DEGREE PROJECT, IN SYSTEM ON CHIP , SECOND LEVEL
STOCKHOLM, *SWEDEN 2014*

Formal Methods in Verification of Interface and Bus Protocols

MASTER THESIS IN ELECTRONICS AND COMPUTER SYSTEMS

KENJI KJELLSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY

KISTA

Formal Methods in Verification of Interface and Bus Protocols

KENJI KJELLSSON

Master Thesis in Electronics and Computer Systems

Stockholm, Sweden 2014

Abstract

This master thesis is performed on behalf of the Swedish technology company Ericsson and is meant to evaluate the efficiency of an assertion-based Verification Intellectual Property (abVIP) against an already existing constrained random Verification Intellectual Property (crVIP).

The abVIP is a verification entity which is connected with a predefined design and proves that the design is adhering to the specification properties through conditional checks, which uses mathematical proofs rather than prolonged simulation based input stimuli.

A market research meant to cover the popularity of abVIPs within the EDA vendors has been conducted. This research showed that the focus of the commercially available VIPs are still on the traditional simulation based crVIP, as only a third of the contacted EDA vendors openly offered any commercial abVIPs for various protocols. These companies are Cadence Design System, Jasper Design Automation and Mentor Graphics.

Based on the market research, the AXI 4 protocol has been chosen to be explored due to its popularity in formal based verification. The exploration phase answered questions related to how commercial abVIPs are built and what methods they considered. This was of particular interest as no industry standard methodology for formal based verification existed at time of writing. The companies for which the abVIPs were obtained from are referred to as company A and company B to avoid any confidentiality liability. Both abVIPs were assessed and shared many similarities although one was clearly more user friendly than the other.

Finally, a custom abVIP have been developed during the course of this thesis for an Ericsson design which utilizes the serial Inter-IC (I2C) bus protocol. This abVIP verification performance is compared against already existing crVIP verification results. This phase showed that the development time of an abVIP is unchanged from that of a crVIP.

The verification environment development time differed however in the favor of the abVIP. By not having to instantiate auxiliary modules, scoreboards or test cases, the abVIP was shown to gain weeks which were needed in the crVIP.

For a significantly smaller subset of the crVIP verification set, the abVIP required more than twice the time needed for the crVIP to yield 100% assertion results. This indicates that protocols which rely on long chains of system clock operations results in large amount of states which the formal tool has to visit. For this reason, one verification run ended up in a state space so large the results did not converge.

Thus it was concluded in this thesis that abVIPs are ultimately not a replacement of crVIPs but rather a complement and should be treated so by Ericsson as well. This is especially the case if the protocol to verify is a serial one.

Acknowledgment

I would like to thank my supervisors from Ericsson Per-Axel Carlhed and Muhammad Usman Tanveer for their continuous supervision and help throughout this thesis work.

Other Ericsson employes who I would like to express gratitude towards are Pierre Rohdin, Max Nilsson, Mikael Eriksson and Lucia Turcanu. Pierre believed in me and offered me a thesis position at Ericsson whilst Max provided me with designs to be verified in the actual thesis despite having no obligation to do so. Similarly Mikael Eriksson helped explaining I2C designs to be verified and Lucia provided me with comparison numbers. I am also grateful to my examiner Ahmed Hemani who have contributed with necessary information covering ASIC design and industry knowledge.

Sincere thanks to Jim Nordin and Yoav Arnon from Jasper Design Automation and Michael Carlberg and Rouzbeh Nikberg from Synopsis for interesting lunch meetings during the course of this thesis. These meeting lead to an increased knowledge regarding formal verification and is greatly appreciated.

I would also like to acknowledge my fellow thesis colleagues at Ericsson who participated in interesting conversations during the lunch breaks and helped me with various discussions despite not necessarily sharing the same thesis topic; Ejaz Sadiq, Emil Lundqvist, Jingying Dong, Sadiq Hemani, Viktor Classon and Yu Yang.

Special thanks to my family and friends who continuously supported me throughout my education at the Royal Institution of Technology (KTH); My brother Tor Kjellsson who helped me throughout the math and physics courses during my bachelors, my father Per Kjellsson and mother Keiko Sampei Kjellsson who taught me the importance of discipline and respect. I am ever grateful to my grandparents Kjell and Mary Nilsson who always believed in me.

Last but not least I want to acknowledge the love of my life, Sandra Trankell.

Table of Contents

Table of Contents

List of Figures

List of Tables

List of abbreviations

abVIP – Assertion Based Verification Intellectual Property

Ack – Acknowledgment

ACE – AXI Coherency Extensions

accVIP – Accelerated VIP

AHB – Advanced High-performance Bus

AMBA – Advanced Micro-controller Bus Architecture

APB – Advanced Peripheral Bus

AXI – AMBA eXtensible Interface

BCP – Boolean Constraint Propagation

BDD – Binary Decision Diagram

BMC – Bounded Model Checking

CNF – Conjunctive Normal Form

COI – Cone of Influence

CRSG – Constrained Random Stimuli Generation

crVIP – Constrained Random Verification Intellectual Property

DFI – DDR PHY Interface

DPLL – Davis-Putnam-Logemann-Loveland

DUT – Design Under Test

DUV – Design Under Verification

EDA – Electronic Design Automation

EC – Equivalence Checking

FSM – Finite State Machine

FPV – Formal Property Verification

GUI – Graphical User Interface

HDL – Hardware Design Language

HVL – Hardware Verification Language

I2C – Inter IC

I/O – Input Output

IC – Internal Circuit

IP – Intellectual Property

LSB – Least Significant Bit

MC – Model Checking

MSB – Most Significant Bit

Nack – Non-acknowledgment
OCP – Open Core Protocol
OVM – Open Verification Methodology
PC – Property Checking
PSL – Property Specification Language
R/W – Read or Write
RTL – Register Transfer Level
SAT – Satisfiability
SCL – Serial Clock Line
SDA – Serial Data Line
SERE – Sequential Extended Regular Expression
SMC – Symbolic Model Checking
SV– SystemVerilog
TVS – Test and Verification Solutions
UVM – Universal Verification Methodology
VE – Verification Environment
VEDT –Verification Environment Development Time
VIP – Verification Intellectual Property
VHDL – Very High Speed Integrated Circuit Hardware Description Language
VMM – Verification Methodology Manual

[This page is intentionally left blank]

Chapter 1

1 Introduction

This thesis is performed on behalf of the Swedish technology company Ericsson. The purpose of this thesis is to compare two different methodologies using Verification Intellectual Properties (VIPs) for functional verification; Constrained Random VIP (crVIP) and Assertion Based VIP (abVIP). Both VIP methods are used for functional verification, for which simulation based verification and formal verification are sub-methods [1]. The main problem statement for this thesis is; “Is abVIP a concept that could help Ericsson increase verification efficiency?”.

Despite the term assertion based VIP , assertions are not exclusive to the formal abVIP; assertions can be used in the simulation based crVIP as well.

1.1 Phase Explanation

The thesis is divided into several phases;

1. Pre-study phase
2. Market research phase
3. Exploration phase
4. Development phase

The various phases and their order can be seen depicted in Figure 1.1 below.

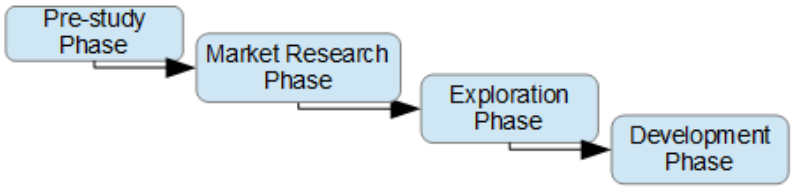

Figure 1.1: Phase structure.

The following subsections will briefly cover the purpose of each different phase.

1.1.1 Pre-Study Phase

The purpose of the pre-study phase is to acquire relevant information and knowledge regarding formal verification and document its' differences with the more common technique of functional verification; simulation based verification. Other relevant information concerning VIPs in general will also be covered.

1.1.2 Market Research Phase

A market research of currently available commercial abVIPs will be performed. The purpose of this phase is to uncover the current focus of VIPs on the EDA market; which vendors offer abVIPs and for which protocols? How many crVIPs does each vendor offer relative to the amount of abVIP as well as for which protocols? These questions will be answered in this phase.

1.1.3 Exploration Phase

The most common protocols for which abVIPs were offered for will be explored in this phase. This phase is meant to evaluate how abVIPs can be built, how the assertions can be defined and how partitioning modes may be achieved. This phase can be considered as a prerequisite for the final development phase. Depending on the availability and access, the abVIPs will be tested on Ericsson designs.

1.1.4 Development Phase

An abVIP will be developed in this phase, based on the knowledge obtained from the previous phases. The target protocol to design an abVIP for will have been determined before this phase. Requirements for the protocol will be that it is used in an Ericsson design, for which the same protocol has been verified through a crVIP. Based on the results, a conclusion for the purpose of this thesis may be answered in this final phase of the thesis.

1.2 Report Structure

The following subsection explains the overall reports structure and content within each heading and connections between various headings.

Chapter 2 - “Background“ on page 5 presents some necessary background information regarding the content of this thesis. This chapter is a part of the result obtained through the pre-study phase.

Chapter 3 - “Market Research” on page 11 presents which Electronic Development Automation (EDA) companies have been investigated and what kind of commercial VIPs are available. This chapter also covers the vendors focus on abVIP compared to crVIP as well as properties regarding their abVIPs. This chapter is a direct result of the market research phase.

Chapter 4 - “Exploration of Commercially Available VIPs” on page 17 covers the evaluation of abVIPs for the selected protocol, as a result from the market research. The abVIPs will be tested on either an Ericsson design or a provided design example, which implements the protocol the abVIPs are designed to verify. This chapter is a direct result of the exploration phase.

Chapter 5 - “Development of Custom Assertion Based VIP“ on page 26 covers the development of a custom abVIP for the I2C bus protocol used in an Ericsson design. The performance of the custom I2C abVIP will be compared against already available verification results from a previously used crVIP. This chapter is a direct result of the development phase.

Chapter 6 - “Conclusion of Thesis Work“ on page 44 summarizes the conclusions from chapters 3 , 4 and 5 respectively and also gives a final conclusion for the whole thesis.

Chapter 7 -”Future work“ on page 47 covers any potential future work that can be done in succession of this thesis, based on any unfinished work from the previous phases.

Appendix A - “ Techniques and Algorithms for Formal Verification” on page 52 presents useful information, algorithms and explanations regarding content linked to formal verification.

Appendix B - “ Market Research Extended Table Section” on page 56 list some extensive VIP protocol list from various vendors researched during the market research phase in chapter 3. These extensive tables is placed in this appendix to prevent disturbing the flow of the thesis.

Appendix C- “ AXI 4 Protocol“ on page 58 presents the protocols that has been chosen to be explored with the commercially available abVIPs deduced from chapter 3 . The specification of the AXI 4 protocol will be covered in this chapter; The ARM IHI 0022D specification which have been used is available through [2] whilst an AXI 4 reference guide used is available through [3].

Appendix D- “ I2C Bus Protocol“ on page 68 presents the I2C bus protocol for which its specifications will be described, as they are a necessity for developing properties to be asserted for the protocol. The openly available I2C specification [4] have been used for this appendix chapter.

Throughout this report, there are several Ericsson specific headings, for which the content will not be presented in the public report. The content is hidden due to confidentiality and is this only present in the Ericsson version of this master thesis report.

1.3 Methodology

The information gathering of the pre-study phase will mostly be conducted through literature studies from various book and papers, all properly referenced throughout the report and presented in the section ”Bibliography“ on page 48.

The market research phase will be conducted through probing the supply of different EDA vendors, either through their homepages or directly through email exchange. In both cases, the sources will be thoroughly referred to.

The exploration and development phase will both be performed in an experimental fashion. The developed abVIP will be a result of careful reasoning, based on knowledge gained from the initial phases explained in section 1.1 -”Phase Explanation“ on page 1, as well as continuous debugging in a trial and error fashion.

Once each phase is done, a conclusion as well as a future work section will be written based on the experience throughout the whole thesis project. These are available in chapter 6 and 7 , as mentioned in section 1.2 above.

Chapter 2

2 Background

This section covers some background information and knowledge necessary to follow the flow of this thesis. This chapter is a direct result of the pre-study phase.

2.1 Verification Intellectual Property

A VIP is much like any other Intellectual Property (IP), a logic block, in this case for verification, which put emphasis on re-usability [5]. An IP is in essence a register transfer level (RTL) code segment which is integrated with other designs [6]. Through constantly reusing logic block, coding time for engineers shrinks and overall coding efficiency increases. Each VIP is often tailored for specific protocols which are used in many designs. A VIP for a certain protocol will most likely not be able to properly verify the property of another protocol, unless they resemble each other to a great extent.

In order to ensure interoperability of IPs with their surroundings, there exist several technology libraries which are encouraged to be implemented, examples being the assertion library OVL from Accellera [7], which is a part of the Open Verification Methodology (OVM). The most recent verification methodology is known as Universal Verification Methodology (UVM) [8] which is the successor of OVM. The used hardware verification language (HVL) is irrelevant if the components are deemed to be compliant with the architectural aspect UVM; as long as the verification methodology is supported by all designs, one can mix designs with different HVLs and they will still be interoperable.

In general, an IP should include its specification, RTL implementation as well as verification plan and verification environment to ensure proper reuse. Older methodologies which serves a similar purpose is OVM and the Verification Methodology Manual (VMM).

This thesis will distinguish between 2 different VIP types, crVIP and abVIP. In a simulation based environment crVIPs are used, whilst in a formal environment abVIPs are used instead. On top of this, simulation based VIPs can be complemented with a VIP utilizing hardware to boost the simulation verification speed. These VIPs are known as accelerated VIP (accVIP). They will henceforth be considered as extensions to the crVIP and thus treated as a simulation based VIP.

The abVIP can however also be a passive verification component, meaning it does not initiate activities, but instead monitors DUT components or bus interface activities [9]. Thus an abVIP may have a mode for which it acts a passive monitor checker, which is re-usable in a simulation based environment as well as in hardware acceleration environment.

2.2 Random Constraints

The simulation based verification through random input stimuli with constraints have since over a decade been a solid choice for verification of design under test (DUT) [10]. The main idea is to randomize input stimuli to a DUT, which follows a set of constraints, instead of coding tedious directed tests for every single property [11]. Consider the following small SystemVerilog (SV) code example, presented in Figure 2.1 below.

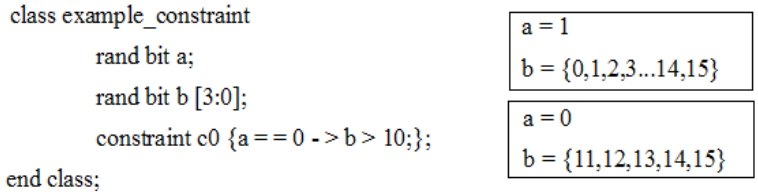

Figure 2.1: Constraint example in SV.

In Figure 2.1 we can see that through few lines of codes, we can achieve custom made input values covering desired ranges, depending on our needs. Through automated generated input stimuli, unanticipated corner cases which would have not otherwise been reached through directed written tests have continuously been reported [12], in other words, a greater functional coverage can be achieved through constrained random stimuli generation (CRSG) in comparison to direct test cases.

2.3 Assertions

An assertion is in essence a statement which must always hold true. Through checking a certain condition at specified intervals, a code automatically checks whether a condition is violated or not [13]. It has been reported that early implementation of assertions in designs can greatly reduce eventual debugging time during development of the RTL code [14].

Assertions are defined as either immediate or concurrent, where the immediate type is event based in procedural code, whilst concurrent assertion probes continuously throughout several clock cycles, triggering at system clock ticks [15]. Assertions are not explicitly used in HVL but also in hardware design languages (HDL) such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog.

In terms of verification, an assertion contributes with the key aspects error detection, isolation and notification [16]; If we consider a waveform generated from input stimuli for a simulation based verification, the assertion can be seen as an automatic waveform checker. Although the visual presentation may depend on the waveform software, the assertion will most likely show where and when the violation occurred. Figure 2.2 below shows an example where an assertion, which claims that one clock cycle after `Ack` is true `Grant` should not be true, is violated.

```
assert property ((@posedge clk) Ack |=> !Grant)
```

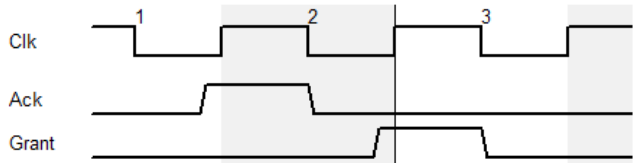

Figure 2.2: Assertion violation example in SV.

A VIP might be connected to its surroundings through an analysis port in simulation based environments or directly to a wire of interest. The VIP does not need to know the details of the surrounding verification components, instead it analyzes its defined properties through this port and reports any deviations from its predefined expectations [17].

2.4 Simulation Based Verification

Simulation based verification is one of the sub methods of functional verification and is also known as dynamic verification. After designing a test bench which instantiates your DUT modules as well as input stimuli for said DUT, tests can be run repeatedly with the same seed, for the same set of stimuli, or unique seeds for unique sets of stimuli [18]. Both assertions as well as constraints are often used in simulations. Through assertions, debugging of your RTL generally reduces in time as stated in section 2.3-”Assertions“ on page 6.

Through constraints it is possible to check the functional coverage, for which an indication of how well the design has been verified against your interpretation of the specification is given [19]. In a sense, the specification can be seen as the golden model for which the output of the DUT is compared against.

One of the drawbacks is that although simple bugs may be found in the first runs of the verification attempt, it is not uncommon that some more nested bugs are found very late in the project or even after production. The most famous and commonly referred to incident for bugs passing through verification checks, thus causing devastating consequences, may very well be the Intel Pentium FDIV bug [20]; Due to an issue in the floating point division, a full precision result could not be acquired and thus yielded wrong results. After this incident, Intel laid some focus on formal verification and the succeeding processor Intel Pentium 4 was their first processor verified through formal verification [21].

2.5 Formal Verification

Formal verification is another sub method of functional verification and is also known as static verification. Assertion and constraints can be used in formal verification as implementations of properties; Assertions results in requirements of the design whilst assumptions constraints the complete state space.

There are three popular methods for formal verification; model checking, equivalence checking and language containment [22]. Equivalence checking (EC) ensures the equivalence between two circuit descriptions. Model checking (MC), also known as property checking (PC), formally proves that set properties holds in all circumstances [23]. For this thesis, language containment and equivalence checking is however not considered.

The idea of formal verification is to drastically shrink the time required for functional verification of designs, through mathematical representations rather than simulations [13]. A formal corresponding finite state machine (FSM) is extracted from a synthesizable RTL code in the form of a Kripke structure [24]. A path through different states in the resulting FSM represents the behavior of the DUT.

Properties which defines the intent of the design can be verified through a proof algorithm referred to as fixed-point proof algorithm [18]. From the initial states in the extracted FSM, the next reachable states are computed. If a fixed-point state which satisfies certain conditions is reached, then the property has been verified. In order to do so, the properties can also be used as constraints, so that a limitation is put on the reachable state space, to avoid a deadlock situation referred to as state explosion [25]. This can be achieved through assuming properties rather than asserting them so that additional constraints is put on the design.

This is also known as limiting the Cone of Influence (COI), which is depicted in Figure 2.3 below. In this figure the initial total state space is indicated by the larger blue cone. Through additional constraints, the state space of interest can be shrunk to the smaller red cone, which is a sub-set of the initial total state space [26].

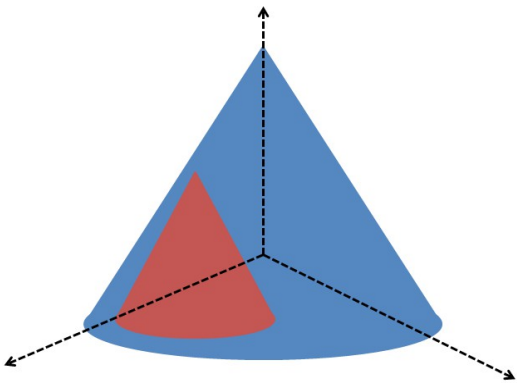

Figure 2.3: Cone of Influence example.

There are two types of properties for formal verification; safety and liveness [25]. Safety properties are considered to be bounded in time whilst liveness properties are infinitive. Safety properties indicates something that must hold true, as an example: “one cycles after Ack rises, Grant must be LOW”. An example for liveness properties could be: “Grant rises sometime in the future after Ack has risen”. In the case a safety property fails, a finite

length counter example is generated, but this is not the case for a liveness property. In order to distinguish between them, one can remember that safety properties are the types which define that “something bad never happens”, while liveness properties defines that “something good eventually happens” [24]. Safety properties are usually more common in comparison to liveness properties. This does not mean liveness is not useful; in the cases where liveness properties are applicable, they are quite powerful.

Apart from the above mentioned formal methodology, there also exists binary decision diagram (BDD), Satisfiability (SAT) and many other verification techniques for formal verification in the Boolean domain [23]. It is most likely that the formal tools used in industry combines a combinations of more than one of these techniques in their solving engines [27]. BDD and SAT are both described more extensively in Appendix A -” Techniques and Algorithms for Formal Verification“ on page 52. These have both been written with influence from [28] and [29].

2.6 Differences Between Simulation and Formal Verification

Formal verification is ultimately a method of complementing simulation based verification rather than a complete substitution [30]. The difference between the two methods of functional verification can be illustrated by the following reasoning, greatly described in the article by A. S. Dabare and S. Verma in [25]: Imagine several possible states illustrated as dots across the periphery of circles, illustrating several clock states as depicted in Figure 2.4 below.

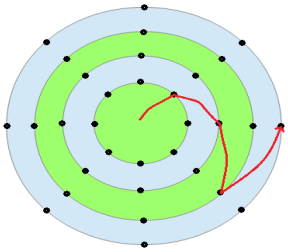

Figure 2.4: Comparison of simulation based and formal verification.

From the initial state, the center of Figure 2.4, a total of 8 states are reachable. The red line shows a possible path which might be taken in a simulation based verification run. This path is achieved through tailored input stimuli with related constraints. At each state, the assertion that are being tested is checked as well. The issue with simulation based verification is that it is cumbersome if not almost impossible to develop test cases for which all possible paths are asserted for complex designs.

