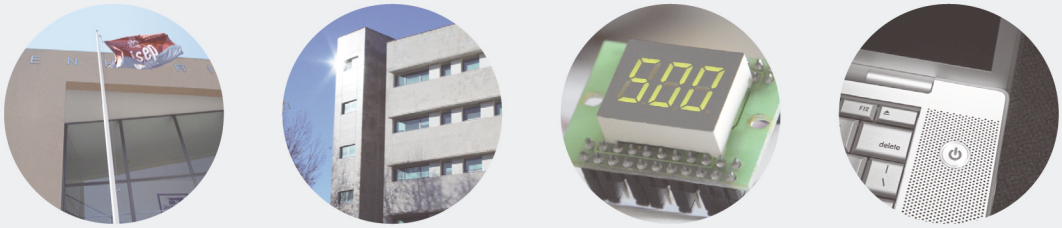
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



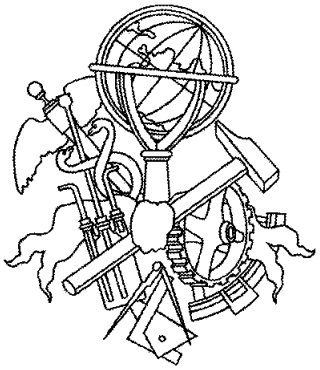
MIPI Display Serial Interface Interoperability Tests

PEDRO FILIPE XAVIER RODRIGUES
Julho de 2016

POLITÉCNICO
DO PORTO

MIPI Display Serial Interface
interoperability tests

Pedro Filipe Xavier Rodrigues
1100466



Tese/Dissertação

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2016

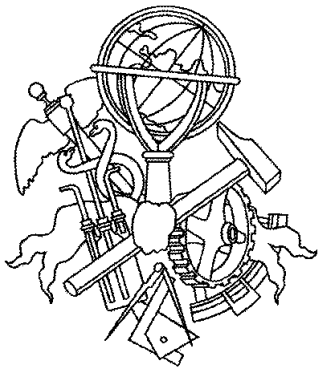
This paper satisfies the requirements present on the Subject Sheet of Tese/Dissertação, of the 2nd year of Electrical and Computer Engineering Master Degree

Candidate: Pedro Filipe Xavier Rodrigues, Nº 1100466, 1100466@isep.ipp.pt

Scientific Orientation: Professor Manuel Gericota, mgg@isep.ipp.pt

Company: Synopsys

Supervision: Eng. António Costa, antonio.costa@synopsys.com



Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

July 6, 2016

Acknowledgments

Para esta tese, e ao longo da minha jornada académica, muitas foram as pessoas que me ajudaram e contribuiram para o meu trabalho e para melhorar a minha pessoa. Por isso, quero, desde já, agradecer a cada uma das pessoas que esteve presente ao longo deste percurso e que de uma forma ou de outra deixaram a sua marca.

Às pessoas que me orientaram, gostaria de agradecer em primeiro ao professor Manuel Gericota, não só pelo tempo despendido com a tese mas também pelos seus ensinamentos nas aulas da unidade curricular PHADI e pelo à-vontade e disponibilidade que sempre demonstrou para comigo. Aos engenheiros António Costa e Rui Ferreira, da Synopsys, que me orientaram durante o meu estágio, estiveram sempre disponíveis para me ajudar e que acreditaram no meu trabalho, o meu muito obrigado.

Quero também agradecer a toda a equipa dos prototyping kits, que me acolheram e me ajudaram em todos os momentos. Um especial obrigado ao Pedro Ricardo Miguel, o meu "*buddy*" dentro da empresa, que esteve sempre presente para me tirar dúvidas, aconselhar e orientar. Ao Terra, Andreia, Silvio e Sá um agradecimento também pelos momentos de descontração no trabalho. Ao Ramiro, Luís, Filipe, João Pinto, Miguel Abreu, Ana, Nelson, Medeiros e Pedro Moreira do *software* e automação o meu muito obrigado também por todos os momentos e por toda a ajuda!

Agradeço aos meus pais e irmão que sempre estiveram presentes em todos os momentos, bons e menos bons da minha vida. Foram incansáveis neste percurso sempre com a preocupação de me perguntarem, diariamente, desde o primeiro ano de escolaridade, "Como é que correu a escola?". Agradeço-lhes muito todo o esforço, não só financeiro, de me manterem ao longo destes anos a estudar e sempre apoiando que o continuasse a fazer.

À Ana que me ajudou muito ao longo do meu percurso académico, que sempre se preocupou em apoiar-me nas minhas decisões e me encorajou

sempre a pensar mais alto, o meu muito e sincero obrigado.

Um agradecimento especial também aos meus amigos Orlando, Cavadas, Esteves, Vítor, Eunice, João, Luísa, Ruh e Flávio que estiveram presentes neste percurso desde o primeiro ano.

Abstract

The increasing demand of mobile devices, with better and faster peripherals, leads to a need of developing new products based on the most recent protocols by the silicon industry companies.

One of the companies with a larger role in the silicon industry is Synopsys. Synopsys is one of the biggest IP producers (Intellectual Property) in the world, offering its clients a large set of components based on the leading protocols on the market such as: USB, MIPI, HDMI, DDR, SATA, Ethernet, PCIe etc.

In this thesis framework were developed two projects regarding MIPI DSI (Display Serial Interface) specification in which is proposed to use Synopsys DSI Host controller to validate the behaviour of two physical layers (PHY). One of the projects has as target test one of Synopsys new PHYs in order to validate its performance on a protocolar level. The other project refers to the protocolar validation of a client PHY that ordered Synopsys services to test it.

To validate both physical layers and accomplish the project goals, both projects passed through the stages of: design development, simulations, Synthesis + P&R (Place and Route) and hardware tests.

Keywords

MIPI, DSI, hardware, validation, PHY, Synopsys

This page was intentionaly left in blank.

Resumo

O aumento da procura de dispositivos móveis com periféricos melhores e mais rápidos, leva à necessidade de se desenvolverem novos produtos baseados nos mais recentes protocolos por parte das companhias tecnológicas.

Uma das empresas com maior papel na industria do silício é a Synopsys. A Synopsys é uma das maiores produtoras de IP (do anglo-saxónico *Intellectual Property*, propriedade intelectual em português) oferecendo aos seus clientes uma enorme gama de componentes baseados nos protocolos líderes de mercado tal como: USB, MIPI, HDMI, DDR, SATA, Ethernet, PCIe entre outros.

No enquadramento desta dissertação surgem dois projectos, ligados à especificação MIPI DSI (Display Serial Interface) nos quais é proposto utilizar o controlador DSI Host, desenvolvído pela Synopsys, para validar o funcionamento de duas camadas físicas (PHY). Um dos projectos tem como alvo testar um dos novos PHY desenvolvidos na Synopsys de forma a validar o seu comportamento e desempenho a nível protocolar. O outro projecto remete a uma validação protocolar de um PHY desenvolvido por um cliente que requisitou os serviços da Synopsys para o testar.

Para efectuar esta validação, ambos os projectos passam pelas fases de: desenvolvimento do projecto, simulações, Síntese + P&R (*Place and Route*) e testes em hardware.

Palavras-chave

MIPI, DSI, hardware, validação, PHY, Synopsys

This page was intentionaly left in blank.

Contents

This page was intentionaly left in blank.

List of Figures

This page was intentionaly left in blank.

List of Tables

This page was intentionaly left in blank.

Acronyms

ACI Analog Control Interface

AMBA Advanced Microcontroller Bus Architecture

APB Advanced Peripheral Protocol

ARM Advanced RISC Machine

ASIC Application Specific Integrated Circuits

AXI Advanced eXtensible Interface

BIF Battery Interface

BPP Bits Per Pixel

BTA Bus Turn-Around

CCI Camera Control Interface

CSI Camera Serial Interface

COS Change Of Specification

DBI Display Bus Interface

DCS Display Command Set

DDR Double Data Rate

DI Data Identifier

DPI Display Pixel Interface

DSI Display Serial Interface

DUT Design Under Test

DVE Discovery Visualization Environment

ECC Error Correction Code

EoT End of Transfer

FPGA Field Programmable Gate Array

FPS Frames Per Second

GUI Graphical User Guide

HAPS High ASIC Performance System

HDL Hardware Description Language

HDMI High-Definition Multimedia Interface

HS High Speed

HSI High Speed Synchronous Interface

I2C Inter-Integrated Circuit

ISEP Instituto Superio de Engenharia do Porto

IOB Input/Output Block

IP Intellectual Property

IPK IP Prototyping Kits

IT Information Technology

LLI Low Latency Interface

LP Low Power

MIPI Mobile Industry Protocol Interface

OS Operating System

OSI Open Systems Interconnection

PCB Printed Circuit Board

PCIe Peripheral Component Interconnect express

PHY Physical Layer

PPI PHY Pixel Interface

P&R Place and Route

RFFE RF Front-End

RIO Reduced Input/Output

RISC Reduced Instruction Set Computer

RTL Register Transfer Level

SATA Serial AT Attachment

SoT Start of Transmission

TCP/IP Transmission Control Protocol/Internet Protocol

TEDI Tese/Dissertação

TSG Technical Steering Group

TWG Test Working Group

USB Universal Serial Bus

UFS Universal Flash Storage

VHDL Very high speed integrated circuit Hardware Description Language

WC Word Count

Chapter 1

Introduction

The silicon industry is in constant development, adopting new methodologies to enable lower production costs and shorter time to market for their products. Along with these, the companies have to adapt to new protocols, larger and more complex circuits and different client requirements.

One of the silicon industry sectors with most evident growth is the mobile devices. Nowadays, mobile devices are sought for their high performances, ease of use, features, and production quality. To address these challenges, the Mobile Industry Processor Interface (MIPI) Alliance defines and promotes open interface specifications, such as Camera Serial Interface (CSI-2) and Display Serial Interface (DSI). These specifications are adopted by many companies, which makes them standard. In this way, it is possible to integrate different companies peripherals easily in large systems in order to produce better products.

Synopsys, as one of the leading companies in digital and mixed-signal Intellectual Property (IP) and one of the MIPI Alliance contributors, develops high-quality, low-power, silicon-proven hardware and software based on its specifications.

1.1 Thesis Context

The projects reported in this document were developed in full collaboration with Synopsys as part of the Tese/Dissertação (TEDI) course of the Electrical and Computer Engineering Masters Degree lectured at Instituto Superior de Engenharia do Porto (ISEP).

As the digital designs get more and more complex, the possibility of the existence of hardware bugs or defects increases. In this way, and because the production of the designs to real silicon is a very expensive process, companies like Synopsys prototype and test the circuits in Field Programmable Gate Arrays (FPGA) systems before producing or selling the design to a client. To perform these, Synopsys has a team (IP Prototyping Kits [IPK] team) that integrate the IP in larger designs and tests it in real world applications.

The work proposed by Synopsys aimed to adapt an existing system, based on MIPI DSI Host IP, to work with different physical layers. One of the proposed physical layers is developed by Synopsys mixed-signal team and aims to one of the most recent MIPI specifications. The other is a physical layer designed and produced by a Synopsys client, that ordered the Synopsys hardware tests in order to validate their mixed-signal design with the Synopsys controllers.

As an IP integration team, the IPK team offers to its clients examples of the IP usage, and to Synopsys internal teams test procedures that validate the designs. Thus, this team's duty varies depending on the project requirements.

1.2 Objectives

As these are new projects, proposed to the IPK team, that will take as basis an existing design, a set of steps needs to be followed to achieve the main objective of this work: to test the compliance of the two physical layers implementation with the MIPI specification. These steps are:

- Study the existing design based on the MIPI DSI Host;
- Document the changes needed to the existing design;
- Modify the design;
- Simulate and validate the design through testbenches;
- Synthesize and constraint the design;
- Run hardware tests;
- Document tests results.

1.3 Document structure

To better understand the structure of this document, this section is used as a guideline to the content of the following chapters:

- **Technology Overview -** states the existing technologies regarding the mobile devices. Informs about questions such as: What is MIPI? What does it stand for? In which products can we find MIPI based peripherals?

  In short, it provides a state of the art for the subjects covered in this thesis;

- **Specifications -** although the MIPI DSI protocol is mentioned on the previous chapter (Technology Overview) this chapter details the protocol specification and how it is used in a low level implementation;

- **Project Flow -** informs how an IPK project is structured and how the MIPI IPK (IP Prototyping Kit) work is done at Synopsys;
- **Project Development -** this chapter states the project developments regarding this thesis in detail, keeping Synopsys, their clients and MIPI Aliance confidential issues protected;
- **Project Management -** presents Synopsys management guidelines that have impact in the IPK team;
- **Conclusion -** sums up the work done, its gaps and virtues, and refers to future work.

Chapter 2

Technology overview

The increasing demand for multimedia devices leads to a growth of the silicon industry and a need to improve the quality of available products. The final customer is always seeking for devices with better performance, ease of use and new features such as high resolution displays or multi megapixel cameras. As a result, the technological companies around the globe compete to provide better solutions and to conquer a greater share of the market [10].

In order to satisfy costumers needs, companies and engineering teams tend to adopt standard protocols and models in the development of new easier and faster electronic devices. One of the examples is the widely adopted Open System Interconnection (OSI) model. This model is a reference tool used to understand data communications between any two networked systems [19].

2.1 OSI Model

The OSI model divides the communication processes into seven layers (see figure 2.1). Each layer performs specific functions to support both the layers above and below it. The three lowest layers focus on passing traffic through the network to an end system. The top four layers come into play in the

end system to complete the process [19].

The seven layers that comprise the OSI model are:

- Application - interacts directly with the end-user like a menu or a Graphical User Interface (GUI);
- Presentation - offers services like encryption and connection confidentiality;
- Session - the application lower level. Allows connections among different applications. However, this layer does not offer any kind of security services;
- Transport - provides transparent transfer of data between end systems and is responsible for end-to-end error recovery and flow control;
- Network - provides routing and switching technologies, creating logical paths for transmiting data from node to node;
- Data Link - data packets are encoded and decoded into bits. Provides transmission protocol information and allows its management. Handles errors in the physical layer, flow control and frame synchronization;
- Physical Layer (PHY) - conveys the bit stream - through electric impulse, light or radio signal - through the network at the electrical and mechanical levels [20][21]. The physical layer is intended to carry information among different devices. Therefore this layer is supposed to convert the above layers information into the minimum possible pins so that it consumes the minimum Printed Circuit Board (PCB) data paths or wires.

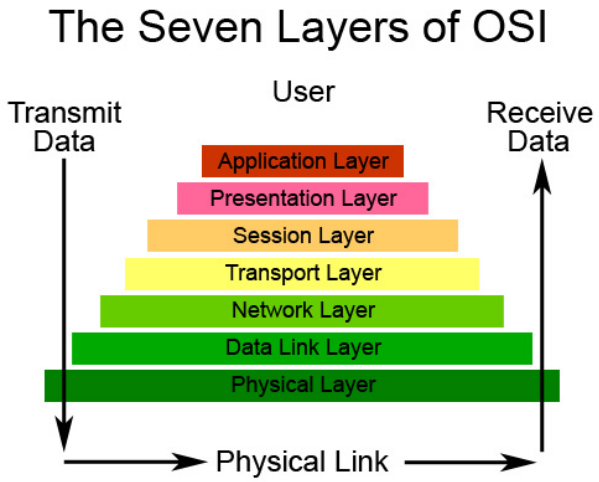


Figure 2.1: OSI Model[1].

This model is widely used in several communication protocols such as MIPI, HDMI, Ethernet, TCP/IP.

2.2 MIPI Alliance

Over the years the mobile industry showed an increasing growth. Almost all electronic devices are changing with tendency to become portable and therefore mobile products.

As the mobile business has grown, the companies sought for protocols and specifications to adopt and create standard and compatible solutions. The Mobile Industry Protocol Interface (MIPI) Alliance is a global open membership organization that develops interface specifications for the mobile ecosystem including mobile influenced industries. It was founded in 2003 by Advanced RISC Machine (ARM), Intel, Nokia, Samsung, STMicroelectronics and Texas Instruments [22].

The MIPI Alliance is a non-profit organization that works as an open membership organization. The organization targets the mobile industry and

mobile influenced industries developing standard specifications. Therefore its mission is to benefit the entire mobile industry by establishing standards for hardware and software interfaces for mobile devices [23].

Today many smartphones, and other mobile gadgets, on the market have at least two MIPI specifications. Some products employ MIPI specifications for a full range of internal connections. MIPI specifications have enabled manufacturers to simplify the design process, reduce design costs, create economies of scale that lower price points, and shorten time-to-market for components, features, and services [23].

2.2.1 MIPI Working Groups

The MIPI Alliance Working Groups are at the heart of the organization. From these targeted groups several specifications are developed progressing along a standard path - from investigation group to specification. Led by a technical chair, each specification is defined by the working group members that request input, proposals, discuss and create a specification draft that is later vetted and reviewed by a board of directors [3].

The released specifications are available to founders, promoters and contributor member companies. However, whilst promoters and member companies work together on the specification, the adopters only have access to the final release.

Currently the MIPI Alliance has fourteen active working groups including:

- Camera Serial Inteface (CSI, see section 2.2.1.1);
- Display Serial Interface (DSI, see section 2.2.1.2);
- High Speed Synchronous Interface (HSI);
- Low Latency Interface (LLI);

- Low Speed Multipoint Link;
- Marketing;
- PHY (see 2.2.1.3);
- Reduced Input/Output Working Group (RIO);
- RF Front-End Working Group (RFFE)
- Sensor Work Group (I3C)
- Software Investigation Group;
- Technical Steering Group (TSG);
- Test Working Group (TWG);
- UniPro.

Other groups were active in the past such as Analog Control Interface (ACI), Battery Interface (BIF), DigRF and System Power Management. They are now currently hibernated once no updates to the available specifications are currently required.

