
   This sub-block distributes the PA frame to the different connected M-TX modules through the M-TX modules data interface. As the number of M-TX modules may be different between various applications, the implementation shall consider the number of M-TX modules as generic parameter which can takes values 1, 2, 3 or 4 lanes.

4. Retry Buffer
   This sub-block is used to store the data frame that are transmitted from the PA- TX to the receiver block. In LLI, only errors are signaled when they occur, i.e. correct reception of a frame is not acknowledged. The PA transmitter attempts retry when an error is detected internally by PA-RX (CRC fail, SEQ incrementing schema or MPHY symbol error).

5.1.2 PA-Rx Data Path

The main function of this block is to process the frames received from the remote device through the M-PHY link and send correct frames to DL layer. This processing consists of merging symbols received from the different connected physical link Rx (i.e. M-PHY Rx) modules to construct the PA frame and checks the correctness of this frame.

When the frame error check fails which means the frame is erroneous, the PA-Rx notifies the PA-TX to enter in Retry mechanism as illustrated by Simulink based block in figure 5.3.

1. Frame Merging
   This sub-block merges the symbols received from the different connected M-RX modules through the M-RX data interface and reconstructs the PA frame which is composed of 12 bytes.

2. Frame Error Checking
   The function of this sub-block is to check the correctness of the received PA frame by computing its CRC to check that the frame is free from error and computing locally its SEQ to check that the received SEQ is the expected one. It also manages the Symbol Error reported by the MPHY.

3. Frame Error Counters
   To check the correctness of the received frame, both the computed CRC and the expected SEQ are compared to the received ones. If they have the same value, the frame is valid and free from errors. Otherwise, the frame is considered as erroneous and the PA enters

