Tampere University

Sami Marin

INTERCONNECT AUTOMATION TOOL IMPROVEMENT

Master of Science Thesis
Faculty of Information Technology
and Communication Sciences
Examiners: Karri Palovuori
Erja Sipilä
November 2021

ABSTRACT

Sami Marin: Interconnect automation tool improvement
Master of Science Thesis
Tampere University
Master's Degree Programme in Electrical Engineering
November 2021

Communication between IP components in System on Chip systems is crucial for providing complex functionalities. The components are connected to each other with interconnects which are responsible of handling the data exchange effortlessly. The interconnects are built to follow a communication architecture which consists of a physical structure and a communication protocol. As the interconnects are often simple and the number of interconnects necessary in a system is large, the generation of the interconnects can be automated to reduce the work effort.

In this thesis, an existing interconnect automation tool is introduced and investigated in order to improve it in various areas. The current implementation of the tool creates an interconnect RTL and files for IP-XACT packaging and verification. The generated interconnect is based on parameters, such as the communication protocol (for example AXI4 protocol) and the number of IPs (slaves/masters), which are inserted into a spreadsheet working as the interface of the tool. The generation itself is done by utilizing a macro in the spreadsheet. Multiple improvement ideas were collected, and the progress done for them during the thesis is explained. The improvement process followed a simple workflow of a feasibility study followed with an implementation of the solution deemed the best.

Only one improvement, support for non-continuous address space for a single master, was finished during the thesis due to an implementation change. However, the feasibility of multiple improvements, such as master-slave visibility and flexible range for address regions, was investigated. Some were unfeasible due to an underlying major restriction. For feasible improvements, possible solutions are provided for future implementation.

Keywords: interconnect, automation, communication protocol, AXI4, IP-XACT

TIIVISTELMÄ

Sami Marin: Väyläkomponenttien automaatiotyökalun parannus
Diplomityö
Tampereen yliopisto
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Marraskuu 2021

Tiedonvälitys IP komponenttien välillä on keskeisessä asemassa monimutkaisten toiminnallisuuksien tarjoamisessa System on Chip -järjestelmissä. Komponentit kytketään toisiinsa väyläkomponenteilla, joiden vastuu on hallita tiedonvaihtoa tehokkaasti. Väyläkomponentit suunnitellaan pohjautuen kommunikaatioarkkitehtuuriin, mikä koostuu fyysisestä rakenteesta sekä tiedonvälitysprotokollasta. Usein väyläkomponentit ovat yksinkertaisia ja niiden määrä on suuri järjestelmissä. Täten väyläkomponenttien generointi voidaan automatisoida työmäärän vähentämiseksi.

Tässä diplomityössä esitellään olemassa oleva työkalu väyläkomponenttien luomiseen. Työkalua myös tutkitaan tavoitteena sen parantaminen useasta näkökulmasta. Nykyinen työkalun toteutus luo väyläkomponentin RTL -tiedoston sekä muita tiedostoja IP-XACT -paketointia ja verifiointia varten. Väyläkomponentit generoidaan parametrien perusteella, mitkä voidaan asettaa työkalun laskentataulukkoon perustuvaan käyttöliittymään. Asetettavia parametreja ovat esimerkiksi tiedonvälitysprotokollana AXI4 -protokolla sekä yhdistettyjen komponenttien määrä ja niiden rooli. Generointi tapahtuu laskentataulukkoon tehdyn makron avulla. Useita parannusehdotuksia kerättiin diplomityön aikana ja edistys niitä kohtaan selitetään. Parannusprosessi oli yksinkertainen, missä aluksi parannuksen soveltuvuus selvitettiin. Tätä seurasi parhaimmaksi todetun ratkaisun implementointi.

Vain yksi parannus saatiin päätökseen diplomityön aikana implementointiratkaisun muutoksen vuoksi. Tämä parannus oli epäjatkuvien osoiteavaruuksien tukeminen, kun ne asetetaan samalle isäntäkomponentille. Tästä huolimatta useiden parannusehdotuksien soveltuvuus selvitettiin. Näitä parannuksia olivat muun muassa isäntä-orja liitosten näkyvyyden parannus sekä osoitealueiden leveyksien joustavuuden parannus. Osa parannusehdotuksista todettiin toteuttamiskelvottomiksi rajoitteen vuoksi, minkä korjaus vaatisi suuremman muutoksen väyläkomponentin toimintaperiaatteeseen. Toteuttamiskelpoisille parannuksille selvitettiin mahdollinen ratkaisu, mikä voidaan implementoida tulevaisuudessa.