The available working groups work together to provide compatible specifications so that the MIPI members are able to design independent solutions of non-MIPI compatible specifications. Thus, the costumers are able to develop products using the MIPI specifications in a wide spectrum range (see figure 2.2):

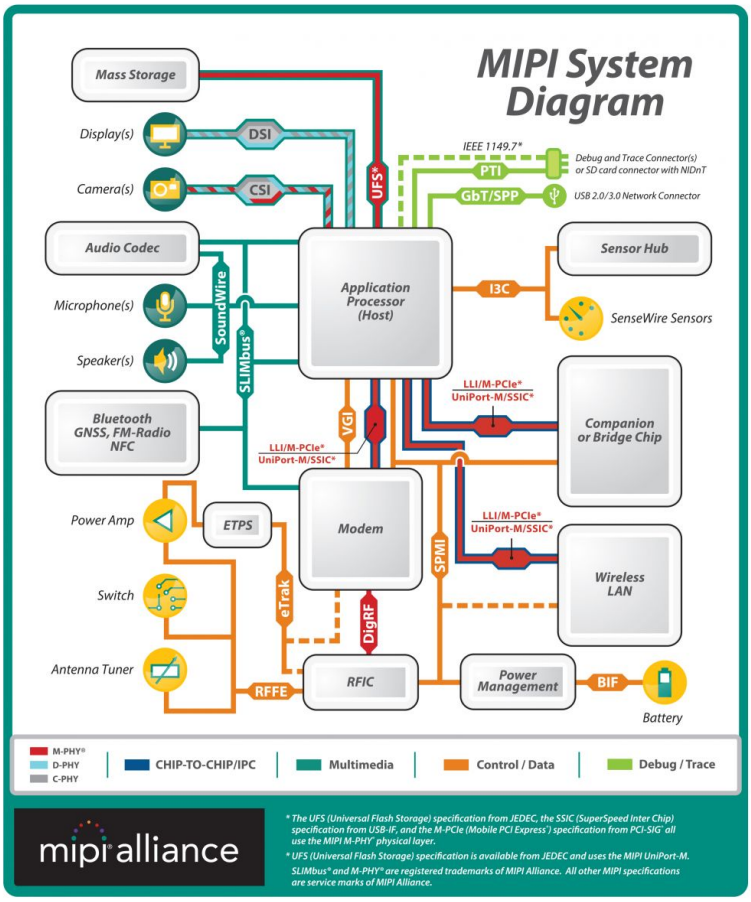


Figure 2.2: Available MIPI specifications [2].

2.2.1.1 Camera Serial Inteface

MIPI CSI-2 and MIPI CSI-3 are the sucessors of the original MIPI camera interface standard [3]. Both interfaces are still evolving, in paralel, since both specifications are still under development. These refer to highly capable architectures that give designers, manufacturers and consumers more options and value while maintaining the advantages of standard interfaces [3].

There are a few differences between the two interfaces since they use different physical and transport layers (see figure 2.3).

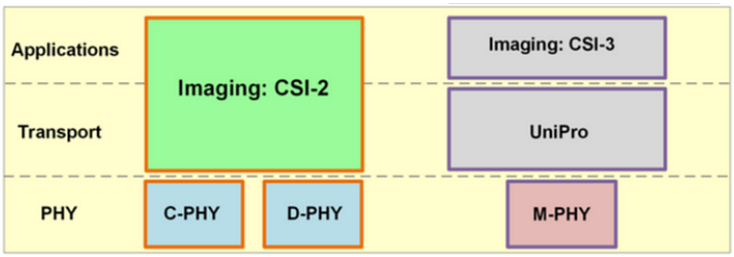


Figure 2.3: Camera Serial Interface [3].

These interfaces bandwidths are being pushed to their limits by the demand of higher image resolution, greater color depth and faster frame rates. The latest MIPI CSI 2 specification release offers higher interface bandwidth, greater channel layout flexibility and support for two different PHYs (the D-PHY and C-PHY). It predicts different performances for both PHY with the same controller (see table 2.1):

Table 2.1: CSI 2 requirements for different PHY [18].

4k @ 30fps and 12 BPP				
Required MIPI Specs	**Required pins per Lane**	**Required Lane Rate**	**Required Bandwidth**	**Control Interface**
CSI 2 D-PHY	2	1.78 Gbps	3.56 Gbps	CCI
CSI 2 C-PHY	3	1.55 Gbps	3.56 Gbps	CCI

D-PHY, as used in CSI-2, is an unidirectional differential interface with one 2-wire clock lane and one or more 2-wire data lanes. The current D-PHY specification introduces lane-based data skew control in the receiver to achieve a peak transmission rate of 2.5 Gbps/lane or 20 Gbps over 8 lanes [3].

On the other hand, C-PHY consists of one or more unidirectional 3-wire serial data lanes each with its own embedded clock. MIPI C-PHY uses 3-phase symbol encoding of about 2.28 bits per symbol operating at 2.5 Gsym/s providing an equivalent of 5.7 Gbps/lane. Three lanes operating at

the C-PHY rate of 2.5 Gsym/s provide 17.1 Gbps over a 9-wire interface.

CSI-2 over C/D-PHY imaging interface does not limit the number of lanes per link. Transmission rate scales linearly with the number of lanes for both C-PHY and D-PHY.

Figure 2.4 illustrates the connections between a CSI-2 Image Sensor transmitter and an Application Processor receiver using 6-pin C/D-PHYs, which are typically used on mobile platforms [3][18]. The high-level diagram represents the connections for the MIPI CSI from the camera to the host processor exposing the difference between using C-PHY or D-PHY.

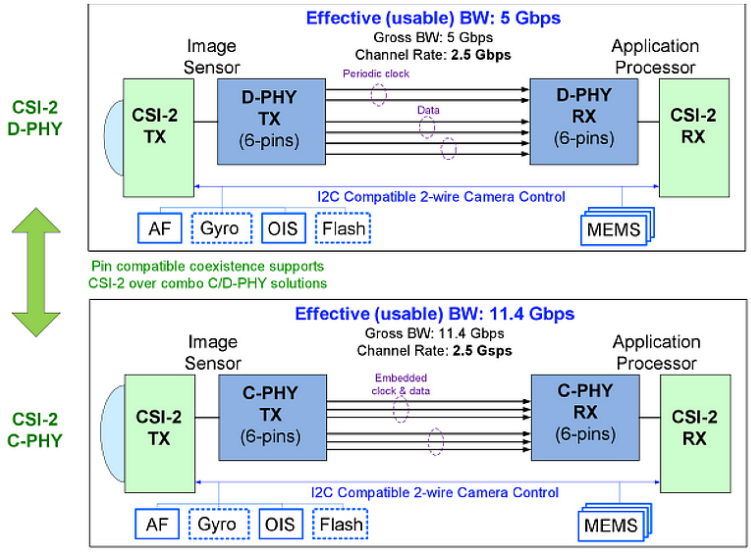


Figure 2.4: Camera Serial Interface with C/D-PHY [3].

2.2.1.2 Display Serial Interface

The MIPI DSI is another of the specifications and working groups of the MIPI Alliance. It was first released back in 2004 being one of the olders MIPI Alliance specifications. It is composed by a set of five active documents, that are still under development, and specify the behavior for each DSI controller interface:

- Display Serial Interface (DSI);
- Display Command Set (DCS);
- Display Pixel Interface (DPI);
- Display Bus Interface (DBI);
- D-PHY.

These documents specify the DSI protocol and define the communication between a host processor and a display using D-PHY physical interface [24] (see figure 2.5).

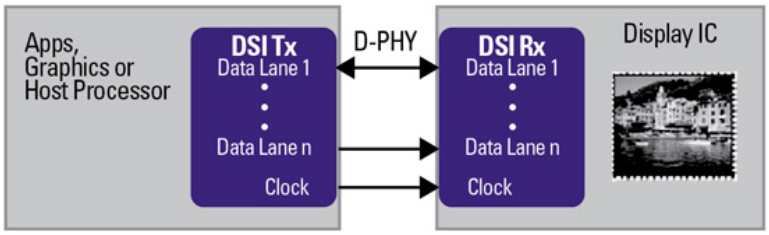


Figure 2.5: Display Serial Interface from processor to display [4].

The current specification defines a minimum of one Data Lane and one Clock Lane and a maximum of 4 Data Lanes (up-to 2.5 Gbps each). The protocol also specifies that the first Data Lane (Lane 0) must be bidirectional to enable the host processor to retrieve information from the device (Display).

In order to allow the information flow from the processor to the display, the data must go through different interfaces. The physical layer is extremely important in this process. It reduces the number of connections on the printed-circuit board since it converts the transport layer's several inputs/outputs to the data lanes. It also protects the communication between the two devices against electric noise since the lanes are differential.

For the image to be transported from the processor to the display, the processor must exercize the DSI Host controller through the DPI interface.

The DSI Host will interpret these electric signals and start generating data packets and control signals that will be sent to the DSI Device through the D-PHY. On the other side the process is exactly the reverse. The packets are received through the D-PHY and decoded into DPI by the DSI Device core. In this way it is possible to have a high bandwidth between the two devices with a reduced number of connections.

DPI

The Display Pixel Interface is a standard video interface used in protocols such as VGA or HDMI.

It defines video formats and signaling for Active-Matrix LCD displays for handheld devices. The DPI may be configured with a data path of 16, 18 or 24 parallel data bits and control signals. The DPI specification defines interface requirements for both ends (host and display) of the link, including the following attributes:

- Clock Timings;
- Data;
- Frame control signals [24].

DCS

Display Command Set is another specification included in the major Display Serial Interface specification.

It defines the means to control the display module parameters for the MIPI members with a standard set of commands that all the DSI devices must support. Thus, the interconnections and integration of products from different manufacturers are made easier since the control of the devices is standardized.

Using DCS simplifies the software drivers for the MIPI controllers and can hasten new feature support such as larger or modified display [24].

DBI

Display Bus Interface is the interface that carries the DCS commands and allows the software drivers to configure the modules inside of the DSI controller. It may have 1, 2, 8, 9 or 16 data signals and controls parameters in the host processor and display module blocks such as:

- Power supplies;
- Clock generators;
- Display drivers;
- Other blocks present on the DSI design [24].

Similarly to DCS the DBI reduces time-to-market and design costs for MIPI DSI since it allows faster integration of different manufacturer modules that support DBI connection with the same data signals.

Although the DBI spec is defined to carry DCS commands, the industry does not always use it in the DSI controllers, adopting other interfaces to configure the modules. These interfaces are also foreseen on DSI specification and are found named on the manufacturer's documents as Generic Interface. Most of the times the interface used to replace DBI is the AMBA APB.

AMBA APB

Advanced Peripheral Bus (APB) is a protocol brought by ARM Advanced Microcontroller Bus Architecture (AMBA), designed for low bandwidth access to system peripheral registers.

This paralel interface protocol uses basic control methods and defines simple write/read processes. There are two major versions used in the in-

dustry: APB2 and APB3. The differences between the two versions lie in the addition of two more signals in version 3:

- PCLK - Peripheral clock;
- PADDR - Register Address;
- PWRITE - Peripheral Write/Read ('1' for write, '0' for read);
- PSEL - Peripheral Select;
- PENABLE - Peripheral Enable;
- PRDATA - Peripheral Read Data;
- PWDATA - Peripheral Write Data;
- PREADY - Peripheral Ready (Available only on APB3);
- PSLVERR - Peripheral Slave Error (Available only on APB3) [5].

All the signals, except the PSEL, PREADY and PSLVERR, are shared by the multiple APB peripherals present on a system.

The protocol allows a single master on the bus, being the PSEL the signal that defines the slave with whom the master wants to transfer data. All data transfers are foreseen to last only two PCLK cycles in APB2. The first cycle to give the PADDR and to inform that it is a read/write access (with PWRITE asserted to write and deasserted to read) and the second cycle to read/write from the address pointed by PADDR. Only the slave that has PSEL asserted will answer to the master request. In APB3 the transfers have no maximum number of PCLK cycles due to the PREADY signal that may extend the transfer if the slave is not ready for it. Thus the transfer will be completed only after PREADY is asserted. This allows to maintain a minimum of two PCLK cycles when the peripheral is ready to answer and an unlimited number of cycles if not.

Figure 2.6 represents a simple APB3 write transaction where there are no wait states, allowing the access to take place in two cycles:

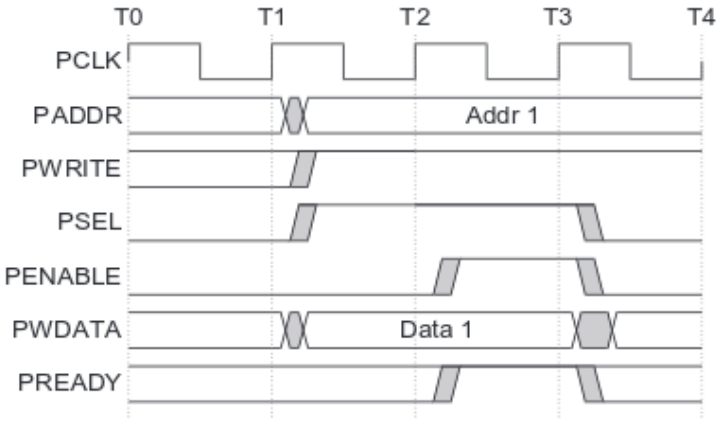


Figure 2.6: APB3 write transfer [5]

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the Setup phase. After the following clock edge the enable signal is asserted, PENABLE, indicating that the Access phase is taking place. The address, data and control signals all remain valid throughout the Access phase. The transfer is complete at the end of this cycle [5].

Figure 2.7 shows a write transfer where the PREADY extends the transfer to the moment it gets asserted.

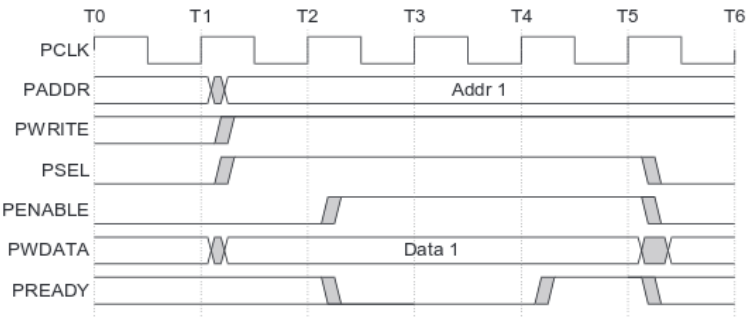


Figure 2.7: APB3 write transfer with wait states [5]

Read transfers follow the same procedures with the PWRITE being de-asserted on the first clock cycle. The data is transmitted by the peripheral on the second clock cycle (see Figures 2.8 and 2.9):

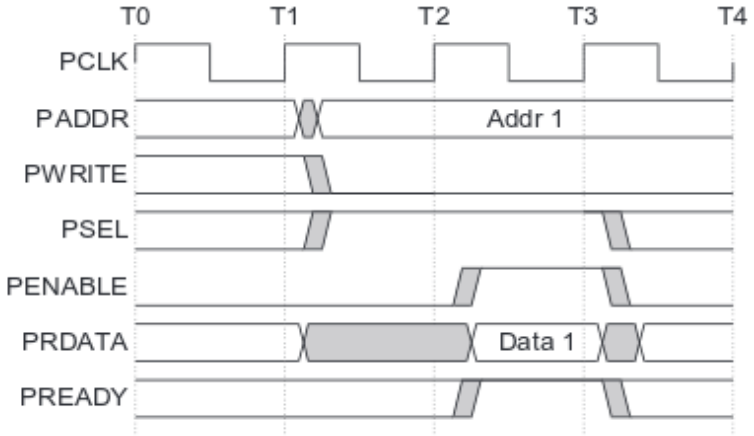


Figure 2.8: APB3 read transfer [5]

The PSLVERR is an optional output signal from the APB3 interface that indicates an error condition on the bus. Error conditions may occur in both read or write transactions and are only considered in the last clock cycle (PSLVERR is only asserted on the last clock cycle independently of the nature of the error).

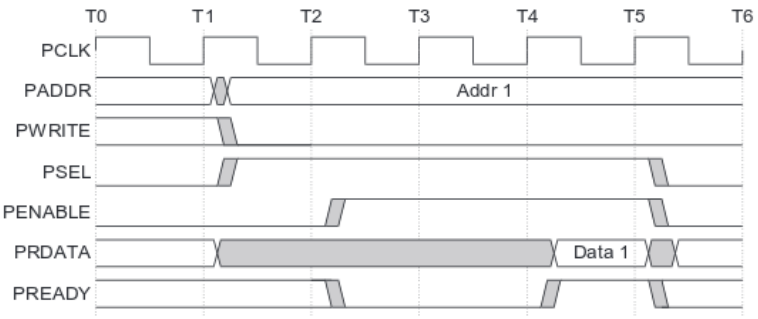


Figure 2.9: APB3 read transfer with wait states[5]

The APB3 specification leaves the PLSVERR signal to be defined by the users and designers of the interface. This means that it can be attributed to any condition that the designer wants to know that occured in a transaction. Transactions that receive an error, might or might not have changed the state of the peripheral. When a write transaction receives an error it does not necessarily mean that the register in the peripheral has not been updated. In the read transaction PSLVERR usually (but not always) means that the returned data is invalid [5].

Figure 2.10 shows an example of a write transfer with an error occurring in T4 clock cycle.

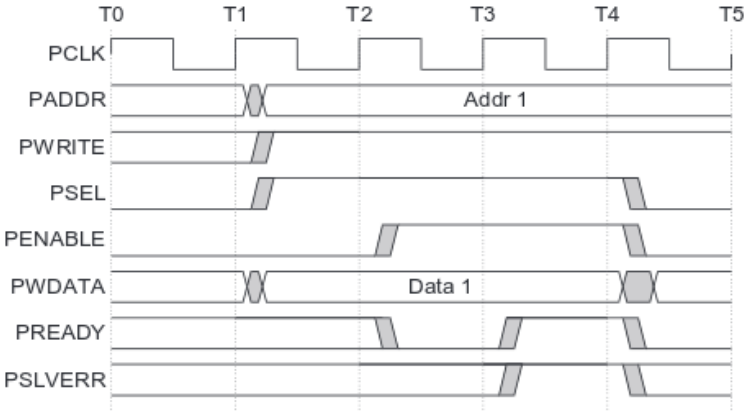


Figure 2.10: APB3 write transfer with error condition[5].

2.2.1.3 PHY

The PHY Working Group is chartered to specifiy high-speed mixed signal physical layer design to support multiple transport and application requirements [25]. The first aim of this working group was to support DSI and CSI specification requirements. As result the working group defined the first D-PHY specification, a low power, differential signaling solution with dedicated clock pair (called Clock Lane) and one or more data pairs (called Data-Lanes). Later specifications of M-PHY and C-PHY were released for different purposes. M-PHY uses differential signaling (on two wires) and supports several industry specifications developed by MIPI as well as other partner organizations. C-PHY provides camera and display applications with 3-phase encoding on a three-wire interface [25].

D-PHY

The PHY is the heart of any advanced, serial interconnect standard.

The MIPI Alliance recognized that different peripherals often share similar requirements from the physical layer. Therefore the D-PHY specification defines a reusable physical layer that supports camera interface (CSI), display interface (DSI) and general-purpose high-speed/low-power interfaces. This helps the adopter and contributor companies to reuse the same technology in different products and reduces time-to-market and PHY engineering investment on different designs.

Currently the D-PHY specification defines a maximum of 2.5Gbps per lane and supports bidirectionality as needed by the applications and protocols. It also meets the demanding requirements of low-power, low-noise-generation, and high-noise immunity which are required by mobile phone designs. [25] .

M-PHY

M-PHY v3.0, the most recent version of the specification, defines bandwidth speeds reaching 5.8 Gigabits per second per lane.