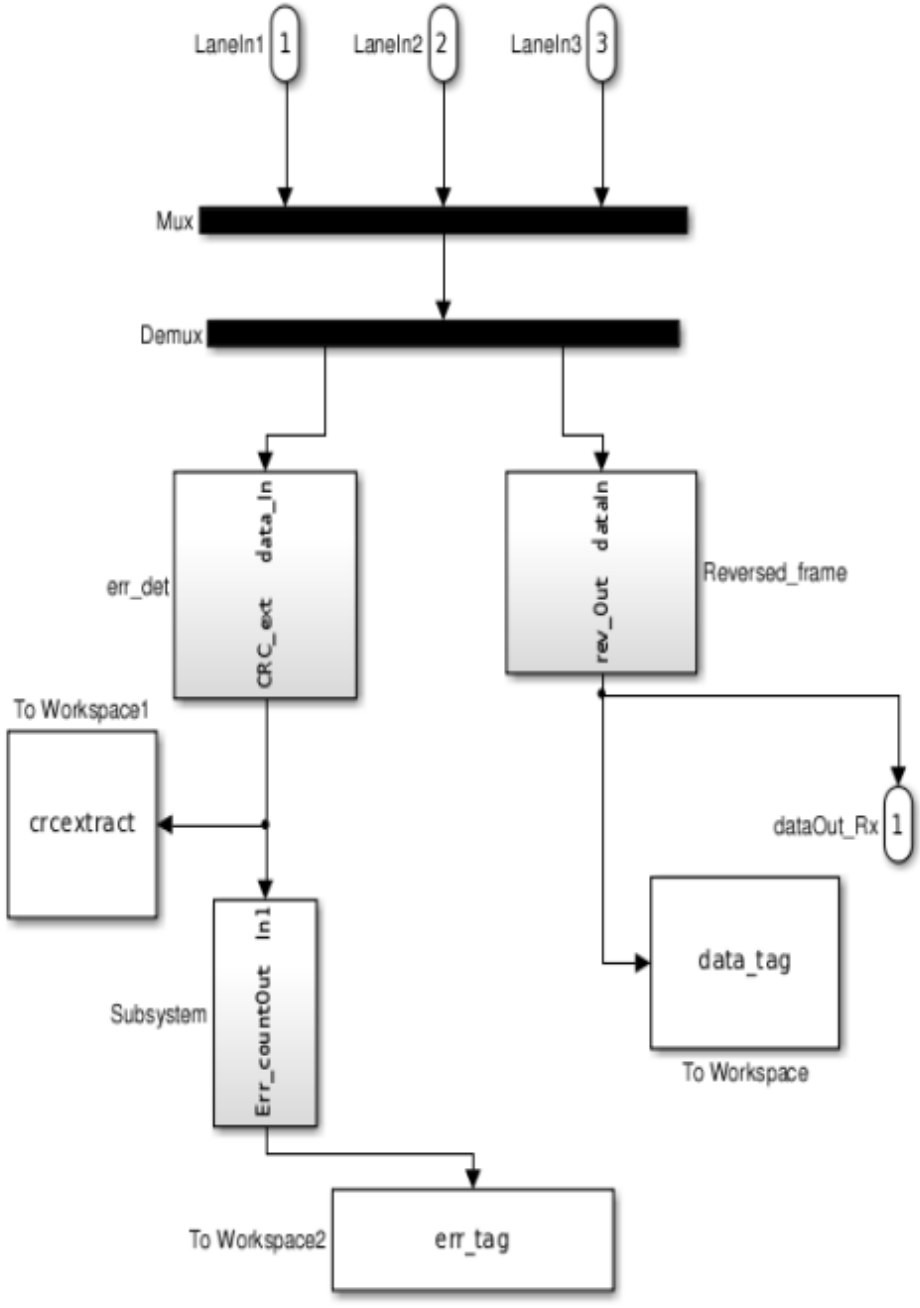

Figure 5.3: Simulink Model of PHY Adapter Receiver Path

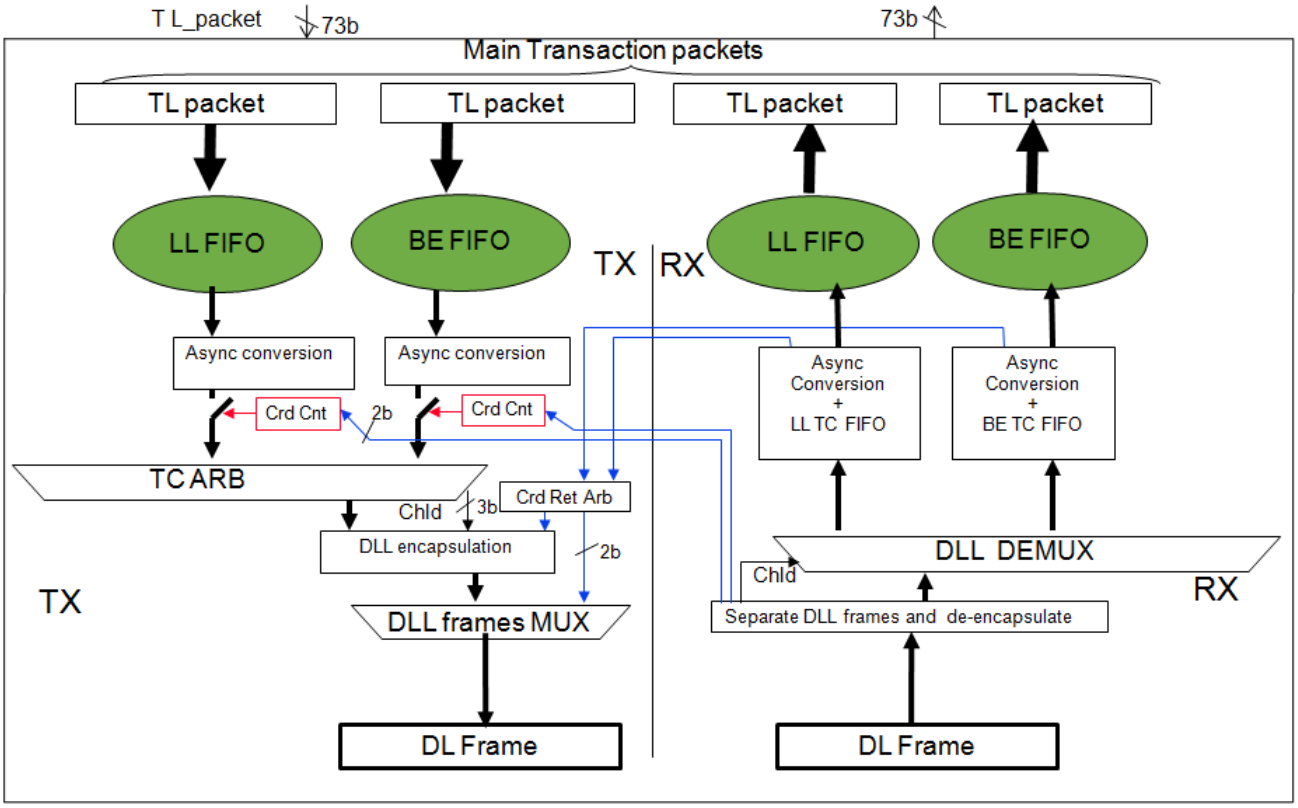

Figure 5.4: Data Link Layer Architecture Model

the Retry state. In case of error, the error notification is also sent to the Error Counters sub-block to increment the PHIT Error Counter value.

4. Frame Post-processing
   The function of this sub-block is to remove the reversion applied on the 10 bytes PA frame by the PA-TX of the remote device and send this frame to DLL.

5.2 Data Link Layer

The Data Link layer is responsible for elaborating frames, and provides flow control to the Transaction Layer. Flow control is necessary since the Transaction Layer is connected to Interconnects that are capable of applying back-pressure to Transactions. Moreover, flow control is independent for the defined traffic classes so that, for example, LL Transactions can proceed while BE Transactions might be blocked by their destinations. Independent flow control is the reason for defining several traffic classes.

Separate DL channels are used by the SVC traffic class (third QoS, as discussed in LLI Overview), enabling Service Transactions to be processed by a LLI stack while the LL and BE Channels are disabled. As a consequence, the Data Link Layer arbitrates between the packets received from the Transaction Layer. The figure 5.4 illustrate the architecture of DLL.

There are two path of Data Link Layer:

5.2.1 DL-Tx Data Path

On the transmit path, the Data Link Layer arbitrates between packets coming from different DL traffic class depending on the priorities of the traffic classes (whether Best Effort or SVC or Low Latency). Channel information, as well as flow control management information (credits), is added to the packets, thereby creating Transaction Frames that are forwarded to the PA Layer. The Simulink model based design of the transmitter path is shown by figure 5.5.

The packet coming from the transaction layer incorporates Channel information, as well as flow control management information (Credits), is added to the Packets, thereby creating Transaction Frames that are forwarded to the PA layer. Packets enter the DL on TX side, in the TL clock domain, and cross the clock boundary to the TX domain. The DL then optionally accumulates packets from each input (store and forward module). The arbitration and frame elaboration stages drives the Credit-based channels packets which arbitrates only after their credit counter is checked to be non-zero. The TL traffic class frame arbiter decides which frame is issued by the DL, and passes the corresponding channel ID to the frame encapsulation function. This function also gets from a Credit Return Channel Arbiter a piggy-back credit return channel and associated number of credits to return, and inserts this information inside the frame. A second stage of frame arbitration, driven by the Credit Return Channel Arbiter, decides if a DL credit frame could preempt the next available frame transmission, in case a remote credit starvation is detected (i.e. many credits to return and/or to many channels).