Avainsanat: väyläkomponentti, automaatio, tiedonvälitysprotokolla, AXI4, IP-XACT

PREFACE

I would like to express my gratitude to my company for providing me the opportunity to write this thesis. I would also want to thank my colleagues and supervisors for aiding me in understanding topics which were not familiar for me. I would like to thank my thesis supervisors as well for providing me feedback on the thesis.

Tampere, 3 November 2021

Sami Marin

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
AXI	Advanced eXtensible Interface
CB	Clock Bridge
CBR	Clock Bridge Receiver
CBT	Clock Bridge Transmitter
CDC	Clock Domain Crossing
EDA	Electronic Design Automation
GUI	Graphical User Interface
IP	Intellectual Property
IP-XACT	XML format for defining electronic circuit designs
QoS	Quality of Service
RTL	Register Transfer Level
SoC	System on Chip
TGI	Tight Generator Interface
VBA	Visual Basic for Applications
VHDL	VHSIC Hardware Description Language
XML	Extensible Markup Language

1. INTRODUCTION

Modern SoC (System on Chip) devices are complex systems with dozens of separate components. These components are often IP (Intellectual Property) blocks with their own functionality and purpose. The combination of the individual functionalities allows complex applications on a single SoC device. [1] The key factor in enabling the complex features is communication between the separate components. Communication is enabled by using interconnects which are often components themselves. Interconnects based on some communication architectures are the commonly used method.

As the system complexity increases, so does the requirements for the interconnects. Flexibility for the number of IPs and their address spaces requires an effortlessly modifiable interconnect. Manually writing interconnects with VHDL (VHSIC Hardware Description Language) for every system is slow and cumbersome since the interconnects themselves are in practice often just wiring between the IPs while allowing communication with some communication protocol. Therefore, the interconnect generation should be automated to reduce the manual work effort which could be used elsewhere.

In this thesis, one implementation of an interconnect automation tool is introduced. The tool can generate interconnects based on certain communication architectures which determine the physical structure and the communication protocol of the interconnect [2]. The implementation is an all-in-one solution based on Excel spreadsheet. As an output, the tool generates the necessary files for the interconnect and various extra files to aid in integration and verification. Flexibility and ease-of-use is enabled by a simple interface, which can be used to determine the parameters of the interconnect. These parameters work as the input data for the generation.

The main goals of the thesis are to determine the current state of the automation tool and improve it in various areas. The need and possibilities for improvement are investigated during the thesis to determine the baseline of the current state. Then the feasibility of the possible improvements is defined. Depending on the feasibility, as many improvements as possible are implemented in the given timeframe of the thesis.

The outline of the thesis is the following. First, the basics of interconnects and communication protocols related to the automation tool are explained in the chapter 2.

Then in chapter 3, the current implementation of the automation tool is introduced in detail. Next the improvement process and the progress done during the thesis are explained in the chapter 4. Lastly the overall improvement of the automation tool is concluded in the chapter 5.

2. INTERCONNECT THEORY

To automate interconnect generation within the scope of this thesis, the required knowledge can be divided into three categories: interconnect principles, communication protocol principles and IP packaging. This chapter introduces these categories and starts with the interconnect principles. Then the communication protocol principles are explained. Next the standardized protocols used in the interconnect tool are introduced. Lastly the IP-XACT packaging methods are explained.

2.1 Interconnect principles

As the communication is the critical factor in performance [3], the used interconnect topology and architecture are important properties of the SoC design. The simple approach for the physical implementation of the interconnect is using buses which could be implemented with single wires. Multiple buses would then form a parallel bus implementation which has been the typical approach for SoC applications. [2] However with increasing complexity in the SoC designs, the simple parallel bus implementation reaches its data transfer limits and most likely bottlenecks the performance [3]. To reduce the effect of the bottlenecks due to the interconnect architecture, various approaches for the physical implementation have been developed. One of the promising architectures, which is also one of the two architectures relevant in this thesis, is a bus-matrix or commonly known as a crossbar. The physical implementation of the crossbar architecture is shown in the figure 1 below.