MIPI M-PHY is a high-frequency, low-power physical layer defined by MIPI Alliance Specification for M-PHY. It can be used as a physical layer for many applications, including interfaces such as:

- Display;
- Camera;
- Audio;
- Memory and storage;
- Power management.

Currently M-PHY supports the following MIPI specifications: DigRF v4, CSI-3, UniPro, and LLI. By special agreement with JEDEC, JC-64.1 Universal Flash Storage (UFS) uses UniPort-M, the combination of M-PHY physical layer and MIPI UniPro specifications .

C-PHY

C-PHY is the most recent design to join the MIPI physical layer. It uses 3-phase encoding on a unidirectional three-wire interface, being its main goal to give the adopters and contributors an alternative to D-PHY. The C-PHY is designed to support camera and display interface.

In contrast to the D-PHY, the C-PHY does not require a separate clock lane. It provides flexibility to assign individual lanes in any combination to any port on the application processor via software control. Due to similarities in basic electrical specifications, C-PHY and D-PHY can be implemented on the same device pins. A 3-phase symbol encoding technology

delivers approximately 2.28 bits per symbol over a three-wire group of conductors per lane. This enables higher data rates at a lower toggling frequency further reducing power [25].

2.2.2 MIPI Solutions

Although the main area where MIPI specifications are found is the smartphones market, it is also used on many other mobile devices such as:

- Tablets;
- Laptops;
- Cameras;
- Multimedia players;
- Smart-watches [26].

To develop these products the MIPI Alliance has several contributors and adopters of its specifications, such as Synopsys Inc, Intel, ARM Limited, LG, Samsung, Raspberry Pi Foundation.

Each company uses the specifications at its own will presenting different and innovative solutions. Some of these companies are only focussed in the development and selling of Intellectual Property (IP). Other companies buy IP and use it to develop real-world use-cases/applications. Others do not develop any MIPI IP and only adopt an IP to develop a higher-level solution such as a smartphone.

For instance, Synopsys and Cadence are MIPI contributors that invest their time in producing MIPI IP and its drivers. On the other hand Raspberry Pi Foundation only adopts developed IP and integrate it in SoC.

The Raspberry Pi board (Figure 2.11) is an example of a real world application that implements MIPI specification in its final stage of production. It is a ready-to-use board that allows its users to use the MIPI DSI

and MIPI CSI interfaces with provided software and devices (display and camera, respectively).

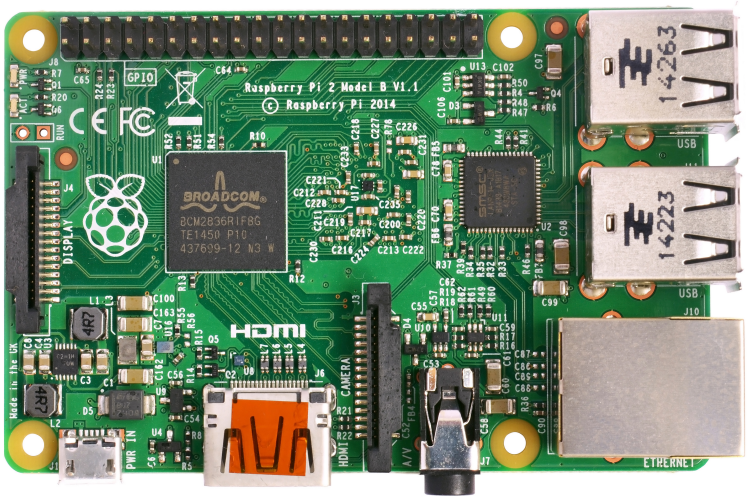
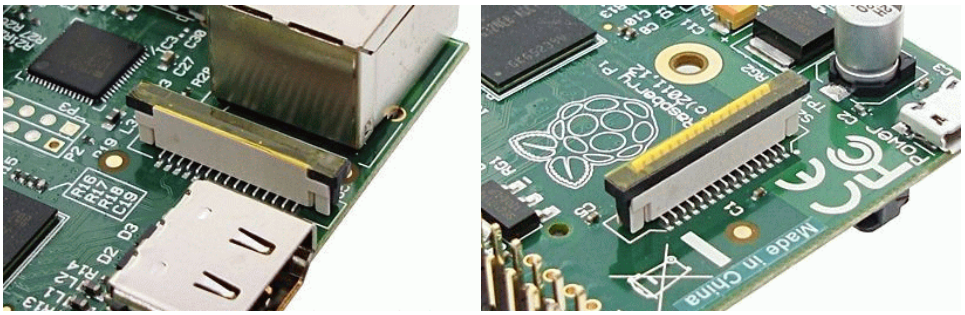


Figure 2.11: Raspberry Pi board [6].

The board provides simple connections for camera and display to the host processor (Broadcom BCM2835, see figures 2.12a and 2.12b). These are normal ports that allow PHYs to transfer information from devices to processor (and vice-versa depending on the protocol).



(a) Raspberry Pi CSI connection (b) Raspberry Pi DSI connection

Figure 2.12: Raspberry Pi MIPI connections.

The DSI connection through D-PHY has two data lanes and a clock lane, being the data lane 0 a bidirectional link. This allows the host processor to

read registers from the display controller.

The CSI connection supports cameras with D-PHY physical layer with two data lanes. All the lanes are unidirectional, which is standard for the CSI2 specification.

2.3 Hardware Description Languages

To implement hardware solutions the designers no longer implement circuits by hand or in proto-boards like breadboards. As the circuits require a huge amount of logic due to its complexity the designers use Hardware Description Languages (HDL) to describe their circuit behavior at the Register Transfer Level (RTL).

HDL are mostly used in digital logic since they allow a precise, formal description of a digital circuit enabling automated analysis and simulation.

Description languages allow the hardware designer to abstract to a higher level than the gate level. It is then possible to synthesize the description into a netlist[1] which can then be placed and routed to produce a set of masks to dope waffers into integrated circuits.

2.3.1 HDL History

The HDL appeared due to exploding complexity of the circuits since 1960s - when Gordon Moore, working at Intel at the time, observed that the number of transistors in a dense integrated circuit, such as a processor, doubles every year [27]. Nowadays, this trend is still observed (Figure 2.13) and it is known as the Moore's Law. Circuit designers needed digital logic descriptions to be performed at a high level without being tied to a single technology such as CMOS or TTL. Thus, HDL primary motivation was to design in RTL abstraction, with a model of data flow and a timing diagram [28].

[1] A netlist is a specification of physical electronic components and how they are connected together

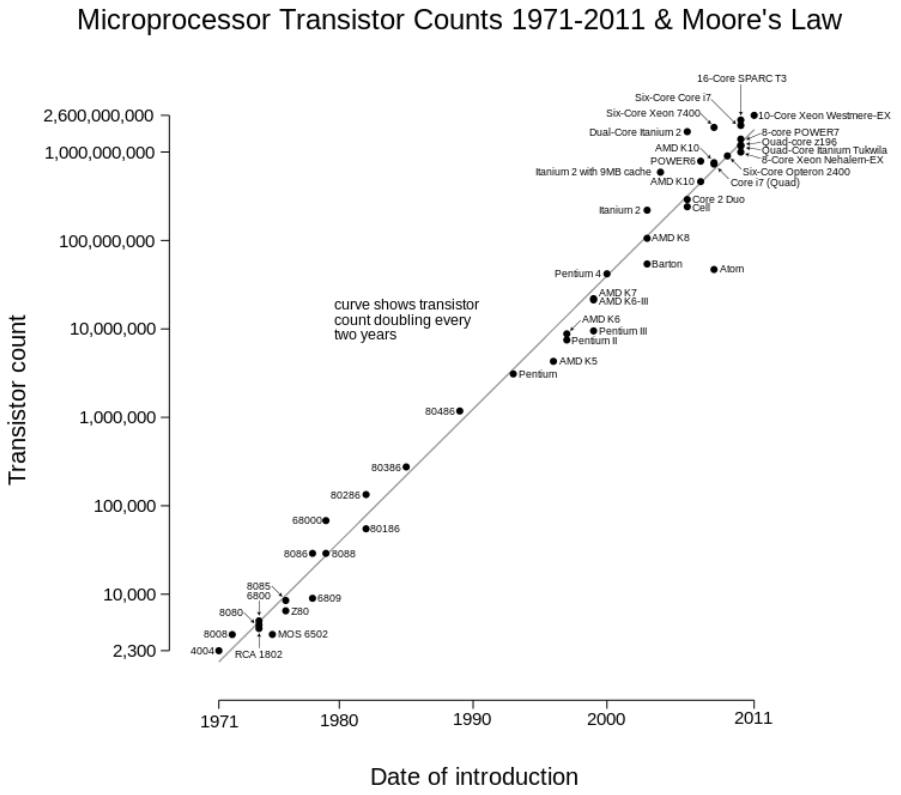


Figure 2.13: Moore's Law through the years [7].

The first HDL that presented a lasting effect was described in 1971 in a text of Gordon Bell and Allen Newell. It firstly introduced the concept of RTL [28] when used to describe the behaviour of equipment produced by Digital Equipment Corporation (DEC, see figure 2.14).

It has became more widespread and known with the introduction of DEC's sixteen RTL modules of a book documenting their use [29].

Figure 2.14: Digital Equipment Corporation [8].

2.3.1.1 VHDL and Verilog

Nowadays, the most well known hardware description languages are VHDL and Verilog, both IEEE standards. They are used in the industry to implement digital and mixed signal descriptions.

VHDL (Very high speed integrated circuit Hardware Description Language) became IEEE standard 1076 [30] in the year of 1987, while Verilog became IEEE standard 1364 [31] in the year of 1995. Nevertheless, Verilog has been used for much longer than VHDL since it was launched (before being an IEEE standard) by Gateway in 1983 [9].

The modeling constructs of VHDL and Verilog cover a slightly different spectrum across the levels of behavioural abstraction (see figure 2.15).

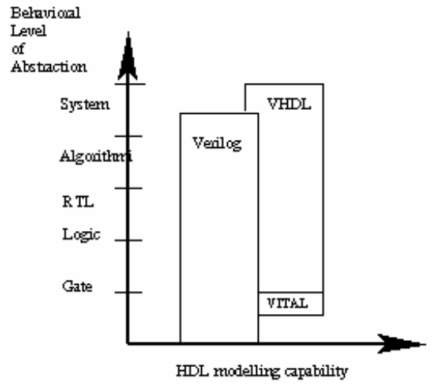


Figure 2.15: VHDL and Verilog comparison [9]

They are very different languages, in terms of syntax, but with the same purpose in terms of hardware. VHDL has similar syntax as Pascal and Verilog as C. VHDL possesses more data types and allows bigger modeling capabilities than Verilog. However, Verilog its known for the simple syntax and ease of use while VHDL is a much more verbose language (see examples 2.1 and 2.2).

```
1  --VHDL upper counter
2
3  signal counter: std_logic_vector (7 downto 0);
4  process(clock, reset)
5  begin
6    if rising_edge(clock) then
7      if reset='1' then
8        counter <= (others=>'0');
9      else
10       counter <= std_logic_vector(unsigned(counter)+1);
11     end if;
12   end if;
13 end process;
```

Listing 2.1: VHDL upper counter

```
1 //Verilog upper counter
2 reg counter [7:0]
3
4 always @(posedge clock or posedge rst)
5   if rst=1'b1
6     counter <= 8'h00;
7   else
8     counter <= counter + 1;
```

Listing 2.2: Verilog upper counter

Although these are different languages with a very different syntax, in terms of hardware both of the above examples produce the same output (Figure 2.16).

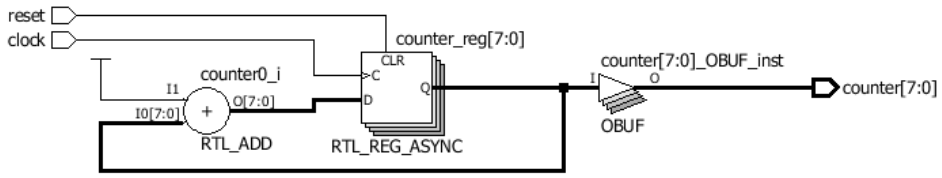


Figure 2.16: Output Circuit.

This chapter summed up some of the MIPI protocols that are used by mobile equipment. The MIPI protocols are behind all current display serial interfaces used in those devices. High level description languages are used to describe both the implementation and the simulation files necessary to test the functionality of the MIPI interfaces. The next chapter presents in higher detail the MIPI DSI specification and show how data flows from the host processor to the display.

Chapter 3

Specification

This chapter presents the MIPI DSI specification. The DSI specification is a confidential document protected by the MIPI Alliance and its contributors and members. Thus this document is based on an earlier release of the specification that may be found online [10].

The purpose of the DSI specification is to define a standard high-speed serial interface between a peripheral, such as an active-matrix display module, and a host processor in a mobile device. By standardizing this interface, the developed components may provide higher performance, lower power and fewer pins than current devices, while maintaining compatibility across products from multiple vendors [10]. Figure 3.1 shows a simplified view of the connection between a MIPI DSI Host and a MIPI DSI Device.

As mentioned in section 2.1, the MIPI protocols refer to the OSI model following its guidelines in order to improve the specifications. Therefore, the MIPI DSI presents, in its specification, the behavior for each of the layers. Figure 3.2 shows a detailed interconnect view of the MIPI DSI Host to the MIPI DSI Device through each layer.

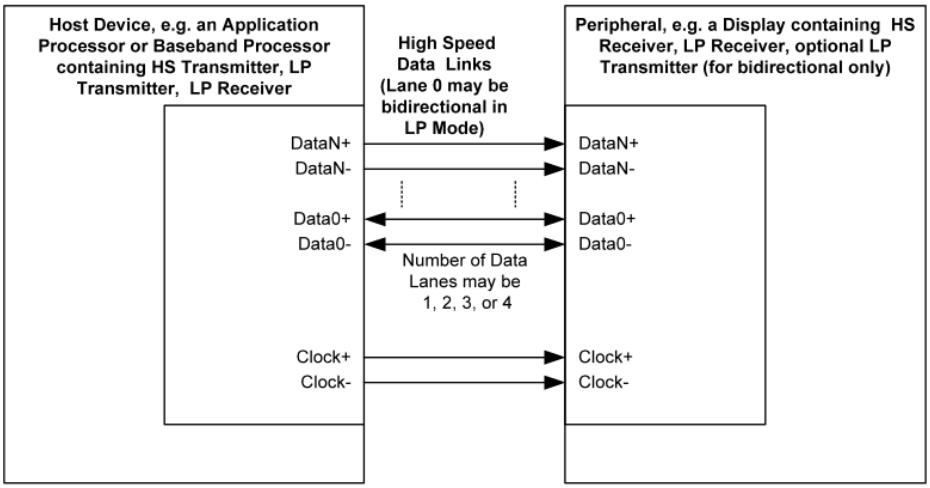


Figure 3.1: DSI Transmitter and Receiver Interface[10].

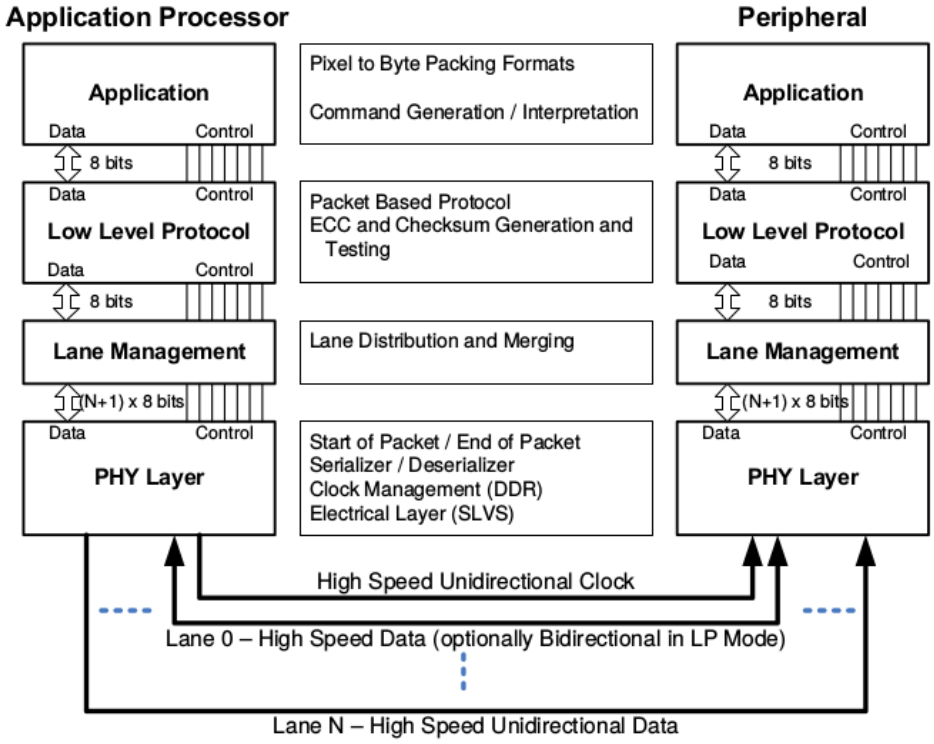


Figure 3.2: DSI Layers [10].

3.1 DSI Physical Layer

The PHY receives the data from the DSI Host controller (located in the Protocol and Lane Management layers) in a paralell byte-to-byte system, then serializes and sends it into the Lanes physical connections to the DSI Device.

The information flows between the DSI Host to the DSI Device using one or more serial data links aaccompanying the serial clock line. The action of sending high-speed data across the bus to the display is called *HS transmission* or *burst*.

Between HS transmissions the DSI Host controller predicts if it has time to go to Low-Power State (LP). The Lanes should be in LP when not sending valid data across the bus. Figure 3.3 shows the normal HS transmission through one data lane.

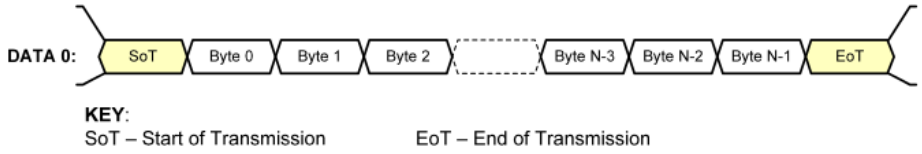


Figure 3.3: DSI HS transmission through one data lane [10].

There is no handshake among Protocol and PHY layers, which allows the controller (protocol layer) to throttle data transfer to (or from) the PHY layer once transmissions are occuring. Packets shall be sent and received in their entirety and without interruption. Therefore the Protocol layer and data buffering on both ends of the Link shall always have bandwidth equal to, or greater than, PHY layer circuitry. A practical consequence is that the system implementer should ensure that DSI Device (display/receiver) has bandwidth capability equal or greater than DSI Host (Host processor/ transmitter).

As mentioned earlier in section 2.2.1.2, DSI protocol specifies a range

of 1 to 4 Data Lanes and a Clock Lane. In all cases the first lane (Data Lane 0) must always be bidirectional and the remaining unidirectional. The Clock Lane must in every case be managed by the DSI Host and never by the peripheral.