5.2.2 DL-Rx Data Path

On the receive path, the Data Link Layer receives frames from the PA layer and forwards them to the appropriate DL traffic classes using the Channel Identifier (ChID) field in the frame. The Data Link Layer also manages flow control credits. Figure 5.6 shows the Simulink model of receiver link layer path.

On the receive path, the Data Link Layer receives Frames from the PA layer and forwards them to the DLL using the Channel Identifier field in the Frame. On the RX side, incoming frame is determined by looking up Channel ID, credits (piggy-back or from DL frame) are extracted and returned to the appropriate local credit counter. The Channel ID selects a RX FIFO that will buffer the extracted incoming TL packet. Each time a packet exits a credit-based FIFO and is accepted by the TL, a credit is sent for the corresponding channel to the Credit Return Channel Arbiter, so that it can be later returned to its remote credit counter. The number of entries in the FIFO's is typically a parameter, also controlling the number of credits issued by the TX side after reset.

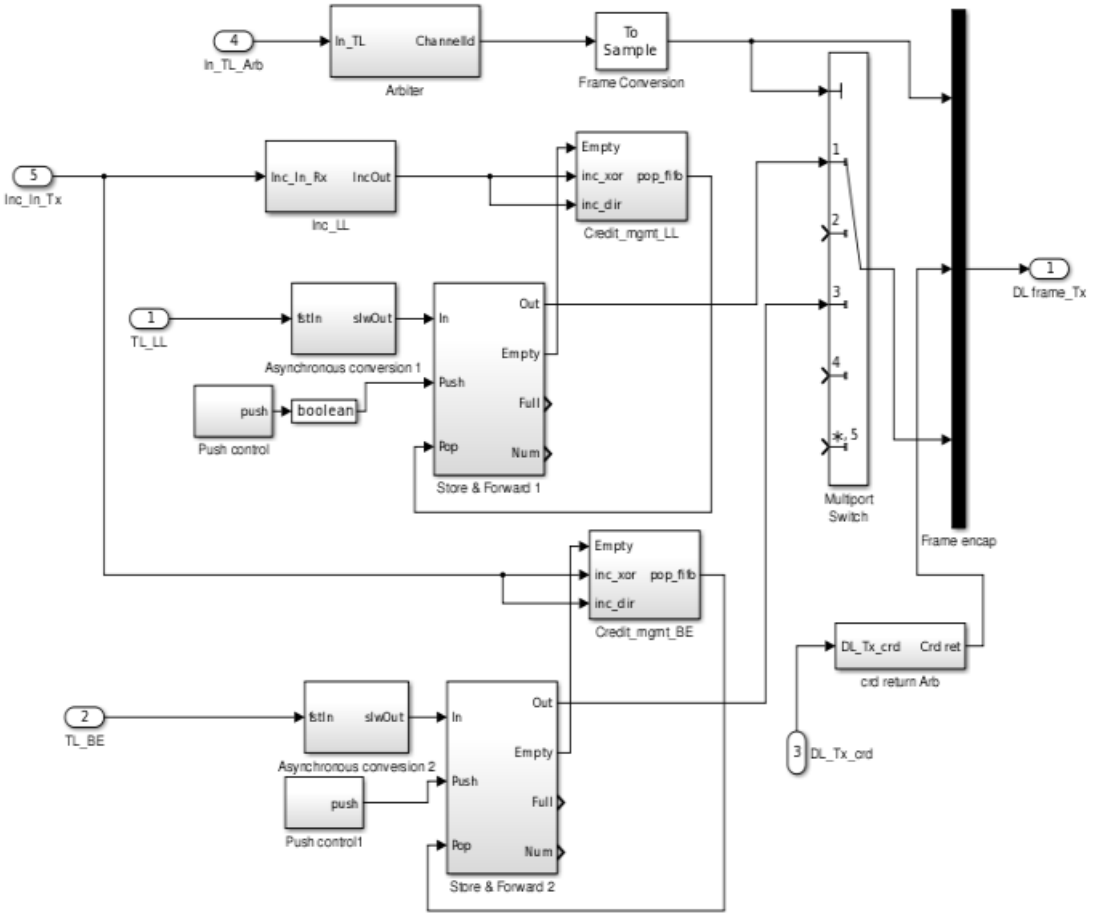

Figure 5.5: Simulink Model of Data Link Transmitter Path

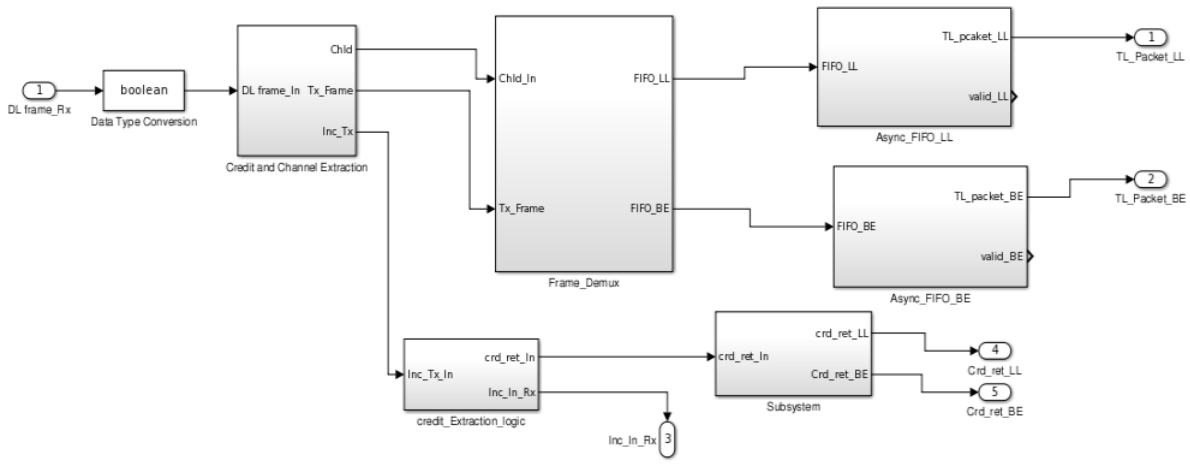

Figure 5.6: Simulink Model of Data Link Receiver Path

5.3 Transaction Layer

The Transaction Layer is the uppermost layer of LLI stack. It provides the interface between the LLI stack and the local chip. This interface execute transactions whenever it gets access request from local interconnect and provides responses to the local interconnect using the Service Primitives (SVC). It act as the wrapper between the bus based protocol (i.e. AXI, AMBA, OPC) and the LLI protocol which is primarily responsible for converting interconnect based protocol to LLI protocol , which can be transported to the Data Link Layer.

Chapter 6

Benchmarking Result

Model Based Design process utilizes a system model as an executable specification throughout design development trajectory. This design technique not only accelerates product development cycle but also offers choice of optimal design alternatives and trade-offs compared to traditional manual prototype based design methodologies, thus enabling to meet predefined performance criteria.

In this section the automatic generated HDL code from the Simulink HDL coder to that of manual hand written code is compared. The comparison targets the time to market, area, power and timing constrains for the Data Link Layer (DLL) of MIPI LLI for both the procedures. The automatic generated code in Verilog from the Simulink Model based design which uses MATLAB R2013a, and the manual hand-written Verilog code for the Data link layer is synthesized for CMOS 45 nm standard cell ASIC technology. Post synthesis result for area, power and timing are presented and the trade-offs are discussed and analyzed.

6.1 Time to Market

MBD, used in conjunction with simulation tools, can lead to accelerated prototyping, software testing and FPGA-in-loop co-simulation. Simulation enables specification variations and architecture modeling errors to be found immediately, early in the design trajectory. Automatic code generation using this approach eliminates the manual steps in implementing the same algorithm to run on the hardware platform. This technique streamlines the design process, lessens errors in hardware design implementation, and reduces the overall time to market.