Now consider formal verification instead. In Figure 2.4 all 8 states from the initial state in the center are each visited and asserted. From each individual states, the 8 following states in the subsequent clock cycle is also individually visited, covering all possible states and checking the assertions within. However, if the design is too large, a state explosion is likely to occur, in the sense that the system memory runs out. The constraints alters the

size of the state space, removing certain states that are no longer relevant, depicted in Figure 2.3 on page 8.

Another big difference is how the perception of clock cycles are treated. For simulation based verification, advancement of the system clock indicates that the past is set and cannot be altered; a sequential nature. The formal mathematical representation is however more combinatorial in the sense that it represents clock cycles as different copies of the design logic, which is connected with other “clock” copies [31]. This is referred to as circuit unrolling, which can be seen in Figure 2.5 below. This figure is a modification of figure 3 in the article written by A. S. Dabare and S. Verma in [31]. Here different colored regions represents different clock cycles.

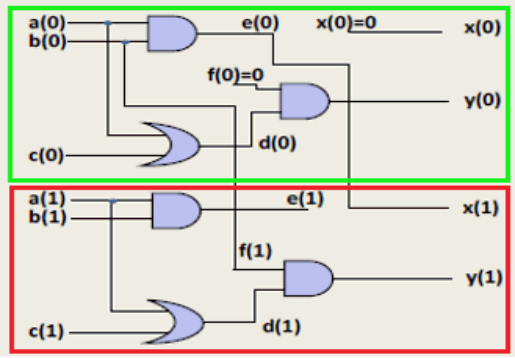

Figure 2.5: Circuit unrolling for several clock cycles.

Thus the past is not set in the sense that previous clock cycle states may very well be revisited for different assertion combinations.

For complex designs, one might begin with a thorough simulation based verification, followed by a formal verification to cover the hard to reach corner cases in order to achieve a coverage of 100%. In that case, the verification method may be refereed to as a hybrid style or semi-formal as a combination of both methods are used.

Chapter 3

3 Market Research

The purpose of the market research phase is to probe the EDA vendor market to observe the current focus on the VIP market; what type of commercially available VIPs can be obtained from which vendor? How many of the vendors are offering abVIPs and to what percentage, relative to the crVIPs? Apart from questions like these, type of protocol, methodologies used and which HVL supported is covered as well.

3.1 EDA Vendors and Their Formal Tools

Table 3.1 presents the EDA vendors and their own formal tools at the time of writing, both EC and PC which may be used together with different abVIPs. This table is meant as reference material so that the reader can get a broader picture which tool belongs to which vendor. The table presents the formal tools of the vendors which are most often referred to by the VIP protocols, established during the research phase presented in section 3.2 - ”EDA Vendors and Their VIPs“.

Vendor	Formal Tool
Cadence	Conformal (EC)
Cadence	Incisive Formal Verifier (PC)
Jasper	Sequential Equivalence Checking App (EC)
Jasper	Formal Property Verifier (PC)
Mentor	FormalPro (EC)
Mentor	Questa (PC)
Synopsys	Formality (EC)
Synopsys	Magellan (PC)

Table 3.1: Table of EDA vendors and their formal tools.

3.2 EDA Vendors and Their VIPs

This sub-section is further divided into EDA vendor specific sections, showcasing their different VIPs and summarize how big percentage of the total amount of commercially available VIPs are abVIPs. The VIPs to be considered are the simulation based crVIP, their hardware assisting extension accVIP and their counter part to be compared against, the abVIPs.

For the vendors who offers abVIPs for any protocol, complete VIP tables are summarized in appendix B on page 56. For the vendors who only offers crVIPs, the supported protocols

are not listed completely. Instead only a summary table is shown with some few example protocols mentioned in their respective sections. The reason for this is because the main purpose of this section is to present EDA vendors which supports abVIPs. For further relevant details regarding abVIPs such as their partitioning and usage modes, see section 4 -”Exploration of Commercially Available VIPs“ on page 17.

The vendors mentioned in the following sub sections are presented alphabetically. Apart from the vendors mentioned in Table 3.1 on page 11, the vendors have been chosen based on a providers list shown on the VIP directory list of the site Design-and-Reuses [32]. The vendors from that list have in turn been chosen from past experience from the supervisors as well as in a random fashion.

3.2.1 Cadence Design Systems

Cadence Design System offers many different VIP solutions for a wide range of protocols [33]. Although simulation based crVIP is the largest portion of their VIP set, there are some few formal abVIP for advanced micro-controller bus architecture (AMBA) protocols, DDR PHY interface (DFI) and open core protocol (OCP) [34]. The total number of VIP protocols offered by Cadence at the time of writing is 61, as can be seen in Table B1 on page 56.

As can be seen in Table B1, out of the 61 available protocols, 55 are offered as crVIPs, 16 as accVIPs and 7 as abVIPs. This showcases that the major focus lies on simulation based VIPs rather than formal assertion based. Out of the available abVIPs, the majority is offered as AMBA protocols. AMBA is an open standard bus protocol and a trademark of ARM architectures [35]. The AMBA protocol have several generations of buses tailored for different needs. Out of the total set of bus protocols advanced peripheral bus (APB), AMBA high-performance bus (AHB), advanced eXtensible interface (AXI) and AXI coherency extensions (ACE) are all protocols for which a formal based abVIP is offered individually by Cadence [36]. They all have support for master and slave device functionality and are run in their PC formal tool Incisive Formal Verifier.

All of the VIPs supports the HVL SystemVerilog (SV) as confirmed by Cadence directly through email. On top of SV support, System C and UVM is used for signal and transaction based simulation verification acceleration. It was mentioned by a representative from Cadence that their abVIPs are developed after the customers demands and the likelihood of development of more than a couple per year is very low [37]. Legacy methodologies, apart from UVM, are also supported.

Table 3.2 summarizes the amount of VIPs and the percentage of abVIPs in comparison to both crVIP and accVIP, together with HVL and some verification component support for the VIPs available.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
Cadence	55	16	7	9.86%	SV	OVM/UVM/VMM

Table 3.2: Cadence VIP summary table.

3.2.2 Jasper Design Automation

Jasper Design Automation is an EDA vendor that offers VIPs which all support assertion-based formal verification through their integration with their Formal Property Verifier tool, listed in Table 3.1 on page 11 [38]. On top of this, these VIPs are claimed to be easily leveraged into a simulation based environment resulting in all of their VIPs being supported both in static as well as dynamic verification environments. Table B2 on page 57 presents the complete list of protocols for which Jasper is claiming to have a VIP for at the time of writing.

Table 3.3 presents the summary table of the percentage of abVIPs in comparison to the crVIPs. Jasper supports both SV and PSL in their VIPs but use their own custom made methodology for their IPs [39]. It is a reasonable observation that a custom made methodology is used, as the verification methodologies OVM, UVM and VMM are for simulation based verification.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
Jasper	17	0	17	100,00%	SV, PSL	-

Table 3.3: Jasper VIP summary table.

3.2.3 Mentor Graphics

On Mentor Graphics' own website a list of their VIPs for certain protocols that are commercially available are presented [40]. These 44 protocols are summarized and shown in Table B3 on page 57. An email conversation with a contact person from Mentor Graphics stated that out of these listed protocols, 15 of them were offered as abVIPs [41], where some protocols in Table B3 on page 57 have multiple abVIPs available. DDR is an example where they offer abVIP exclusively.

Methodology wise it is stated on their homepage that they support OVM and UVM. Additionally, these methodologies can be accelerated with a specific test environment referred to as Veloce, enabling all crVIPs to simultaneously act as accVIPs [42]. Table 3.4 summarizes the amount of VIPs and the percentage of abVIPs in comparison to both crVIP and accVIP, together with HVL and some verification component support for the VIPs available.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
Mentor	43	44	15	17,24%	SV, PSL, OVL	OVM/UVM

Table 3.4: Mentor VIP summary table.

3.2.4 PerfectVIPs

Providing mainly interface and storage protocols, PerfectVIPs offers VIPs for a total of 7 protocols, for which none is implemented as an abVIP. [43]. Through email conversations it was established that the HVL was SV whilst the supported methodologies were OVM/UVM/VMM [44].

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
PerfectVIPs	7	0	0	0,00%	SV	OVM/UVM/VMM

Table 3.5: PerfectVIPs VIP summary table.

3.2.5 Sibridge Technologies

At the time of writing, Sibridge offers VIPs for 14 protocols, for which all appear to be implemented as crVIPs [45]. An email was sent to Sibridge Technologies in order to confirm whether this was the case or not, but a reply was not given within the time frame for the market research phase. Examples on supported protocols are the AMBA buses, Ethernet, MIPI, PCI express and some more. The results are summarized in Table 3.6 below.

The majority of the VIPs are written with SV and their I2C protocol supports master and slave functionality as an example. Integration with OVM/UVM/VMM is supported with their VIPs.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
Sibridge	14	0	0	0,00%	SV	OVM/UVM/VMM

Table 3.6: Sibridge VIP summary table.

3.2.6 SmartDV Technologies

The VIP vendor SmartDV offers a total of 75 protocols which are all automatically generated by their own developed tool [46]. None claim to be assertion based. Apart from SV being supported, all VIPs are also claimed to be supported in native SystemC. Some examples are the AMBA buses, MIPI, Ethernet, I2C, JTAG and many more. The results are summarized in Table 3.7 below.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
SmartDV	75	0	0	0,00%	SV	OVM/UVM/VMM

Table 3.7: SmartDV VIP summary table.

3.2.7 TrueChip Solutions

TrueChip offers VIPs for 14 protocols on their homepage [47]. 4 out of the 14 protocols state that they have “static assertion protocol check” which could be interpreted as abVIPs. Based on e-mail conversations, no pure abVIP is available at the time of writing but their assertions for their crVIPs can be packed into a module or an interface [48]. This module/interface can then be used to formally verify some protocols through formal means with JasperGold Formal Property Verifier.Examples on offered protocol VIPs are the AMBA and MIPI family, I2C and USB.

Thus, at the time of writing, it is concluded that TrueChip does not offer any pure abVIP, instead everything is originally simulation based crVIP. All their VIPs are written in SV and supports OVM, UVM and VMM. The results are summarized in Table 3.8 below.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
TrueChip	14	0	0	0,00%	SV	OVM/UVM/VMM

Table 3.8: TrueChip VIP summary table.

3.2.8 Synopsys

Synopsys have 31 protocol VIPs commercially available on their site at the time of writing, where none of these are implemented as an abVIP [49]. Instead, all protocols are offered as simulation based crVIPs in Synopsys respective verification tool shown in Table 3.1 on page 11. Through an email conversation, a contact person from Synopsys have confirmed that there are no commercially available abVIPs at the time of writing [50]. Despite this, he also mentioned that abVIPs are heavily invested in and is offered non commercially to certain customers. Thus, Synopsys is most likely going to commercially offer abVIPs in the future. Examples on supported protocols are AMBA buses, Ethernet, I2C, MIPI and many more.

Table 3.9 summarizes the amount of VIPs and the percentage of abVIP in comparison to both crVIP and accVIP, together with HVL and some verification component support for the VIPs available. It is worth re-mentioning however that Synopsys do have abVIPs, however not commercially available at the time of writing.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
Synopsys	31	0	0	0,00%	SV	OVM/UVM/VMM

Table 3.9: Synopsys VIP summary table.

3.2.9 Test and Verifications Solutions

Supporting OVM/UVM, Test and Verification Solutions (TVS) offers VIPs for a total of 22 protocols at the time of writing [51]. The HVL is SV and none of the listed VIPs are implemented as an abVIP. Instead all of them are offered as crVIPs, as confirmed by the contact person through email conversations [52]. Said contact person did however claim that it was possible to convert some of the crVIPs to abVIPs if requested explicitly by a potential customer. Some examples on supported protocols are AMBA buses, MIPI, Ethernet and some more. The summary table can be seen in Table 3.10 below.

Vendor	crVIP	accVIP	abVIP	% abVIP	HVL Support	Supported Methodology
TVS	22	0	0	0,00%	SV	OVM/UVM

Table 3.10: TVS VIP summary table.

3.3 Research Conclusion

What can be deducted from section 3.2 on page 11 and appendix B on page 56 is that SV is supported in all VIPs, regardless if the VIP is simulation based or formal based. Some of the most common protocol to develop a VIP for was the AMBA bus family, Ethernet and MIPI whilst Cadence, Jasper and Mentor all offered abVIPs for the AMBA bus family.

Regarding the supply of abVIP versus crVIP it is apparent that the current main focus is still on simulation based crVIPs. However, based on the email responses from various vendors it is also apparent that abVIPs are gaining popularity and will continue to be a solution to be reckoned with [37][41][50].

The methodologies OVM and UVM had most support through the vendors, while some vendors also supported VMM. This methodology support could also differ between protocols within the same vendor.

Since Cadence, Mentor and Jasper all offered abVIPs for the AMBA bus family, it was decided that one of the protocols to be explored would be the AMBA AXI 4 protocol, using one or more abVIPs from some of these vendors.

Chapter 4

4 Exploration of Commercially Available VIPs

The complete list of VIPs for various protocols found during the market research phase of this thesis is presented and listed in section 3.2 - “EDA Vendors and Their VIPs“ on page 11 as well as in appendix B on page 56. The result of this market research yielded the AXI 4 interface protocol to be the candidate for exploration.

This section describes the evaluation of the commercially available abVIPs from company A and B for the AXI 4 interface protocol, as stated in section 3.3 -”Research Conclusion“ on page 16. The main purpose of this phase is to assess the ease of use and ease of integration into the Ericsson verification environment (VE) of the abVIPs from both vendors. Other parameters to compare are requirements on said Ericsson VE, use mode, effort and time lapsed to get a complete verification result of the DUV. The VE is essentially the working environment for Ericsson employes where they perform their daily design and verification tasks.

In the case the abVIPs were not completely encrypted, another purpose of this phase was to see how commercial abVIPs are built. This worked as inspiration whilst developing a custom made abVIP for the I2C bus protocol. The specification of the AMBA AXI 4 interface protocol can be found in appendix C-” AXI 4 Protocol“, starting on page 58.

In order to ensure that the report can be publicly published, each vendor is referred to as company A and company B. The identity of these companies is only available in the Ericsson version of this report, as Ericsson have access to the abVIPs from each vendor. These identities are revealed in section 4.1.7 -”Ericsson Specific: Company A VIP Evaluation Procedure and Results“ on page 22 as well as in 4.2.7 -”Ericsson Specific: Company B VIP Evaluation Procedure and Results“ on page 24, but its contents are excluded from the public report.

An AMBA interface interconnect design and an AXI 4 Lite Slave have been provided by Ericsson, to be verified for this phase. The interconnect is a slim version of an interconnect used in different Ericsson project, whilst the slave is the very same component from yet another Ericsson project.

The evaluation for both company A and B's respective abVIPs are performed similarly through connecting the abVIP in master mode, to verify the interface protocols through their internal slave assertions. Ideally, the desired configuration is depicted in Figure 4.1 below.

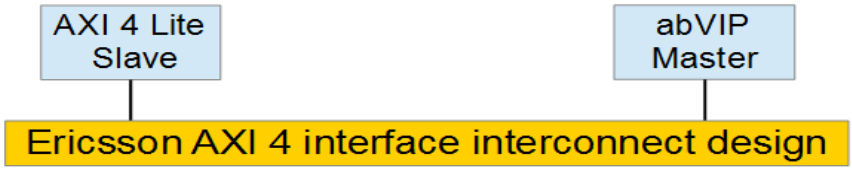

Figure 4.1: AXI 4 evaluation base scenario.

Although there are room for a total amount of two AMBA masters and three AMBA slaves on this interconnect, only one master and one slave was initially connected throughout the evaluations. This does however mean that several ports were unconnected, which in general is no problem for a formal tool as this simply results in less signals to be driven.

In the case that the configuration depicted in Figure 4.1 above runs satisfactorily, additional AMBA abVIPs are added to the interconnect to see how the verification results are affected. This further verification of the interconnect is presented in the Ericsson specific sections 4.1.9 and 4.2.8 on page 23 and 24 respectively.

4.1 Company A and AXI 4 Protocol

This subsection begins with showcasing the methodology company A used in their own abVIP for the AXI 4 interface protocol. The following section does not present direct extraction of their actual code expressions. Instead this section presents vague but general examples which are equivalent to their implemented methodology. Exclusive to the Ericsson version of this report, subsection 4.1.7 on page 22 describes the process of replicating the behavior depicted in Figure 4.1 above more thoroughly.

4.1.1 Company A abVIP Setup

The abVIP from company A can be instantiated in two different ways;

1. Instantiation in top file
2. Instantiation in parallel property file

In the 1st case, it is possible to insert an external AXI 4 DUV to be verified in the top module. Then the abVIP can also be instantiated in said top module and configured as the opposite device, enabling verification of the external DUV, as can be seen in Figure 4.2 below.

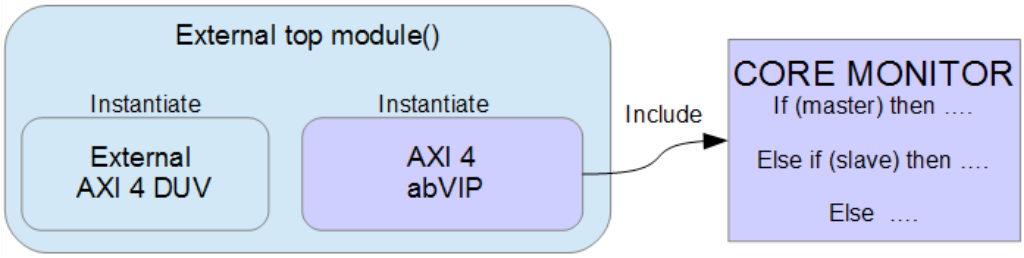

Figure 4.2: Company A AXI 4 abVIP top instantiation.

In the 2nd option, by including the abVIP in a separate property file, it is possible to bind the configured abVIP to the AXI 4 DUV top module with the SV bind feature, enabling further re-usability by not having to alter the top file for the external DUV. This is illustrated in Figure 4.3.

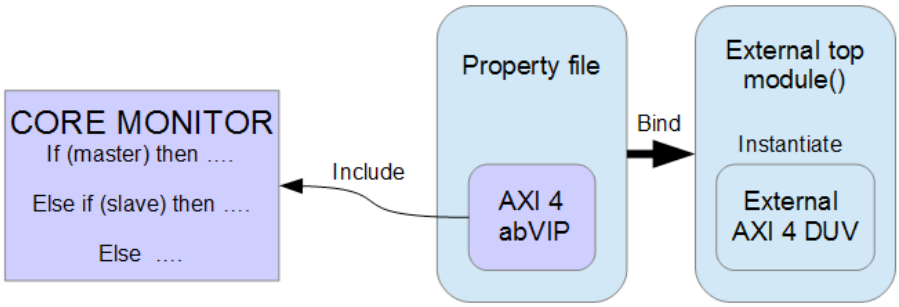

Figure 4.3: Company A AXI 4 abVIP property file bind.

4.1.2 Company A abVIP Configuration

The abVIP checking the AXI 4 interface protocol from company A utilizes a core monitor which may be configured into three different modes:

1. Master mode
2. Slave mode
3. Checker/Monitor mode

Regardless of the selected mode, the same core file is loaded, which contains all supported assertions, constraints and remaining logic for the whole protocol family. This core file is the previously mentioned core monitor, which can be seen in both Figure 4.2 and in Figure 4.3 above. The selection of different modes is achieved through the core file being loaded with different parameters, which enables different subsets of the whole core. Thus the corresponding choice of either master mode, slave mode or the monitor yields different assertions and properties being evaluated against the external AXI 4 DUV. This is depicted in Figure 4.4 below.

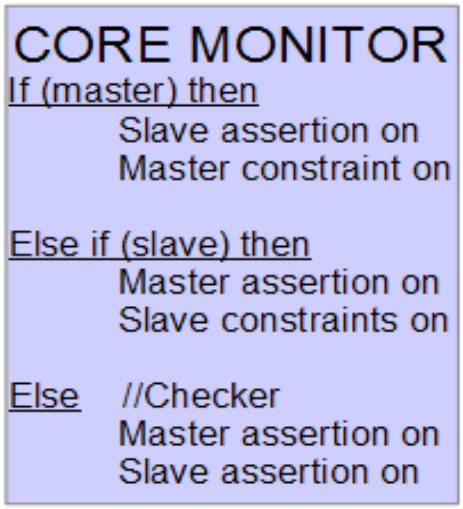

Figure 4.4: Company A core monitor mode subsets.

The 1st case, master mode, is only relevant for formal verification. The master properties are implemented as constraints while the slave properties are implemented as assertions.

The 2nd case, slave mode, is also only relevant for formal verification and the characteristics are inverse in relation to the master mode; slave properties are implemented as constraints while the master properties are implemented as assertions.

In both above cases, the constraints yields implication limits on the device during formal analysis, acting as a counterpart to constraining input stimuli in simulation based verification, thus reproducing the device behavior for evaluation purposes.

The last mode is the checker/monitor mode, which may be used for both formal as well as simulation based verification environment. It is a passive component mode, only yielding assertions for the applicable properties since it should not alter the input regardless of verification method.

In order to select one of the three modes, simply instantiate one of the master, slave, checker modules in the company A's AXI 4 supply stock. The abVIP is also parametrized in the sense that it is possible to tweak the modeling device which should verify the external DUV; length of address, length of data, response time and so forth. Other parameter examples can be whether to check if the AXI signals, presented in section C.1.2-” AXI 4 Signals “ on page 60, have an X or Z value.

Although it is possible to formally verify both the master and the slave at the same time, it is often of more use to only verify either the master or the slave at a time, while having the ability to verify both.

4.1.3 Company A abVIP Formal Methodology

When using one of the three modes, it is not necessary to delete any assertions that will not be used in the core file in order to run the formal verification; If the user wants to verify a slave, the master properties will instead be assumed automatically.

If required, it is possible to add assertions or constraints if the DUV requires so. These user added assertions and constraints are checked through company A's formal verification tool, which can detect contradicting statements, indicating dead ends and locks, most likely not intended by the user. Much like any other formal tool, the engine selection, effort mode selection and maximum time spent for proofs all determine how well the formal verification runs, and thus the abVIP comes with predefined recommended verification settings in a script.