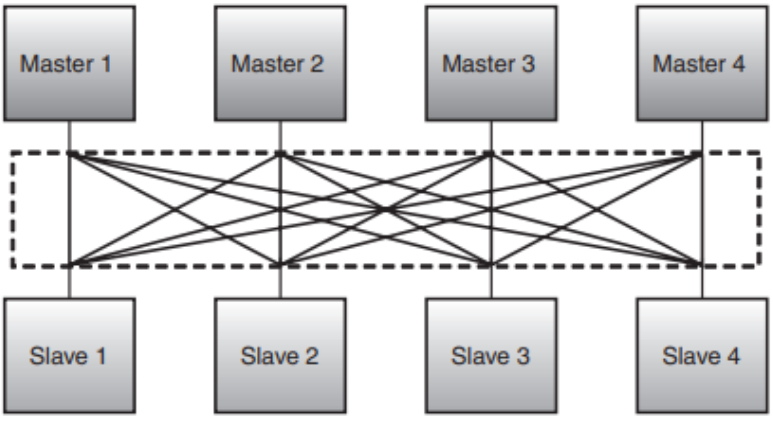


Figure 1. *Crossbar architecture [2]*

In the figure 1, there are blocks which represent components for example the IPs in the SoC design. Every block also has a role of a master or a slave. They are distinguished

from each other by their functionality in regards of the communication. Master blocks start and control the communication between the other blocks. Slave blocks can only respond to the communication attempts from the master blocks. [4] As seen in the figure 1, there are buses from each master to each slave. Communication between the blocks is packet switched as in the data transfer contents determine the destination. The destination information is decoded from the packet and then multiplexed to the correct end point. In a case where multiple data transfers occur to the same end point; arbitration is present to decide the order of receival. [2] The crossbar architecture thus allows high parallelism where multiple data transfers can occur simultaneously between different slaves and masters [2,4]. The benefit of high data throughput has increased the popularity of the architecture even though the implementation comes with high costs in terms of area, power consumption and manufacturing [2,3,4]. The other relevant and popular architecture is the ring bus architecture, which is introduced in the figure 2 below.

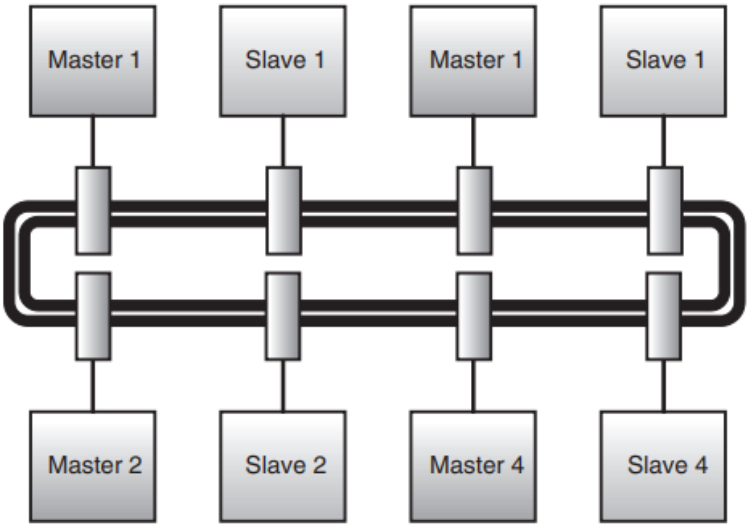


Figure 2. *Ring bus architecture [2]*

As shown in the figure 2, the ring bus architecture is quite different from the crossbar architecture. Instead of having buses for every connection, there is a single ring bus connection. The ring consists of nodes which operate as the end point for the components and are connected serially to each other. The communication in a ring bus is often token based [5]. The data transfer is passed to the ring and the nodes check the contents of the transfer. If the destination information points to the component connected to the node, the data transfer is appointed to component. Otherwise, the data transfer continues through the ring until the current destination is reached. [2] The major benefit of the ring interconnect is the great performance to area ratio. High performance is achieved due to the short length between the nodes in the ring which allows higher

operating frequency. [5] The benefit stems from the location of the nodes. If the data transfer is between adjacent nodes, the transfer is fast. The locations of the nodes are however the drawback of the ring architecture as well. In worst case the data transfer goes through the whole ring before it reaches its destination. This drawback can be reduced by supporting transfers to both directions depending on for example distance [2]. The nodes can also be allocated in proper locations where additional latency due to distance is less crucial.