D-PHY supports Low Power (LP) and High Speed (HS) data transmissions for Command and Video Mode, respectively. The Command Mode (in Low Power) must only use Data Lane 0 whilst the High Speed (Video Mode) transactions use all the available lanes.

As the only bidirectional lane is Data Lane 0, when the peripheral is transmitting it must always be in Command Mode. For the Device to be transmitting, it must be requested by the host. Therefore, when the host requires a response from a peripheral, e.g. returning read data or status information, it must assert a turn-request pin to its PHY during the last packet of the transmission. This tells the PHY layer to issue a Bus Turn-Around (BTA) command.

When the peripheral receives the BTA command, its PHY layer asserts Turn-Request as an input to the Protocol layer. This tells the receiving Protocol layer that it shall prepare to send a response to the host processor. Normally, the last packet received tells the Protocol layer what information to send once the bus is available for transmitting to the host processor.

3.1.1 Bidirectional Control Mechanism

Turning the bus around is controlled by a token-passing mechanism: the host processor sends a BTA request, which conveys to the peripheral its intention to release, or stop driving, the data path after which the peripheral can transmit one or more packets back to the host processor. When it is finished, the peripheral shall return the control of the bus back to the host processor.

In bidirectional systems, there is a remote chance of erroneous behavior

(device and host driving the data path) resulting in bus contention. Mechanisms are provided in the DSI specification for recovering from any bus contention event without forcing "hard reset" of the entire system. One of these methods specifies that, in case of contention, both host processor and display must release the data path to stop the contention to occur. After this procedure the host must start driving the path and restart the transaction.

3.2 Multi-Lane Distribution and Merging

As MIPI DSI is a lane scalable (from 1 to 4 Data Lanes) protocol it must manage the packets through its available lanes. To do that, the DSI specification defines that the Host controller must distribute a sequence of packet bytes across the available Lanes (Figure 3.4) where each Lane is an independent block of logic and interface circuitry. In the receiver (shown in Figure 3.5) the layer collects incoming bytes from the same number of Lanes and consolidates the bytes into complete packets to pass into the following packet decomposer.

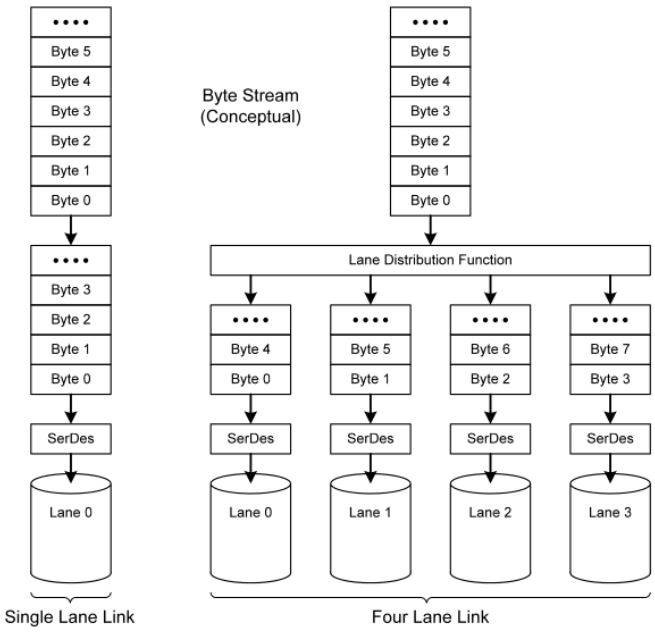


Figure 3.4: Packet distribution through the lanes [10].

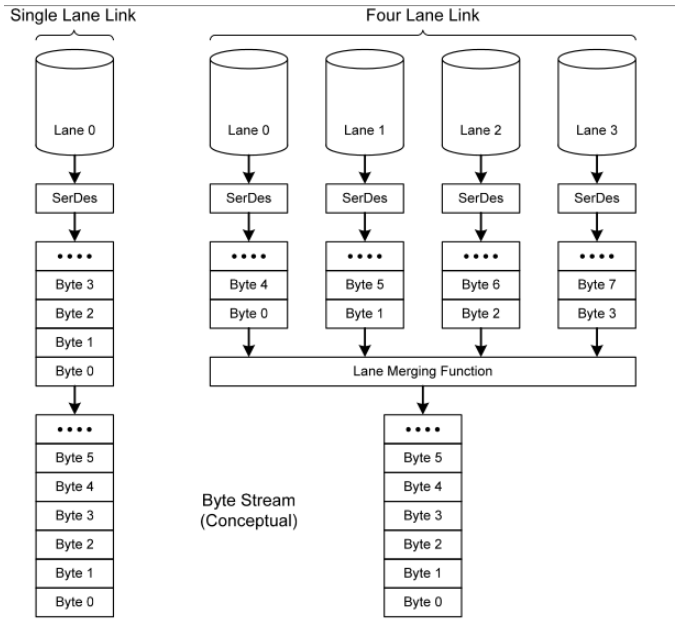


Figure 3.5: Packet merge through the lanes [10].

The Lane Distributor, present on the host processor and on the display, takes a HS transmission of arbitrary byte length and sends them in groups in parallel across the Data Lanes. Before sending data, all Lanes perform the Start of Transmission (SoT) sequence in parallel to indicate to their corresponding receiving units that the first byte of a packet is beginning. After SoT, the Lanes send groups of N bytes, where N is the number of available Data Lanes, from the first packet in parallel. For example, with a two Lane system, byte 0 of the packet goes to Lane 0, byte 1 goes to Lane 1, byte 2 to Lane 0, byte 3 to Lane 1 and so on (as seen in Figures 3.4 and 3.5).

Figure 3.6 gives two examples of HS transmission with multiple and non-multiple bytes of the number of Data Lanes.

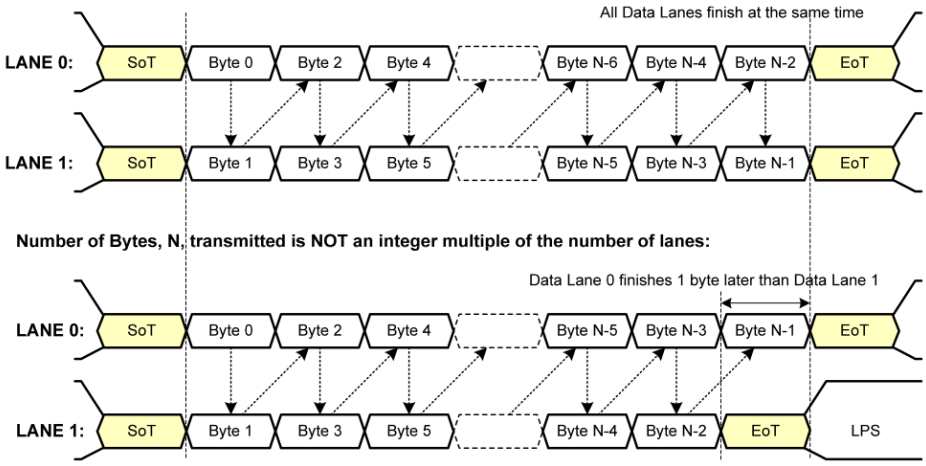


Figure 3.6: Two Lane HS transmission example [10].

3.2.1 Multi-Lane Interoperability and Lane-number Mismatch

The number of Lanes used shall be a static parameter. It shall be fixed at the time of system design or after an initial configuration and may not change dynamically. The host processor shall be configured to support the same number of Lanes required by the peripheral. Specifically, a host processor

with N-Lane capability ($N > 1$) shall be capable of operating using fewer Lanes, to ensure interoperability with peripherals having M Lanes, where $N > M$.

Figure 3.7 shows a host processor with 4 available Data Lanes, operating with a device that only includes 2 Data Lanes, making the Data Lane 2 and Data Lane 3 of the host processor unused.

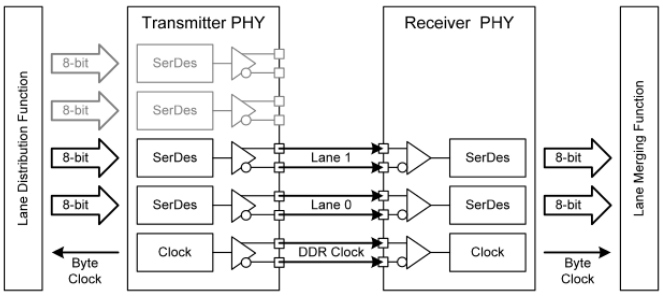


Figure 3.7: Host processor interoperability with 2 Data Lane Device [10].

3.3 DSI Protocol

The DSI protocol must be presented and implemented on the host processor and on the display controllers. On the transmitter side of a DSI Link, parallel data, signal events and commands are converted in the Protocol layer to packets, following the packet organization documented in this section. The Protocol layer appends packet-protocol information and headers, and then sends complete bytes through the Lane Management layer to the PHY. Packets are serialized by the PHY and sent across the serial Link. The receiver side of a DSI Link performs the converse of the transmitter side, decomposing the packet into parallel data, signal events and commands.

3.3.1 Multiple Packets per Transmission

In its simplest form, a transmission may contain one packet. If many packets are to be transmitted, the overhead of frequent switching between LP and HS will severely limit bandwidth if packets are sent separately one packet per transmission.

The DSI protocol allows multiple packets to be concatenated, which boosts effective bandwidth. It is useful for events such as peripheral initialization, where many registers may be loaded with separate write commands at system start up. Figure 3.8 shows, on the above part, packets being sent separately, and bellow the same number of packets being sent concatenated in a single HS transmission.

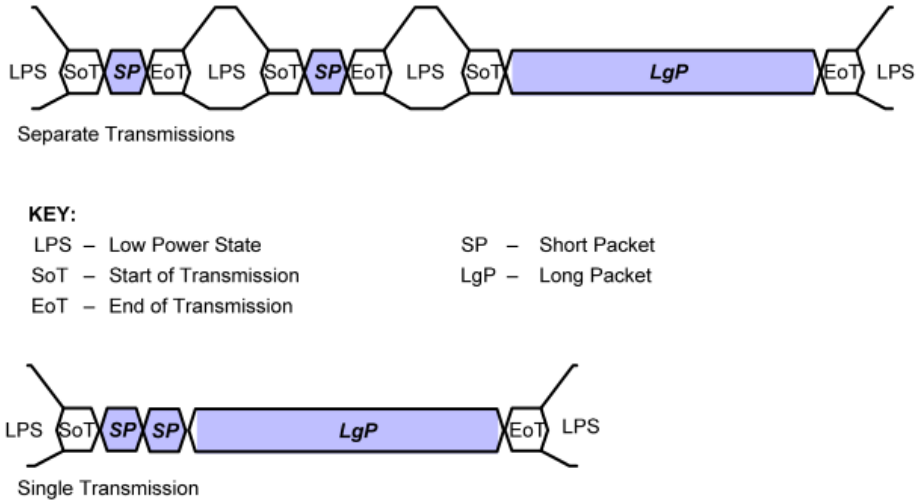


Figure 3.8: Difference between separate and concatenated packet transmission [10].

3.3.2 Packet composition

The DSI Protocol defines that each packet must obey some rules that every MIPI DSI Hosts and Devices must satisfy. This way every DSI controller will be compatible among different manufacturers, which makes the controllers

easier to integrate in a full scale system.

The MIPI DSI controllers are able to send/receive two types of packets: Long and Short packets, depending on the information that both of them can carry.

3.3.2.1 Long Packet

The first byte of the packet must contain the Data Identifier (DI) which includes information about the type of the packet and its length (short or long). For instance, in Video Mode systems a packet may be one horizontal display line. DI contains in its 8 bits the Virtual Channel (used only for multiple displays) and the Data Type for the application specific payload data (Data types will be discussed later in detail in this same section).

After the DI the next two bytes, named Word Count (WC), contain the information that defines the number of bytes of the packet. For example, if two bytes of valid information are to be sent, the WC must store among its 16 bits (2 Bytes) the value of 2.

Afterwards it is sent an Error Correction Code (ECC) byte that allows single-bit errors to be corrected and 2-bit errors to be detected. The ECC only covers errors in the DI and WC fields.

DI, WC and ECC are contained in the first 32 bits of a MIPI DSI packet. These are always present and all together form the Packet Header, or just Header.

After the end of the Packet Header, the receiver reads the next Word Count $\times$ bytes of the Data Payload. Within the Data Payload block, there are no limitations on the value of a data word, i.e. no embedded codes are used.

At the end of the Payload two additional bytes of checksum are added to the packet in order to detect payload errors.

Figure 3.9 shows a long packet structure.

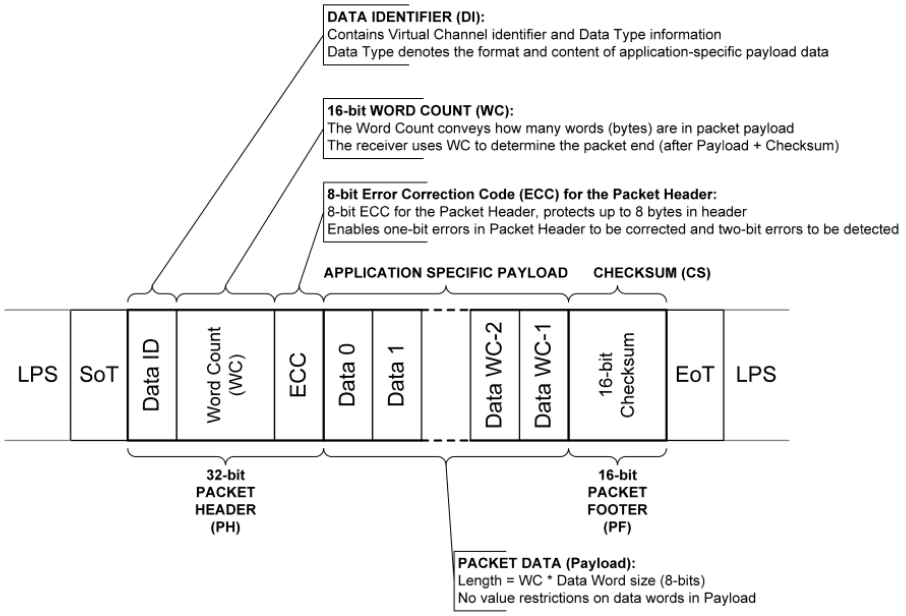


Figure 3.9: Long Packet structure [10].

With 16 bits WC, the Long Packet is able to transmit from 2^0 to 2^{16} Bytes of valid data (1 to 65536 Bytes) and a total of $2^{16} + 6$ counting with Header and last two checksum bytes.

3.3.2.2 Short Packet

The Short Packet follows almost the same method described for the Long Packet with the difference that there is no WC included between DI and ECC. Instead the Short Packet includes their valid data between the DI and ECC, and may contain from two to nine bytes. Figure 3.10 shows the Short Packet structure.

3.3.2.3 Data Identifier Byte

The first byte of any packet is the DI (Data Identifier) byte. Its two most significant bits (bit 7 and bit 6) identify one of four virtual channels. Virtual Channels may be used in case of using multiple displays.

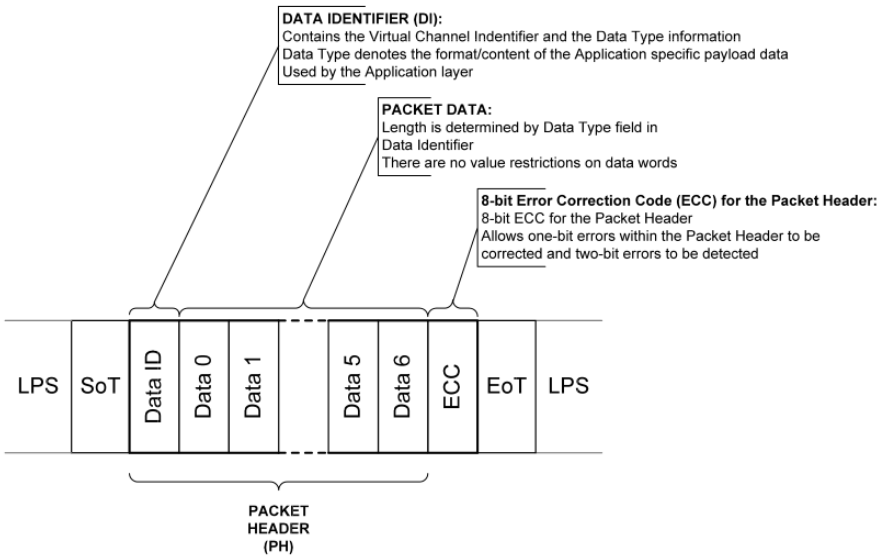


Figure 3.10: Short Packet structure [10].

It identifies to which display is the data directed to. If only one display is being used, the Virtual Channel must remain 0 through all the packets.

The six less significant bits specify the Data Type. The Data Type field specifies the size, format and, in some cases, the interpretation of the packet contents. For example, when sending a VSYNC packet the Data Type is the packet information itself, as there is no aditional data being sent.

The last Byte of DI is the ECC. It must always be calculated by the host processor. The peripherals are not required to calculate an ECC byte but to check if the Data Type was correctly sent. If the ECC is not used, a single byte of all zeros (00h) shall be transmitted.

Data Type

MIPI DSI specification allows several Data Types to be sent/received. Table 3.1 shows the available Data Types.

Table 3.1: Available Data Types [10].

Data Type (hex)	**Description**	**Packet Size**
01	Sync Event, V Sync Start	Short
11	Sync Event, V Sync End	Short
21	Sync Event, H Sync Start	Short
31	Sync Event, H Sync End	Short
02	Color Mode Off	Short
12	Color Mode On	Short
22	Shut Down Peripheral	Short
32	Turn On Peripheral	Short
x3 and xB	Generic Write	Short
x4 and xC	Generic Read	Short
x5 and xD	DCS Write	Short
06	DCS Read	Short
37	Set Maximum Return Packet size	Short
09	Null Packet	Long
19	Blanking Packet	Long
29	Generic Non-Image Packet	Long
39	DCS Long Write Packet	Long
0E	Packed Pixel Stream 16-bit RGB 5-6-5	Long
1E	Packed Pixel Stream 18-bit RGB, 6-6-6	Long
2E	Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6	Long
3E	Packed Pixel Stream, 24-bit RGB, 8-8-8	Long
x0 and xF	Reserved, do not use	

In this chapter the MIPI DSI specification was summed up in order to provide the basis for understandig the following chapters. Next it is presented the MIPI IPK (IP Prototyping Kit) work flow at Synopsys and how projects are structured.

This page was intentionaly left in blank.

Chapter 4

Project Flow

This chapter presents how the projects flow at Synopsys and explains the details of the two projects developed in the context of this thesis.