Company A emphasize the importance of debugging the external DUV through its abVIP. Thus company A's formal verification tool together with the abVIP automatically creates signals groups for organization of relevant signals. Waveforms showing important transactions of the AXI signals, together with tables corresponding to the highlighted waveform, gives additional details of the transactions.

4.1.4 Company A abVIP Simulation Methodology

As stated previously, company A's abVIP can be used in a simulation environment as well. In this case, the abVIP is instantiated in a top module of a simulation environment, as a passive monitor checker. As the abVIP will be used in passive mode, it can not produce any input stimuli nor drive any signals. Instead it would be connected with the interface signals of the DUV and assert the transactions of those signals. The unused signals can not float and will require some fixed values assigned to them, which is a difference from the formal methodology where ports can be left open. Apart from assertions and constraints, the abVIP can also gather coverage results in its simulation mode.

4.1.5 Company A abVIP Verification Plan

The company A's abVIP offers a verification plan for both the master and the slave, which serves as a coverage report for the DUV properties and its assertions. A 100% coverage result means that the DUV is compliant with the AXI 4 interface protocol and is the percentage to strive for.

There are attached instructions regarding how the verification plan for either master or the slave mode of the abVIP may be generated. More detailed information on how to use the verification plan is available in section 4.1.7 -”Ericsson Specific: Company A VIP Evaluation Procedure and Results“ on page 22 in the Ericsson report.

4.1.6 Company A's abVIP Ericsson VE Compatibility

Company A's abVIP is easily used and is easily integrated with the Ericsson VE; it is provided with complete RTL examples for both master and slave DUV, which may be formally verified with the previously mentioned core monitor. In each example, the core monitor is configured to the correct mode. This shows potential use on how the instantiation of the abVIP would be done in practice, with an external master or slave DUV. As stated in section 4.1.1 -”Company A abVIP Setup“ on page 18, it is possible to connect an external DUV which would be formally verified, and is the main purpose of the company A abVIP.

The external DUV to be connected for this thesis is the Ericsson provided AXI 4 Lite slave together with the AXI 4 interconnect design. Just like the configuration desired, shown in Figure 4.1 on page 18, the interconnect together with the AXI 4 slave have been connected with company A's abVIP in master mode. The top module is a generic SV top module, just instantiating the sub modules and connecting them with appropriately defined wires.

The provided interconnect did not use all available signals presented in section C.1.2 on page 60, meaning these ports for the interconnect are not connected. Any assertions related to these signal will thus automatically fail, as there is no way of disabling them. An example is that the Company A AXI 4 abVIP does not offer an AXI4 Lite mode of the abVIP, which could have removed several assertions not included in the Lite version. The verification results also includes some false passes, as all safety type assertions with trigger commands trivially passed without their trigger commands getting issued.

Out of all implemented assertions, slightly less than 50% failed after a run of 8 minutes, although most assertions were trivial false passes. More detailed results from the evaluation phase will be presented in section 4.1.7 -” Ericsson Specific: Company A VIP Evaluation Procedure and Results“, which is limited to the Ericsson report. The reason for the limitation is because the detailed information will reveal which EDA vendor Company A is. The heading will still be available in the public report but will lack any content in the public report due to confidentiality. What can revealed however is that extended verification test for the Ericsson designs increased from 8 minutes to 65 minutes, indicating the effect of the added complexity.

4.1.7 Ericsson Specific: Company A VIP Evaluation Procedure and Results

4.1.8 Ericsson Specific: Company A Verification Plan

4.1.9 Ericsson Specific : Company A Extended Interconnect Verification

4.2 Company B and AXI 4 Protocol

Much like for company A, this section vaguely describes the methodology implemented by company B regarding their implementation of their abVIP for the AXI 4 interface protocol. Exclusive to the Ericsson version of this report, subsection 4.2.7 on page 24 describes the process of replicating the behavior depicted in Figure 4.1 on page 18 more thoroughly.

4.2.1 Company B abVIP Setup

The abVIP from company B can either be instantiated in a local top module or bound to an external top module through a property file, much like Figure 4.2 on page 19 and Figure 4.3 on page 19, showcased for the abVIP for company A. The abVIP is delivered with an executable script which may be edited for this purpose; there are predefined environmental labels for design files to bind or to include in the script.

4.2.2 Company B abVIP Configuration

Similar to company A, company B utilize a large core file which may be used either as a master or slave component depending on the need. Parameters enables the choice, and if an interconnect needs to be verified, like showcased in Figure 4.1 on page 18, an auxiliary device can be generated so that the abVIP provides an AXI 4 master or a slave to exercise the AXI 4 DUV. For example, in the case that the user desires to verify a Slave DUV, a master abVIP is generated which asserts the master properties whilst the slave properties are assumed. This behavior is inverted in the case the desire is to evaluate the master behavior.

4.2.3 Company B abVIP Formal Methodology

The AXI 4 abVIP provided by company B is highly configurable through a vast usage of parameters, which may be tailored according to the need of the DUV. Through simple binary values, irrelevant properties may be excluded as a result. Examples on parameters that can be set is the usage of AXI 4 Lite or regular AXI 4, whether to verify a master or a slave, whether to include low power properties or not and so forth.

The provided script can be run immediately without the user having to worry about any tool based settings, as the formal tool selects predefined engines for the user if the run is initiated from the GUI. It is however possible to modify the provided script in order to tailor the engine proof settings after the specific run needs.

4.2.4 Company B abVIP Simulation Methodology

It is possible to use the abVIP in a dynamic simulation based environment through the

usage of the parameters mentioned in section 4.2.3 on page 23. The provided documentation however lacks more detailed information on how this is actually performed beyond changing the value on a simple parameter.

4.2.5 Company B abVIP Verification Plan

Unlike company A, the abVIP from company B is not delivered with any verification plan, instead the result of the properties being asserted are summarized in the GUI of the formal tool of company B. As a result, this is an unsupported feature.

4.2.6 Company B's abVIP Ericsson VE Compatibility

The abVIP contains a specification regarding the usage of the AXI 4 abVIP from company B. This specification greatly documents the extracted specifications of the openly available AMBA AXI 4 specification, available through [2]. There are however few executable examples delivered with the abVIP, in comparison to company A. The specification mentions methods on how to use the abVIP, but as mentioned, lacks actual examples showcasing these claims. This puts some requirements on the user regarding HVL knowledge as compensation.

In order to evaluate the AXI 4 Lite Slave together with the AXI 4 interconnect design, an AXI 4 master abVIP was generated after setting the appropriate parameters in the provided script. Both the AXI 4 Lite slave and AXI 4 interconnect was instantiated in the provided top module but this resulted in an over-constrained environment; The abVIP from company B is not suited for a chained designs, instead it is meant to connect either a master or a slave DUV directly with their abVIP.

When the AXI 4 Lite Slave DUV was attached directly with the generated master abVIP, 70% of the assertion passed after only 1 minute. In order to obtain these result, some manual modification of the provided abVIP had to be done however, apart from simply setting certain parameters.

More detailed results from the evaluation phase is presented in section 4.2.7 on page 24, which is limited to the Ericsson report. The reason is because the detailed information reveals which EDA vendor Company B is. The heading is still available in the public report but the content is absent in the public report, due to confidentiality.

4.2.7 Ericsson Specific: Company B VIP Evaluation Procedure and Results

4.2.8 Ericsson Specific : Company B Extended Interconnect Verification

4.3 Exploration Phase Conclusion

The exploration phase of this thesis showed that both commercial abVIP consisted of a larger file which was divided into different regions, depending on the usage mode of the abVIP. Apart from formal verification, the abVIPs also offered a usage mode of the abVIP as a passive checker, which could be used in a simulation based environment as well. As a result of this phase, this abVIP model is used for the development of the custom abVIP described in chapter 5 on page 26.

The ease of use, ease of integration as well as requirements of the Ericsson VE varied depending on which company abVIP were used:

- For company A the abVIP was very easily used and integrated in the Ericsson VE. Setup of the abVIP was performed under an hour and a verification result of roughly 50% was achieved after 8 minutes. The majority of the failures were trivial as they depended on unconnected signals.

- For company B the abVIP was not as easily integrated in the Ericsson VE, but yielded 70% assertion results once connected. This result was however obtained through bypassing the interconnect depicted in Figure 4.1 on page 18, and directly connecting the AXI design to the abVIP.

The Ericsson specific extended conclusion will be presented in section 4.3.1 below and is only available in the Ericsson report of this thesis. The heading will still remain in the public report however.

4.3.1 Ericsson Specific: Extended Exploration Phase Conclusion

Chapter 5

5 Development of Custom Assertion Based VIP

Ericsson requested that the bus protocol which the development of an abVIP would be based on would be the Inter-IC (I2C) bus protocol. This chapter covers guidelines to consider in order to produce a formal friendly abVIP, as well as a summary extraction of the actual development of the I2C abVIP.

For any interested reader, the I2C protocol itself is covered in appendix D-” I2C Bus Protocol“ on page 68. The abVIP's slave mode was initially developed together with an I2C master module from a previous Ericsson project. The I2C abVIP's master mode was then developed with the help of the previously developed abVIP's slave mode.

5.1 Formal Assertion Guidelines

There are multiple documentations regarding the development for formal based assertions and what to consider whilst writing them at the time of writing this report [53] [54] [55] [56]. Similarly, Ericsson has its own design rules, which were unfortunately overlooked during this thesis project. For this reason, these guidelines and conventions might deviate from the Ericsson design rules.

This subsection summarizes the main points of the suggestions from these sources and were all considered in the actual development of the custom I2C abVIP.

- **Develop a verification plan:** Through a verification plan, tracking the status of properties regarding their evaluation for the design is possible. Properties which have been exercised should be noted in the verification plan so that the verification focus can be laid on non exercised properties.

- **Use sequences:** Sequences, or Sequential Extended Regular Expressions (SERE), increases the readability of the code and partition away complex logic in smaller formats to avoid confusion. Properties should be built by sequences to the extent which it is possible. Assertions and assumptions should in turn consist of properties.

- **Avoid under- or over-constraining:** In order to evaluate an abVIP, the behavior must be properly set. DUT without its top level environment might no longer have other components to generate relevant input stimuli to it. Thus these inputs have to be assumed through constraints interpreted from the design specification. Undesired behavior is most likely going to occur if some constraints are missing (under constraining) or too many constraints are included (over constraining). Under constraining results in injection of illegal input scenarios to the DUV whilst Over constraining results in legal scenarios getting excluded.

- **Cover your assertions:** An assertion may very well have a triggering condition, which is a sequence that must be true before a property is implied. If the implied

property is asserted and passes, the triggering condition is expected to have been exercised. However, there can occur false passes, where the assertion passes without the triggering condition being exercised. Thus it is important to cover the trigger conditions to judge whether or not the passing assertion is valid or not.

- **Use consistent labels:** Use consistent labels for sequences, properties, assertion, assumptions and constraints. The purpose is to quickly distinguish and identity the nature of the label through looking at it. The convention shown in Table 5.1 below has been used as an example.

- **Avoid state explosions:** It is important to take care when expressing properties; if a property heavily relies on a clock for a large amount of cycles, then several states will exponentially be inferred, which can result in a state explosion, meaning that the formal tool will likely crash. Thus in formal verification, the expressions of the properties are very important.

Prefix Label	Identifies
s_*	Sequences
p_*	Properties
a_*	Assertions
c_*	Constraints
cov_*	Cover statements

Table 5.1: Prefix label naming convention.

5.2 I2C Design Top Structure

The initial top structure when developing the I2C abVIP can be seen in Figure 5.1 below.

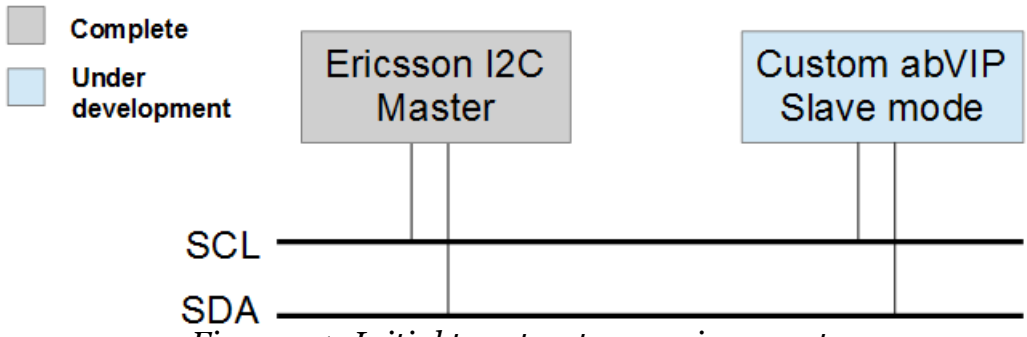

Figure 5.1: Initial top structure environment.

The provided Ericsson I2C master is a functional I2C master module used in other Ericsson projects. It was used in order to develop the slave mode of the custom I2C abVIP developed during this thesis. Once the slave mode was deemed to be satisfactorily functional, it was used as a reference for the development of the master mode of the I2C

abVIP, as depicted in Figure 5.2 below.

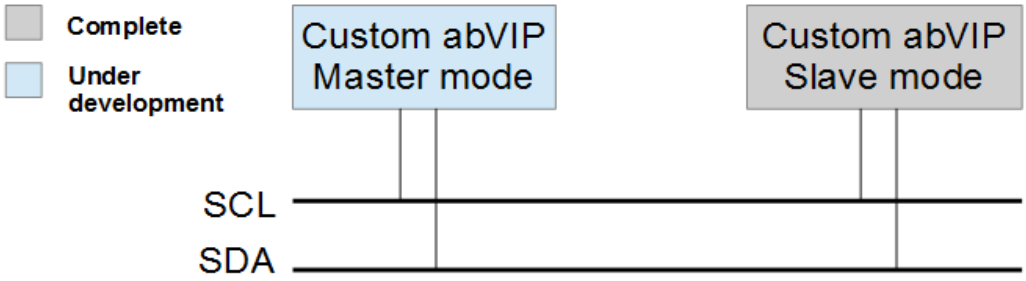

Figure 5.2: Secondary top structure environment.

In order to properly implement the 7 bit and 10 bit I2C logic in the slave, one additional abVIP slave instance was created and connected to the SDA and SCL line of the top structure. This configuration was ultimately used until the end of the development phase and is depicted in Figure 5.3 below.

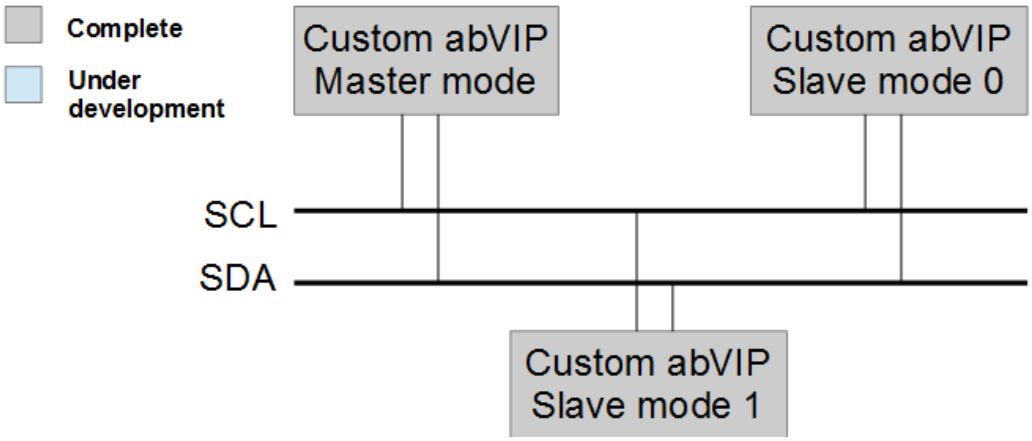

Figure 5.3: Final top structure environment.

Much like company A in chapter 4 on page 17, the I2C abVIP have been made to consist of one single common file, which is divided into either master or slave mode depending on a given parameter. Much like company B, additional abVIP parameters have been implemented to limit the COI and the associated assertions as well.

Ideally, the common assertions should be placed in the bottom of the file and be used by both abVIP modes to emphasize re-useability. If the assertions alone could be extracted to a separate module and be reused without the abVIP, even further re-useability would be attained. During this thesis project the assertions were however placed in their respective mode block due to design limitation. This experience is documented in order to urge any reader of this documentation to early on plan the structure of any VIP to make it as re-useable as possible.

With re-useability in mind. the developed abVIP can also be used as a passive monitor checker in a simulation based environment. This can be achieved by simply instantiating the abVIP in a monitor mode, which is based on the slave mode. Although being based on the slave mode, the monitor mode checks the transactions between a master and slave, and asserts if either component conflicts with the I2C protocol.

This is in general a concept that must be kept in mind in order to re-use any formal abVIP component in a simulation based environment; as long as the design depends on formal generation of inputs stimuli through constraints, then that design cannot be re-used in a simulation based environment as it is.

5.2.1 Ericsson I2C Master Behavior Replication

In order to achieve the initial development environment depicted in Figure 5.1 on page 27 the provided I2C master design had to be properly constrained. The provided master design had only been used in a simulation based verification environment, where input stimuli were provided from external components. For the formal verification environment in this thesis, several constraints in the form of assumptions had to be written in order for the master design to behave correctly without its missing modules. The assumptions were written with the help of the I2C master design specification. The process of replicating the master designs behavior took roughly 4 weeks, mostly due to inexperience with design.

5.3 I2C SCL and System Clock Relation

One important characteristic of the I2C protocol's clock line, SCL, is that it scales with the actual system clock to allow different operation frequencies; an SCL period can consist of several system clock periods as depicted in Figure 5.4 below, to correspond to various frequencies.

Figure 5.4: System clock and SCL relation.

The formal environment do not however have a notion of physical time for a clock cycle; a clock cycle is treated like an atomic unit, of no specific time unit. This prevents any assertion that is related with timing requirements, seen in Table D2 on page 75, to be constructed in the I2C abVIP. It is however still possible to have a scaling clock, which the SCL essentially is, during a data transfer if I2C clock stretching is disregarded.

5.4 Custom I2C abVIP Shared Assertions

Based on Table D4 on page 79, there are numerous characteristics that may be suitable to be expressed as properties to be asserted in the I2C abVIP. Table 5.2 summarizes the characteristics that have been chosen to be covered as assertions in both modes of the custom I2C abVIP . The extracted property numbers are relabeled from Table D4 in order to avoid further verbose labeling.

Extracted property number	section - page
1. **8:** Data on SDA must be stable during the HIGH period of SCL. Similarly HIGH or LOW state of SDA can only toggle when SCL is LOW.	Sec 6 – 8/46
2. **15:** No starting nor stopping condition may occur during a byte transfer.	Sec 7 – 10/46
3. **16:** An acknowledgment or non-acknowledgment must occur after each byte.	Sec 7 – 10/46
4. **19:** If receiver can not acknowledge an address, either a (P) or an (Sr) is generated, which must be generated after a non-acknowledgment.	Sec 7 – 11/46
5. **30:** Stop condition is not allowed to immediately follow a start condition.	Sec 9 – 14/46
6. **34:** There are multiple illegal address byte combinations; see Table D3 on page 76	Sec 10 – 16/46
7. **38:** 00000000 (H00) is not allowed to be used as 2nd byte for a general call.	Sec 10 – 16/46
8. **40.** High dummy acknowledge sent after START byte 0000_0001, no slave is allowed to acknowledge this byte.	Sec 10 – 18/46
9. - : Master/slave shall not drive SDA when authority belongs to slave/master.	-

Table 5.2: Applicable assertion property table for the I2C abVIP.

In a simulation based environment all timing requirements of the I2C bus protocol in Table D4 would be a perfect candidate for assertions to be covered in a VIP. As stated in section 5.3 on page 29 however, formal verification do not have a notion of time and thus these properties can not be verified by an abVIP. The last applicable assertion in Table 5.2 above is not explicitly written in the I2C specification, but is implemented as an assertion in the slave mode abVIP as it is implied.

5.4.1 I2C Assertion Properties

Table 5.3 below summarizes properties used in both modes of the I2C abVIP in order to verify the I2C protocol. The actual code expressions are hidden due to confidentiality, but the purpose of each property is defined below Table 5.3. This table also presents which property number of Table 5.2 above is covered by the property in the same row.

Property	Property Number
p_PROPER_PULSE_LENGTH	1, 2, 3
p_NACK_START_STOP	4
p_NO_IMMEDIATE_START_STOP	5
p_LEGAL_ADDR_BYTE_X	6
p_LEGAL_GNRL_ADDR	7
p_LEGAL_START_BYTE	8
p_SDA_AUTHORITY	-

Table 5.3: Property table for master mode I2C abVIP.

- **p_PROPER_PULSE_LENGTH** checks that all address and data transactions occurs in sets of 9 SCL pulses. Once an evaluation signal has been asserted through the design logic this property is checked as an assertion.

- **p_NACK_START_STOP** checks that an I2C start or stop command have been issued by the master after a non-acknowledgment, within one SCL pulse after the non-acknowledgment was detected.

- **p_NO_IMMEDIATE_START_STOP** defines that an immediate jump between the START and STOP, seen in Figure 5.5 on page 33 is not allowed to occur.

- **p_LEGAL_ADDR_BYTE_X** is a property which groups together illegal or not supported combinations of the address byte sent from the master and received by the slave. There exist 2 versions of this property, depending on whether the addressing mode is 7 or 10 bit exclusively. The property checks the content of the address storage array to see whether this result is illegal or not. The origin of the actual illegal addresses can be seen in Table D3 on page 76.

- **p_LEGAL_GNRL_ADDR** is similar to p_LEGAL_ADDR_BYTE_X mentioned above, in the sense that it checks the content of the general address array storage once a proper evaluation signal has been asserted by the design logic.

- **p_LEGAL_START_BYTE** is a property which is triggered by an evaluation signal. Once this evaluation signal rises, this property checks that a non-acknowledgment have resulted on the SDA line during a start byte, meaning no slave instance have acknowledged this byte.

- **p_SDA_AUTHORITY** is a property which ensures that SDA, which in accordance to the I2C protocol is a tri-state buffer, has the proper value when its enabling condition is on. If this is not the case, then some external instance is falsely trying to alter the SDA line, both for the master as well as for the slave.