Both architectures have their own uses. Crossbars are used in applications, for example high end processors, where data throughput and parallel operation are the most important factors. Ring bus interconnects are used in applications which are cost effective while allowing moderate performance.

2.2 Protocol principles

Now that the architecture for the physical implementation is clear, one important factor missing are the common rules for communication. Often the interconnected components are not new components. Some of the components are reused IPs or components provided by external component sources. To ease the communication between the components, communication protocol standards have emerged.

Communication protocol standards define two major features for the interconnected components. First defined feature is the interface for the components which is the physical pins on the device. A set number of pins are defined with their own functionalities and purposes. [2] The second feature defined are the data transaction rules. The rules define the properties for the data transmission such as the correct way to start and end the transmission, order of the possibly multiple data packets, transaction and receival acknowledgements and arbitration rules. [6] Following the features defined in the protocol standard increases the flexibility of the system and decreases the complexity of the interconnect designs as they are responsible of the data transmission between the components. Simpler interconnect designs are possible due to the common interface pins for the connections and the logic needed for the data transactions are not dependent on the receiving components as they share the data transaction rules.

2.3 AXI protocols

In the scope of this thesis the communication protocol standards relevant are bus based AXI (Advanced eXtensible Interface) protocols from AMBA (Advanced Microcontroller Bus Architecture) 4 specification by ARM [7]. Three protocols are introduced in the

specification and each of them are focused on high performance systems. The protocols are AXI4 protocol, AXI4-Lite protocol and AXI4-Stream protocol. These protocols have slightly different features and use-cases, but they share the underlying basics for communication. The combination of features makes these protocols a preferred choice for SoC interconnect designs.

The first shared feature is the principle for interface definition. The protocols define only master and slave interfaces. These interfaces are symmetrical between each slave and master respectively [8]. This allows simple connections and data transmissions, thus reducing the logic necessary in the interconnects. The second shared feature is allowing independent topology [8]. In other words, the physical structure of the interconnect does not have restrictions caused by the protocol. This allows the optimization of the interconnect by choosing a suitable topology. Another feature is the interface signal principles. Even though the interface signals are specific for each protocol, the principle for them are to support high parallelism. This is achieved by having own channels for address, read, write and response signals. The channels are independent from each other. Thus, the operation is largely asynchronous and parallel. [8] The channels also have their own purpose as in the address channels are only meant for the addresses and so forth. Independent channels open the option to optimize the signal widths for each of the channels since for example the addresses often require less bits than the data blocks.

The protocols share similar data transaction rules as well. Firstly, the data transaction is burst based. Each transaction consists of control information and the data itself. The control information contains the address of the destination and a description of the data which are relayed using the address channels. The data is relayed using either the write channel or the read channel depending on the direction such as from the master to the slave or vice versa. In both cases, the master starts the transfer by assigning the control information on the respective address channel for either read or write operation. Then the data is transferred in bursts until every burst is transferred. Indication for the last burst is relayed though a separate channel. Throughout the transaction, handshake channels are utilized. A response is given after the full transfer as well indicating a successful transaction or certain error occurring. [9] The parallelism of the channels enables the transactions for different IP blocks to occur out-of-order which is implemented in the protocols by assigning an ID for each occurring transaction. The only restriction is that a transaction with the same ID must be completed in order. [8] Out-of-order transactions are highly beneficial in interconnects as the IP blocks connected most likely have different response times.

The following sub-chapters introduce the protocol specific features and the interface signals relevant for the thesis.

2.3.1 AXI4 protocol

As mentioned earlier, the AXI protocols have separate channels for write and read transactions. The channels present in AXI4 protocol are the following: read address, read data, write address, write data and write response channels [10]. The data transaction procedures are visualized in the figure 3 below.