At Synopsys, such as in other IP developer companies, the projects flow from the simplest hardware blocks to the integration of larger systems that carry multiple blocks and software. Thus, to carry the projects along its multiple mutations and changes, the companies need to have multiple teams working on the same projects.

The IPK team at Synopsys works, along with other teams, on the integration stage of the projects. This is the final stage of the project and occurs in parallel with the IP releases. Whenever an IP controller RTL is validated in simulation, the IPK team integrates it into larger designs and test its features in real world applications to validate it in hardware. Figure 4.1 represents the project flow at Synopsys before it is released to customers.

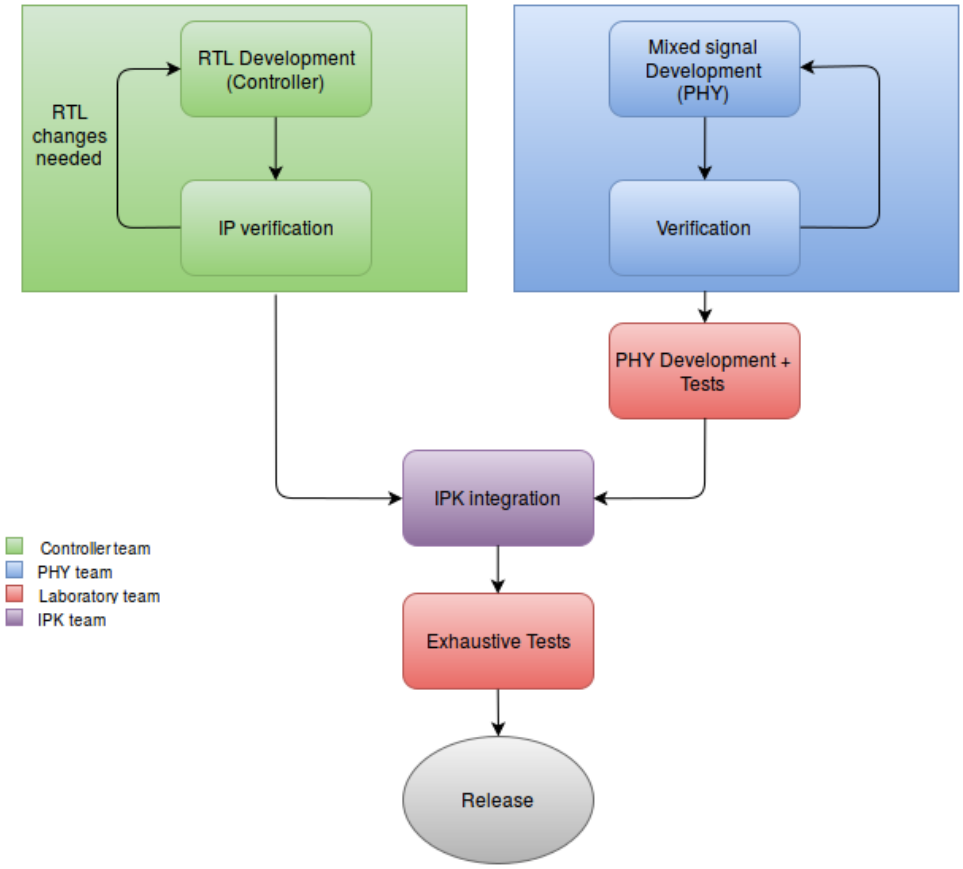


Figure 4.1: Synopsys Project Flow.

4.1 Different teams for the same projects

As presented in Figure 4.1 the projects pass through diferent teams before being officially released. This implies long term projects, with huge complexity, short time windows between deliveries and low tolerance to delays as it may compromise the work of other teams.

The projects follow the specification guidelines and, whilst the controller team develops its design regarding the transport layer specifications (such as DSI, DBI and DPI), the PHY team refers to the physical layer specification (D-PHY for DSI controller). Sometimes this leads to unexpected incompatibilities that need to be discussed and agreed between both teams.

Therefore, although the teams work on the same projects, such as producing solutions based on MIPI DSI protocol (integrating controller and PHY), their worl will not always be synchronized due to the different specification release dates. As the specifications relating to the transport or physical layer get released, a new controller or a new PHY is produced. In this way, the IPK team performs different tasks, like integrating a new D-PHY or integrating a new controller onto the design. The two different tasks for the IPK team are:

- Hardware Validation (when validating a new PHY);
- Prototyping kits (when prototyping a new controller).

Each of these are intended for different purposes regarding the different projects, requiring work from both the IPK team parts, hardware and software.

4.1.1 Hardware Validation

Hardware validation may be requested from inside the Synopsys or by outside companies. Its purpose is to validate hardware behaviour integrated into the IPK environment. Therefore, an hardware validation procedure at Synopsys may be, for example, testing a new PHY with a previously tested controller, testing a display panel or validating some board. This requires effort analysis to evaluate the changes or enhancements needed on the available systems. After the hardware tests a report is delivered to whom requested the hardware validation explaining whether the hardware is working correctly or not.

The hardware validation is frequently requested to the IPK team to validate new PHYs. This is done to avoid the PHYs to reach the market without being tested in a real hardware system. Thus the PHYs are produced and

integrated with the controllers in order to validate its behaviour and prevent design bugs.

4.1.2 Prototyping Kit

Design prototyping is the main work developed by the IPK team at Synopsys. Prototyping kits are designed or updated whenever a new controller release is scheduled. This is necessary not only to test the controller performance on a FPGA kit, but also to assess its behaviour when integrated in a real user application. When selling an IP, Synopsys not only delivers the IP to its costumers but also a complete example of its integration in a SoC environment, a full functional implementation comprising both hardware and software parts.

Prototyping kits are designed around the IP controller and a PHY. The controller is wrapped inside an FPGA along with other hardware blocks, such as an interface to the PHY, in order to form a full stack hardware transport layer. The PHY is manufactured and turned into a testchip/ASIC and is further soldered into a board to be tested with the controllers.

4.2 IPK Environment

The Synopsys controllers are blocks of hardware that require a huge amount of digital logic. Therefore the designs are complex and require computers with high performance hardware to synthesize and simulate them. As the complexity of the developed systems increase, there is the need of using large servers to store and process the IP/IPK data.

Each and every hardware project is developed using common servers located at strategic places on the world globe.

The IPK team at Porto site uses a server located in Munich to run regressions[1] and programs that have limited access licenses. Thus, the tools may be shared by different users without owning licenses to each and everyone of the employees at Synopsys.

4.3 Remote Machines at Synopsys

The machines, installed in Munich, run Unix based OS like Red Hat and Solaris and use *tcsh* shell. This gives the designer a complex and open environment to work on.

In order to use these machines, a remote connection needs to be established to the server using a client application such as Citrix Receiver. Software like Citrix Receiver allows the user to share the desktop using an interactive session (see Figure 4.2).

After establishing the connection to the server using Citrix, the user is still not able to run regressions nor programs. Instead, the user has the possibility to request two types of jobs to the Munich servers:

- ilight - user requests ilight job whenever wants to access license servers and run binaries such as shell scripts;
- iheavy - user requests iheavy job whenever wants to access servers that provide software licenses and are able to run software programs with GUI that require higher hardware performance.

With these types of jobs the servers management is made easier since different users/tasks will require different machines. After the access to one of these types of machines is granted, the user is ready to work on the Munich server remotely and launch programs and scripts under a Unix environment.

[1] Regressions are sets of scripts, or programs, that run sequencially with the purpose of giving the user a desired output. For example, a synthesis regression will output a bitfile to transfer to an FPGA, a simulation will output a waveform database.

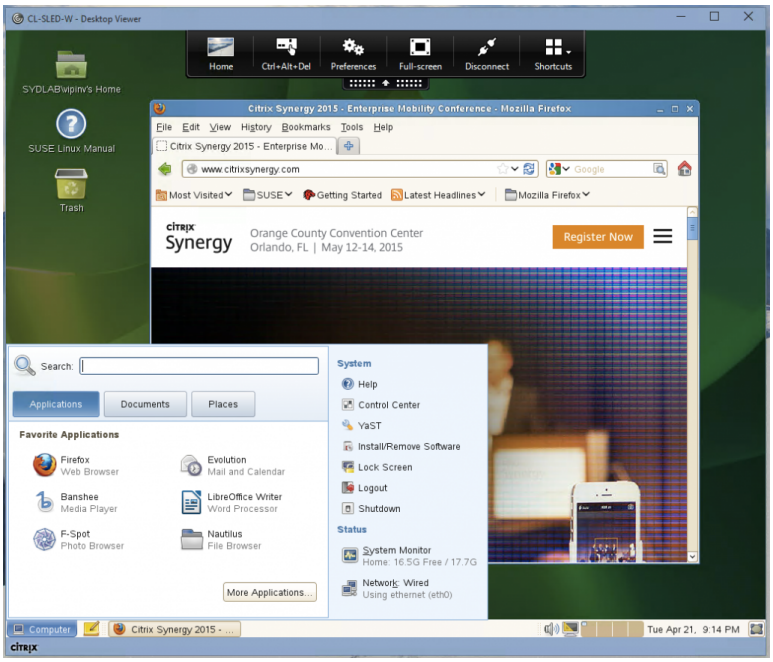


Figure 4.2: Citrix Receiver software [11].

To launch one of these jobs, the user needs to use the *qsc* Linux command on a *tcsh* terminal. The following prompt shows an example on how to launch an interactive iheavy job:

```
1 [prodrig/]# qsh -P iheavy
```

After executing the *qsc* command, a ticket will be queued into the server. Another terminal will be launched on Citrix meaning the job was granted for the user to work on the respective "ilight" or "iheavy" machine.

Although both iheavy and ilight requests launch interactive sessions on Munich, when launching scripts that consume long processing time from the CPU the user needs to use another type of job. This other kind of job will not be interactive and will send the regression onto a cluster to process it and dump some required files. This cluster is named at Synopsys as GRID and all the regression scripts need to be sent to it in order to avoid occupy a lot of processing resources on the interactive machines.

4.3.1 GRID

To launch regressions, after having a valid job granted, the user needs to request GRID access. To do so, another job, named "bnormal" needs to be requested. This will, again, launch a ticket to the server queue and wait for a set of machines (depending on the script) to be available. The script will then run on the cluster and dump the output files onto the user workspace.

The machines that compose the GRID are allocated to run regression scripts and are available within the Synopsys community. The user may send other scripts that are not regressions to the GRID, however this is not required within the Synopsys policy, as they may run on the normal interactive machines.

To have GRID access and run the regressions another command needs to be ran in the new terminal.

4.4 Projects Environment

At the IPK team all the projects follow the same work flow. Thus all share the same directory structure and the same scripts behaviour. This allows the workers to interact and help each other while developing and debugging the hardware designs.

As the scripts are the same among different projects it allows the IPK team members to focus on the team main tasks, such as:

- RTL design:
  - Controller integration;
  - Adding new blocks to the design;
  - Enhance existing RTL blocks for better performance.
- Simulations:

  - Adapting IP controller tests to the prototyping kit;
  - Run and use Simulations to replicate hardware problems.

- Synthesis:
  - Constraint the design;
  - Place and Route (P&R) enhancements.

- Tests.

Thus, to maintain a common work flow, every prototyping kit directory tree has the same structure (as shown in Figure 4.3).

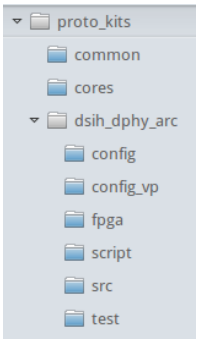


Figure 4.3: Prototyping kits directory structure.

4.4.1 Common

The *common* folder, shown in parallel with *cores* and *dsih_dphy_arc*, is an independent folder of the prototyping kit that is maintained by all users. It keeps all the scripts, hardware blocks, documents and other files that are used among the different projects.

Files present on *common* have one or multiple owners. This means that only people that are the owners of the file are allowed to edit it. This helps to report and correct bugs on all projects.

4.4.2 Cores

The *cores* folder provides the IPK team access to the IP controllers source code, testbenches, scripts and others. The files inside this folder are read-only to the IPK team and enable the user a faster integration of the IP on the prototyping kit.

4.4.3 Dsih_dphy_arc

The *Dsih_dphy_arc* folder stores all the files that are respective to the DSI Host prototyping kit. Although the files on this folder belong to this project, they share similarities among other projects and the way things are done inside it is the same in all prototyping kits.

4.4.3.1 Config and Config_vp

Config and *config_vp* are folders that store a Makefile, that when run configures the workspace, sets ambient variables, and copies files from *cores* and *common* to the *dsih_dphy_arc* folder.

The Makefile stored under *config_vp* is responsible for unpacking and encrypting the cores used on the DSI Host IPK:

- APB-I2C core;
- AXI-APB (also known as x2p) core;

These are dynamic cores that can be configured to several purposes. Thus, along with the Makefile it is stored a .config file that specifies the configuration parameters of the current project.

The DSI Host controller does not need to be encrypted, as the customers that buy the prototyping kit will have access to the IP source code. Thus, only the APB-I2C and x2p need to be encripted by the Makefile.

However, the DSI Host controller also needs to be configured and imported from the *cores* folder with a set of parameters. This is done by the

Makefile stored under *config* folder along with a .config file that stores the controller configurable parameters. Thus, it is always possible to configure the controller by editing this file and running the Makefile again.

4.4.3.2 FPGA

The *FPGA* folder has all the files needed to run synthesis and P&R. As the other folders it has a Makefile that contains some targets allowing users to run different configurations/synthesis.

After having the bitfile configuration as the Makefile output, it is passed to an FPGA board. The boards used at Synopsys are known as High ASIC Performance System (HAPS, see Figure 4.4) and are based on Xilinx Virtex FPGA family.

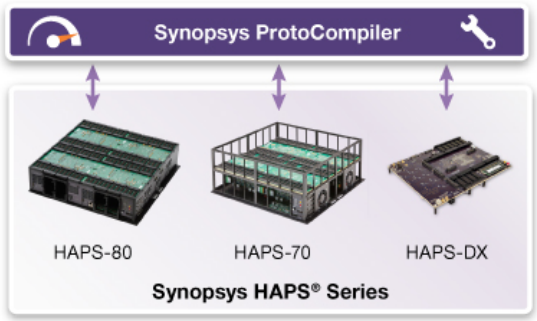


Figure 4.4: HAPS available systems [12].

As the FPGA are not Synopsys property there is no access to the P&R method used by Xilinx. Thus to run synthesis and P&R tasks, the Makefile will call ProtoCompiler and Vivado. ProtoCompiler (see Figure 4.5) is a Synospsys developed software that translates the RTL source code into a netlist.

Netlist's syntax are standard among electronics circuits. In its simplest forms netlists consist on a list of pins of the electronic components and a list of electrical conductors that interconnect them.

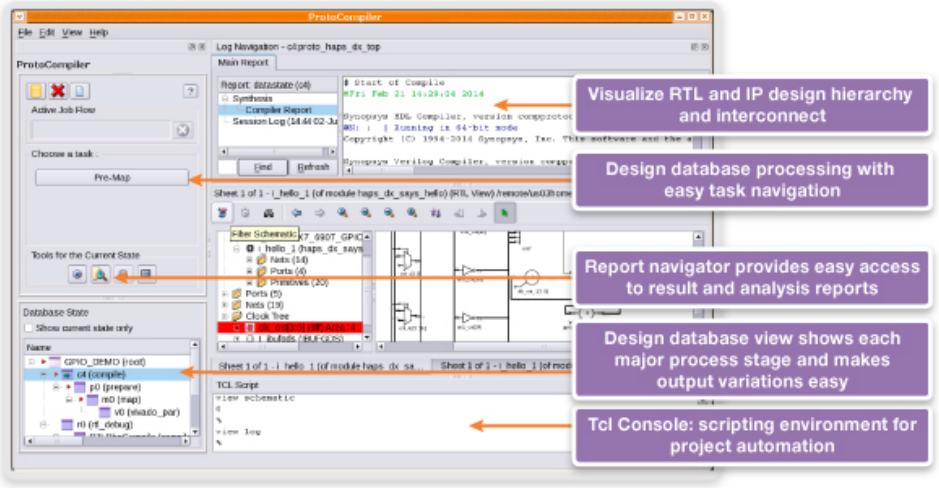


Figure 4.5: ProtoCompiler Software [13]

After having the netlist, Vivado (Figure 4.6) performs the P&R task and generates the bitfile needed to configured the designed circuit into the FPGA present on HAPS.

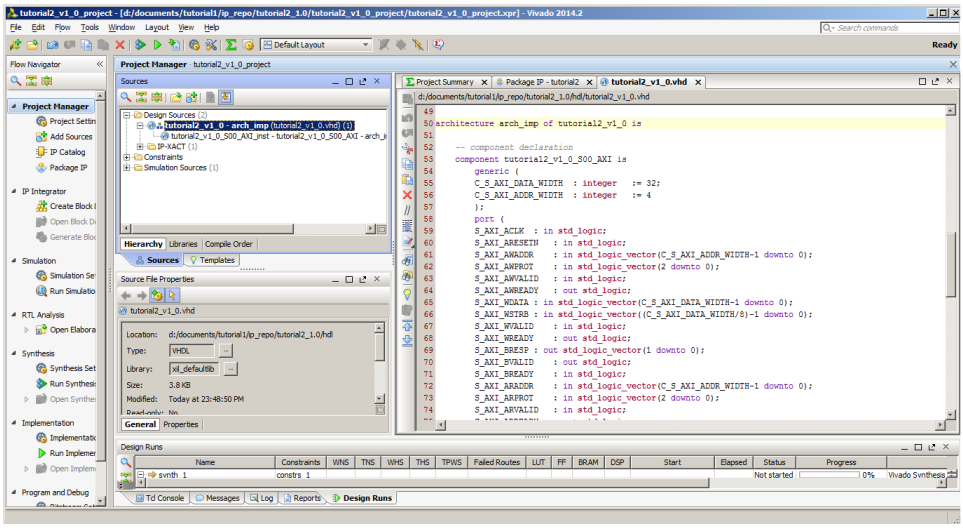


Figure 4.6: Vivado GUI.

As both of them support tcl scripts, the Makefile will not need to open the GUI of both software applications in order to run synthesis and P&R.

Instead, it will call the tcl scripts and run all the flow automatically. The synthesis process is known as a regression and for that reason it needs to be ran at the GRID.

4.4.3.3 Script

The *script* folder stores the prototyping kits scripts needed to set the environment.

When accessing ilight or iheavy machines, the licenses are still not granted to the users. To access them, the user needs to issue a command on the prompt in order to request the required software license. For example, when wanting to access ProtoCompiler license the user needs to issue the following command:

```
1 [prodrig/scripts]# module load protocompiler
```

The scripts, on the *script* folder, will load the needed licenses to run the DSI Host prototyping kit regressions.

4.4.3.4 Src

The *src* folder stores all the source files related to the prototyping kit RTL. From the top level to the simplest blocks all the source files are stored inside this folder. All but the controllers, that are stored under the *cores* folder.