Apart from the above mentioned assertions, there are abVIP mode specific assertions as well, which are not applicable in both modes. One example for both modes will be

mentioned below:

- **p_LEGAL_7BIT_ADDR** is a slave property which checks that the slave address associated with the instance is not illegal. The MSBs 5'b1111_0 should not be used by a 7 bit slave as an example, as these 5 bits are reserved for 10 bit addressing. This assertion checks the address even after a general call have reconfigured the LSBs.

- **p_LEGAL_SCL_PERIOD** is a master property which has no triggering condition. It checks whether the master parameter SCL_PERIOD, described in section 5.7.1 on page 37, has a legal value or not during instantiation. This is not a generic I2C assertion but rather an abVIP specific assertion.

5.5 Custom I2C abVIP Slave Mode Development

An I2C slave component do not have to produce any stimuli to the verification environment like an I2C master. Instead it is sufficient if it drives acknowledgment pulses according to the I2C specification, found in [4]. The slave mode has thus been designed without any assumption constraining any of its internal signals. Furthermore, by excluding the driving logic which puts acknowledgments on the SDA line, the slave mode offers a passive monitor checker. This monitor checker takes a subset of the slave assertions, which is shared by the master, to check the I2C transactions on the SDA and SCL line pair.

The following subsections describes how the slave mode was implemented on a basic level; which methodology was used for instance. For Ericsson, there is a detailed abVIP specification documentation prepared for the developed I2C abVIP which covers the complete design implementation. For the public report, information is withheld on purpose due to confidentiality reasons. The I2C abVIP slave mode have been verified together with the I2C Master design and the I2C abVIP master mode.

5.5.1 Slave Mode Parameters

The I2C abVIP is designed so that certain parameters can be given during instantiation of the abVIP. The following parameters are a subset of the total amount of parameters one can set when instantiating the custom I2C abVIP in its slave mode:

- MODE
- SLAVE_ADDRESS
- ...
- GENERAL_CALL

The abVIP can be set to either master or slave mode, which is handled by the parameter MODE. The parameter SLAVE_ADDRESS corresponds to the 7 bit or 10 bit address the slave instance responds to. GENERAL_CALL is a parameter which defines whether or not the slave instance will be able to respond to a masters general call on the SDA/SCL line pair.

Apart from the above mentioned parameters additional parameters exists in order to implement the slave instance of the I2C abVIP according to the I2C bus protocol, but is withheld in this public report. The parameters were implemented in accordance to knowledge gained from the exploration phase.

5.5.2 Slave Mode FSMs

The slave mode has been designed with 2 levels of FSMs in order to properly respond to the SDA transactions which the master issues on the SDA line. For this public report, only the first and most basic level of the FSM is shown in Figure 5.5 below.

The various states correspond to the nature of the SDA and SCL line pair; IDLE is when the line pair is free, START is when an I2C start condition has occurred, quickly followed by the TRNSFR state, which indicates a data transaction. The STOP state is corresponding to an I2C stop condition which is followed by the IDLE state.

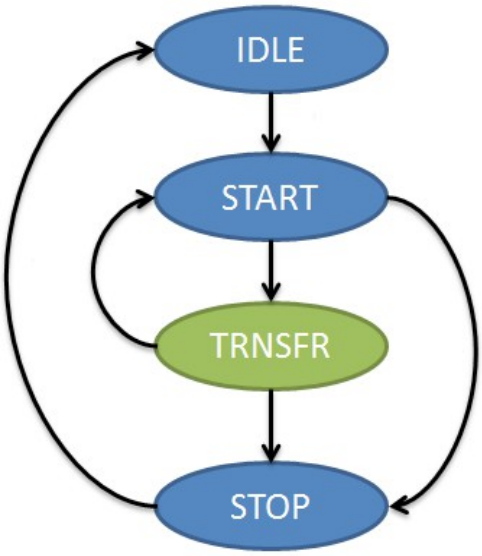

Figure 5.5: I2C abVIP Basic Slave FSM.

It is the TRNSFR state in Figure 5.5 above that consist of yet another level, which corresponds to the 7 bit and 10 bit addressing and data transfer part of the I2C protocol, which is explained in section D.1.4 and D.1.5 on pages 70 and 71 respectively. Based on the data read on the SDA line by the slave, various jumps are made between these different states on the 2 different FSM levels.

5.5.3 Slave Mode Auxiliary Logic

The slave mode of the custom I2C abVIP consist of 4 different categories of auxiliary groups which are used in order to implement the I2C slaves behavior. These are:

1. Counters
2. Storage
3. Evaluation

4. Command

1: Counters is the group which consist of index variables, which ensures that for each SCL pulse, the value of SDA is stored on the proper index. Examples would be reading the issued address on the SDA line by the I2C master, or any of the data transactions.

2: Storage is the group which consist of any array which purpose is to store values. Example would be the address array which gets data from the SDA line when leaving the IDLE state depicted in Figure 5.5 above. Another example is the data storage when the slave is reading data from the I2C master.

3: Evaluation is the group which contains all the signals which triggers the assertions for the custom I2C abVIP. On the proper occasion in the nested slave FSM, certain evaluation signals are asserted. The slave assertions in turn triggers on rising values of said evaluation signals and the validity of a certain conditions are checked. The applicable slave assertions are covered in section 5.4.1 on page 30.

4: Command is the last of the 4 groups used in the slave mode of the I2C abVIP. It contains several signals to recognize certain acknowledgments in the slave. Examples can be whether an address has been acknowledged or not. This groups also contains signals to further divide the TRNSFR state in an address and data region, depending on which type of signal is asserted.

5.6 Custom I2C abVIP Slave Mode Performance

The performance of the slave mode of the custom I2C abVIP is decided through comparing various results with a previously used simulation based I2C crVIP at Ericsson. Both the crVIP and the developed abVIP was tested on the same Ericsson I2C master design in order to evaluate both VIPs performance. Factors to consider were

- Development time of the VE until verification results could be obtained
- Time to achieve verification results
- Quality of said results

Although this performance rating is ultimately for the specific I2C abVIP developed for this thesis, these comparison numbers will enable a general conclusion to be made regarding the efficiency of abVIPs in comparison to crVIPs. Based on the results, conclusions are drawn whether or not abVIPs for formal verification is a concept that might further benefit Ericsson. For the remaining sections, the VE Development Time (VEDT) will be a factor to consider for both performance comparisons.

5.6.1 Slave Verification Environment Development Time

The first comparison is between the VEDT for simulation based and formal based verification. As mentioned in section 5.2.1 on page 29, replicating the master design took 4 weeks of this project. Setting up the correct run configurations took an additional week, where the majority of this week consist of wrong parameter setting due to human error.

It is important to note that the replication of the master was performed on an early stage of the thesis project, meaning extra time might have been consumed due to simple inexperience within the subject and not necessarily due to a complex master design.

Figures related to the VEDT of the crVIP has been provided for comparison from block level verification tasks performed in an ASIC project at Ericsson. The VEDT time for the crVIP, apart from the crVIP environment build up, includes setting up an OVM based test bench which further consist of a top level environment. This in turns contains build, elaborate and connect phases. Apart from this, test classes, sequences, propagation and connection of interfaces are all factors that need to be considered for the VEDT.

The different VEDTs have been estimated for both types of VIP and are listed in Table 5.4 below.

VEDT abVIP	VEDT crVIP
5 weeks	14 weeks

Table 5.4: Slave mode VEDT comparison.

5.6.2 Slave Verification Run Time

Table 5.5 below summarize the time required for both abVIP and crVIP for the Ericsson I2C master design in order to achieve the verification results shown in section 5.6.3 below.

Run	Verification run time abVIP	Verification run time crVIP
1	11 hours 8 minutes	8 hours 3 minutes
2	7 hours 34 minutes	

Table 5.5: Slave mode verification run time comparison.

The test cases used from the crVIP verified both read and write transaction for data transaction up to 16 bytes for combined formatting and up to 32 bytes for non combined formatting. These test cases were applied on both a 7 bit as well as a 10 bit slave. The crVIP run time was the time required to achieve 100% code and functional coverage in a simulation based environment with all passing assertions.

In order to get more confidence in the verification results, one has to continuously perform verification with additional test cases. This process continues until the verification project team considers the verification sufficiently exhaustive. This additional time to acquire further confidence is not included in the 8 hours and 3 minutes reported in the table above. This time is the sum of several test cases, where as the time required for the formal runs are the full time for only 1 test case each.

Table 5.6 below shows the constraints applied to the Ericsson I2C master design to be verified with a limited COI. Although the crVIP results issued several test cases up to 32 data bytes, only 2 data bytes transactions were allowed for the formal run. This was done

in order to limit the COI to ensure converging results.

The main reason why the verification run time has decreased between Run 1 and Run 2 is because the Ericsson master design were constrained in a way to not generate start bytes nor general calls for Run 2. The reason for this constraint is because the Ericsson master design does not implement start byte nor general call in 10 bits, as it does for 7 bit addressing.

Run	Constraints
1	Only 7 bit addressing
	1 slave specific address, start byte and general call
	Both read/write transactions
	Both combined and no combined formatting
	2 transaction instructions
	2 bytes data transactions
	2 bytes in slave storage
2	Only 10 bit addressing
	1 slave specific address, no start byte nor general call
	Both read/write transactions
	Both combined and no combined formatting
	2 transaction instructions
	2 bytes data instructions
	Only 2 byte in master storage

Table 5.6: Slave mode of abVIP COI constraints.

5.6.3 Slave Quality of Results

Table 5.7 below shows the quality of the results obtained from both verification runs. Important to note here is that the crVIP had the ability to verify the timing requirements of the I2C protocol in the Ericsson I2C master design, whilst the developed abVIP do not have such an option due to reasons stated in section 5.3 on page 29.

Furthermore, the coverage percentage in formal, Coverage F, and the coverage percentage in simulation, Coverage S, are not indicating the same type of coverage. Examples of what the coverage results in the simulation based environment consist of are line coverage, code coverage and functional coverage. The coverage in the formal run only indicates whether the trigger conditions for the assertions were exercised or not.

Run	Quality of results abVIP		Quality of results crVIP	
	Assertion	Coverage F	Assertion	Coverage S
1	100,00%	100,00%	100,00%	100,00%
2	100,00%	100,00%	100,00%	100,00%

Table 5.7: Slave quality of results comparison.

5.7 Custom I2C abVIP Master Mode Development

In contrasts to an I2C slave component, an I2C master design has to produce proper stimuli which in turn has to be constrained. In formal verification one has to tell the tool what to not produce in contrast to simulation based verification, where you instead have to tell the tool what to produce. Thus in the custom I2C abVIP master mode, several assumption have thus been designed to control the master modes internal signals.

The following subsections describes how the master mode was implemented on a basic level; which methodology was used for instance. For Ericsson, there is a detailed abVIP specification documentation prepared for the developed I2C abVIP which covers the complete design implementation. For the public report, information is withheld on purpose due to confidentiality reasons. The I2C abVIP master mode have been verified together with the I2C abVIP slave mode.

5.7.1 Master Mode Parameters

The I2C abVIP is designed so that certain parameters can be given during instantiation of the abVIP. The following parameters are a subset of the total amount of parameters one can set when instantiating the custom I2C abVIP master mode:

- MODE
- MASTER_ADDR_MODE
- ...
- SCL_PERIOD

As stated in section 5.6.1 on page 34, the custom abVIP can be set to either master or slave mode, which is handled by the parameter MODE. The parameter MASTER_ADDR_MODE determines whether the master should generate 7 bit or 10 bit addresses exclusively, or both. This parameter is implemented in order to have the possibility to reduce the COI if a slave only supports one type of addressing, as explained in section 2.5 on page 7. The parameter SCL_PERIOD determines how many clock cycles a periodic SCL pulse should consist of, as depicted in Figure 5.4 on page 29.

Apart from the above mentioned parameters, additional parameters exists in order to implement master instance of the I2C abVIP according to the I2C bus protocol. An example being the address the master may issue; it is recommended that as few slave addresses as possible is given to this parameter array, so that once can minimize the COI. The parameters were implemented in accordance to the knowledge gained from the

exploration phase.

5.7.2 Master Mode FSMs

Much like the slave mode of the abVIP, the master mode consist of 2 levels of FSM which are nested. In this public report, only the first basic level is shown and explained. The base FSM is shown in Figure 5.6 below.

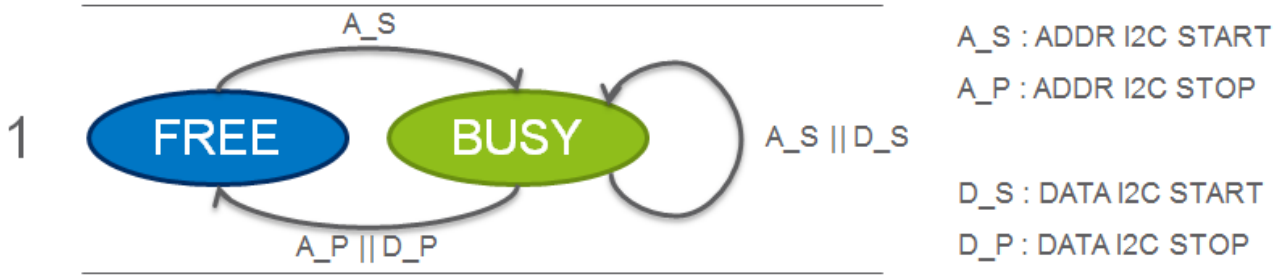

Figure 5.6: Master mode base FSM.

This base FSM describes the binary nature of an I2C bus; it is either FREE or BUSY. If an I2C start condition occurs whilst the bus is FREE then the bus becomes BUSY. If an I2C stop condition occurs whilst the I2C bus is BUSY, then the bus becomes FREE. Any start condition which occurs during the BUSY state is a repeated start, in accordance to the I2C bus protocol.

It is the BUSY state consist of yet another FSM, which is not shown in the public report. It can however be revealed that it consist of an address part and a data part. In the address part, the generated address is put on the SDA line which a slave instance later reads. Based on the acknowledgment response from the slave, the master mode either performs a write or a read transaction in accordance to sections D.1.4 and D.1.5 on pages 70 and 71 respectively.

5.7.3 Master Mode Auxiliary Logic

The master mode of the I2C abVIP consist of several category groups which are listed below:

1. Counters
2. Storage
3. Commands
4. Assumptions
5. Configurations

1 ~ 3: The explanation of the 3 first categories are very much similar to the explanations given in points 1, 2 and 4 in section 5.5.3 on page 33 but instead for the master mode of the I2C abVIP. For this reason, please refer to this section for further details.

4: The assumption category consists of all the constraints placed on various signals in

order for the master mode of the I2C abVIP to behave in accordance to the I2C protocol. The majority of the assumptions are somehow related to the start/stop signals shown in Figure 5.6 above. An example of an assumption which is of great importance of the I2C protocol is a constraint labeled c_NO_START_STOP_SIMUL. What this constraint achieves is the basic property of the I2C protocol that an I2C start and a stop condition cannot occur simultaneously. Through this constraint, the formal tool never issues a start and a stop command simultaneously.

5: The configuration category is the last of the categories of auxiliary logic used by the master mode of the I2C abVIP. It consist of several matrices which can be assigned any values and are not constrained by any assumption. Out of the complete COI, it is possible to create cover statement for a certain desired pin pointed test cases, in order to quickly extract a tailored transaction sequence. An example of a desired sequence could be writing 4 bytes to a certain address, followed by a combined read of 2 bytes to another address. Although the formal tool would exercise this test case out of the complete COI, by writing a cover statement for this exact condition, one can quickly get multiple subsets of interest.

5.8 Custom I2C abVIP Master Mode Performance

Just like in section 5.6 on page 34, the performance of the master mode of the custom I2C abVIP is desired to be decided through comparing various results with a previously used simulation based crVIP used at Ericsson. Both said crVIP and the developed abVIP would be tested on the same Ericsson I2C slave design in order to evaluate the abVIPs master mode. Factors to consider would be:

- Development time of the VE until verification results could be obtained
- Time to achieve verification results
- Quality of said results

Unfortunately, a crVIP was never used to verify the Ericsson slave design, instead directed test cases were used instead. The performance of the master abVIP will however still be documented in the following sub-section for a potential future work. A conclusion for the efficiency of abVIPs can still be drawn based on the performance of the slave mode however.

5.8.1 Master Verification Environment Development Time

The first desired comparison would be between the VEDT for the formal based and simulation based verification environments. Setting up the slave design took 6 hours in total. It is however important to take note that this was performed on a later stage of the thesis project, meaning less time, in comparison to setting up the master design, might have been consumed due to more experience at this point of the thesis.

The VEDT for the crVIP environment was not available at the time of writing, since no such test had been performed previously. This has instead been added as future work task.

VEDT abVIP	VEDT crVIP
6 hours	-

Table 5.8: Master mode VEDT comparison.

5.8.2 Master Verification Run Time

Table 5.9 below summarize the time required for the abVIP in order to achieve the verification results shown in section 5.8.3 below for the Ericsson I2C slave design. Although the Ericsson slave design is less complex than the Ericsson I2C master design, its minimum SCL to system clock relation is close to double that of the Ericsson I2C master design. This unfortunate requirement results in a state explosion, meaning the complete restricted COI cannot be visited, despite using the minimum SCL to system clock ratio. The verification run was terminated after 24 hours.

Run	Verification run time abVIP	Verification run time crVIP
1	24h	-

Table 5.9: Master mode verification run time comparison.

The constraints for this run on the abVIP is shown in Table 5.10 below. These constraints were applied through proper values on the master parameters of the abVIP. The Ericsson slave design only supports 10 bit addressing however, and thus 7 bit addressing have been disregarded through parameter settings. Similarly, start byte nor general call is not supported, and thus these options are removed from the COI as well.

Despite limiting the COI even further by only allowing write transactions and 1 byte data transactions, the high SCL to system clock relation was sufficient to result in a state explosion.

Run	Constraints
1	Only 10 bit addressing
	Both read/write transactions
	1 slave specific address, no start byte nor general call
	Both combined and no combined formatting
	2 transaction instruction
	2 bytes data transactions
	2 bytes in master storage

Table 5.10: Master mode of abVIP COI constraints.

5.8.3 Master Quality of Results

Table 5.11 below shows the quality of the results obtained from both verification runs.

Run	Quality of results abVIP		Quality of results crVIP	
	Assertion	Coverage F	Assertion	Coverage S
1	?%	100,00%	-	-

Table 5.11: Master mode quality of results comparison.

Although the cover statements managed to be visited by the formal tool, it was the assertions of type “safety” which could not converge due to the failure to reach the complete COI for the proof run. Since the COI was not completely visited due to state explosion, the assertions percentage could not be determined.

As no crVIP was used to verify the Ericsson I2C slave, no numbers will be reported in the table above, instead this will be left as a future work task. It is important to note the difference between the term coverage in formal and in simulation, as explained in section 5.6.3 on page 36.

5.9 Development Phase Conclusion

The development phase took 10 weeks for which the slave mode constituted of 4 weeks and the master mode the remaining 6 weeks. This also includes the debugging time required to make each mode work according to the I2C protocol. Thus the complete developing time for the I2C abVIP took 10 weeks.

A custom crVIP for the I2C protocol have been developed at Ericsson and numbers for the development time including debugging are reported as approximately 10 weeks as well.

Development time abVIP	Development time crVIP
10 weeks	10 weeks

Table 5.12: Development time for abVIP and crVIP.

This observation is within the expectations as the design themselves are not different whether one uses formal or simulation based verification; what differs is the generation of the input stimuli. This in turn is the only point for which one could see a benefit with abVIPs over crVIPs; as indicated by Table 5.4 on page 35, the VEDT differed with 9 weeks in the favor to the abVIP. By letting the formal tool handle the generation of stimuli instead of having to produce several test cases, scoreboards and assertion modules the benefit in time clearly showed, as shown in Figure 5.7 below.

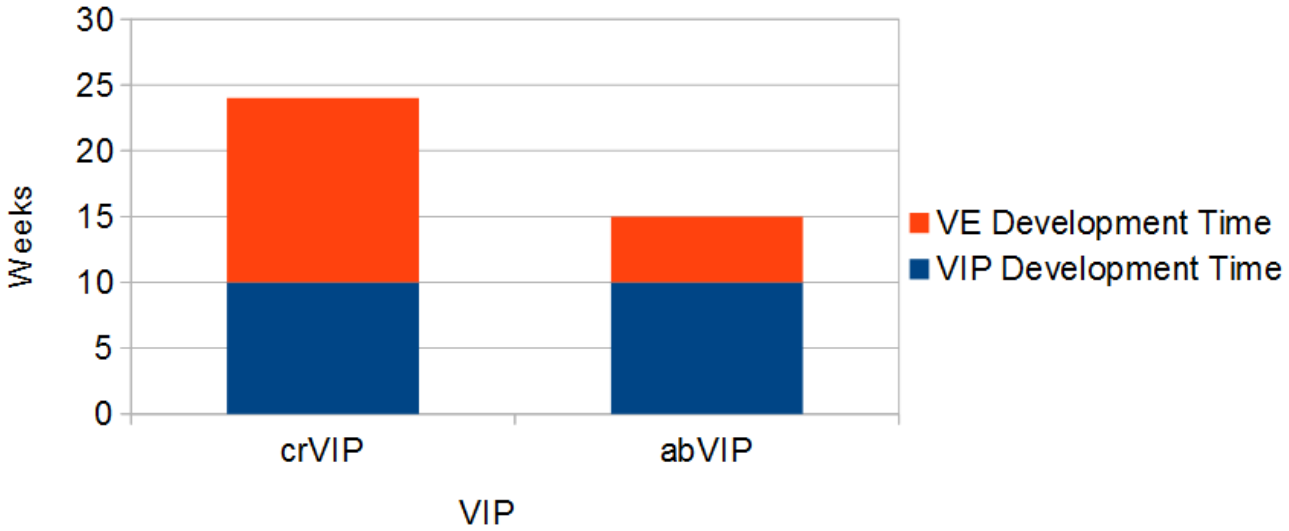

Figure 5.7: Slave mode VEDT and development time comparison, crVIP vs abVIP.

The comparison of the individual verification run times in Table 5.5 on page 35 shows rather similar numbers between the two VIP types for 100% assertion percentage results. These numbers are graphically represented in Figure 5.8 below.