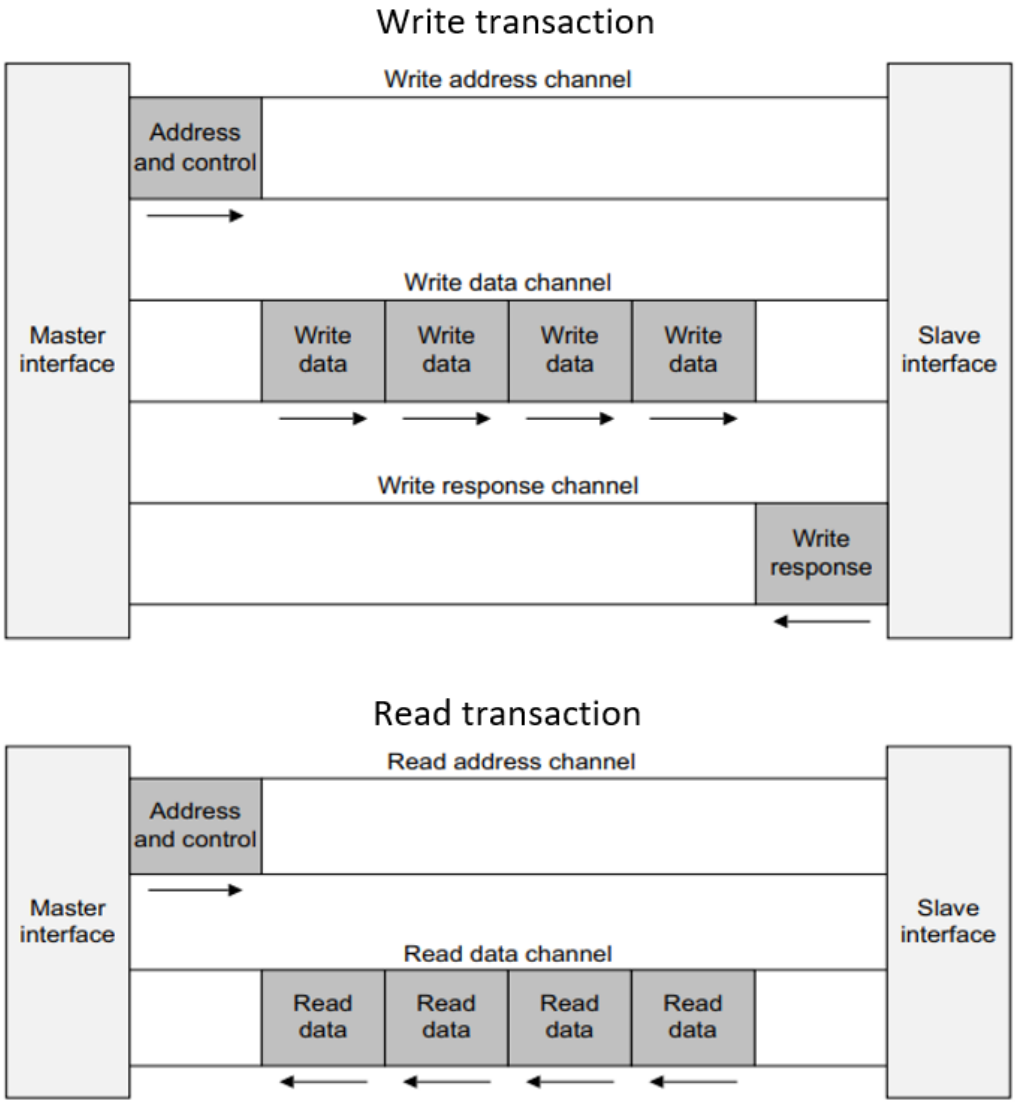


Figure 3. *Write and read transaction procedure for AXI4 protocol [10]*

As can be seen from the figure 3, the write and read transactions follow the basic principles as earlier mentioned. The write transaction starts with the master sending the control information and then the data is transferred. After every burst is transferred, a separate response channel is used by the slave to send the status of the transaction.

The read transaction follows the same procedure with the difference being the channels used and a separate response channel is not used. The separate channels consist of interface signals which are listed in the table 1 below for each channel.

Table 1. *AXI4 protocol interface signals [10]*

Global signals					
Name		Description			
ACLK		clock signal, synchronous signals sampled on rising edge			
ARESETn		reset signal, active-LOW			
Channel signals					
Write/Read address channel		**Write/Read data channel**		**Write response channel**	
Name (*x*=W/R)	Description	Name (*x*=W/R)	Description	Name	Description
A*x*ID	ID tag	*x*ID	ID tag	BID	ID tag
A*x*ADDR	address	*x*DATA	data	BRESP	response
A*x*LEN	burst length	*x*LAST	last burst indicator	BUSER	user extension
A*x*SIZE	burst size	*x*USER	user extension	BVALID	valid handshake
A*x*BURST	burst type	*x*VALID	valid handshake	BREADY	ready handshake
A*x*LOCK	atomic type	*x*READY	ready handshake		
A*x*CACHE	memory type	RRESP	read response		
A*x*PROT	access permission	WSTRB	valid write datalanes		
A*x*QOS	QoS identifier				
A*x*REGION	region indicator				
A*x*USER	user extension				
A*x*VALID	valid handshake				
A*x*READY	ready handshake				