Along with the source files there is a .lst file that points to every RTL file needed to the project. In this way, it is possible to have files that are not under the *src* folder (for example the controllers) and files that are under *src* folder and are not used.

4.4.3.5 Test

The *test* folder includes all the testbench files. Inside it has a Makefile with several targets to perform all supported simulations. For instance, when

wanting to run a test to the DPI interface the following command must be ran:

```
1 [prodrig/test]# make test_vtb_dsi_dpi
```

In this way, only DPI interface is tested. If all the tests are to be run the user just needs to type "make" on the prompt as the default target will run all the simulations.

The simulations use a Synopsys property software named VCS. VCS runs the tests pointed by the scripts and dumps the results to dynamic folders with the test name. Within the dumped files, there are a log of the simulation and a waveform database (.vcd or .vpd).

To open the database, the user needs to use another Synopsys tool that is framed inside the VCS tools package named Discovery Visualization Environment(DVE). DVE is an advanced, full-featured debug and visualization environment. Inside it, it is possible to better analyze the RTL source code and perform advanced debug such as, trace a driver of a net or find out what is causing the state 'X' on a logic cell. DVE has been specifically architected to work with all of the advanced bugfinding technologies in VCS and shares a common look and feel with other Synopsys graphical-based analysis tools. DVE enables easy access to design and verification data along with an intuitive drag-and-drop or menu-and-icon driven environment [14]. Figure 4.7 shows the DVE GUI and some of its features.

With the waveforms and the test log, the simulations are debugged and the behavior of the Design Under Test (DUT) validated.

The files included within the mentioned folders, in all the projects, are stored in servers and are accessible to the Synopsys users to download. The management tool used in Synopsys to download and upload the files to these servers is named Perforce.

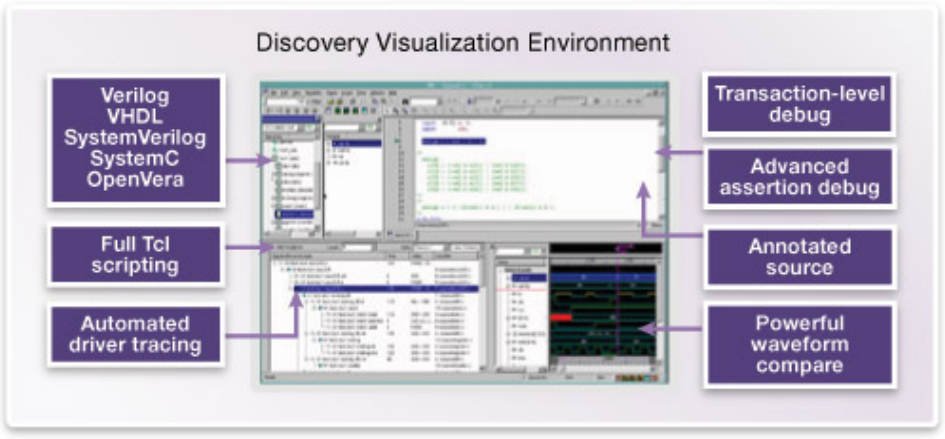


Figure 4.7: DVE software [14].

4.5 Perforce

All the files that form the projects are stored using a tool named Perforce. It allows the users to keep versions of the work preventing code losses. Thus, Synopsys designers work freely without taking risks of losing files.

The projects are stored under master directories. These are the root to all the prototyping kits and are called "depots". The depot stores all the versions of the projects allowing the user to sync the desired one to a workspace.

To access the files stored on depot, a client needs to be created specifying the files/projects to be synchronized with his workspace. One client is needed to sync files from the depot . Nevertheless, a user may have more than one client, for instance, to sync different projects into different workspaces.

To sync the desired files, specified when creating the client, the user needs to go to the workspace and use the Perforce sync command. The following prompt lines show the process of synchronisation of all the files from the depot to the workspace:

```
[prodrig/]# cd $WORKSPACE
[prodrig/workspace]# p4 sync ...
```

After synchronizing from depot, all files are in read-only mode to prevent the users to edit them. In order to edit the files, the user needs to run a Perforce command to inform the depot that the client is editing the file:

```
[prodrig/workspace]# p4 edit file.txt
```

Thus, the file opens for edition and lets other clients with the same file synchronized to know that conflits may occur. After the desired file edition, the user may submit the file to the depot using the command:

```
[prodrig/workspace]# p4 submit file.txt
```

While submiting the files to depot, Perforce forces the user to write a comment about that submission. This comment is stored on Perforce database along with the submited version of the files and helps the users to understand what was edited on that submission.

This page was intentionaly left in blank.

Chapter 5

Project Development

As mentioned in the introductory chapter, two projects, developed in cooperation with Synopsys, are described in detail in this chapter.

The aim of both projects is the integration of the PHYs with the DSI controller and their validation using real hardware. As a base to the projects the latest DSI prototyping kit was synced from the Perforce depot. Figure 5.1 shows the block diagram of the DSI prototyping kit available at the beggining of the project.

As seen in Figure 5.1, the available prototyping kit was designed with two DSI Host controllers and therefore targeting two PHYs. In this way it can target two displays with up to 4 lanes each, using different virtual channels. Each of the PHYs is directly driven by the PHY interface, which means that every signal that goes from the design to the PHY or from the PHY to the design passes through it. The design was also prepared for AXI communication allowing processor control over the registers. A Video input that passed through a video bridge was available, allowing the prototyping kit to connect to other video protocols such HDMI or CSI.

Based on this design, it is visible that the RTL is not a standalone solution. Therefore the prototyping kits, along with the RTL design, possess PHY integration and the software component. The software, based on an

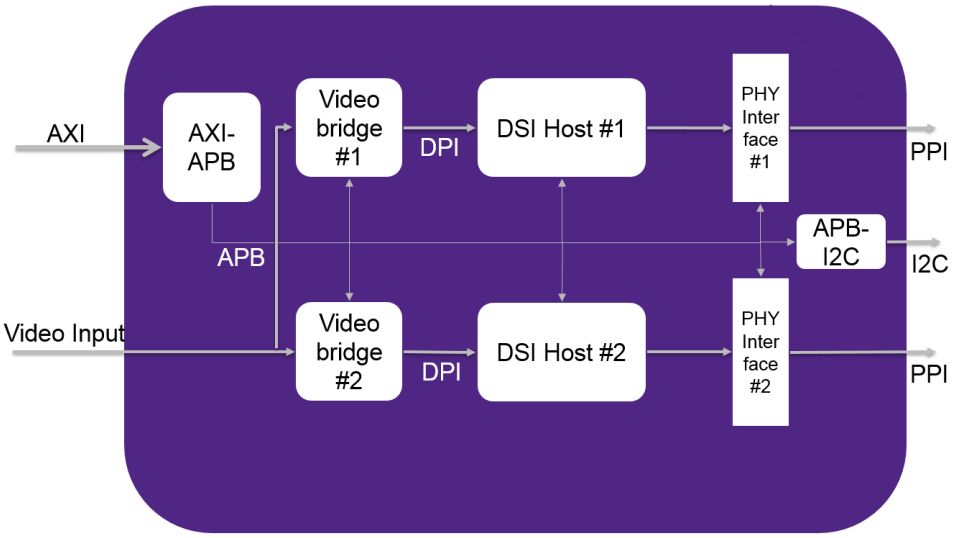


Figure 5.1: DSI prototyping kit.

ARC processor (see Figure 5.2), connects to the RTL design through AXI to program the hardware registers. The mixed signal PHY converts the PHY Pixel Interface (PPI) digital information into analog to be transmitted through the data lanes.

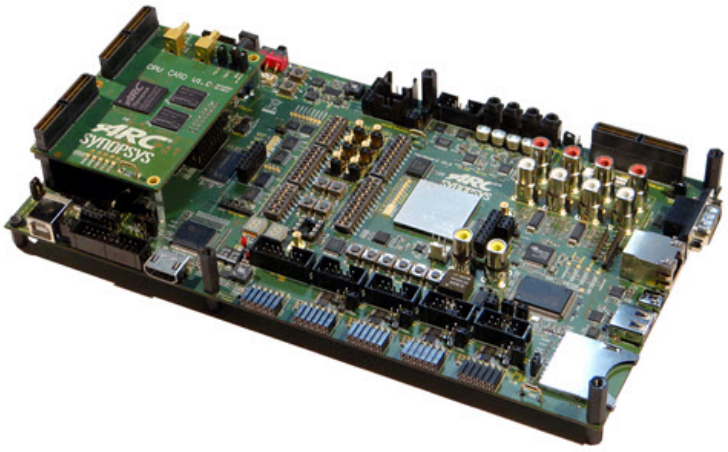


Figure 5.2: ARC-SDP Board.

With the three boards integrated in one prototyping kit (HAPS, ARC and PHY) it is possible to emulate a full stack solution that could be implemented in a SoC. Figure 5.3 shows the normal prototyping kit structure.

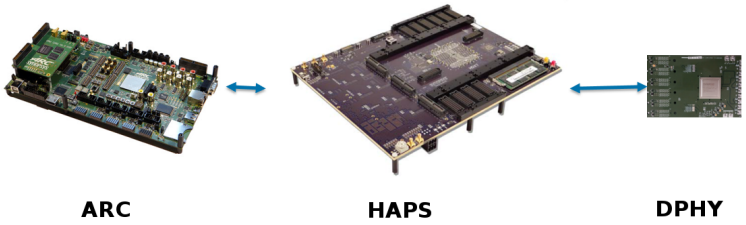


Figure 5.3: Prototyping kit structure.

Although it is called PHY to the board (in which it is actually contained), it is important to refer that the real PHY is contained inside a testchip[1] that is soldered to the board.

5.1 Project requirements

For the internship, Synopsys proposed two hardware validation designs, one targeting a client company PHY, that required the Synopsys services to validate their testchip, and other targeting a 3rd Generation Synopsys PHY.

Thus, for the hardware validation purposes it was proposed to:

- change the DSI prototyping kit design in order to match the PHY's pinout;
- run the available simulations and analyse the hardware behaviour;
- change the Synthesis constraints;
- validate in real hardware the interoperability of the PHY within the kit.

[1]The testchip is a wrapper that comprises the PHY itself and some aditional logic needed for correct functional behaviour

5.2 RTL changes

Whenever there is a change to any of the parts of the prototyping kit, RTL changes are required. On these projects changes were nedded as the PHY was different from those used among the prototyping kit.

As only one PHY was to be tested in both of the projects, the first required modification was to remove one controller, one video bridge and one PHY interface from the design. Thus, no logic inside the FPGA was wasted and the synthesis and P&R tasks run faster.

To do so, and to keep supporting the two PHY feature from the prototyping kit, one *define* (IPK_DUAL_PHY) was added to the design constants. Using *define* and *ifdef* or *ifndef* in Verilog, it is possible to add or subtract code to the design. Therefore this constant, whenever defined, adds to the design the second instance of the controller, video bridge and PHY interface.

Figure 5.4 shows the design change after adding the *ifdef* of the constant.

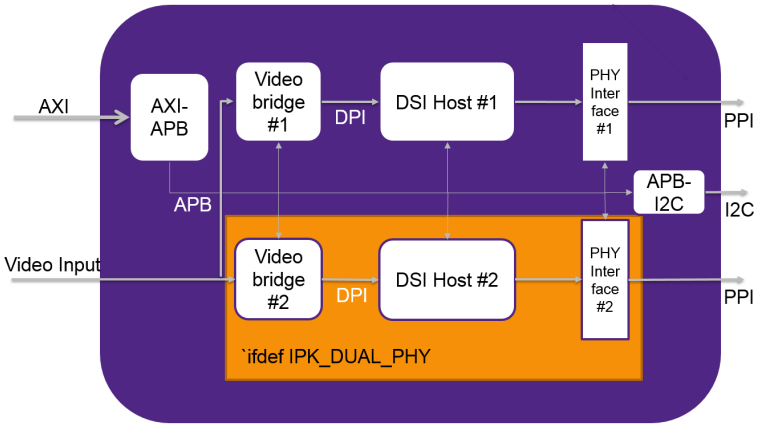


Figure 5.4: Design when added the *ifdef*.

After the removal of the logic that was driving the second PHY, there was the need to adapt the remaining RTL to support the new PHYs. Thus, a comparison between prototyping kit PHY and the new PHYs was done in order to understand the differences.

To support the new PHYs the main changes needed were to the pinout and to the configuration interface. As the pinout had changes in both PHYs the configuration interface remained the same in the Synopsys PHY. On the other hand, the client PHY was configured through APB.

5.2.1 Client PHY

The first changes were done to the project envolving the client PHY. An analysis was done to the prototyping kit available pins to understand how they matched the PPI of the new PHY. Some of the available pins were the same and matched the new PPI interface, others needed to be added and others removed. For this reason, changes were done from the top level module to the controller to match the new PPI. This was done, again, using a define to this project (IPK_IAG_PHY) keeping the support for the prototyping kit PHY. It allows to remove the undesired pins with *ifndef IPK_IAG_PHY* or add pins that are only used by this project with *ifdef IPK_IAG_PHY*. The pins that were not on the design were included on the PHY interface register bank allowing them to be toggled through software.

Besides the PPI interface, this PHY had the particularity of being configured by an APB interface. Therefore all APB signals needed to be extended to pass from the PHY interface to the top level pinout. As mentioned in section 2.2.1.2, each peripheral needs to have a PSEL pin connected. In the DSI Host prototyping kit the management of the PSEL pins is done by the AXI-APB block based its .config file stored in config_vp. The .config file specifies which PSEL must be asserted to each register:

- From 0xD0000000 to 0xD0000fff - design main APB memory (PSEL0);
- From 0xD0001000 to 0xD0001fff - APB-I2C block (PSEL1);
- From 0xD0002000 to 0xD0002fff - DSI Host controller #1 (PSEL2);
- From 0xD0003000 to 0xD0003fff - Video bridge #1 (PSEL3);
- From 0xD0004000 to 0xD0004fff - Reserved (PSEL4);
- From 0xD0005000 to 0xD0005fff - DSI Host controller #2 (PSEL5);
- From 0xD0006000 to 0xD0006fff - Video bridge #2 (PSEL6);
- From 0xD0007000 to 0xD0007fff - Reserved (PSEL7);
- From 0xD0008000 to 0xD0008fff - Reserved (PSEL8);
- From 0xD0009000 to 0xD0009fff - PHY Interface #1 (PSEL9);
- From 0xD000A000 to 0xD000Afff - PHY Interface #2 (PSEL10);

As AXI is a 32 bit address protocol and all the APB peripherals had only 12 bit addresses, on the block AXI-APB configuration is described that, to each AXI address corresponds a PSEL pin and its 12 least significant bits. This allows to connect multiple APB peripherals using bits 12 to 15 to define the peripheral the processor is reading/writing to.

The PHY Interface, through which all the PHY signals pass, had PSEL9 and 12 bit address connected. Thus there was no need to add another PSEL to connect directly to the PHY. Instead, as the PHY interface did not need the 4096 register addressing (0x000 to 0xFFF) some logic added to it could split the two peripherals. As the client PHY needs more addresses than the PHY interface, the registers were split on the following way:

- 0x000 to 0xEFF - PHY;
- 0xF00 to 0xFFF - PHY interface;

Figure 5.5 shows the added RTL logic to support the split version of the register adressing.

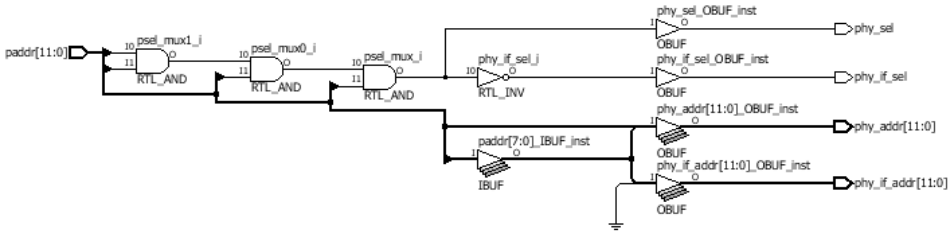


Figure 5.5: Logic added to split addresses and select the peripheral

As the PHY interface registers did not follow the new version of the addresses (0xF00 to 0xFFF) and an 8 bit adressing was enough then bits 11 to 8 were manipulated to the logic level '0' whenever the PHY Interface was selected.

After the RTL changes, the simulations to the PHY interoperability with the prototyping kit were performed, as described in section 5.3.

5.2.2 Synopsys PHY

The new Synopsys PHY was related to the 3rd Generation DPHY. This points to the specification version on which its RTL was based.

The 3rd Generation of DPHY aims to frequencies up to 2.5Gbps per lane, which allows to transmit video at higher resolution and with more frames per second.

DSI Host prototyping kit was designed to use a 2nd generation PHY. As this is a different PHY, the pinout and the registers were different from the previous, therefore requiring RTL and software changes to the prototyping kit.

When the testchip of this PHY was manufactured, two different PHYs were included inside it. One prepared to receive HS data (Rx), for proto-

cols like CSI2 Host, and another to transmit HS data (Tx) for DSI Host. Therefore, some of its pins correspond only to the Rx PHY and others only to the Tx PHY. This is a different architecture from what was designed to the 2nd generation PHY testchip included on the DSI Host prototyping kit. This one included bidirectional pins that could be an input or an output depending on the design it was connected to. Thus, the PHY board that connected to the HAPS had a couple more pins that needed to be added onto the design.

Figure 5.6 displays the pinout differences between the 2nd Generation testchip and the new 3rd Generation testchip.

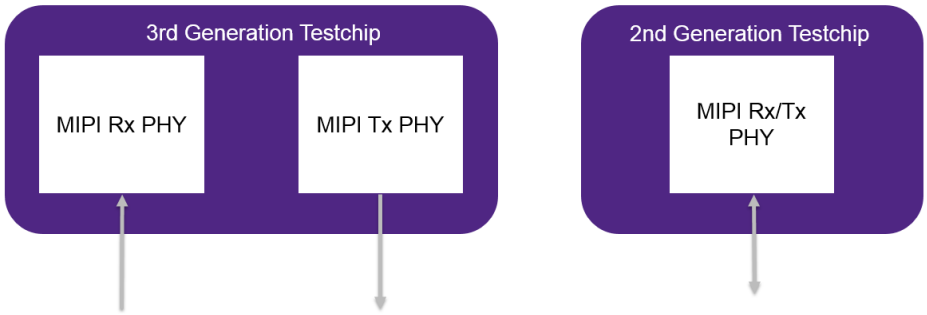


Figure 5.6: Testchip's differences.