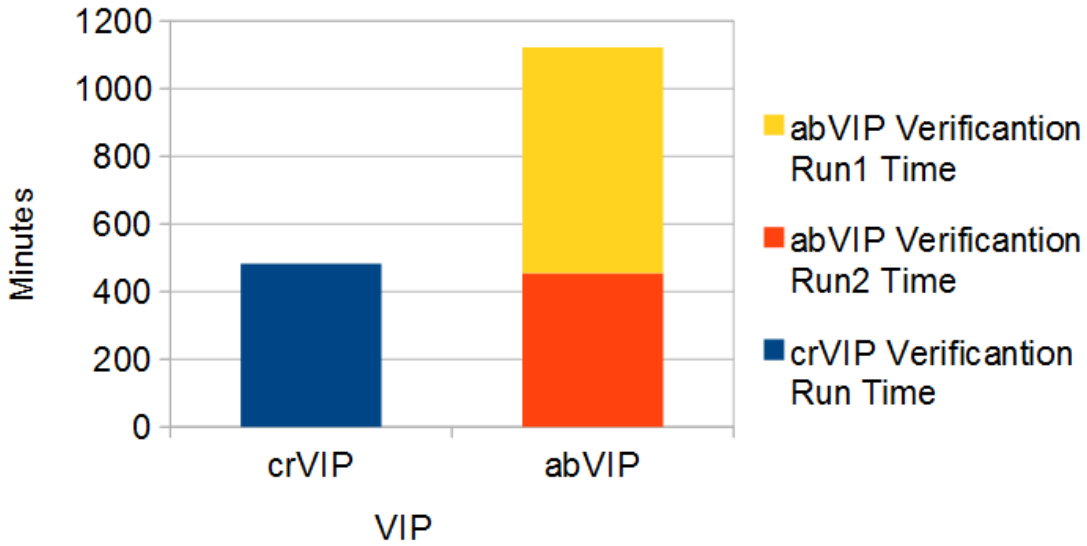

Figure 5.8: Slave mode verification run time comparison, crVIP vs abVIP.

Important to note here however is that the although both VIP verified both 7 bit and 10 bit addressing, the crVIP results verified I2C transactions up to 32 data bytes in comparison to only 2 data bytes for the abVIP. When combining the abVIP run times for only 2 bytes, the total time required is already more than twice that of the crVIP with up to 32 data bytes.

The reason for these large run times despite the difference in data bytes is due to the how design complexity is treated in formal verification. The Ericsson master and slave design had a large SCL and system clock relation which leads to bigger COI for the formal tool, depicted in Figure 5.4 on page 29. This means that each and every I2C transaction gets

affected and requires a very large amount of equivalent mathematical states, risking state explosion. As mentioned in section 5.8 on page 39, a state explosion actually occurred for the Ericsson I2C slave design due to a high SCL and system clock relation. Based on the time required for only 2 data bytes in the abVIP run, the time required for up to 16 data bytes are feared to exceed that of the crVIP result by far even if the proof runs would converge.

This is in general the drawback of using a serial protocol for formal verification. Thus a limitation of abVIPs is that for serial protocols formal verification has to create a very large state space which in turn is time consuming, relative to simulation based verification.

Even though a 100% assertion result was achieved from the crVIP run, the design is not necessarily error free. Consider an assertion which checks that an illegal value never occurs. In a dynamic simulation environment, if you never produce all of the input stimuli which has the potential to assign an illegal value, you might miss a design flaw. In static formal environment, the complete state space can be exhaustively visited, meaning that a passing assertion of this kind may yield greater confidence. For this reasons, a 100% assertion result in formal can bring more confidence to a design, in comparison to a 100% result in simulation based verification. This fact can be useful if one desires to perform pin-pointed verification test, which can bring high confidence with a run over a night, with very low man-hours needed.

Important to remember is that the crVIP consisted of more assertions than the abVIP; the assertions for the timing requirements of the I2C protocol. As the formal tool do not have a notion of physical time apart from atomic clock cycles with no unit, formal verification can not be used to verify a protocol completely, as crVIP can. This is a further limitation for formal verification.

All VIP should emphasize re-usability, as stated in section 2.1 on page 5. The developed abVIP for this thesis is no exception. In order to achieve re-useability, the monitor mode of this abVIP can be used in a simulation based environment. With the help of an I2C abVIP specification written during this thesis work, simplification to re-use this abVIP is hoped to be achieved; this specification is meant to quickly introduce users to proper use of the I2C abVIP.

The specification begins with an explanation regarding both modes parameters together with explanations on proper value settings to limit the COI. For any interested user the I2C abVIP have been decomposed in its various block segments and explained in detail for both the master and the slave mode with various figures.

Chapter 6

6 Conclusion of Thesis Work

This chapter begins with summarizing the conclusion from each phase described in this thesis report in order to draw a final conclusion whether abVIP is a concept that may help Ericsson increase verification efficiency.

6.1 Market Research Phase Conclusion

Through the market research conducted for this thesis, which is presented in chapter 3 on page 11, it was found that only the major EDA companies offered abVIPs commercially. The ratio of abVIP against crVIP were generally very small, with exception of Jasper Design Automation who only offers abVIPs. The only HVL that was supported by all EDA vendors was SV.

Cadence Design Systems mentioned that it was unlikely that they would develop more than a couple of abVIPs per year, indicating that crVIP is still the stronger trend of the two. They and Mentor Graphics were the only companies to also offer a third kind of VIP, the accVIP, which is an assisting VIP for crVIPs in a simulation based environment.

6.2 Exploration Phase Conclusion

Through the exploration phase, knowledge about abVIP structuring was obtained and used during the later development phase. The ease of use of the commercial abVIPs varied between the two commercial abVIPs explored as well as their verification results. Much like any other abVIP, the input stimuli was automatically generated through the constraints implemented in the abVIPs, which contributed to the ease of usage.

Early indications that increased complexity in a verification design increases the verification time significantly were observed during this phase as well.

6.2.1 Ericsson Specific : Exploration Phase Extended Conclusion

6.3 Development Phase Conclusion

Through the development phase it was found that the development time for an abVIP did not differ from the development time of a crVIP for the same I2C protocol. What did differ however were the VEDTs and the time required to get a complete verification result, which was highly dependent on the actual design to be verified. The VEDT showed the benefit of using abVIP over crVIP by not having to generate additional auxiliary modules or input stimuli.

The verification run time suffered however; In a serial protocol like the I2C, the depth of a transaction can be very large, which increases the complete space for the abVIP to visit,

resulting in very long verification run times. For the Ericsson I2C slave design, the COI was so large it resulted in state explosion.

During this phase it was realized that industry designs may very well not be naturally formal friendly. An example being the Ericsson I2C designs which had larger minimum SCL to system clock relation than the minimum requirement for the developed I2C abVIP. As every I2C transaction scales with this relation, the state space for the Ericsson I2C designs ended up being quite large in formal. When developing a design, the fewer amount of clock cycles a logic is dependent upon, the more formal friendly it is. If the minimum requirement of the amount of clock cycle is large however, as was the case in both Ericsson I2C designs, then there is unfortunately not much that can be done.

A requirement to be able to use an abVIP in a simulation environment is that it should offer at least one mode that does not depend on formally generated constraints through assumptions. Thus any abVIP that has a mode for which no input is generated through constrains, is a viable target to be re-used in a simulation environment. The I2C abVIP's monitor mode is an example of a passive component which can be re-used in a simulation based environment for this reason.

6.4 Final Conclusion abVIP vs crVIP

The EDA market clearly focuses more on crVIP than abVIP based on the customers demands, and the customer is most likely demanding solutions that not only work satisfactorily, but yields fast results.

The commercial abVIP which were explored showed a difference in ease of use, documentation procedure and coding structure; indicating that there are no present methodology for formal verification which companies can follow at the time of writing.

The developed abVIP yielded exhaustive proof, but in terms of protocol assertions, the same assertion percentage results were achieved faster with a crVIP. The benefits of using an abVIP was shown during the VEDT for the VIP, as the burden of having to generate stimuli and other auxiliary modules were removed by the abVIP in contrast to the crVIP. As the verification designs becomes larger and more complex however, the benefit gained from the VEDT will shrink by the increased verification run time, especially if the dependency on the system clock is high. In the worst case of a state explosion, the verification result will never converge as well. It was also concluded that a crVIP may contain more assertions, as properties related to physical time requirements can not be asserted in an abVIP in contrast to a crVIP.

It was revealed in the exploration phase that the controlling and partitioning of an abVIP is achieved through setting instantiation parameters, which in turn affects condition checks and assumptions which constrains the design.

The development phase showed that the design structure of an abVIP is no different to that of a crVIP. This observation is supported by the fact that the development time of a crVIP is unchanged from an abVIP. Auxiliary logic in an abVIP can however be less complex in comparison to a crVIP; the abVIP can let the formal tool drive internal signals instead of

equivalent driving logic in a crVIP.

The development phase also showed one clear limitation of abVIPs when they are used in a serial protocol, which heavily depends on the system clock; the more clock cycles a transactions requires, the more equivalent states the formal tools need to generate. In the worst case, this limitation can result in a state explosion with non-converging verification results.

An abVIP mode which relies on formally generated input stimuli can also not be re-used in a simulation based environment due to the inputs to the design not being generated in the same way. For this reason, for the abVIP to be re-used in a simulation based environment one must consider to include a mode which does not generate input in this fashion during the development. The monitor mode of the I2C abVIP is an example of a mode that may be re-used in a simulation based environment.

Based on the results from all of the phases conducted through this master thesis, it is apparent that static formal abVIP is not a replacement for the traditional dynamic simulation crVIP, but rather a complement. This is the role which the abVIP is believed to serve Ericsson as well; an extended pin-pointed verification test for smaller designs, preferably with non-serial protocols, if the crVIP results are deemed not sufficient. In that sense, it is a concept that may help increase Ericsson verification efficiency.

Chapter 7

7 Future work

This chapter covers some topics that were not covered due to various reasons in the time scope of this master thesis, and are potential follow up topics to be conducted.

7.1 Evaluation Phase

In order to properly conclude the evaluation phase, comparison numbers for the AXI 4 interconnect must be extracted with an AXI 4 crVIP. First when evaluations numbers concerning the points below have been extracted, the performance of commercial abVIPs compared against the performance of commercial crVIPs can truly be rated:

- Ease of use of the crVIP.
- Ease of integration into the Ericsson verification environment (VE).
- Requirements on said Ericsson VE.
- Usage mode, effort and time lapsed to get a complete verification result.

7.2 Development Phase

Based on a code review at Ericsson, the current structure of the developed abVIP might not be ideal in the sense of re-useability. The common assertions for the slave and master are placed in their respective block in the common abVIP file. Due to limitation of the design implemented, the assertion are not placed in a common block at the end of the file. This would be desirable in order to keep the structure of the code file simple and understandable. This is more a matter of code re-structuring than a major implementation change. A further step in the right direction would be to allow the assertions to be extracted into an external module, to be re-used in another VIP.

On top of this, verification results from using an I2C crVIP for the Ericsson slave design is also desired, for which none were available at the time this thesis was conducted. Although the drawn conclusion in section 6.4 on page 45 is unlikely to change, one may never know with 100% certainty until such numbers are produced.

Bibliography

[1] J. Jose, S. A. Basheer, (2009, May 07) “A Comparison of Assertion Based Formal Verification with Coverage driven Constrained Random Simulation, Experience on a Legacy IP”, [Online]. Available: http://www.design-reuse.com/articles/18353/assertion-based-formal-verification.html (Accessed 2014/01/23)

[2] ARM, “AMBA® AXI™ and ACE™ Protocol Specification”, [Online]. Available: http://ee427plblabs.groups.et.byu.net/wiki/lib/exe/fetch.php?media=ihi0022d_amba_axi_protocol_spec.pdf (Accessed 2014/02/17)

[3] Xilinx, “AXI Reference Guide”, [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf (Accessed 2014/03/21)

[4] Phillips, “I2C Bus Specification and User Manual”, [Online]. Available: http://i2c2p.twibright.com/spec/i2c.pdf (Accessed 2014/02/11)

[5] R. Wilson, (2005, Feb 02) “What's with verification IP?”, [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1152824 (Accessed 2014/01/23)

[6] A. Fedeli, M. Moriotto, U. Rossi, F. Toto (2005, Jan 21) “Addressing IP Reuse with Formal Verification and Assertion Based Verification”, [Online]. Available: http://www.design-reuse.com/articles/9511/addressing-ip-reuse-with-formal-verification-and-assertion-based-verification.html (Accessed 2014/01/24)

[7] H. D. Foster, A. C. Krolnik, “Libraries,” in C*reating Assertion-Based IP*, 1st ed. New York City, Springer, 2008, ch.1, pp 11

[8] K. A. Meade, S. Rosenberg, “What is the Universal Verification Methodology (UVM)?,” in *A Practical Guide to Adopting the Universal Verification Methodology (UVM)*, 2nd ed. San Jose California, Cadence Design Systems, 2013, ch.1, pp 1

[9] H. D. Foster, A. C. Krolnik, “Definitions,” in C*reating Assertion-Based IP*, 1st ed. New York City, Springer, 2008, ch.2, pp 31

[10] C. Pixley, J. Havlicek, (2003, Apr 13-15) “A Verification Synergy, Constraint-Based Verification”, [Online]. Available: http://www.eda.org/edps/edp03/submissions/pixleyVerificationFinal.pdf(Accessed 2014/01/22)

[11] C. Roseburgh (2005, Mar 04) “Using Vera and Constrained-Random Verification to Improve DesignWare Core Quality”, [Online]. Available: http://www.design-reuse.com/articles/9818/using-vera-and-constrained-random-verification-to-improve-designware-core-quality.html (Accessed 2014/01/23)

[12] H. D. Foster, A. C. Krolnik, D. J. Lacey , “Do assertions really work?,” in *Assertion-Based Design*, 2nd ed. Boston/Dordrecht/London, Kluwer Academic Publishers, 2004, ch.1, pp 7

[13] R. Stolzman (2002, Aug 16) "Understanding Assertion-Based Verification", [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1275847 (Accessed 2014/01/23)

[14] H. D. Foster, A. C. Krolnik, "Stakeholders," in C*reating Assertion-Based IP*, 1st ed. New York City, Springer, 2008, ch.1, pp 5

[15] C. Spear, "Connecting the Testbench and Design," in *SystemVerilog for Verification*, 2nd ed. New York City, Springer, 2008, ch.4, pp 107-109

[16] H. D. Foster, A. C. Krolnik, D. J. Lacey , "What is an assertion?," in *Assertion-Based Design*, 2nd ed. Boston/Dordrecht/London, Kluwer Academic Publishers, 2004, ch.1, pp 3

[17] H. D. Foster, A. C. Krolnik, "Guiding Principles," in C*reating Assertion-Based IP*, 1st ed. New York City, Springer, 2008, ch.3, pp 38

[18] C. Spear, "Parallel Random Testing," in *SystemVerilog for Verification*, 2nd ed. New York City, Springer, 2008, ch.1, pp 12

[19] C. Spear, "Feedback from Functional Coverage to Stimulus," in *SystemVerilog for Verification*, 2nd ed. New York City, Springer, 2008, ch.1, pp 13

[20] L. Hoyle (1994, Nov 30) "The Pentium FDIV bug - a Picture", [Online]. Available: http://www.ipsr.ku.edu/stafffil/hoyle/pentium_fdiv/ (Accessed 2014/01/28)

[21] T. Schubert, "High Level Formal Verification of Next-Generation Microprocessors," in *DAC '03 - Proceedings of the 40th annual Design Automation Conference* , 2003, pp. 1–6.

[22] Y. Kukimoto (1996, Feb 06) "Introduction to Formal Verification", [Online]. Available: http://embedded.eecs.berkeley.edu/research/Vis/doc/VisUser/vis_user/node4.html (Accessed 2014/01/23)

[23] R. Drechsler, "Introduction," in *Advanced Formal Verification*. New York/Boston/Dordrecht/London/Moscow, Kluwer Academic Publishers, 2004, ch.0, pp xix-xx

[24] H. D. Foster, A. C. Krolnik, D. J. Lacey , "Formal verification framework," in *Assertion-Based Design*, 2nd ed. Boston/Dordrecht/London, Kluwer Academic Publishers, 2004, ch.1, pp 45-47

[25] A. S. Dabare, S. Verma (2011, Dec 11) "Understanding Formal Verification Concepts ", [Online]. Available: http://soccentral.com/results.asp?CatID=488&EntryID=36389 (Accessed 2014/01/24)

[26] H. D. Foster, D. L. Perry, "Cone of Influence Reduction," in *Formal Verification for Digital Circuit Design*, 1st ed. New York, McGraw-Hill Professional, 2005, ch.8.1, pp 135

[27] R. Drechsler, "Challenges," in *Advanced Formal Verification*. New York/Boston/Dordrecht/London/Moscow, Kluwer Academic Publishers, 2004, ch.0, pp xxi-xxiii

[28] K. Claessen, N. Een, M. Sheeran, N. Sörensson, “SAT-solving in practice,” in *Proceedings of the 9th International Workshop on Discrete Event Systems* , 2008, pp. 61–67.

[29] R. E. Bryant, (1992) “Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams,” [Online]. Available: http://www.cs.utah.edu/~ritwik/papers/p293-bryant.pdf , pp. 3–4.

[30] M. C. McFarland, “Formal Verification of Sequential Hardware: A Tutorial,” in *Transactions on computer-aided design of integrated circuits and systems*, may 1993, vol 12, no 5 pp. 633–654.

[31] A. S. Dabare, S. Verma (2012, Jan 31) “Understanding Formal Verification Concepts – Part 3”, [Online]. Available: http://soccentral.com/results.asp?CatID=488&EntryID=37653 (Accessed 2014/01/25)

[32] Design and Reuse, “Verification IP Directory”, [Online]. Available: http://www.design-reuse.com/vip/provider.php (Accessed 2014/02/11)

[33] Cadence Design Systems, “Verification IP”, [Online]. Available: http://ip.cadence.com/ipportfolio/verification-ip (Accessed 2014/02/11)

[34] Cadence Design Systems, “Assertion-Based VIP”, [Online]. Available: http://ip.cadence.com/ipportfolio/verification-ip/assertion-based-vip (Accessed 2014/02/11)

[35] ARM, “AMBA Open Specifications”, [Online]. Available: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php (Accessed 2014/02/11)

[36] Cadence Design Systems, “Cadence Verification IP for ARM AMBA Protocols”, [Online]. Available: http://www.cadence.com/downloads/rl/vip/AMBA.pdf (Accessed 2014/02/11)

[37] F. Pouyet, “Cadence: questions on ABVIPs”, Personal e-mail (Feb. 12, 2014)

[38] Jasper Design Automation, “Intelligent Proof Kits and VIPs ”, [Online]. Available: http://www.jasper-da.com/products/intelligent-proof-kits (Accessed 2014/02/13)

[39] Ø. Kolsrud (2009, March 19) “OCP and Verification of Configurable OCP Interfaces”, [Online]. Available: http://ocpip.org/uploads/documents/OCP_IP_3_19_2009.pdf (Accessed 2014/02/13)

[40] Mentor Graphics, “Mentor Verification IP ”, [Online]. Available: http://www.mentor.com/products/fv/verification-ip (Accessed 2014/02/12)

[41] B. Janfalk, “RE: Mentor Assertion-Based VIP”, Personal e-mail (Feb. 14, 2014)

[42] Mentor Graphics, “Veloce Emulation Systems ”, [Online]. Available: http://www.mentor.com/products/fv/emulation-systems/ (Accessed 2014/06/05)

[43] PerfectVIPs, “Hardware Verification Products”, [Online]. Available: http://www.perfectvips.com/products.html (Accessed 2014/02/11)

[44] N. Patnaik, “Re: PerfectVIPs Assertion-Based VIPs?”, Personal e-mail (Feb. 14, 2014)

[45] Sibridge Technologies, “Verification IP”, [Online]. Available: http://www.sibridgetech.com/verification-ip.aspx (Accessed 2014/02/12)

[46] SmartDV Technologies, “Verification IP's”, [Online]. Available: http://www.smart-dv.com/products.html (Accessed 2014/02/12)

[47] TrueChip Solutions, “Verification IPs”, [Online]. Available: http://www.truechip.net/products.html (Accessed 2014/02/12)

[48] S. Gupta, “Re: TrueChip Assertion Based VIP - corrected”, Personal e-mail (Feb. 13, 2014)

[49] Synopsys, “Verification IP - Overview”, [Online]. Available: http://www.synopsys.com/Tools/Verification/FunctionalVerification/VerificationIP/Pages/default.aspx (Accessed 2014/02/11)

[50] M. Carlberg, “RE: Synopsys Assertion-Based VIP”, Personal e-mail (Feb. 17, 2014)

[51] Test and Verification Solutions, “Verification IP”, [Online]. Available: http://testandverification.com/solutions/vip/ Accessed 2014/02/13)

[52] M. Bartley, “RE: D&R Web Activity Notification - asureVIP_AHB_30_Lite_M”, Personal e-mail (Feb. 16, 2014)

[53] R. Drechsler, “ASSERTION-BASED VERIFICATION,” in *Advanced Formal Verification*. New York/Boston/Dordrecht/London/Moscow, Kluwer Academic Publishers, 2004, ch.5, pp 168,180

[54] A. Gurung, V. Roy (2007, Apr 15) “Recommendations for Developing an Assertion Based Protocol VIP for Formal Analysis”, [Online]. Available: http://www.cadence.com/rl/Resources/conference_papers/PP%20GurungA%20Cadence.pdf (Accessed 2014/03/11)

[55] P. Dasgupta, “Writing Unsatisfiable Specifications,” in *A Roadmap for Formal Verification.* The Netherlands, Springer, 2006, ch 4, pp 103-104.

[56] J. Andersen, P. Jensen (2005, Oct 27) “Leveraging Assertion Based Verification by using Magellan”, [Online]. Available: https://www.syosil.com/files/publications/bsnug05_final_andersen_jensen.pdf (Accessed 2014/03/11)

Appendix A

A Techniques and Algorithms for Formal Verification

The following section presents and discuss different techniques and algorithms used for formal verification from a mathematical perspective. Although hidden from the user at the user interface during usage of industrial formal tools, the author consider it relevant to understand the basics of the engines running underneath the hood of the formal verification tools. Depending on the intent of the formal verification, different engines optimized for different intentions can be chosen.