The first signals in the table 1 are the global clock and reset signals. All the channel signals are sampled on the rising edge of the clock signal [11]. Then there are the channel specific signals. The write and read address channels have similar interface signals. There are the ID and the address signals. Then the transaction settings such as burst length, size and type signals. Additional settings include atomic operations, memory types, access permissions, QoS (Quality of Service) and region interface settings. There is also the user extension signal which the user can define for example to add extra information about the transaction. The last signals in the read and write address channels are the handshake signals. The data channels also share similar signal interfaces with some differences. There are the ID and the data signals. The signal used to indicate the last burst of the transaction. Similarly, there are the user and the handshake signals. The differences between the data channels are the last two signals which are specific for the read or the write channel. The read data channel has a

response signal, and the write data channel has an indicator signal for valid data lanes. The last channel in the table 1 is the separate write response channel. The channel is only used for write responses thus the channel consists of ID, response, user and the handshake signals.

As indicated in the table 1, every transaction channel has the handshake signals. These are utilized throughout the transactions and they are a crucial part of organizing the communication. The channels have slightly different rules for handshaking which must be followed. However, they follow the same process for handshaking. The process starts with the source setting the VALID signal to a HIGH state. The source can be the master or the slave as both can control the flow of information. The HIGH state for the VALID signal indicates that the channel appropriate information is available for transferring. Then the destination sets the READY signal to a HIGH state which indicates the readiness for receival of information. The transfer is completed on the rising edge of the clock signal when both handshake signals are in the HIGH state. [10,11] There are three different allowed handshake cases which are introduced in the figure 4 below.

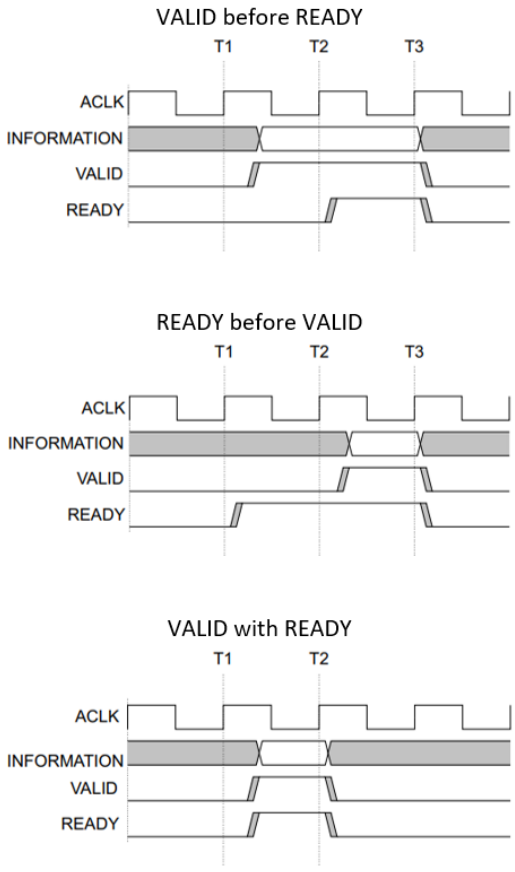


Figure 4. *Allowed handshakes for AXI4 protocol channels [10]*

The first handshake in the figure 4 is the basic process where the VALID signal is set to HIGH before the READY signal. The other cases are also possible due to the READY

signal being allowed to be HIGH before the VALID is set. The benefit of doing so is to reduce the time the transaction takes. In the second handshake the READY signal is HIGH before the VALID signal. Thus, the transfer is done in one clock cycle instead of the information being held in the channel until both signals are HIGH like in the first case. In the last case, both handshake signals are set to HIGH at the same time. Therefore, the transfer is completed at the next rising edge. [10,11]

In addition, there are channel specific rules for the handshakes and dependencies between the handshake signals for the channels as well. These are present to avoid deadlock and they must be followed. [10]

2.3.2 AXI4-Lite protocol

The next relevant AXI protocol is the AXI4-Lite protocol which is a simplified version of the AXI4 protocol in the previous sub-chapter. Some of the features regarding the burst based transfers are fixed instead of user-defined. The major settings being the burst length fixed to value 1 and the usage of full width of the data channel for all accesses. The simplified protocol is intended for simple control interfaces which do not need the full capabilities of the AXI4 protocol. [10] The protocol has the same channels and follows the same transaction processes as the AXI4 protocol. The simplified interface signals are collected in the table 2 below.