Adaptation of Simulink Model Based Design methodology for designing and implementation of Data Link Layer of LLI, shorten development time by more than 80%, reduces verification time by more than 60% and creates FPGA prototypes 50% faster than before as show in figure 6.1. It is evident from the figure that development time of Model Based Design flow involves two steps. First step is model based executable specification modeling followed by HDL translation. Unlike the manual work flow which involves textual specification, followed by rigorous and time

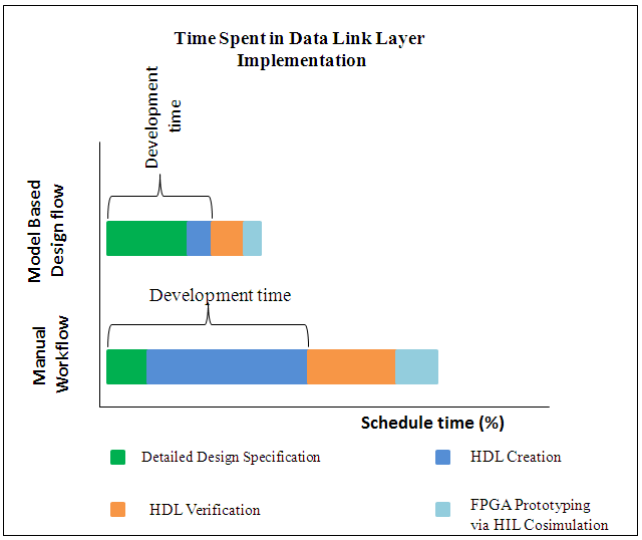

Figure 6.1: Comparison of Model Based Design flow and manual workflow timelines for DLL

Area (μm^2)	MBD Automatic Generated Methodology	Manual Hand-written Methodology	Change Comparison
Data Link Layer Tx	11024.94	7950.53	27.87%
Data Link Layer Rx	9845.25	8824.94	10.36%

Table 6.1: Area Comparison Result

taking hand-written HDL creation, which increases the development time.

6.2 Implementation Area Trade-off

The Data Link Layer of LLI is implemented and synthesized with 45 nm CMOS standard cell library for both the automatic generated code using HDL coder as well as manually hand-written code. The resulting area obtained is shown in table 6.1.

It reveals that the overall required area for the automatic generated code is 24% more than that of the manual hand-written code. This area overhead may have occurred due the reason that usage of multiple data conversion blocks in Simulink model design for converting the floating point data type to fixed point. Moreover the Simulink lookup table blocks used in the model design of DLL Tx also seem to add considerable area overhead compared to manual hand-written design.

6.3 Power Trade-off

In this subsection power comparison is done between the code generated using Model Based Design approach to the hand written code for the DLL, and the resulting power obtained is shown in table 6.2.

Parameter	Power (μW)	MBD Automatic Generated Methodology	Manual Hand-written Methodology	Change Comparison
DLL Tx	Internal	1035.56	721.57	30.32%
	Switching	61.34	49.28	19.63%
	Leakage	29.68	24.13	18.69%
	Total	1126.58	794.98	29.43%
DLL Rx	Internal	932.65	798.68	14.29%
	Switching	69.43	56.11	19.18%
	Leakage	31.56	26.29	16.69%
	Total	1033.64	881.10	14.75%

Table 6.2: Power Comparison Result

Clock Type MHz	MBD Automatic Generated Methodology	Manual Hand-written Methodology	Change Comparison
RClk	137 MHz	153 MHz	10.45%
WClk	334 MHz	402 MHz	16.91%

Table 6.3: Timing Comparison Result for DLL Transmitter Path

The dynamic power comprises of switching power and internal power. Switching power is dissipated when charging and discharging the load capacitance at the cell output. The load capacitance is composed of interconnect (net) capacitance and gate capacitances the net is connected. The amount of switching power depends on the switching activity (is related to the operating frequency) of the cell. The more there are logic transitions on the cell output, the more switching power increases. Internal power is consumed within a cell for charging and discharging internal cell capacitances. Internal power also includes short-circuit power. when transistors switch, both NMOS and PMOS networks may be momentarily ON at once, this leads to a small short circuit current. Whereas, Leakage power is the power dissipation primarily due the subthreshold current in the transistor when it is turned off. As the subthreshold current is exponentially dependent on threshold voltage, so the leakage power is drastically influenced by the threshold voltage.