A.1 Binary Decision Diagram

The BDD is used in the symbolic model checking (SMC), for which a binary decision tree is reduced by merging common states with the same transition paths while maintaining an ordering hierarchy. Through optimizing the state diagram tree, the load on the formal verification tool drastically shrinks for a design which consists of a large amount of design blocks. Consider the example circuit depicted in Figure A1 below.

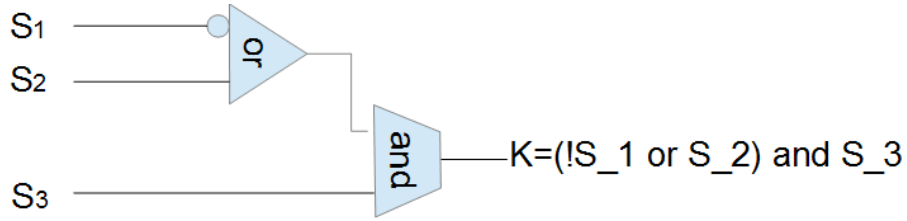

Figure A1: Example circuit for BDD explanation.

The inputs of the circuit and its resulting outputs can be summarized in the following table:

S_1	S_2	S_3	K
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table A1: Resulting table of example circuit

This result of Table A1 can further be depicted as a decision tree, which is illustrated in

Figure A2. Each variable goes down through different directions depending on if that variable is low or high. This is also shown by a dashed line for low and filled line for high. Thus through traversing down the tree, the output of the three inputs are shown in a square box.

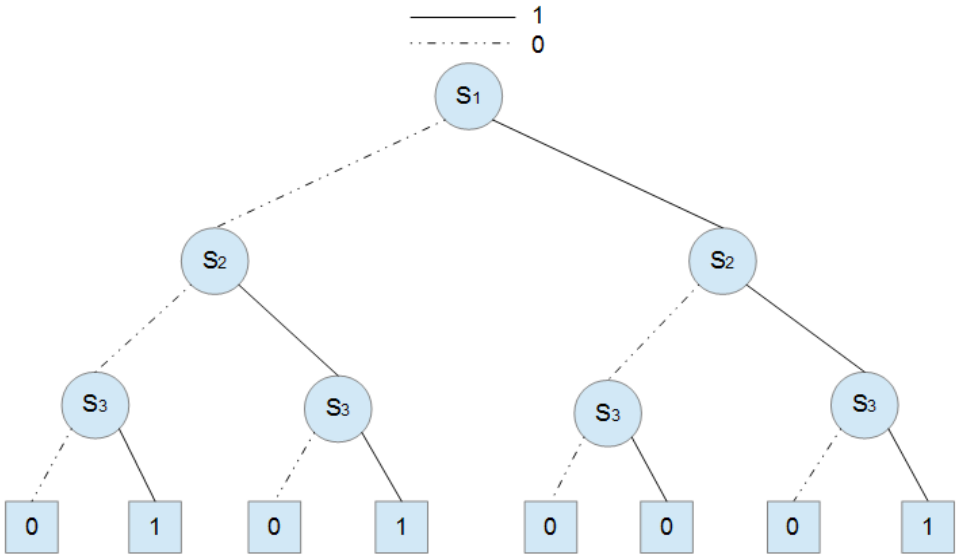

Figure A2: Decision tree of example circuit.

This tree can further be optimized through merging common leaves which would yield the following, intermediate tree in Figure A3

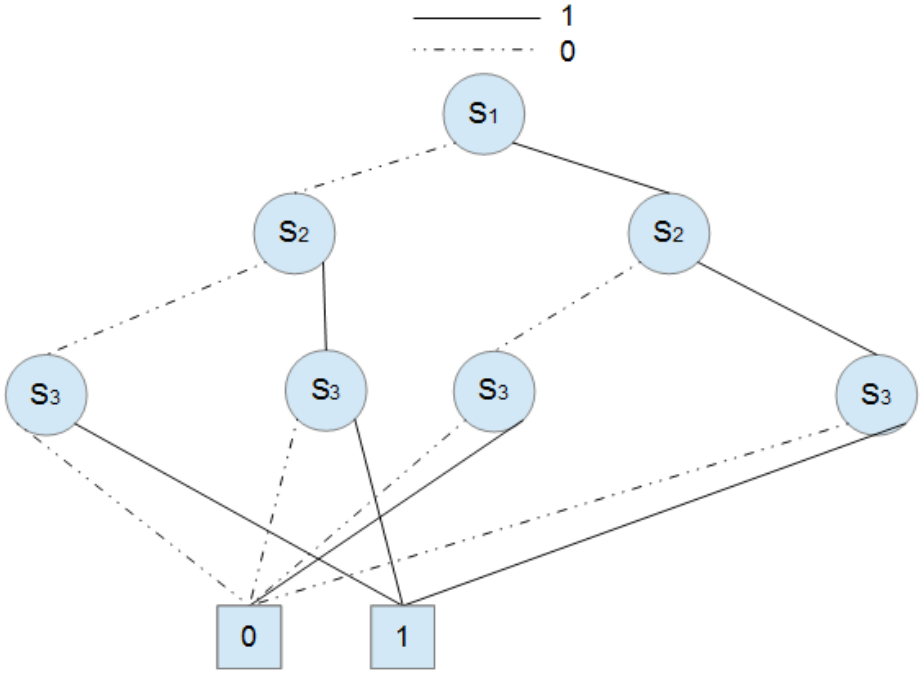

Figure A3: Merging of leaves.

Further optimization is performed on the common intermediate vertices which shares the same transition downwards in the tree, depicted in Figure A4 below.

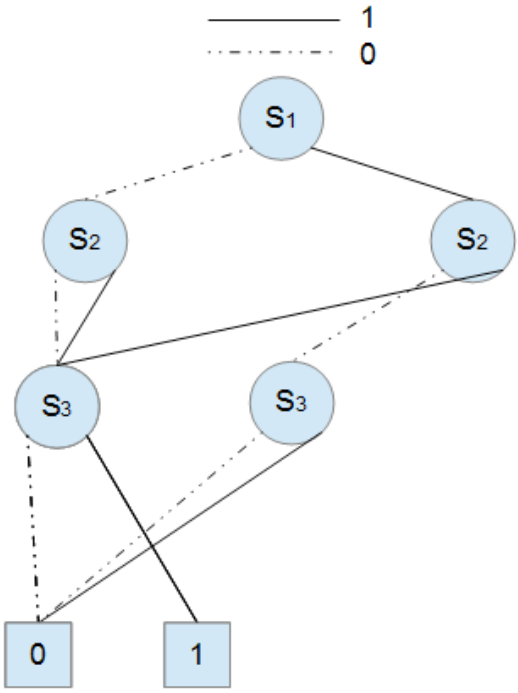

Figure A4: Further reduction of tree.

In the end the tree structure ends up with redundant states for which the following transition or outcome is the same regardless of the value on the variable state. Thus the optimal BDD is achieved in Figure A5 and still corresponds to the result in Table A1 on page 52.

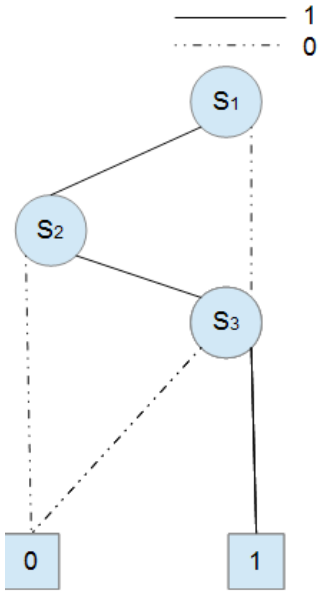

Figure A5: Optimal tree structure through reduction.

A.2 Satisfiability

In order to formally verify a circuit, its functionality may be expressed in conjunctive normal form (CNF) which uses a conjunction of AND operators, disjunctions of OR operators, both for literals of a variable of interest. Said variable might be s , which in that case its literals are either false, $\neg s$, or true s . The disjunction are called clauses for which one clause has to evaluate to true in order for the formula to be considered verified. An example is given below.

$$(a+\neg c)(b+\neg c)(\neg a+\neg b+c) \tag{A1}$$

In equation (A1) we can see that the formula consists of three clauses with literals inside of them. If any of these clauses evaluates to false, the whole formula becomes false and becomes unsatisfiable. If one literal, say $a=1$ is assigned, then $c=0$ is compulsory in order for the first clause to become true. This referred to as implication on contrast to assignment. In the third clause in (A1) this forces $b=0$ so the whole equation becomes true. These forced implications are also known as Boolean Constraint Propagation (BCP).

A conflict occurs if assignments results in implications which can not be resolved. Assume the following equation below.

$$(a+\neg b)(c+b)(b+a+\neg c) \tag{A2}$$

In equation (A2), if $a=0$ is assigned, then $b=0$ is implied for the first clause. In order to make the second clause pass, $c=1$ is implied as well. The remaining third clause however results in a conflict since through the forced implications this clause will always be false.

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm is used to traverse through assignments of variables and implications. It states that if a conflict occurs through an implication for which it cannot be resolved, then one has obtained proof that the CNF is non-satisfiable. If one however find a case for which all clauses are true after all variables have been assigned through implications, then a witness has been found which states that the CNF is satisfiable.

In order to formally verify a property of a design, the tool converts said property to logic which can be combinatorial logic or more complex sequential logic. The goal is to find a witness who can claim that the property is satisfiable. If not, it find proof that it can never exist. This proof is also referred to as counter example when SAT is used in bounded model checking (BMC).

Appendix B

B Market Research Extended Table Section

This appendix section presents extensive tables from the market research phase in chapter 3 on page 11 for the interested reader.

B.1 Cadence Complete VIP Protocol List

Protocol col 1	crVIP	accVIP	abVIP		Protocol col 2	crVIP	accVIP	abVIP
AMBA ACE	Yes	Yes	Yes		MIPI DSI	Yes	Yes	
AMBA AHB	Yes	Yes	Yes		MIPI LLI	Yes		
AMBA APB	Yes	Yes	Yes		MIPI M-PHY	Yes		
AMBA AXI 3, 4	Yes	Yes	Yes		MIPI SLIMbus	Yes		
AMBA Stream	Yes				MIPI UniPro	Yes		
CAN	Yes				Mobile PCI-E	Yes		
DDR2	Yes				NVM Express	Yes		
DDR3	Yes				OCP 2.2	Yes		Yes
DFI			Yes		OCP 3.0	Yes		Yes
DisplayPort	Yes				PCI	Yes		
eMMC 4.5	Yes				PCIe Gen 2	Yes	Yes	
eMMC 5	Yes				PCIe Gen 3	Yes	Yes	
Ethernet 10/100/1G/10G	Yes	Yes			PCIe MR-IOV	Yes		
Ethernet 40G/100G	Yes				PCIe SR-IOV	Yes		
Flash ONFi 3.0	Yes				PLB 6	Yes		
Flash Toggle	Yes				PPN Flash DDR	Yes		
HDMI 1.4	Yes	Yes			SAS 6G	Yes		
HDMI 2.0	Yes				SAS 12G	Yes		
I2C	Yes	Yes			SATA 3G		Yes	
I2S		Yes			SATA 6G	Yes		
JTAG	Yes				SD Card 4.0	Yes		
Keypad		Yes			SIMCARD		Yes	
LIN	Yes				SRIO	Yes		
LPDDR 3	Yes				SSIC	Yes		
LPDDR 4	Yes				UART	Yes		
LRDIMM	Yes				UFS 1.0	Yes		
LRDIMM DDR 4	Yes				USB 2.0	Yes		
MIPI CSI-2	Yes	Yes			USB 3.0	Yes		
MIPI CSI-3	Yes				Wide I/O	Yes		
MIPI D-PHY	Yes				Wide I/O 2	Yes		
MIPI DBI		Yes						

Table B1: Cadence complete VIP protocol availability list.

B.2 Jasper Complete VIP Protocol List

Protocol col 1	crVIP	accVIP	abVIP	Protocol col 2	crVIP	accVIP	abVIP
AMBA ACE	Yes		Yes	Ethernet 10/100/1G/10G	Yes		Yes
AMBA AHB	Yes		Yes	Ethernet 40G/100G	Yes		Yes
AMBA APB	Yes		Yes	LPDDR	Yes		Yes
AMBA AXI 3, 4	Yes		Yes	LPDDR 2	Yes		Yes
AMBA Stream	Yes		Yes	LPDDR 3	Yes		Yes
DDR	Yes		Yes	OCP 2.2	Yes		Yes
DDR 2	Yes		Yes	OCP 3.0	Yes		Yes
DDR 3	Yes		Yes	SDRAM	Yes		Yes
DFI	Yes		Yes				

Table B2: Jasper complete VIP protocol availability list.

B.3 Mentor Complete VIP Protocol List

Protocol col 1	crVIP	accVIP	abVIP	Protocol col 2	crVIP	accVIP	abVIP
AMBA ACE	Yes	Yes		JTAG	Yes	Yes	
AMBA APB	Yes	Yes	Yes	LPDDR 2	Yes	Yes	
AMBA AHB	Yes	Yes	Yes	LPDDR 3	Yes	Yes	
AMBA AXI 3,4	Yes	Yes	Yes	MIPI CSI-2	Yes	Yes	
DDR		Yes	Yes	MIPI DigRFv4	Yes	Yes	
DDR 2	Yes	Yes	Yes	MIPI DSI	Yes	Yes	
DDR 3	Yes	Yes		MIPI LLI	Yes	Yes	
DDR 4	Yes	Yes		MIPI M-PHY	Yes	Yes	
DisplayPort	Yes	Yes		PCI	Yes	Yes	Yes
eMMC 4.5	Yes	Yes		PCIe Gen 1	Yes	Yes	
Ethernet 10/100/1G/10G	Yes	Yes	Yes	PCIe Gen 2	Yes	Yes	
Ethernet 40G/100G	Yes	Yes	Yes	PCIe Gen 3	Yes	Yes	
Flash ONFi 2.1	Yes	Yes		SAS 6G	Yes	Yes	
Flash SAMSUNG	Yes	Yes		SAS 12G	Yes	Yes	
Flash SPANSION	Yes	Yes		SD Card	Yes	Yes	
Flash SPI	Yes	Yes		Smartcard	Yes	Yes	
GDDR 5	Yes	Yes		SPI	Yes	Yes	Yes
HDMI 1.4	Yes	Yes	Yes	SPI4-2	Yes	Yes	Yes
HDMI 2.0	Yes	Yes	Yes	UART	Yes	Yes	
I2C	Yes	Yes	Yes	USB 2.0	Yes	Yes	
I2C EEPROM	Yes	Yes		USB 3.0	Yes	Yes	
I2S	Yes	Yes		VByOne	Yes	Yes	

Table B3: Mentor complete VIP protocol availability list.

Appendix C

C AXI 4 Protocol

The protocols which was explored with the commercially available abVIPs is the AMBA AXI 4 interface protocol as stated in the section 3.3 -”Research Conclusion“ on page 16. This appendix covers some basic characteristics of the AXI 4 interface protocol. The ARM IHI 0022D specifications and an AXI 4 reference guide have been used for this appendix and is available to the public [2].

The AXI 4 is a high-performance interface protocol which is suited for designs demanding high bandwidth and low latency. Among many of its key features it supports out of order transaction completion.

C.1 AXI 4 Basic Interface Concept

The following subsections covers some fundamental properties of the AXI 4 interface concept. If any of the tables are referring to page or section number, they are referring to the previously mentioned openly available specification which can be found in [2]. Figure C1 shows an example where the AXI 4 interface might by found in an interconnect scheme.

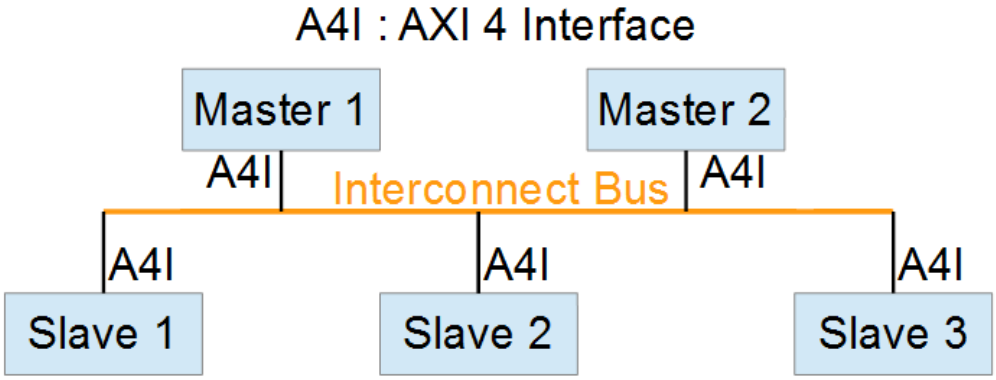

Figure C1: AXI 4 Interconnect example.

There exist interface definition between master and interconnect, slave and interconnect as well as between master and slave directly. For simplicity, only the interaction between a single master and a single slave is considered in this appendix chapter, to highlight the basic functionality of the AXI 4 interface protocol. The reason is that this interface protocol's abVIP is used as an evaluation and not for development. For additional specifications for multiple transaction, refer to the openly available specifications at [2].

C.1.1 AXI 4 Channel Types

The AXI 4 interface protocol is burst based, meaning data is transmitted repeatedly without any interruptions or need of external instructions. The AXI 4 consist of the 5 following independent transaction channels, which transmit only in one preset direction:

- read address
- read data
- write address
- write data
- write response

Much like the name suggests, data channels carries data information that should either be read or written. The address channels carriers the address to the destination devices, as well as control information of the data to be transmitted. The last channel type, "write response", is used to signal the completion of a transfer, for example a slave signaling a master that the data transmission is done.

Each of these 5 channels consists of two internal signals; a VALID and a READY signal, which together acts as a handshake synchronization mechanism. On top of this, the read and write data channels have a LAST signal which indicates that the last item of the data has been transmitted in the transaction.

The read data channel relays both the read data and the response from the destination slave to the source master, indicating the completion of the transaction. It also includes the data bus which can take the values 2^N with $3 \leq N \leq 10$.

The write data channel carries only the data to be written from the master to the slave. Again, the data bus can take the the values 2^N with $3 \leq N \leq 10$, but also a byte strobe signal after each 8 bit. This strobe signal indicates which bytes are valid and not. There is no communication with the slave, instead the writes are buffered so no acknowledgment is required from the slave. Unlike read data channel, the response is not sent from this channel; write response channel is a stand alone channel.

The write response channel is used by the slave to tell the master that a write has successfully been performed. Although this acknowledgment is required for each write transaction, it does not have to occur immediately. The AXI 4 interface protocol supports out of order transactions while the write data channel is buffered, so it is not required for a master to receive an acknowledgment from a slave before starting a new write. Instead, in the end, an acknowledge for every write is required.

Figure C2 on page 60 summarizes the 5 different channels and their constant data

transactions direction. The two upper channels are associated with the read transaction and the three lower channels with write transaction, hence their different shades of the color gray.

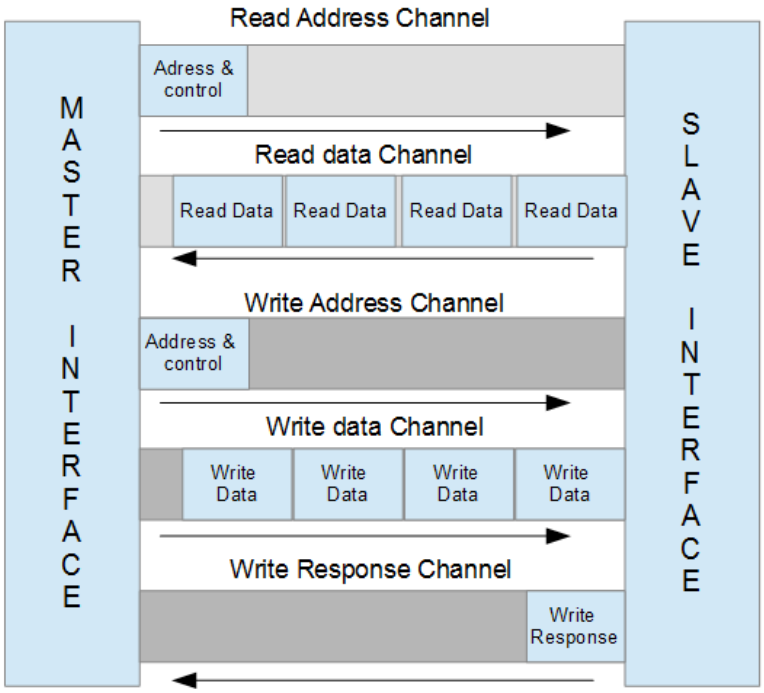

Figure C2: AXI 4 channel types and directions.

C.1.2 AXI 4 Signals

Apart from the global signals, each channel have their own set of internal signals which are presented throughout tables in this subsection. All signals in this subsections are sampled on the rising edge of the global clock and each AXI components uses this single global clock. The following Table C1 presents the global clock and active low reset used for the interface.

Signal Name	Source Device	Explanation
ACLK	Clock source	The device global signal
ARESETn	Reset source	The device global reset, active LOW

Table C1: AXI 4 global signals.

Table C2 on page 61 summarizes the signals used for the write and read address channel, which is indicated with the "AW" and "AR" label respectively. These two channel signals shares the same table for their similarity in signal name and explanation.

The write data channel signal begins with the label "W", as can be seen in Table C3 on page

62 while the read data channel signal uses "R" as its label, shown in Table C4 on page 62. Similarly, write response channel uses "B" as a label for its signal, which can be seen in Table C5 on page 62.

Signal Name	Source Device	Explanation
AWID/ARID	Master	Write/Read address ID, used as an identification tag for the destination
AWADDR/ARADDR	Master	Write/Read address, yields address of first data in a burst transfer
AWLEN/ARLEN	Master	Burst length, yields number of transfers of address
AWSIZE/ARSIZE	Master	Burst size, determines size of transfer in the burst
AWBURST/ARBURST	Master	Burst type, determine how each burst type is calculated
AWLOCK	Master	Lock type, details about atomic characteristics
AWCACHE/ARCACHE	Master	Memory type, describes how transactions traverse through the system
AWPROT/ARPROT	Master	Protection type, determines privilege and security type
AWQOS/ARQOS	Master	Quality of Service
AWREGION/ARREGION	Master	Region Identifier, may enable one interface on slave to be used for multiple interfaces
AWUSER/ARUSER	Master	User Signal, user defined signal
AWVALID/ARVALID	Master	Write/Read address valid, indicates valid write/read address and control information
AWREADY/ARREADY	Slave	Write/Read address ready, indicates whether or not slave is ready to accept address and control information

Table C2: AXI 4 write/read address channel signals.