As DSI Host is a transmitter, only the Tx PHY included on the 3rd Generation testchip is needed. However, in order to avoid unstable behaviour from the testchip interoperability with HAPS, all the pins were included on the design (Tx and Rx). Tx PHY connects normaly to the design propagating its inputs and outputs to the DSI Host controller. The Rx PHY inputs locked to a given logic level - for example the reset pin of this PHY (phyrstz, when '0' the PHY is in reset state) was stuck at the logic level '0' in order to not use it - and the PHY outputs were left with no load - only connected to the design but with no effect at all. Thus, the main differences

resided again on the PHY interface block, that had to support several new pins (from the Rx PHY) and propagate the Tx PHY pins to the DSI Host controller.

5.3 Testbench

The testbench is one of the important tasks that needs to be performed in order to validate the hardware behaviour and to find bugs.

At the IPK team, oppositely to the RTL, there is little testbench development. Instead the controller team files are used and adapted to the prototyping kit in order to validate it. To do so, the DUT is changed to the prototyping kit and then the stimulus and verification files are applied to it.

Figures 5.7 and 5.8 show what is done inside the IPK team to adapt the verification process of the controller to the prototyping kit.

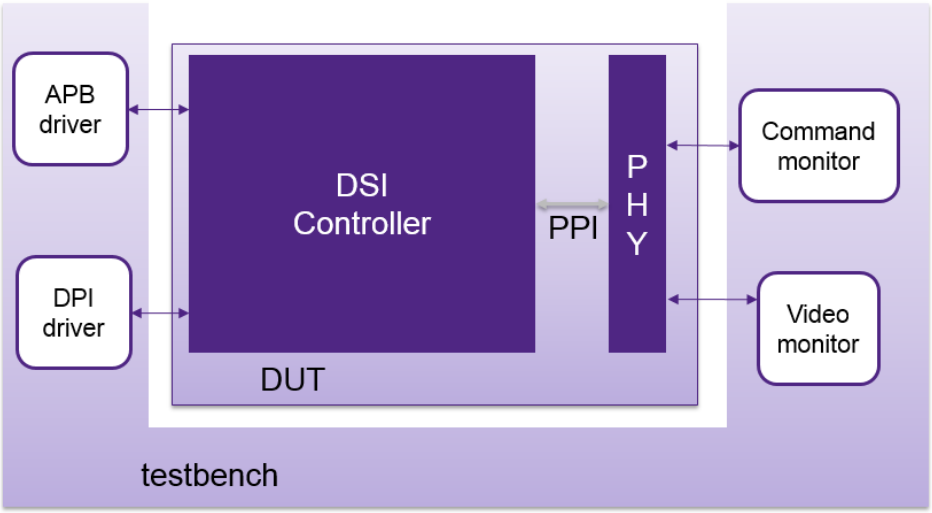


Figure 5.7: Testbench applied to the controller.

As shown, the testbench was "extended" to be applied to the prototyping kit ports. These are tasks done by the IPK team to perform the testbench - adapt the testbench from the controller to the prototyping kit. However,

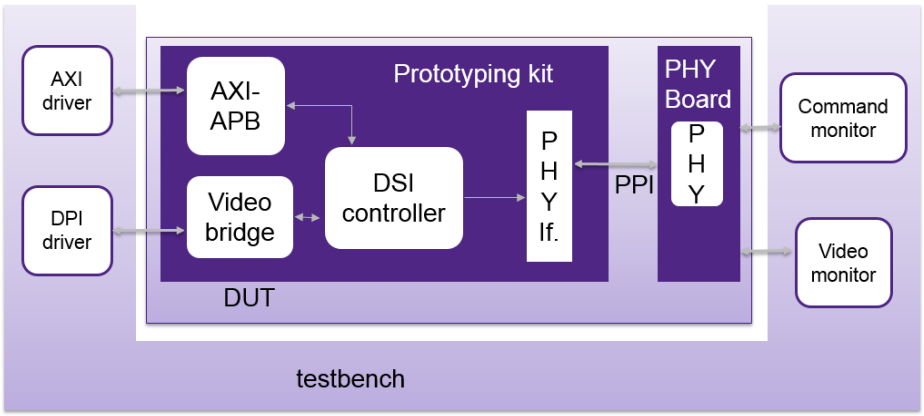


Figure 5.8: Testbench adapted to the prototyping kit.

the tasks that run on the controller testbench transmit data through APB and, as the prototyping kit has no APB interface, it is needed to adapt the APB transmitions into AXI. Therefore, the data is sent through AXI, and converted by the AXI-APB block into APB, mantaining the same stimulus to the controller.

With the integration of a new PHY, each project demands changes made to the DUT in order to match the hardware setup. Thus, modifications to the previous testbenches need to be performed to adapt the existing test cases to each of the PHYs.

5.3.1 Client PHY

The client PHY was difficult to integrate due to lack of information about the board and the testchip. An initialization sequence of the PHY was sent with APB transactions. However, as there were no register description, each APB write/read transaction was unexplainable. The PHY was seen like a black box and along with the initialization sequence it was sent an encrypted model of it to be integrated on the prototyping kit.

Along with the sequence initialization it was also needed to change the

PPI connection inside the DUT. However, the PHY board had more pins than the ones included on the encrypted model. Some of these from the board itself, like a clock that came from a cristal oscillator, others from the testchip logic. Therefore the board and the testchip logic were created, wrapping the PHY and adding the missing pins to simulate the correct behaviour of the real received hardware.

5.3.2 Synopsys PHY

In Synopsys PHY the main changes were again to the DUT and the initialization sequence. As there were more pins due to the two PHYs inside the testchip there was the need to include them on the DUT to test.

As this was a Synopsys PHY all the needed documentation was open for the IPK team to read and debug the PHY behavior. Also included with the documentation was register description, from the PHY and the testchip, and an initialization sequence example easing the simulation.

5.4 Synthesis and P&R

After the testbench validation on both of the projects it was needed to synthesize the projects in order to convert the RTL into a bitfile.

On Xilinx FPGA platforms it is common to use .ucf (user constraint file) to constraint the pinout of the RTL. As the HAPS platforms use Xilinx FPGA, the ucf must be created in order to lock the RTL pins into the IOBs (Input/Output Blocks). At Synopsys the IPK team uses a tool called hapsmap that generates the .ucf file giving as input other 3 files (.pas .con .map). These files create an abstraction to the user and turn the pinout constraints easier to do. They create a virtual path from the RTL design to the peripheral to which it will be connected. To do so, each file has a role:

- **.map** - emulates the peripheral that the user wants to connect to the

HAPS. On the case of these projects the desired peripherals to connect to the HAPS are the ARC and the PHY. Thus a .map file is needed for each of the peripherals, containing their pinout (according to which HAPS pin are connected to);

- **.con** - creates a macro to each .map file allowing to connect the RTL pins to each different board;
- **.pas** - with the created macro, it is possible to connect each RTL pin to a pin of the desired peripheral.

Figure 5.9 shows an abstraction of the hapsmap.

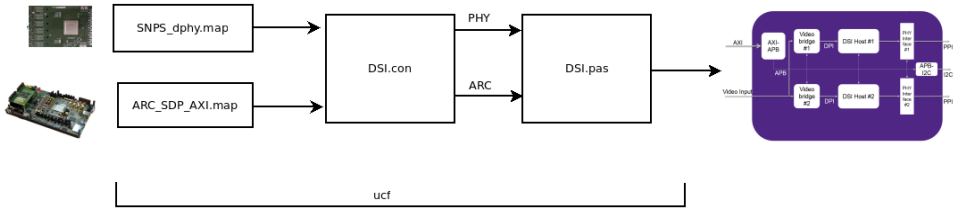


Figure 5.9: Hapsmap example.

These three files are combined by the hapsmap tool creating the ucf file.

To connect to the HAPS board, it is necessary to use a HAPS Track 3 (HT3) connector (see figure 5.10). The HT3 connector has a specific pinout that is known by the hapsmap tool. Thus the .map pins are converted into the FPGA specific pins to be placed in the .ucf file.

On both projects, the connection to the ARC-SDP processor remained the same. Therefore, no changes were done to its .map files. As the PHY was the changing variable on both projects, it was needed to have both .map files in order to synthesize the projects. The client company provided along with the hardware (PHY and board) a .map file. On the Synopsys PHY, as it is produced and tested by the LAB team, a .map was provided along with the board.

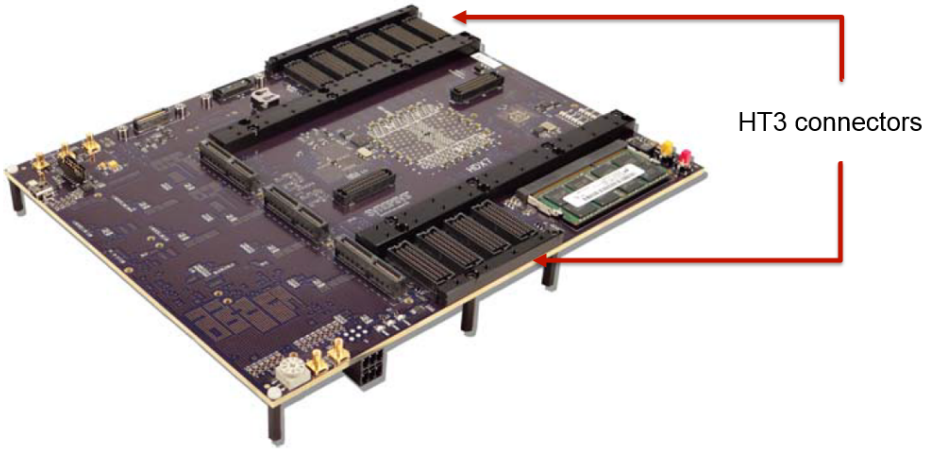


Figure 5.10: HT3 connectors location on HAPS board.

Whenever the hapsmap files were ready, the synthesis was run using hapsmap and Protocompiler to produce the netlist and the P&R with the Vivado tool to generate the bitfile.

5.5 Hardware tests

Having bitfiles ready, the tests began integrating the ARC with the software (developed by the software team at Synopsys), the HAPS with the generate bitfile and the PHY. To integrate and validate the systems it was also necessary a DSI Device - a display - to correctly evaluate the behaviour of the PHY. To do so, two different displays and a logic analyzer were used in order to validate the correct packet handling and the PHY compliance.

5.5.1 Client PHY

The client PHY required some debug to bring it up. Therefore, it was needed to return to the simulations more than once in order to match the stimulus that were occuring on the hardware and realize what was going

on. However, sometimes the testbenches were not enough as the bugs faced on hardware were not occuring in simulation. Thus, instrumentations[2] were run to better understand and debug the system.

The target display, as a DSI Device, works at a constant frequency of 500 Mbps per lane and supports only the RGB 8-8-8 data type. As the tests (agreed with the client) only involved the interoperability of the PHY on the DSI Host prototyping kit with this display as target, only the 500 Mbps frequency and the RGB 8-8-8 data type were used to test the PHY. Thus, the tests required by the client involved:

- validating the configuration of the display through LP communication with the PHY;
- 4 different patterns to be displayed on the screen (Figures 5.11, 5.12, 5.13 and 5.14);
- BTA request.

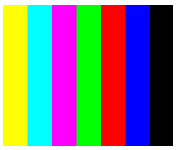


Figure 5.11: Color bar pattern

Figure 5.12: Red pattern

Figure 5.13: Grey pattern

Figure 5.14: Crosshair pattern

[2]Instrumentation is the process of debugging the hardware in real time. It is a feature present on Protocompiler and the HAPS boards that is able to capture the FPGA internal signals and dump them into a database for debug processes.

These tests were run to validate the PHY by the presented order. Although the display configuration was done correctly and validated through a scope, the patterns were not appearing on the screen. Thus, some troubleshoot was done with a protocol analyzer to understand what was wrong with the packets sent to the display. On the protocol analyzer the packets seemed good and the pattern image was being received correctly. However, the display was unable to capture them.

After some tests, it was possible to understand that the mistake was being done when passing from LP to HS. The sequence performed by the PHY was defective as there was a shift on the first clock sent by the Clock Lane. After discussion with the client it was possible to invert this with a register write and therefore validate the correct behavior of the PHY with the correct image output to the display. Figure 5.15 shows the output image of the system being presented on a display.

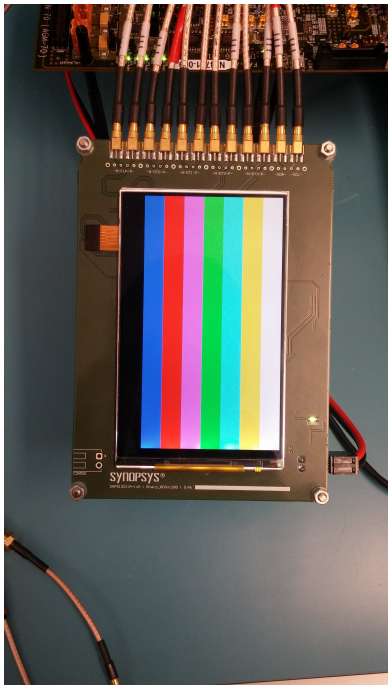


Figure 5.15: Client PHY image output.

5.5.2 Synopsys PHY

The Synopsys PHY, after being integrated on the simulations and having the bitfile ready, immediatly worked on hardware. One bug was found on the PHY board due to additional resistors connected to the I2C ports that affected its behaviour.

Figure 5.16 shows the full setup assembled for the PHY hardware validation.

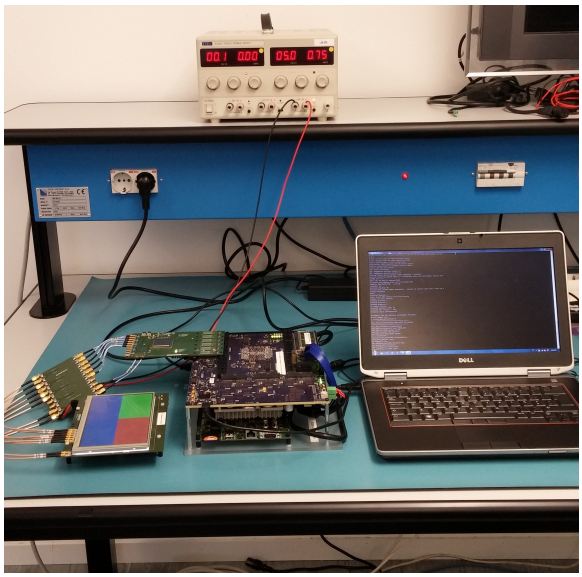


Figure 5.16: Synopsys PHY full setup.

As this was a new Synopsys PHY, the tests performed required more detail than the ones on the client PHY. For this PHY, it was necessary to determine its maximum frequency of operation with the DSI Host prototyping kit and if data types were transmitted correctly using a display (as seen in Figure 5.16) and a protocol analyzer (see figure 5.17).

The specification of the D-PHY determines that the 3^{rd} generation should reach 2.5 Gbps per lane. However, the DSI Host needs to keep up with a frequency of 312.5 MHz ($\frac{2500Mbps}{8}$), which is a frequency too high to run on a FPGA with a design like this. Thus, the tests were run from the 80 Mbps

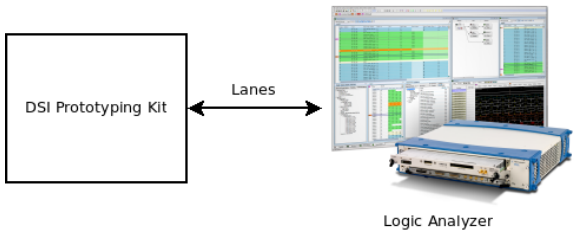


Figure 5.17: DSI Host tests using a protocol Analyzer.

to the maximum frequency allowed by the design that settled at 572 Mbps. This value can be improved in the future by adding some enhancements to the design constraints.

This chapter presented the workflow at the Synopsys IPK team and some of the aspects addressed on both projects. The next chapter focus the Synopsys internal project management, how hardware is shared and how project dates are handled.

This page was intentionaly left in blank.

Chapter 6

Project Management

In any business, management is very important to support the work flow and accomplish the company/team goals. Thus, this chapter gives an overview of what is the management process at Synopsys that impact IPK team.

At Synopsys there is a set of values followed by the company and by the people who work there: Integrity, Customer Success through Execution Excellence, Leadership, and Passion. Figure 6.1 shows the Synopsys pyramid of values.

Figure 6.1: Synopsys pyramid of values [15].

These are the values followed by Synopsys employees and taught to the newcomers. They are followed in Synopsys Management decisions and are the basys of Synopsys as a company.

Synopsys is more than an IP development company. Along with it, Synopsys is well known by its software tools (like VCS or ProtoCompiler) and its services. Thus, the company has a great number of teams that cover several areas on the silicon industry. However, no matter what part of Synopsys an employee belongs to (IP, IPK, Software tooks, Human Resources, etc.), all the workers are treated equally. In this way, when a new employee, intern or contractor begins its work, in one of Synopsys offices across the globe, his/her integration is facilitated by a background structure (such as Human Resources, IT, etc.). At the IPK team, newcomers have to read a set of documents and trainings whose aim is to help them to understand how the company works, to give them information about tools they have at their disposal or simply to welcome them. Trainings are given to clarify several subjects, from tools usage to ergonomics. The newcomer has also a "*Buddy*" inside the company, a person that guides and helps through the integration process, clarifies the doubts and follows her/his work on the first weeks. To better integrate newcomers, and to help the workers through the rest of their career, good management politics are required at Synopsys.

6.1 Management

The management at Synopsys is done hierarchically, which means that employees are ranked at various levels within the organization. Each level is one above the other and at each stage in the chain, one person has a number of workers directly under them, within their span of control. Figure 6.2 shows what a hierarchical organization chart looks like [16].

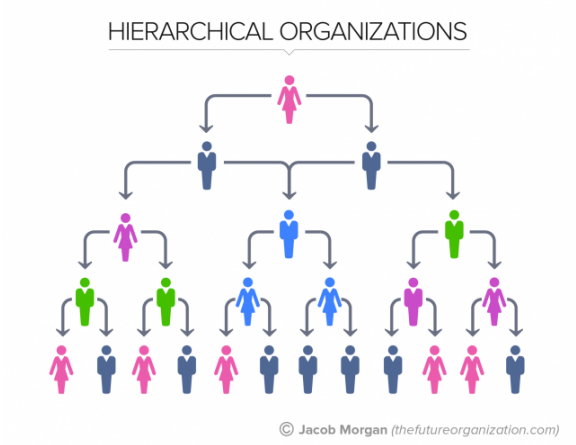


Figure 6.2: Hierarchical organizations [16].

The person that has others directly under his/her control, guides them and passes information from the top hierarchies. In this way it is possible to synchronize big teams and accomplish large scale projects. The IPK team at Porto is a good example of this hierarchical chart as it is composed by three sub-teams, the Hardware team, the Software team and the Test Automation team. Figure 6.3 shows the IPK team hierarchical chart.