It is evident from table 6.2 that the power consumption of automatic generated code is 28.88 % higher than that of manual hand-written code. This increase in power is due to the inclusion of extra data type conversion block in the Simulink model as well as due to fact that asynchronous FIFO is not optimized in terms of power, which increases the power consumption.

6.4 Timing Constrain Trade-off

In this subsection timing constrain of the MBD automatic generated code and manual hand written code is compared. The DLL design involves two clock, read clock (RClk) and write clock (WClk). The table 6.3 and table 6.4 illustrate the maximum frequency offered by both read and write clocks for DLL transmitter path and receiver path respectively.

Clock Type MHz	MBD Automatic Generated Methodology	Manual Hand-written Methodology	Change Comparison
RClk	235 MHz	301 MHz	21.92%
WClk	119 MHz	144 MHz	17.36%

Table 6.4: Timing Comparison Result for DLL Receiver Path

Thus it can easily be deduced from table 6.3, that the write clock for manual hand written procedure is almost 17% faster than its counterpart. Nevertheless the read clock shows 10.45% difference between both the different approaches. Undoubtedly manual workflow has significantly better operating speed compared to Model Based Design methodology. Later, table 6.4 reveals that the write clock for both the procedures considered in the paper has notable difference of 17.36%. Furthermore read clock for the receiver shows the difference of almost 22% for both the different modeling approaches. So in nutshell operating frequency of the automatic generated code is significantly decreased compared to its manual counterpart which may be very critical for the design's having tight timing constrains.

Chapter 7

Limitations of Model Based Design Methodology

The ability to be able to perform rapid prototyping and thus helps to accelerate the product development cycle, Simulink HDL coder plays a crucial role in achieving the objective. However, Model Based Design methodology has some limitations associated with it which prevent it from achieving rapid prototyping as well as fails to provide complete automation to the product design trajectory, which is one of the essential factor responsible for the delayed time to market. The problem associated with the MBD methodologies as discussed as follows:

1. The target library for the HDL supported Simulink block set are not exhaustive and rich enough as it lacks even the elementary blocks such as: 8b10b encoding, BCH encoder etc.
2. The support for the customizable external HDL code is also not available.
3. Another problem with Simulink HDL Coder is the choice of block implementations because of the characteristics inherent by the blocks/model in the library. For example: A block in a subsystem requires Boolean or fixed data type as the input while the output of the block previous to it delivers double data type as the output thus this requires the data conversion block in between which results in increase of area and power as well as delays the critical path of the design.
4. The other limitation associated with this technique is that, it did not provide any synchronization or built-in handshaking mechanism which could ensure that the previous blocks is done with computation and the next are ready to input data. So this imparts latency in the design and the architecture design such MIPI LLI which has tight latency and timing constrain this modeling approach is not productive enough.
5. There are also more tenuous failings difficult to analyze or describe for the HDL generation in Simulink HDL coder such as the inability to use the lookup table.

The above mentioned limitation of MBD methodology by Simulink HDL coder reveals that, however Simulink HDL coder is of great help, as it minimizes the amount of time spent in manually writing the HDL code by automatically generating the same. Moreover automatic generation of test benches saves the time from redundant test benches that need to be written. However, there are some restrictions to this, as the ability to use hand-written HDL code is still required. The following section discusses the limitation of the MBD technique with a test case of single cycle latency CRC checksum architecture.

7.1 A Test Case: Single Cycle Latency CRC Checksum Algorithm

7.1.1 Introduction

To discuss the limitation offers by the Simulink Executable Model Based Design technique a test case has been taken, single cycle latency CRC checksum algorithm. In the case study of LLI modeling we came across the case where it is required to calculate the CRC-8 for incoming data frame of width 80 bits in one clock cycle. Since the Simulink HDL coder library has CRC block available but it imparts the latency of 13 clock cycles to calculate the checksum of CRC-8 for the given 80 bit data frame, therefore this fails to fulfill the requirement. Thus, necessitating the requirement of single cycle latency CRC algorithm model. In order to achieve the requirement or fulfill the purpose of single latency CRC computation.

7.1.2 Proposed CRC Algorithm

Background

CRC is a polynomial-based division method which computes position-dependent checksums for detecting multi-bit errors in data frames. For each frame transmitted through a medium that may introduce errors, a checksum is computed and appended to the end of the frame. This checksum also called Frame Check Sequence (FCS), is computed as a remainder of a polynomial division of the data frame by the CRC generator polynomial. Consider an m bit wide data frame D which can be represented as a polynomial $D(x)$ of degree $m-1$ whose coefficients are the bit values of D.

$$D(x) = D_{m-1}x^{m-1} + D_{m-2}x^{m-2} + \cdots + D_1x + D_0 \tag{7.1}$$

Further consider generator polynomial $G(x)$ of degree n such that

$$G(x) = G_nx^n + G_{n-1}x^{n-1} + \cdots + G_1x + G_0 \tag{7.2}$$

With $G_n = 1$ and $G_i \in \{0, 1\}$ with $0 \leq i \leq n-1$.