Signal Name	Source Device	Explanation
WDATA	Master	Write data, explanation in name
WSTRB	Master	Write strobes, indicates which byte lane hold valid data
WLAST	Master	Last write, indicates which entry is the last in the transaction
WUSER	Master	User Signal, user defined signal
WVALID	Master	Valid signal, indicates if valid data and strobe are available
WREADY	Slave	Ready signal, indicates whether the slave can accept and more write data

Table C3: AXI 4 write/read data channel signals.

Signal Name	Source Device	Explanation
RID	Slave	Read ID tag
RDATA	Slave	Read data, explanation in name
RLAST	Slave	Last read, indicates which entry is the last in the transaction
RRESP	Slave	Read response, indicates status of transaction
RUSER	Slave	User Signal, user defined signal
RVALID	Slave	Valid signal, indicates if valid data is available
RREADY	Master	Ready signal, indicates whether the master can accept and more read data

Table C4: AXI 4 read data channel signals.

Signal Name	Source Device	Explanation
BID	Slave	Response ID tag
BRESP	Slave	Write response, indicates status of transaction
BUSER	Slave	User Signal, user defined signal
BVALID	Slave	Valid response, indicates if channel signaling valid response.
BREADY	Master	Ready signal, indicates whether the master can receive a write response.

Table C5: AXI 4 write response channel signals.

C.1.3 AXI 4 Interface Reset Behavior

The global reset ARESETn can be asserted whenever necessary, but has to fall in synchronization with ACLK. During the reset, all VALID signals has to be driven low by their respective source device whilst any other signal may be driven to any value. The specifications mentions that the earliest time ARVALID, AWVALID and WVALID may be HIGH is at the first rising edge of ACLK after ARESETn is HIGH. This can be seen in Figure C3.

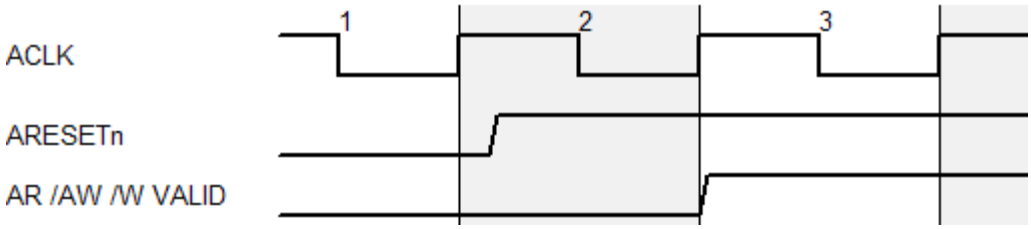

Figure C3: AXI 4 Reset behavior.

C.1.4 AXI 4 Handshake Mechanism

There are two signals which exists in every channel type; VALID and READY. These two are used as a handshake mechanism to synchronize and process data transfer, which can be considered as the main mechanism of the AXI 4 interface protocol. The source controls the VALID signals to initiate a data transfer through indicating whether address, data or control information is available. On the other end, the destination controls the READY signal to indicate that the device is ready to accept this information. A data transfer can only occur when both VALID and READY is both HIGH, as depicted in Figure C4. Here shaded waveform indicates unknown/irrelevant values. The actual transfer occurs on the first rising edge of ACLK while both VALID and READY is HIGH. Once data is transmitted, the data toggles and VALID and READY falls to LOW.

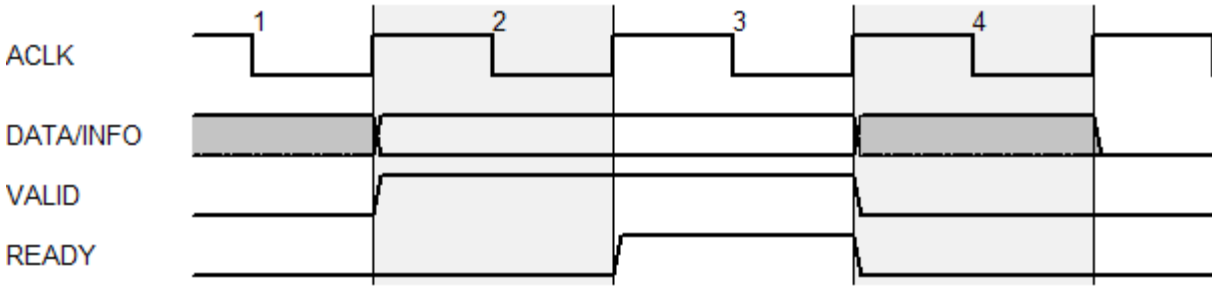

Figure C4: AXI 4 handshake mechanism, valid-ready.

Notice in Figure C4 that the information is held stable during the time VALID is waiting for READY. Here VALID is not allowed to fall whilst waiting for READY to rise. If READY is asserted before VALID however, it may fall before VALID is asserted. A transaction can still occur, even if READY is HIGH before VALID, which is depicted in Figure C5. Notice in this picture however that the information is only relevant while the VALID is HIGH.

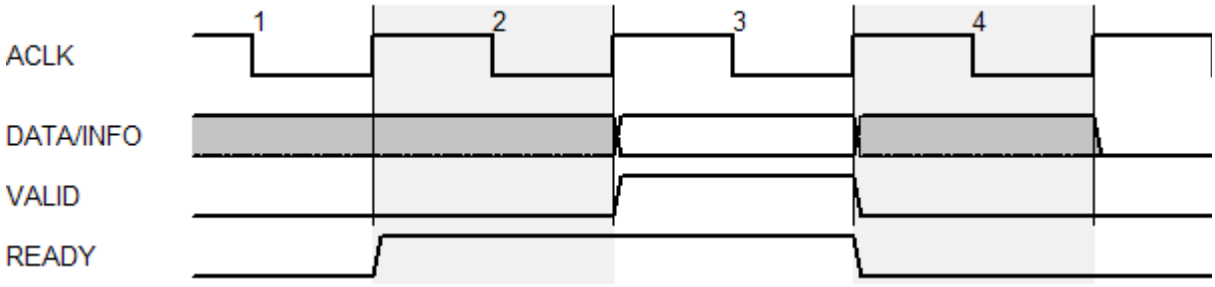

Figure C5: AXI 4 handshake mechanism, ready -valid.

The third case would be that VALID and READY rises at the same time, meaning in Figure C5 the READY signal would be postpone to be parallel with VALID. In that case, the data is transferred at first rising edge of ACLK as well.

This handshake procedure is performed by all channels, as they all have their own set of VALID and READY signals, which is summarized and can be seen in Table C6 on page 64.

Channel Type	Handshake signal pair
Read address channel	ARVALID - ARREADY
Read data channel	RVALID - RREADY
Write address channel	AWVALID - AWREADY
Write data channel	WVALID - WREADY
Write response channel	BVALID - BREADY

Table C6: AXI 4 channel type handshake pairs.

Common for each handshake pair is that after a master has asserted its respective VALID signal, it must remain stable until its respective READY signal is asserted as well.

C.1.5 AXI 4 Channel Handshake Relation

A set of rules exists in order to prevent any deadlocks in the system, which could be a result of the 5 different synchronization mechanisms communicating with each other. Depending whether the data direction is a read, response or a write the dependencies may vary. These dependencies together with their direction type is listed in Table C7 on page 66.

The read dependencies are shown in Figure C6 where single arrow heads show that the appointed signal may rise regardless whether the signal, said arrow starts from, is asserted or not. Double headed arrows indicates that the origin signal has to be asserted before the appointed signal may be asserted. Gray signals indicates master signals while blue ones

indicate slave signals. These figure properties are the same for Figure C7 and Figure C8 on page 67 as well.

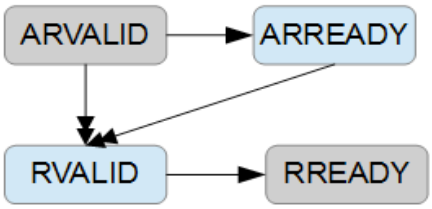

Figure C6: AXI 4 read channel dependencies.

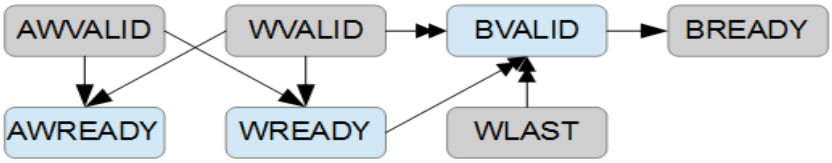

Figure C7: AXI 4 write channel dependencies.

Dependency	Direction	Figure
Master shall not wait for ARREADY before asserting ARVALID	Read	Figure C6
Slave may wait for ARVALID before asserting ARREADY, but may also assert ARREADY before ARVALID is asserted	Read	Figure C6
ARVALID, ARREADY both required to be HIGH before slave may assert RVALID	Read	Figure C6
Slave shall not wait for master to assert RREADY before asserting RVALID	Read	Figure C6
Master may wait for RVALID before asserting RREADY, but may also assert RREADY before RVALID is asserted	Read	Figure C6
Master shall not wait for slave to assert AWREADY or WREADY before asserting AWVALID or WVALID	Write	Figure C7
Slave may wait for AWVALID,WVALID or both before asserting AWREADY or WREADY	Write	Figure C7
Slave may assert AWREADY or WREADY before master assert AWVALID, WVALID or both	Write	Figure C7
Slave must wait for WVALID and WREADY and WLAST before asserting BVALID	Write	Figure C7
Slave shall not wait for master to assert BREADY before asserting BVALID	Write	Figure C7
Master may wait for BVALID before asserting BREADY, but may also assert BREADY before BVALID is asserted.	Write	Figure C7
Master shall not wait for slave to assert AWREADY or WREADY before asserting AWVALID or WVALID	Response	Figure C8
Slave may wait for AWVALID,WVALID or both before asserting AWREADY or WREADY	Response	Figure C8
Slave may assert AWREADY or WREADY before master assert AWVALID, WVALID or both	Response	Figure C8
Slave must wait for AWVALID, AWREADY, WVALID,WREADY and WLAST before asserting BVALID	Response	Figure C8
Slave shall not wait for master to assert BREADY before asserting BVALID	Response	Figure C8
Master may wait for BVALID before asserting BREADY, but may also assert BREADY before BVALID is asserted.	Response	Figure C8

Table C7: AXI 4 Channel Dependency table.

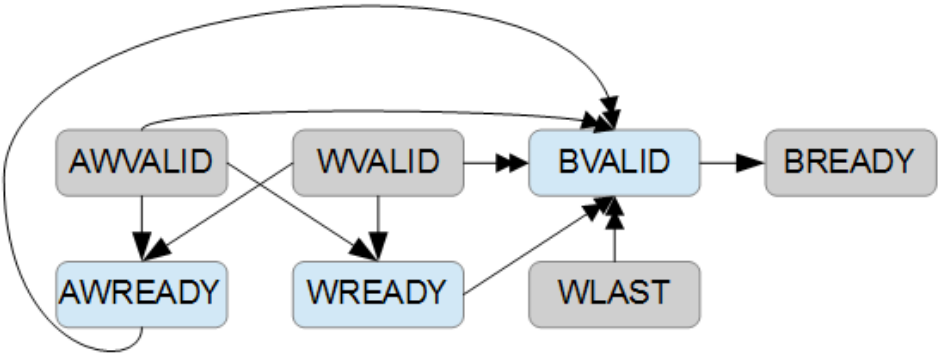

Figure C8: AXI 4 response dependencies.

C.2 AXI 4 Remaining Specification and AXI 4 Lite

The complete specification of the AXI 4 interface protocol can be found, as previously mentioned, in the publicly available specification [2]. Apart from the basic functionality concept described in section C.1 -” AXI 4 Basic Interface Concept“ on page 58, there exist many more specifications and restriction for the AXI 4 interface protocol, examples being ordering models, atomic access, low power signaling schemes and more. The main purpose of this appendix was to give the reader broad overview of the AXI 4 interface protocol and how it functions on its fundamental level, rather than an in depth description.

There is also a smaller version of the AXI 4 interface protocol, which is the AXI 4 Lite protocol. This is simply a subset of the complete AXI 4 protocol, which excludes certain features and ports from the bigger set [3]. Table C8 summarized which of the ports presented throughout section C.1 on page 58 that are excluded in the Lite version. Any features related to these signals the table below are thus not supported.

Signal Name	Signal name
AWID/ARID	WLAST
AWLEN/ARLEN	WUSER
AWSIZE/ARSIZE	BID
AWBURST/ARBURST	BUSER
AWLOCK	RID
AWQOS/ARQOS	RLAST
AWREGION/ARREGION	RUSER
AWUSER/ARUSER	

Table C8: AXI 4 Lite unsupported signals.

Appendix D

D I2C Bus Protocol

This appendix covers basic specification properties of the I2C protocol for any interested reader, to simplify the understanding of the development phase presented in chapter 5-”Development of Custom Assertion Based VIP“ on page 26.

The abVIP which Ericsson has requested adheres to the version 2.1 of the I2C protocol and implements a subset of these specifications, depending on which set of specifications are of interest for their design under verification (DUV) which uses the I2C protocol.

D.1 I2C Basic Bus Concept

The I2C is a bi-directional 2 wire bus which is a bus protocol that has been in the industry for quite some time. Version 1.0 was introduced by Philips at 1992 and has received some version updates since. The I2C bus protocol is today no longer a copyrighted protocol and its specifications are openly available for the public [4]. This sub section is an extracted summary of a subset of the openly available specification.

The bi-directional 2-wire bus consist of a serial data line (SDA) and a serial clock line (SCL) for which every basic operation of the bus protocol can be traced back to. There is at least one master and one slave which communicate with each other through sending sets of 9 bits in bursts; an address or data byte being 8 of those bits together with a 9th acknowledgment bit. The following sub sections describes the general behavior of the I2C bus protocol and the devices attached to the bus more thoroughly, to give the reader a overview of the I2C bus protocol.

D.1.1 I2C Master and Slave

Devices attached to the SDA and SCL lines can be a transmitter, receiver or both. Depending on which device is initiating a data transfer, and which device as a result is addressed by this transfer, devices are temporarily masters and slaves. This is described more thoroughly with the help of Figure D1.

Figure D1: I2C Master and Slave example.

Assume that both device A and B are capable of transmitting and receiving. At one time, device A may initiate a data transfer to device B, either to transmit or receive data to or from it, and is thus the temporal master at that time whilst B at the same time is the slave. Immediately after A is done with its transfer, B may very well be the next temporal master by initiating a data transfer to A, making device A the latest temporal slave.

It is the responsibility of the master to generate a clock behavior on the SCL line during its control duration of the bus.

D.1.2 I2C Bus Availability

Before devices can attempt to access the bus, the bus has to be free. The bus is considered to be free when both SDA and SCL is constantly HIGH ('1'). Once a master has been assigned, the master pulls down SDA from HIGH to LOW ('0') when SCL is still HIGH. This condition is known as a Start condition (S). Once an S condition has been issued, the bus is considered to be busy and the I2C transmission occurs, toggling both SDA and SCL depending on its needs. Once the master pulls SDA from LOW to HIGH when SCL is HIGH, a Stop condition (P) has occurred and the bus is considered to be free again. The time between an S and a P condition is the time slot a certain master occupies the bus. This behavior can be seen in Figure D2. In this figure, dashed lines means unknown values for a prolonged time.

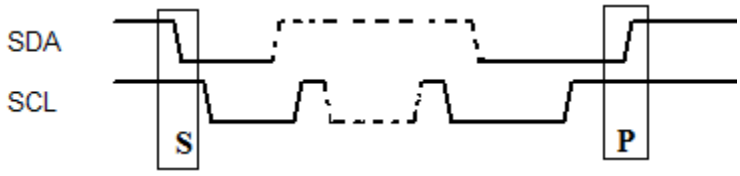

Figure D2: I2C Start and Stop condition example.

Apart from the previous conditions mentioned, there is a 3rd alternative known as the repeated Start condition (Sr), when a temporal master do not wish to release the hold of the bus and instead want to possibly address another device, which then becomes the newest slave. The behavior on the SDA and SCL lines are identical for both S and Sr conditions; an Sr condition is simply a repeated S condition. The master is always the device that produces the S or P conditions, regardless if it reads or writes from/to the slave.

D.1.3 I2C Data Validity

Data on the SDA line must be stable when SCL is HIGH to be considered valid during a data transfer. Thus during a data transfer, data on the SDA line may only toggle when SCL is LOW to be considered valid data. Note that the S and P conditions are dependent on the change of the value on SDA when SCL is HIGH. They do not occur during a data transfer however; starting/stopping conditions are the extreme values for the data transfer and is thus not a violation to the data stability requirement.

D.1.4 I2C 7 Bit Addressing and Data Transfer

Devices can address each other through either through 7 bit or 10 bit addressing schemes. In the case of a 7 bit data transfer, the SDA line gets filled with the 7 bit address together with one bit for the data direction, read or write (R/W). The initial bytes' LSB, the R/W bit, is LOW for write and HIGH for read operations. This resulting byte is followed by either an acknowledgment (Ack), which is a stable LOW on the SDA line, or a non-acknowledgment (Nack), which is a stable HIGH on the SDA line during the 9th SCL pulse. The type of acknowledgment indicates the status of the transfer. A Nack is always followed by an Sr or a P condition generated by the master, regardless of when the Nack occurs or data direction. A Nack is however not required for an Sr or P condition to occur; even if the slave produces an Ack, the master can still terminate the transaction through a P condition.

If the address byte is acknowledged, then a data byte is produced on the SDA line on the following periodic SCL pulses. The amount of data bytes to be sent can be unrestricted and is decided by the masters command but also the slaves capability to handle the data sent. Figure D3 demonstrates an example waveform for a data transfer. In this figure, the bits of the address and data bytes can be either HIGH or LOW, indicated by a box.

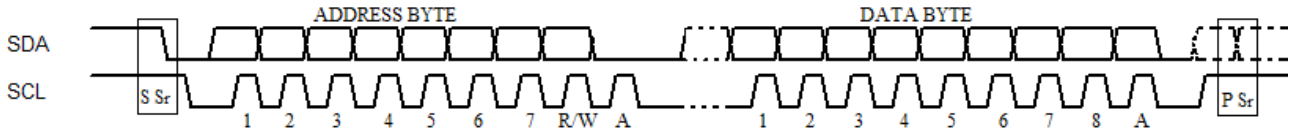

Figure D3: I2C Data validity, addressing and transfer example.

The first byte, which begins with the most significant bit (MSB), sent by the master is the address to the slave it wants to initiate a transfer with, together with the transaction direction read or write. Between the Ack and the first bit of the next byte (9th and 10th bit counting from left), the slave have the permission to force the SCL line to LOW so that master is put in a wait state. This is the only occasion the master does not have complete control of the SCL line. The master is put in a wait state if the slave needs additional time to process the byte sent, so any data put in by the master is not valid at that time. This behavior is known as clock stretching and is the only time the SCL is not periodic during a data transfer.

For the address byte, it is always the master, regardless if its receiving or transmitting, which addresses the slave. For the address byte, it is always the master which initially drives SDA HIGH to make the default choice of a Nack. If no slave acknowledges the address put on the SDA line, the default, untouched, result on the SDA line is a Nack. This is indicated in Figure D4 below. Thus during the acknowledgment pulse for the very first byte, the so called address byte, it is always the slave which drives the SDA LOW for an Ack or leaves SDA HIGH for a Nack.

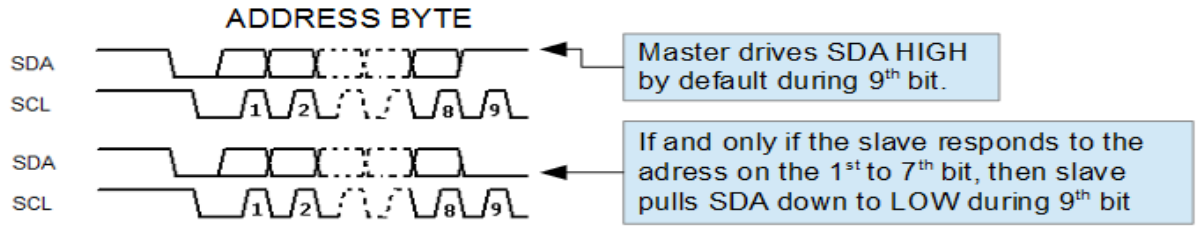

Figure D4: I2C Nack (top) and Ack (bottom) behavior for the address byte.

This behavior is however only true for the address byte. For the remaining data bytes, regardless of how many, it is the transmitter who automatically puts a Nack on the SDA lines, whilst the receiver drives it low, to indicate that it can still receive data. If the receiver can not receive any more data, it produces a Nack on the SDA line. A stop or repeated start condition can however occur even if the receiver produces an Ack; an example on this scenario could be if the transmitter sends 1 byte while the slave is capable of storing 2 bytes. During a write transmission, the master is the transmitter and the slave the receiver. During a read transmission, the master is the receiver and the slave the transmitter.

D.1.5 I2C 10 Bit Addressing and Data Transfer

In the case of a 10 bit addressing mode, the transaction scheme is a bit different. Two bytes have to be sent in order to cover the 10 necessary addressing bits, where the direction bit in the first byte should initially always be 0. As 7 and 10 bit devices can be connected to the bus at the same time, the master indicates that it wants to address a 10 bit device through sending the 7 bits 1111_0XX on the SDA line. The first 5 bits 1111_0 are reserved for this purpose, and XX indicates the two first bits of the desired slave to address, followed by a 0 for the R/W bit regardless of data transfer direction.

The two MSB XX , for the actual address, will result in the first acknowledgment, A1. Here multiple slave candidates may very well answer the call, but no data transaction is performed until a 2nd acknowledgment, A2, is generated. A2 is generated when the remaining 8 bits in the 2nd byte indicates the remaining address bits in the 10 bit addressing scheme. This base case transaction is shown in Figure D5.