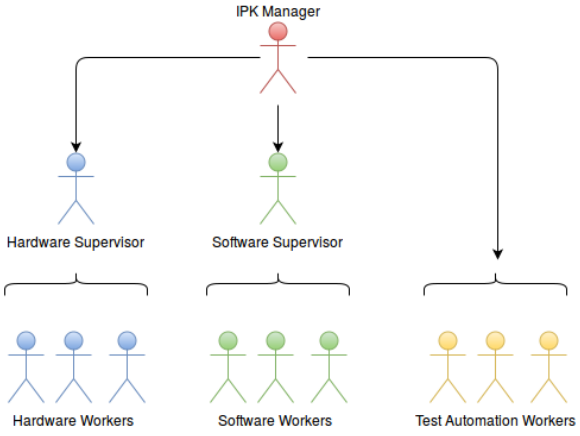


Figure 6.3: IPK team Hierarchical chart.

As represented in Figure 6.3, all the teams share the same manager. In this way it is simpler to synchronize projects and to keep ideas and

information aligned among the three teams. However, each one is a sub-team of the IPK team and, for that reason, a person that follows and guides the specific work of each sub-team is needed. Hardware and Software teams have a supervisor that is responsible for managing their sub-team work, Test Automation is managed directly by the IPK manager. To synchronize the work among these different sub-teams there are meetings on which the projects are discussed. Nevertheless, inside each sub-team there are groups for the projects, for instance MIPI, DDR and HDMI. Thus, to keep track of the projects' status, to help and clarify doubts or to brainstorm, there are some different meetings that happen occasionaly on the IPK team among sub-teams and projects.

6.2 IPK Meetings

The meetings are an important part of the management process at Synopsys. They help the workers and the management entities (Managers and Supervisors) to keep track of the projects, to help solving problems that may stall the projects and to share ideas. This lead to different types of meetings that involve different subjects and people. Most of the meetings occur once a week and last, in average, an hour.

The Hardware sub-team has a weekly meeting that involves every person working on this team, to brainstorm and exchange ideas about the projects. The main purpose of this meeting is to solve hardware problems. Thus, each person, in turn, exposes the problems that are facing and the other members of the team present solutions and ideas to solve (or debug) those problems. This meeting is also intended for the information to flow from the Supervisor to the rest of the team. In this way, beyond the solving problem part, this meeting may also include discussions about team changes (like switching projects) or about the company itself.

Also inside the Hardware sub-team there are meetings that involve group

members, like a MIPI hardware meeting or HDMI hardware meeting. The purpose of these meetings are to track the of RTL development. As these are more restrict meetings, with people that works on the same subjects, it is possible to share much more detailed ideas about projects - for instance, share ideas about packet formats on MIPI.

As almost all the projects demand hardware and software, meetings that involve both teams are also needed. These meetings are very important since they synchronize both teams during the project duration. Thus, each week inside the IPK team there are:

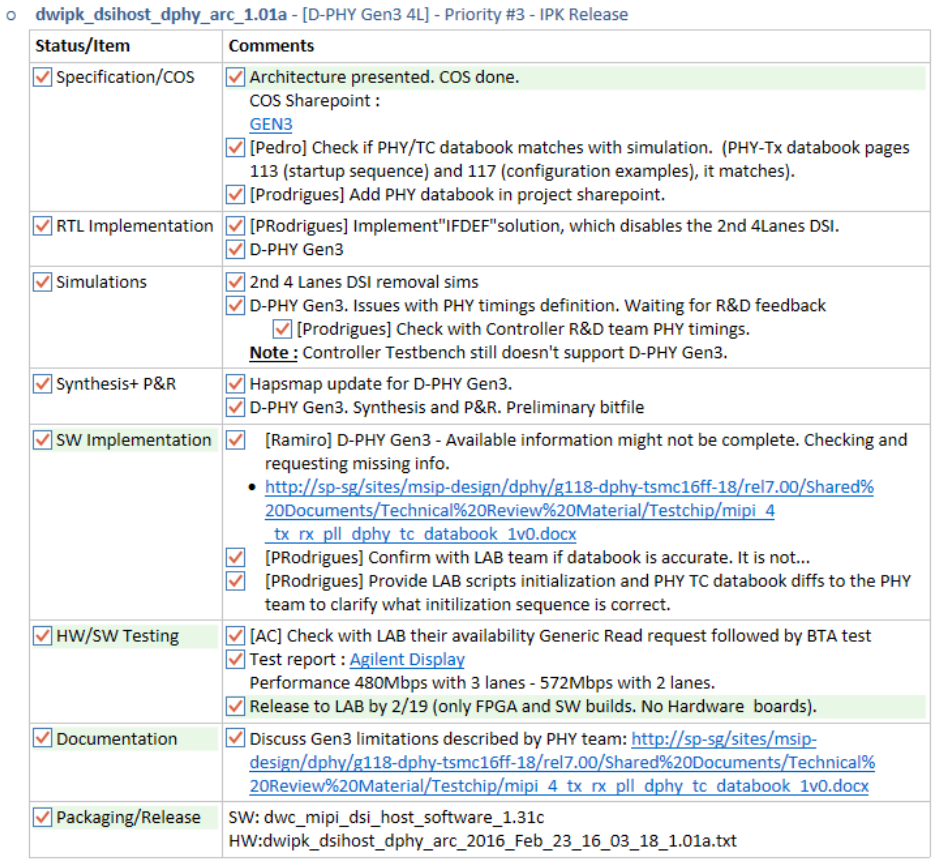
- MIPI hardware and software meeting;
- HDMI hardware and software meeting;
- DDR hardware and software meeting.

In these meetings participates the people that work on the same projects, inside both teams, along with the supervisors and the team manager. They help to keep track of the projects, in terms of hardware and software, to synchronize both parts and to clarify management subjects.

6.2.1 MIPI Hardware and Software meeting

On the MIPI hardware and software meeting are discussed projects like DSI Host, CSI2 Host, CSI2 Device, UFS and others. They, have a shared page that everyone can access to see and update the status of the projects. This is where the track of the project is done and where tasks are assigned.

Figure 6.4 shows the page used for tracking the DSI Host interoperability with Synopsys 3^{rd} generation PHY.



dwipk_dsihost_dphy_arc_1.01a - [D-PHY Gen3 4L] - Priority #3 - IPK Release

Status/Item	Comments
☑ Specification/COS	☑ Architecture presented. COS done. COS Sharepoint : GEN3 ☑ [Pedro] Check if PHY/TC databook matches with simulation. (PHY-Tx databook pages 113 (startup sequence) and 117 (configuration examples), it matches). ☑ [Prodrigues] Add PHY databook in project sharepoint.
☑ RTL Implementation	☑ [PRodrigues] Implement"IFDEF"solution, which disables the 2nd 4Lanes DSI. ☑ D-PHY Gen3
☑ Simulations	☑ 2nd 4 Lanes DSI removal sims ☑ D-PHY Gen3. Issues with PHY timings definition. Waiting for R&D feedback ☑ [Prodrigues] Check with Controller R&D team PHY timings. **Note :** Controller Testbench still doesn't support D-PHY Gen3.
☑ Synthesis+ P&R	☑ Hapsmap update for D-PHY Gen3. ☑ D-PHY Gen3. Synthesis and P&R. Preliminary bitfile
☑ SW Implementation	☑ [Ramiro] D-PHY Gen3 - Available information might not be complete. Checking and requesting missing info. • http://sp-sg/sites/msip-design/dphy/g118-dphy-tsmc16ff-18/rel7.00/Shared%20Documents/Technical%20Review%20Material/Testchip/mipi_4_tx_rx_pll_dphy_tc_databook_1v0.docx ☑ [PRodrigues] Confirm with LAB team if databook is accurate. It is not... ☑ [PRodrigues] Provide LAB scripts initialization and PHY TC databook diffs to the PHY team to clarify what initilization sequence is correct.
☑ HW/SW Testing	☑ [AC] Check with LAB their availability Generic Read request followed by BTA test ☑ Test report : Agilent Display Performance 480Mbps with 3 lanes - 572Mbps with 2 lanes. ☑ Release to LAB by 2/19 (only FPGA and SW builds. No Hardware boards).
☑ Documentation	☑ Discuss Gen3 limitations described by PHY team: http://sp-sg/sites/msip-design/dphy/g118-dphy-tsmc16ff-18/rel7.00/Shared%20Documents/Technical%20Review%20Material/Testchip/mipi_4_tx_rx_pll_dphy_tc_databook_1v0.docx
☑ Packaging/Release	SW: dwc_mipi_dsi_host_software_1.31c HW:dwipk_dsihost_dphy_arc_2016_Feb_23_16_03_18_1.01a.txt

Figure 6.4: Page used in DSI Host interop with 3^{rd} generation PHY project.

These pages start blank, with no marked items. Whenever a task is done successfully, its item is filled with a visa marking their completion.

As it is visible in Figure 6.4, the project is divided in stages that need to be completed for it to be finished. Project stages are labeled on Figure 6.4 as **Status/item** and are targeted to software and hardware teams. Each of these has comments and tasks that are assigned throughout its duration as seen in Figure 6.4 as **Comments**.

6.3 Project Stages

The stages that each project follows are:

- Specification/Change Of Specification (COS);
- RTL Implementation;
- Simulations;
- Synthesis + P&R;
- Software Implementation;
- Hardware and Software Tests;
- Documentation Development;
- Packaging and Release.

These take place by the described order with one exception: Software Implementation, which occurs in paralel with the initial tasks (see Figure 6.5).

At the beggining of a project the hardware responsible member needs to write a document - Specification - in which are registered the details of the project. However, if an old/previous project is taken as a basis for a new one, it is called Change of Specification. For instance, on both DSI projects reported in this document, the base was the DSI Host prototyping kit. Therefore there was the need to write a COS.

When the specification/COS document is ready, it triggers the RTL and Software implementations. The software implementation occurs in parallel with the hardware stages of RTL implementation, Simulation and Synthesis + P&R. Whenever both teams have these stages completed, the tests begin. The tests stage may be the one that requires more time, depending on the required debug.

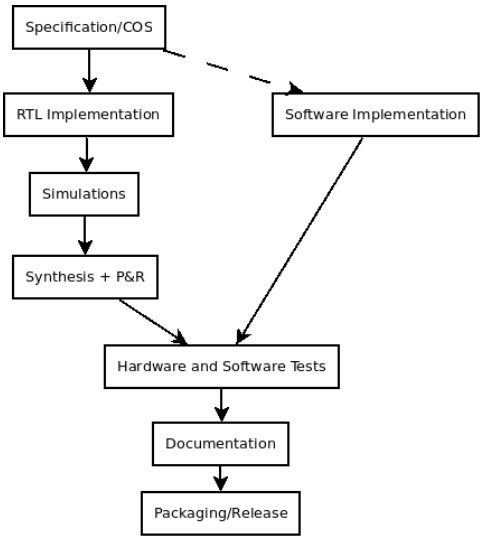


Figure 6.5: Hardware and Software stages chart.

After tests are completed, the documentation stage follows. The Documentation stage differs among the projects. When it is a prototyping kit project, it involves some documents that report the kit usage, how the clients should assemble all its parts together and how to change it at their will. When it is a hardware validation project, documentation is reduced to a unique test report that states the required tests. The projects reported on this document, regarding DSI Host interoperability with two PHYs, were hardware validation projects. Therefore both of the projects were released with a test report document.

The last stage is the release/packaging that, such as the documentation, may differ from project to project. On the prototyping kit projects, the release is done by packing all the files into a single one (a file with .run extension that when run, unpacks all the files) that includes documentation, software and hardware source files. On the hardware validation the release is variable, sometimes it may be just the test report itself; others it may be the binary files that are used in HAPS and ARC systems.

Whenever all these stages are completed, the project is concluded.

6.3.1 Projects Management

All the projects are timely scheduled by the management entities - sometimes with the help of the Marketing team that better knows the clients needs. Thus, on the meetings, the manager or supervisor assigns the newer projects to the teams.

As new projects appear, they are assigned and placed in a shared document (PoR) to which every worker as access. This document is constantly updated because new projects are always being assigned to the IPK team.

Documents like PoR are stored at Synopsys using SharePoint (see Figure 6.6).

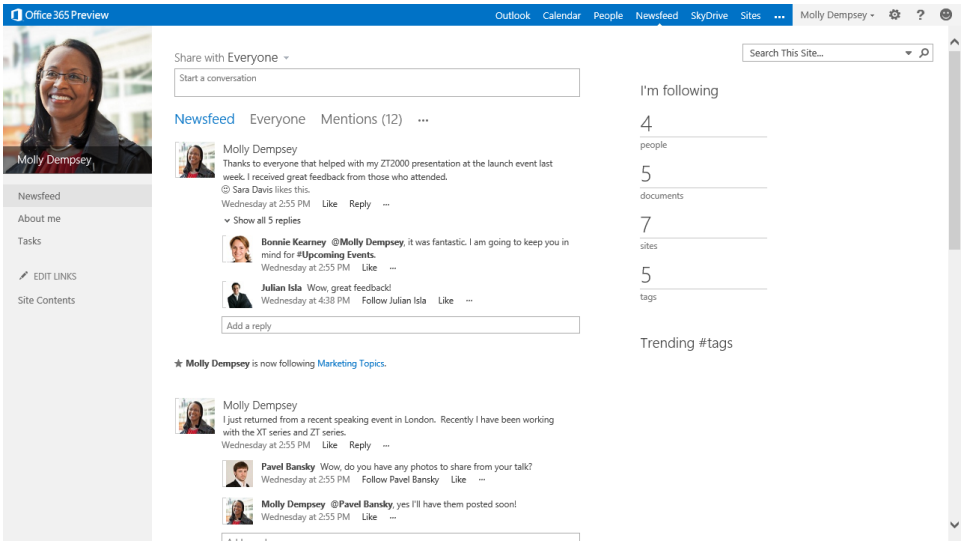


Figure 6.6: SharePoint[17].

Each team at Synopsys has its own SharePoint page to store and share documents among projects. These SharePoint pages are splited by the team manager to better organize them. In this way, everyone has access to the documents from the different projects.

Besides sharing documents, at SharePoint there is a section called Issue Tracker. Its main purpose is to report problems or bugs found on the sys-

tems. Whenever a new problem or bug is found, it is placed in Issue Tracker and it is assigned to the owner of the project to solve it. The owner receives an e-mail with the problem details and has the duty to solve it and update its status.

This chapter reported, in short, some management methodologies used at Synopsys and by the IPK team, to improve the products and the work quality. The final chapter presents the conclusions and future work for the MIPI DSI Host prototyping kit.

Chapter 7

Conclusion

This chapter presents an overview of the work developed during the internship, presenting its major gaps along with the future work to correct/improve them.

Synopsys, as one of the leadership companies in the silicon industry, is in constant improvement to present to its clients better products and new solutions. In this matter, the prototyping kits are an enhancement, made on the IP production, that accelerate time to market and ease the controllers use. These provide consistent examples on IP use cases allowing the client to modify and adapt it to its systems and designs.

During the internship time at Synopsys, topics and subjects were studied in order to successfully accomplish the proposed projects. The developed designs had to take into account some internal commonly used blocks, for instance AXI-APB (to communicate between ARC system and the inner blocks of the design), attributes of the MIPI DSI protocol, such as support from one to four data lanes, and incompatibilities between the controller and the PHY, for example the pinout (that had to be changed on the PHY interface block in both designs).

Both projects were completed within the scheduled time limit, presenting minor gaps that can be corrected in the future. Regarding the Synopsys

PHY, the maximum frequency obtained was lower than expected as the previous design (with a 2nd generation PHY) was communicating at almost the same bit rate. To improve this frequency an analisys needs to be performed to the design constraints in order to understand how the critical paths timings can be shortened. On the client PHY, only the referred tests were performed (see section 5.5.1), however some unpredicted software changes were added in order to prevent issues like contention, controller interruptions and bad image on the display panel.

Bibliography

[1] Networks mania *OSI Model- OSI 7 layers*, 2012.

[2] MIPI Alliance, "About mipi," *http://mipi.org/momentum*, 2016.

[3] MIPI Alliance, "Mipi working groups," *http://mipi.org/working-groups*, 2014.

[4] G. Global, "The future of mobile electronics interconnectivity," 2010.

[5] ARM, "Amba 3 apb specification," 2004.

[6] Raspberry Pi Foundation, "Introducing raspberry pi model b+," *https://www.raspberrypi.org/blog/price-cut-raspberry-pi-model-b-now-only-25/*, 2010.

[7] D. Cardinal, "Moore's law at 50: Its past and its future," *Extreme Tech*, 2015.

[8] D. W. Jones, "The digital equipment corporation pdp-8," 2001.

[9] G. Budzyń, "Programmable logic design," *http://www.ue.pwr.wroc.pl/pld/pld_1.pdf*.

[10] MIPI Alliance, "Standard for display serial interface," 2006.

[11] V. Borkar, "Linux virtual desktop: Tech preview is now available for evaluation," 2015.

[12] Synopsys, "Fpga-based prototyping made easy,"

[13] Synopsys, *The Fastest Way to Deliver Synopsys HAPS Series FPGA-Based ASIC Prototypes.*

[14] Synopsys, "Functional verification choice of leading soc design teams," *http://www.synopsys.com/prototyping/fpgabasedprototyping/pages/protocompiler.aspx.*

[15] Synopsys, "Synopsys culture, value and ethics," *https://www.synopsys.com/company/synopsyscareers/Pages/culture.aspx*, 2016.

[16] F. J. Morgan *The 5 Types Of Organizational Structures*, 2015.

[17] Microsoft, "Sharepoint revolves around you," *https://blogs.office.com/2012/07/23/sharepoint-revolves-around-you/*, 2012.

[18] MIPI Alliance, "Camera interface specification," *http://mipi.org/specifications/camera-interface*, 2016.

[19] P. Simoneau, "The osi model: Understanding the seven layers of computer networks," *Global Knowledge*, 2006.

[20] The International Telegraph and T. C. Commitee, "Data communication networks: Open systems interconnection(osi)," *Internationa Telecommunication Union*, 1991.

[21] V. Beal, "The 7 layers of the osi model," 1999.

[22] R. Merritt, "Mobile chip interface gets real," 2006.

[23] P. Lefkin and R. Wietfeldt, "Understanding mipi alliance interface specifications," 2014.

[24] MIPI Alliance, "Display interface specifications," *http://mipi.org/specifications/display-interface*, 2016.

[25] MIPI Alliance, "Phy working group," 2016.

[26] MIPI Alliance, "Mipi interfaces beyond smartphones," 2013.

[27] G. Moore, "Cramming more components onto integrated circuits," *Electronics Retrieved*, 1965.

[28] M. D. Ciletti, "Advanced digital design with verilog hdl," *Prentice Hall*, 2010.

[29] G. B. . A. Newell, "Designing computers and digital systems," *Digital Press*, 1972.

[30] IEEE *1076-2008 - Standard VHDL Language Reference Manual*, 2009.

[31] IEEE *1364-2005 - Standard for Verilog Hardware Description Language*, 2006.