With equations 7.1 and 7.2 the Frame Check Sequence $FCS(x)$ for a data frame $D(x)$ using a CRC generator polynomial $G(x)$ of degree n can be computed by using polynomial arithmetic in the Galois field of two elements, or $GF(2)$ as given by equation 7.3:

$$FCS(x) = D(x)x^n \; mod \, G(x) \tag{7.3}$$

Consider D as a row matrix of size $1 \times m$ representing coefficients of input data frames.

$$D = \begin{bmatrix} D_{m-1} & D_{m-2} & \dots & D_1 & D_0 \end{bmatrix}_{1\times m}$$

Using property of Identity matrix I, D can be expressed as below :

$$D = D \times I \tag{7.4}$$

With,

$$I = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}_{m\times m}$$

Identity matrix $I_{m\times m}$ is a group of m one-hot encoded data frames.

In $GF(2)$ the addition and multiplication operation corresponds to logical XOR and AND operation respectively. Moreover CRC is a linear function, hence it satisfies the following properties:

Property 1. Linear function over addition

$$CRC[A(x) + B(x)] = CRC(A(x)) + CRC(B(x)) \tag{7.5}$$

with, $A(x)$ and $B(x)$ being data frame polynomials.

Property 2. Linear function over scalar multiplication

$$CRC[AB(x)] = A \, CRC(B(x)) \tag{7.6}$$

With A being scalar number and $B(x)$ being data frame polynomial.

The proposed architecture exploits the following theorem:

Theorem 1:

$$CRC(D) = D_{m-1}CRC(I_1) \; + D_{m-2}CRC(I_2) \; + \dots \; + D_0 CRC(I_m) \tag{7.7}$$

Proof:

$$\begin{aligned} CRC(D) &= CRC([D_{m-1}\ D_{m-2}\ \dots\ D_1\ D_0]_{1\times m}) \\ &= CRC(D \times I) \\ &= CRC[\{D_{m-1}I_1\ + D_{m-2}I_2\ + \cdots +\ D_1I_{m-1} +\ D_0I_m\}] \end{aligned}$$

where I_1,I_2 $\dots I_m$ are the subset of Identity matrix I. with,

$$I_1 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix}_{1\times m}$$

$$I_2 = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}_{1\times m}$$

$$I_m = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \end{bmatrix}_{1\times m}$$

From property 1,

$$CRC(D) = CRC(D_{m-1}I_1)\ + CRC(D_{m-2}I_2)\ + \dots\ + CRC(D_0I_m)$$

Now, from property 2,

$$CRC(D) = D_{m-1}CRC(I_1)\ + D_{m-2}CRC(I_2)\ + \dots\ + D_0CRC(I_m)$$

From Theorem 1, it can be inferred that to compute CRC of any data frame of width m, it is not required to store CRC values of all possible combinations of data frame with width m i.e. 2^m. By only knowing CRC of m one-hot encoded data frames, CRC of any m bit wide data frame can be easily computed.

Single Cycle CRC Algorithm

Traditional parallel implementation of CRC computation uses LUT based approach in which CRCs of all possible combinations of data frames is stored. For computing $CRC-8$ for data frame of width 16, the LUT needs to store $2^{16} \times 8$ bits. Increase in data frame width becomes a bottleneck for such CRC computation approaches. Slicing techniques are used in which data frame is split into smaller blocks of width 8. So instead of single LUT, multiple LUTs are used for CRC computation. However, the implementation cost of such algorithm is very high.

This dissertation discusses a novel CRC algorithm based on the mathematical analysis done in the previous subsection.

Theorem 1 forms the main pillar of the proposed algorithm which implies that, to compute

	One-Hot Encoded Data Frame	CRC-8 value
I_1	0000000000000001	0x07
I_2	0000000000000010	0x0E
I_3	0000000000000100	0x1C
I_4	0000000000001000	0x38
I_5	0000000000010000	0x70
I_6	0000000000100000	0xE0
I_7	0000000001000000	0xC7
I_8	0000000010000000	0x89
I_9	0000000100000000	0x15
I_{10}	0000001000000000	0x2A
I_{11}	0000010000000000	0x54
I_{12}	0000100000000000	0xA8
I_{13}	0001000000000000	0x57
I_{14}	0010000000000000	0xAE
I_{15}	0100000000000000	0x5B
I_{16}	1000000000000000	0xB6

Figure 7.1: A sample of a 16×8 LUT

$CRC - n$ of m bit wide data frame, it is only required to know $CRC - n$ of only m one-hot encoded data frames ordered from $CRC[I_1]$ to $CRC[I_n]$. Thus, this algorithm requires just one LUT of size $m \times n$ to store $CRC - n$ of m one-hot encoded data frames. Figure 7.1 shows a LUT for $CRC - 8$ computation of 16 bit wide data frame as an example. Now to compute $CRC - n$ of m bit wide data frame, $CRC - n$ values of m one-hot encoded data frames stored in LUT are multiplied by corresponding bit of m-bit data frame and the multiplied values are XORed to get the result. This is implemented by using logic array $LA1$ of AND and XOR gates to perform multiplication and XOR operations respectively as shown in Figure 7.2. The algorithm presented uses CRC properties which reduce the area drastically. Typically, using a traditional LUT based approach, the size of LUT for 10 bit wide data frame for $CRC - 8$ computation is $2^{10} \times 8 = 1024$ Bytes, whereas using the proposed algorithm it is only $10 \times 8 = 10$ Bytes. As the width of data frame increases size of LUT for conventional Look-up-Table based architectures increases exponentially unlike the proposed algorithm which scales linearly. The complete architecture of the single cycle latency CRC computation algorithm is shown by figure 7.3.