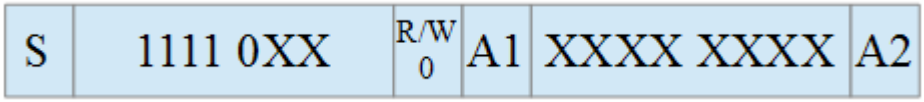

Figure D5: I2C 10 bit addressing base.

If the intention of the data transfer is a write from the beginning, then the data can now be transmitted from the master to the slave, quite similar to the 7 bit addressing mode.

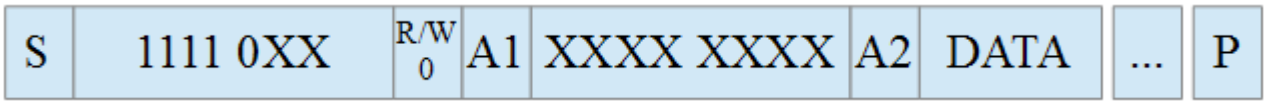

Figure D6: I2C 10 bit addressing simple write.

It can be seen in Figure D6 that once a slave has been properly located and an A2 has been assigned, the write data transfer can commence and eventually result in a P condition. In the case for a simple read the very same base case used in Figure D5 will still be used. This sequence will be complemented with an Sr condition followed by the reserved 1111 0XX again. Due to the repeated start condition, the slave recognizes the remaining 8 bits between the previously assigned A1 and A2 and yields another acknowledge, A3, after a read direction has been given on the LSB of the 3rd byte. Data can then be transferred as usual. This complete sequence is shown in Figure D7.

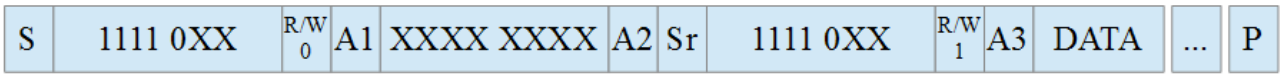

Figure D7: I2C 10 bit addressing simple read.

Using these simple write and read operations, it is possible to combine them so a master may both write and read to a device in a desired order. This is referred to as a combined format.

Further more, the 10 bit read format also has a special form of combined format; if a 10 bit read instruction is issued after a 10 bit write instruction, then the master can initiate the actual read transaction after the very first repeated byte. This behavior together with all the 10 bit transaction schemes can be seen in Figure D8.

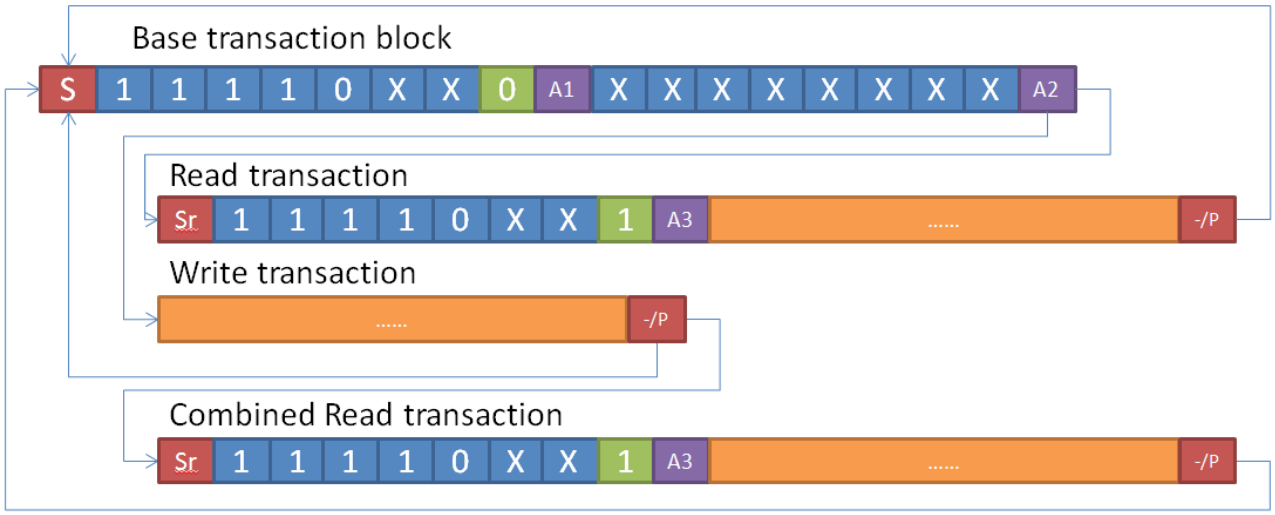

Figure D8: I2C 10 bit transaction chart.

D.1.6 I2C General Call Addressing

General call addressing is a potential I2C method of the I2C master where it address all connected slave instances to the SDA and SCL line. The slaves which are capable of answering this call have the opportunity to either change their address or receive data from the master through write transactions. In both cases, the master sends the address 8'h00. The second byte is however different depending on the type of general call.

The first case is known as non-hardware general call, where the LSB of the 2nd byte is 0, as indicated by Figure D9. For the non-hardware call, there are two scenarios of the 2nd byte; 8'h06 (H06) and 8'h04(H04). Both these scenarios have the potential of changing the current address of the slave who accepts this type of the general call. In practice, this can be done by replacing certain address pins with other pins.

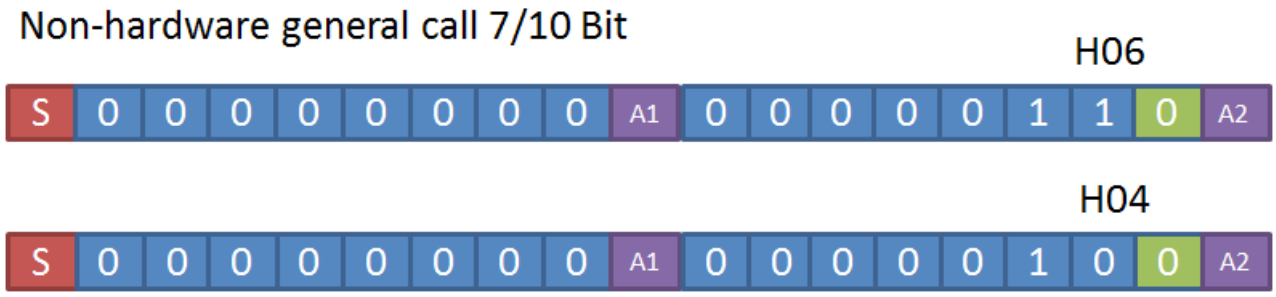

Figure D9: I2C non-hardware general call.

The 2nd type of general call is the hardware general call, where the LSB of the 2nd byte is 1, as indicated by Figure D10. Although the exact procedure differs slightly from 7 and 10 bit, the idea to send the master devices address is the same. Once the master address has been sent, the master initiates a write transaction to all the slave who answered the call.

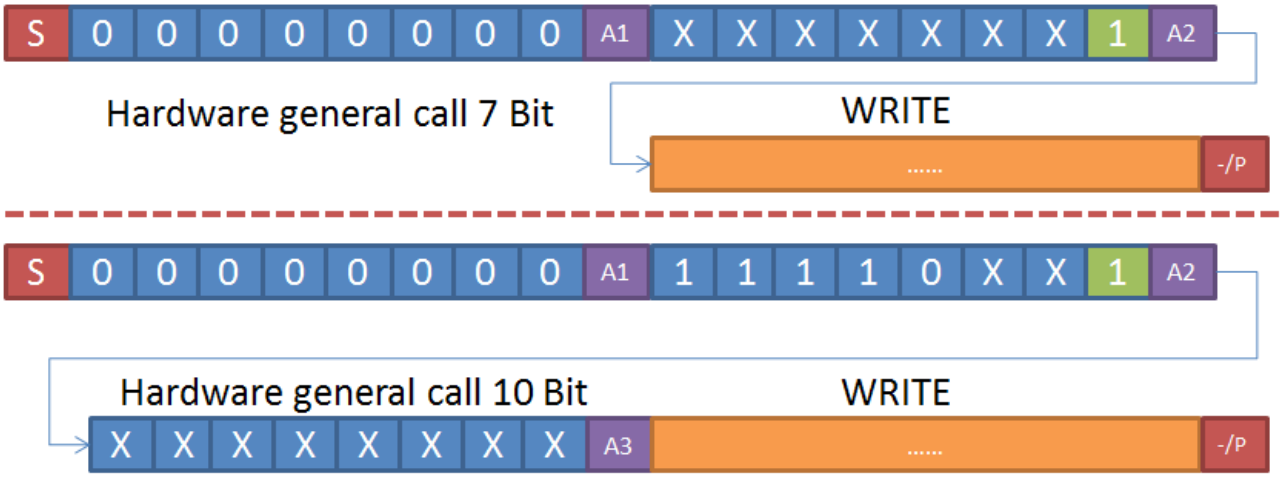

Figure D10: I2C hardware general call, both 7 and 10 bit.

Hardware general call is often used by an external micro-controller who directs instructions on a higher level than the I2C protocol.

D.1.7 I2C Start Byte

Start byte is essentially a prolonged start condition which no slave instance is allowed to acknowledge. It is used in order to synchronize I2C devices which operates on different frequencies for which skew can occur.

D.1.8 I2C Multi Master Arbitration

The I2C protocol allows multiple devices capable of being masters to be present on the bus at the same time. In order to avoid data loss, collision and corruption the potential masters have to compete in order to earn the right to be the current master of the bus. When multiple master candidates initiates an S condition on a free bus in the same time slot, an arbitration process takes places from the succeeding cycle. Through a wired-AND connection to the SDA line, each master tries to put their own information on this connection. The first masters input to the ANDed connection which generates a 1, whilst the other master generates a 0, loses the arbitration. This behavior is summarized in Table D1.

Candidate 1 - Input	Candidate 2 - Input	Output	Master
0	0	0	-
0	1	0	Candidate 1
1	0	0	Candidate 2
1	1	1	-

Table D1: I2C arbitration decision table example.

This process of arbitration is occurring in parallel to a synchronized SCL line, where the arbitration on the SDA line occurs when the synchronized SCL is HIGH. Similarly to the SDA, the synchronized SCL is achieved through a wired-AND connection, for which the competing candidates put their own clock inputs on. The combination of these inputs results in the synchronized SCL, visible in Figure D11.

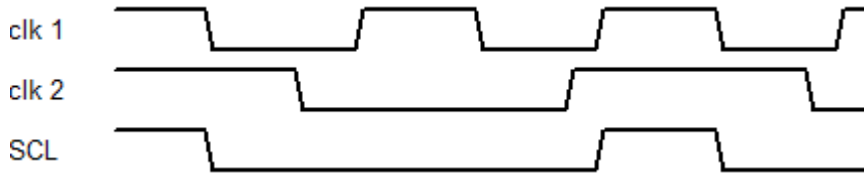

Figure D11: I2C Synchronized clock example.

Together with the synchronized SCL, the arbitration of the master can look as in Figure D12.

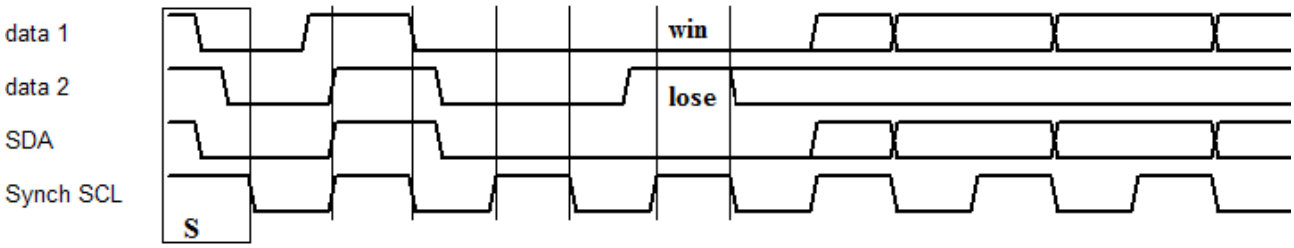

Figure D12: I2C Final arbitration example. C1 wins while C2 loses.

Since the candidate 1 (C1) produces a LOW value on its data input whilst candidate 2 (C2) produce a HIGH value while the synchronized SCL is HIGH, C1 wins the arbitration and becomes the master shortly after. The losing C2 produce a constant HIGH value, which does not interfere with the master.

D.1.9 I2C Timing Requirements

The openly available specification presents various electrical and timing related requirements for their signals in the I2C protocol. For this section, only the timing requirements for the SCL and SDA lines in standard as well as fast mode are mentioned. These are presented in Table D2 on page 75 where S stands for standard mode, F fast mode while min and max stands for the minimum and maximum values allowed for that mode.

The timing requirements presented in Table D2 can be checked in simulation based verification. In formal based verification, the notion of time is restricted to atomic clock cycle and are thus hard to prove.

Parameter	S min	S max	F min	F max	unit
Hold time (Sr)	4	-	0,6	-	µs
LOW period SCL	4,7	-	1,3	-	µs
HIGH period SCL	4	-	0,6	-	µs
Setup time (Sr)	4,7	-	0,6	-	µs
Setup time SDA	250	-	100	-	µs
Rise time SDA SCL	-	1000	20	300	ns
Fall time SDA SCL	-	300	20	300	ns
Setup time (P)	4	-	0,6	-	ns
Free time between (P)&(S)	4,7	-	1,3	-	µs

Table D2: I2C timing requirement.

D.1.10 I2C Predefined Address Bytes

Table D3 presents some predefined bytes that may or may not be used depending on the supported modes of the I2C protocol for the first address byte. “X” is a wildcard, meaning that it does not matter whether that value is 0 or 1.

7 Bits	8th bit	Meaning
0000_000	0	General call address
0000_000	1	START byte
0000_001	X	CBUS address
0000_010	X	RESERVED
0000_011	X	RESERVED
0000_1XX	X	HS mode
1111_1XX	X	RESEVRED
1111_0XX	0	10bit addressing mode

Table D3: I2C predefined address bytes.

D.2 I2C Extracted Characteristics

This sub section lists the extracted characteristic for the I2C protocol interface, for which assertions may be written for the DUV implementing the I2C bus protocol. This extraction is presented in Table D4 on page 79 and may very well repeat the behavior of the I2C protocol previously described throughout section D.1 -” I2C Basic Bus Concept“ starting on page 68.

Extracted characteristic number	Found in section - page
1. Two bidirectional wires, SDA (Serial Data Line) and SCL (Serial Clock Line) is the major back bone of the bus; they both carry information and is used together in order to ensure harmony in the procedure.	Sec 4 – 6/46
2. Each device is recognized through their unique address, which can be either 7 bit or 10 bit. In the case of 7 bit, one bit for the direction of the transaction, either read or write depending on the value. In case of 10 bit, 2 bytes are used.	Sec 4 – 6/46
3. Each component may have the ability to be a Master and a Slave. The components which initiates a data transfer is the Master while the component which it refers to is the Slave. The Slave afterward might very well become a Master and the previous Master might very well be a Slave. The Master generates the clock signal to permit a data transfer.	Sec 4 – 6/46
4. I2C is a multi-master bus, which means that more than one component capable of controlling the bus may be connected to the bus at the same time. The selection of which master is active is made by the arbiter. This process relies on a wired-AND connection on both SDA and SCL. First candidate to produce a one when the other one produce a zero loses, on the SDA (while SCL HIGH).	Sec 4 – 7/46
5. Generation of the clock signal on the I2C bus is always the masters responsibility.	Sec 4 – 8/46
6. When bus is free, both SDA and SCL is HIGH.	Sec 5 – 8/46
7. Number of interfaces connected to the bus dependent on the maximum capacity of 400 pF.	Sec 5 – 8/46
8. Data on SDA must be stable during the HIGH period of SCL. Similarly HIGH or LOW state of SDA can only toggle when SCL is LOW.	Sec 6 – 8/46
9. HIGH to LOW transition on SDA whilst SCL is HIGH is a Start (S) condition. Note that this is not an issue of stable data for data transfer.	Sec 6 - 9/46
10. LOW to HIGH transition on SDA whilst SCL is HIGH is a Stop (P) condition. Note that this is not an issue of stable data for data transfer.	Sec 6 - 9/46
11. Bus is considered to be busy after an (S) condition and free after a (P) condition.	Sec 6 - 9/46
12. A Master can prolong its authority of the bus by sending a repeated Start (Sr) condition. During this time, the bus is still considered to be busy.	Sec 6 – 9/46
13. Each data is sent with the MSB first. LSB must be followed by an acknowledgment (Ack) or not acknowledgment (!Ack) on the SCL line.	Sec 7 – 10/46
14. A Slave can halt a Master from sending data by holding the SCL LOW. This is known as clock stretching.	Sec 7 – 10/46
15. No starting nor stopping condition may occur during a byte transfer.	Sec 7 – 10/46
16. An acknowledgment or non-acknowledgment must occur after each byte.	Sec 7 – 10/46

17. Transmitter releases SDA from HIGH during acknowledgment pulse.	Sec 7 – 10/46
18. Receiver holds SDA LOW during acknowledgment pulse.	Sec 7 – 10/46
19. If receiver can not acknowledge an address, it hold the SDA HIGH so that either a (P) or an (Sr) is generated, which must be generated after a non-acknowledgment.	Sec 7 – 11/46
20. Masters generate their own clock on the SCL line, for which data is only valid during the HIGH period. These clocks are used for the clock synchronization with the ANDed SCL wire.	Sec 8 – 11/46
21. Clock which belongs to the Master who won, can affect the SCL to go from HIGH to LOW when it does. Normally when this clock goes from LOW to HIGH, so shall SCL. But SCL can not go from LOW to HIGH unless all clocks are HIGH.	Sec 8 – 11/46
22. Master may only start a data transfer when bus is free.	Sec 8 – 12/46
23. Arbitration takes place when SCL is HIGH. Master loses if its data output is HIGH while the other Masters data output is LOW (Same procedure as for SDA).	Sec 8 – 12/46
24. Losing Master candidate of the arbitration must immediately switch to Slave mode, as it may be addressed by the winning master.	Sec 8 – 12/46
25. Losing Masters data output is turned off in the sense that it produces a constant high signal. This does not affect the winning Master.	Sec 8 – 12/46
26. Arbitration is not allowed between an (S) and data bit, data bit and (P) or (Sr) and (P).	Sec 8 – 13/46
27. Data transmission is achieved as follows for 7 bit addressing: 7 bit address sent together with direction bit (R/W) (8 bit in total). 8^{th} bit LOW indicates a write while a HIGH indicates a read.	Sec 9 – 13/46
28. After its first data transmission is over, a Master can request an (Sr) and address another slave without giving up the right of the bus.	Sec 9 – 13/46
29. There are three possible data transfer formats: -Master transmitter transmit to slave-receiver (transfer direction not changed). -Master reads slave immediately after first byte (Master-transmitter becomes Master-Receiver and vise verse for the Slave). -Combined format; Master can at one time read or write and at another time write or read.	Sec 9 – 14/46
30. Stop condition is not allowed to immediately follow a start condition	Sec 9 – 14/46
31. General call address can call all devices, but some devices can be made to ignore this call.	Sec 10 – 15/46
32. Several identical devices may be connected to the bus at the same time, they can distinguish themselves from each other through fixed and programmable address bits.	Sec 10 – 15/46
33. Not acknowledge is used when data should not be handled for whatever reason.	Sec 10 – 16/46

34. There are multiple illegal address byte combinations; see Table D3 on page 76	Sec 10 – 16/46
35. In a general call, it is the second byte which determines what will happen, as the first byte is used to identify a general call (reserved sequence).	Sec 10 – 16/46
36. If 2nd byte is 0000 0110 (H06) then all listening devices will reset and write the programmable part for a general call.	Sec 10 – 16/46
37. If 2nd byte is 0000 0100 (H04) then all listening devices will not reset but write the programmable part for a general call.	Sec 10 – 16/46
38. 00000000 (H00) is not allowed to use as the 2nd byte for a general call.	Sec 10 – 16/46
39. Other than H00, H04 and H06 in 2nd byte must be ignored for general call.	Sec 10 – 18/46
40. High dummy acknowledge sent after START byte 0000_0001, no slave is allowed to acknowledge this byte.	Sec 10 – 18/46
41. In a mixed bus structure. I2C bus devices must not respond to the CBUS messages. This for this reason, a special CBUS address has been reserved.	Sec 10 – 18/46
42. High speed mode (Hs) can only be achieved from F/S mode through a S → Master code → !Ack. The reason for !Ack is because no device is allowed to acknowledge the master code.	Sec 13 – 21/46
43. The Hs mode has additional pull up resistors and current sources, in order to meet the extra harsh timing requirements. Thus their waveform shape is slightly different from the S and F mode.	Sec 13 – 22/46
44. F mode is downwards compatible with S mode.	Sec 13 – 23/46
45. Hs mode devices are fully downwards compatible with both F and S mode. If changed from Hs to F/S, the current-source will be ignore, and everything will operate on the F/S speed.	Sec 13 – 24/46
46. Both 7 bit and 10 bit devices can be connected to each other, and used in all modes.	Sec 14 – 27/46
47. In the case of 10 bit addressing, 2 acknowledgments will be sent for each byte. Only one device will however answer to both acknowledgments.	Sec 14 – 27/46
48. Minimum hold time for repeated start condition. See Table D2 on page 75.	Sec 15 – 32/46
49. There are minimum low and high period of the SCL clock. See Table D2.	Sec 15 – 32/46
50. There is a minimum setup time for start, stop conditions and data. See Table D2.	Sec 15 – 32/46
51. Minimum rise and fall time for both SDA and SCL. See Table D2.	Sec 15 – 32/46
52. Minimum bus free time between a start and a stop condition. See Table D2.	Sec 15 – 32/46

Table D4 : I2C Extracted properties from [2].

Do keep in mind that this extraction of the I2C characteristics is made by the author of this thesis and not an official statement by the original developer Phillips. Not all characteristics are considered to be extracted from the specifications, but those who are may be referred back to when further extraction of properties for the I2C bus protocol is performed.

Also, not all of these observations in the previous table is applicable as actual properties to assert. Those who are is mentioned and elaborated on further in section 5.5 -”Custom I2C abVIP Slave Mode Development“ on page 32.

TRITA TRITA-ICT-EX-2014: 127