7.1.3 Implementation Result

The proposed algorithm has been manually hand-written code described in Verilog and implemented for $CRC - 8$ over 64, 128, 256 and 512 bit wide input message. Latency, area in terms of resource utilization, operating frequency and throughput for the corresponding data frame input has been performed and analyzed using Xilinx ISE on Virtex-6 xc6vlx760-11760 FPGA. The result obtained is shown in table 7.1.

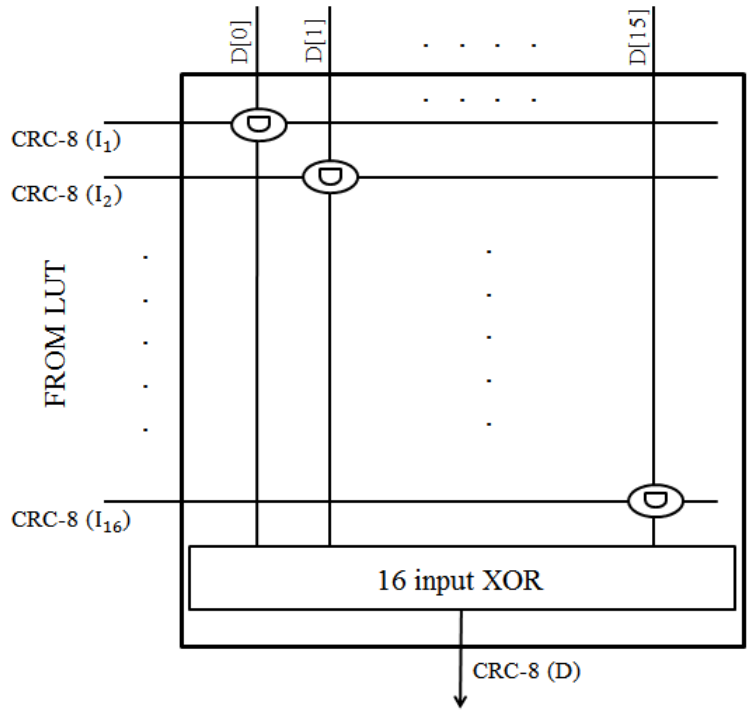

Figure 7.2: A sample Logic Array $LA1$ architecture for $CRC-8$ computation of 16 bit wide data

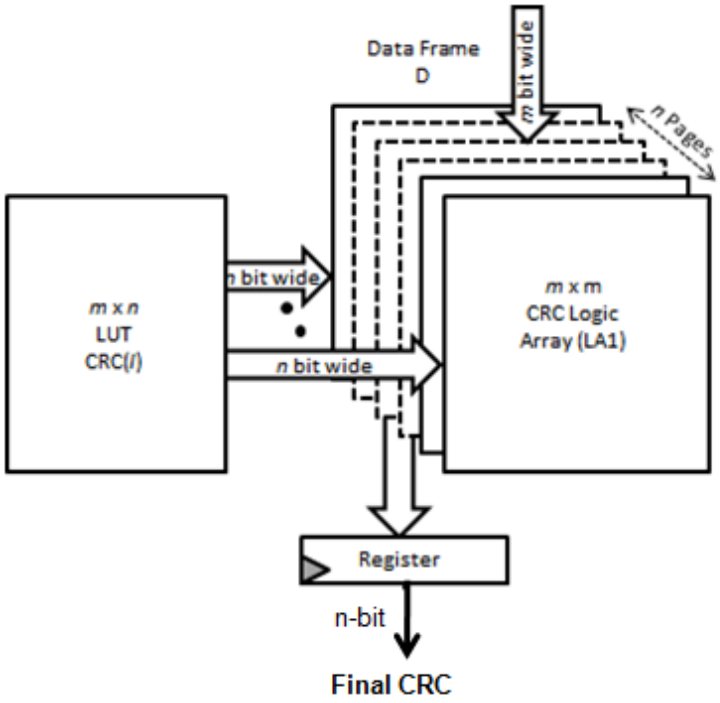

Figure 7.3: Complete Design Architecture of Proposed Algorithm

Algorithm	Datawidth (bits)	Latency (cycles)	Area (LUTs)	Frequency (MHz)	Throughput (Gbps)
Proposed Algorithm	64	1	65	552	35.3
	128		107	404	51.7
	256		209	363	92.9
	512		431	328	167.9

Table 7.1: Implementation Result for Vertix-6 FPGA

7.2 Simulink Model for the Proposed CRC Algorithm

The Simulink Model Based Design for the proposed algorithm is difficult to model, as the proposed algorithm requires a Look up Table (LUT) for storage of one-hot encoded weights. The Simulink library for the look up table block has limited capability, so it fails to accommodate 8 bit wide 80 one-hot encoded weights, as it is the essential requirement for computing the CRC-8 for 80 bit wide data frame. Similarly it is difficult to model Logic Array (LA1) due the reason that library lack multiple input XOR gates and using the traditional input XOR gate will increases the Level of the design and thus adds to exponentially increasing delay in the critical path, which will fail to provide substantial throughput and operating frequency. Moreover Simulink HDL coder lacks the built-in handshaking or synchronization mechanism to tell that when the block Logic Array (LA1) performs CRC computation and ready to take next input data thus this often imparts combinatorial delay. Later, the translation of the hand written code to Simulink block set is also not feasible because Simulink HDL coder lacks support for customizable external HDL code.

Chapter 8

Conclusion

As the market for mobile communication system matures equipment vendors are under increasing pressure to provide low cost solutions and that too on time. With today's complex and rapidly evolving mobile technologies, raises a set of challenges in their development. However, these evolving mobile technologies impart extra functionalities to be taken care of in the design implementation which enlarges the architecture modeling, RTL development and simulation time thus consequently increases the time to market. This thesis work presents a solution for improving the prototyping time by using Model Based Design approach and discuss their trade-offs. Adoption of Model Based Design helps us to develop a high level abstraction that automatically facilitates efficient FPGA implementation. Moreover, FPGA in loop capability enables co-simulation of FPGA implementations within Simulink. This capability provides a straightforward and reliable method to verify hardware implementation, accelerate simulations and save prototyping time.

The result obtained after the analysis of MBD automatic generated code to that of manual hand-written code for MIPI LLI Data Link Layer reveals that the time to market can be improved by more than twice for the Model Based Design approach whereas this will impart an area overhead of almost 25%. Moreover power consumption of the former methodology is raised by almost 29% compared to latter methodology. Timing constrains for both read (RClk)and write (WClk) clock have crucial variation of almost 11% to 20% and almost 17% respectively for both the different modeling approaches. So it might be unfavorable for the design's targeting tight timing and frequency constrains. So this trade off with respect to time to market gain, power and area overhead unwrap the fact that designs targeting low implementation area and power constrains and at the same time not exposed to high pressure of time to market should tend to follow traditional manual hand-written methodology. However, those designs which are not constrained to tight area and power requirement but exposed to high pressure of time to market requirement should follow the Model Based Design methodology in their design process.

Bibliography

[1] Push-button Performance using Sytem Generator for DSP. Xilinx Inc. Available at: http://www.xilinx.com/products/software/sysgen/sysgen_sellsheet.pdf.

[2] Simulink Documentation. Mathworks Inc. Available at: http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/index.html?/access/helpdesk/help/toolbox/simulink/.

[3] System Generator User Guide. Xilinx Inc. Available at: http://www.xilinx.com/support/documentation/swmanuals/sysgenbklist.pdf.

[4] Put Hardware in the Loop with Xilinx System Generator for DSP. Xilinx Inc. Available at: http://www.xilinx.com/publications/xcellonline/xcell47/xcpdf/xcsysgen47.pdf,XcellJournalArchives,2003.

[5] AccelDSP Synthesis Tool, User Guide. Xilinx Inc. Available at: http://www.xilinx.com/support/swmanuals/acceldspuser.pdf,2008.

[6] Accelware reference designs, user guide. Xilinx Inc. Available at: http://www.xilinx.com/support/swmanuals/accelwareuser.pdf,2008.

[7] MATLAB for Synthesis, Style Guide. Xilinx Inc. Available at: http://www.xilinx.com/support/swmanuals/acceldspstyle.pdf,2008.

[8] HDL Coder and Verifier: High-Level Synthesis Expands to MATLAB Users. Inside DSP Article, MathWorks. Available at: http://www.bdti.com/InsideDSP/2012/09/05/MathWorks.

[9] Low Latency Interface 2 (LLI-2). In *MIPI Alliance Specification* (August 2012), vol. 0.1, MIPI Alliance.

[10] Hardware in the Loop from the MATLAB/Simulink Environment, Sept. 2013. Available at: http://www.altera.com/literature/wp/wp-01208-hardware-in-the-loop.pdf.

[11] National Aerospace Laboratories Proves Benefits of Model-Based Design for DO-178B Flight Software Development. In *Mathworks Article* (2013), Mathswork Inc.

[12] Simulink HDL Coder. Mathworks, Inc. Available at: http://www.mathworks.in/help/pdf_doc/hdlcoder/hdlcoder_ug.pdf.

[13] Simulink HDL Verifier. Mathworks, Inc. Available at: http://www.mathworks.in/help/pdf_doc/hdlverifier/hdlv_ug_book.pdf.

[14] Executable specification for system design. Mathworks Inc. Available at: http://www.mathworks.in/help/simrf/examples/executable-specification-for-system-design.html.

[15] Beek, S., and Sharma, S. Best Practices for FPGA Prototyping of MATLAB and Simulink Algorithms. In *Electronic Engineering Journal* (August 2011).

[16] Brown, S., and Vranesic, Z. Fundamentals of Digital Logic with VHDL Design.

[17] Goel, P.

[18] Lindberg, K., and Nissbrandt, K. An evaluation of methods for fpga implementation from a matlab description. Masters Degree Project, Stockholm, Sweden.

[19] Maxfield, C. The Design Warrior's Guide to FPGAs. In *Elsevier Ltd.,Oxford* (2004).

[20] MediaTek. Automatic Hardware Implementation of Digital Filters for an Audio Codec. Mathworks Newsletter. Available at: http://www.mathworks.in/company/newsletters/articles/automatic-hardware-implementation-of-digital-filters-for-an-audio-codec.html.

[21] Narinder, L. FPGA-Based Wireless System Design. Mathworks Newsletter. Available at: http://www.mathworks.in/company/newsletters/articles/fpga-based-wireless-system-design.html.

[22] Roth, C., and John, L. Digital Systems Design using VHDL. In *CL-Engineering* (2007).

[23] Saade, J., Pétrot, F., Picco, A., Huloux, J., and Goulahsen, A. A system-level overview and comparison of three High-Speed Serial Links: USB 3.0, PCI Express 2.0 and LLI 1.0. In *IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems* (April 2013), pp. 147–152.

[24] Salcic, Z., and Smailagic, A. Digital systems design and prototyping using field programmable logic. In *Springer* (Oct. 2000).

[25] Schubert, P., Vitkin, L., and Winters, F. Executable Specs: What Makes One, and How are They Used? SAE TECHNICAL PAPER.

[26] Smith, P., Prabhu, S., and Friedman, J. Best Practices for Establishing a Model-Based Design Culture. Mathworks Inc.

[27] Specification, M. A. Application note for IPC. vol. 0.9. Dec 2013.