Table 2. *AXI4-Lite protocol interface signals [10]*

Global signals					
Name		Description			
ACLK		clock signal, synchronous signals sampled on rising edge			
ARESETn		reset signal, active-low			
Channel signals					
Write/Read address channel		**Write/Read data channel**		**Write response channel**	
Name (*x*=W/R)	Description	Name (*x*=W/R)	Description	Name	Description
A*x*ADDR	address	*x*DATA	data	BRESP	response
A*x*PROT	access permission	*x*USER	user extension	BUSER	user extension
A*x*QOS	QoS identifier	*x*VALID	valid handshake	BVALID	valid handshake
A*x*REGION	region indicator	*x*READY	ready handshake	BREADY	ready handshake
A*x*USER	user extension	RRESP	read response		
A*x*VALID	valid handshake	WSTRB	valid write datalanes		
A*x*READY	ready handshake				

The interface for the AXI4-Lite protocol is much simpler due to the fixed burst length as shown in the table 2. The signals for the transactions such as the address, data and handshake signals have the same functionality. The additional signals are also present. One major difference is the missing ID signals which means that the transactions must be in order. The handshake process with the AXI4-Lite protocol is the same, and the same rules and dependencies are present like in the AXI4 protocol.

2.3.3 AXI4-Stream protocol

The last protocol from the AMBA 4 specification is the AXI4-Stream protocol which is for streaming data between the masters and the slaves. The protocol has a single interface which is used for the streaming. The interface is similar with the AXI4 protocol write data channel [8]. Data is transferred in data streams which are a series of transfers or transfers grouped up as packets. Multiple data stream types are supported such as byte streams or continuous streams. Byte stream is a collection of data bytes and null bytes. The null bytes can be whenever in the byte stream and the data bytes are transferred only on valid handshake. The continuous stream contains only data bytes. The stream can be aligned or unaligned. In aligned stream, there are no extra bytes between the packets. Unaligned stream is the opposite. [12] The interface signals used for the data streaming are introduced in the table 3 below.

Table 3. *AXI4-Stream protocol interface signals [12]*

Protocol interface signals	
Name	Description
ACLK	clock signal, synchronous signals sampled on rising edge
ARESETn	reset signal, active-low
TVALID	valid handshake
TREADY	ready handshake
TDATA	data
TSTRB	data description
TKEEP	data qualifier
TLAST	transaction boundary
TID	data stream ID
TDEST	destination
TUSER	user extension

As shown in the table 3, there are the necessary signals for the handshakes and the data. Additionally, there are extra signals to add extra information about the data. The TSTRB signal describes the type of the data and the TKEEP signal informs about the null bytes. The TLAST signal is used to indicate the end of the packet. The multiple

masters and slaves are handled with the ID signal which separates the different data streams. The destinations for the packets are determined with the TDEST signal. The protocol has the optional user extension signal like in the other AXI protocols as well. Like in the AXI4-Lite protocol, the handshake process is identical with the previous AXI4 protocol. [12]

2.4 IP-XACT packaging

As mentioned earlier, modern SoC devices can contain IPs from different sources. The IPs and the interconnects could have different principles for designing which could hinder the design process of the complete system. To improve the reusability of IPs and compatibility with tools, IP-XACT standard can be used to describe IPs in a manner which is consistent and machine readable [13]. The separate designs in the system using the common documentation style enables less problematic integration which is a key factor in the design of modern SoC devices [16].

The level of detail in the IP-XACT descriptions can be chosen for example only top-level details such as ports are presented. For interconnects, the useful details to describe with IP-XACT could be the top-level ports, bus definitions and internal memory maps. To describe the necessary structures in the IP, the standard provides document types. These types have their own purpose as they are used to describe certain details of the IP. The document types important for interconnect designs are listed below.

- Component
- Design
- Design configuration
- Bus definition
- Abstraction definition

The first document type in the list is component which is the top-level type. It is used to describe for example IP blocks. In this case, the component type describes the interconnect and its top-level details. The necessary top-level information would be the master and slave ports, buses and address mapping information. The next document type is design which describes the hierarchy of the component. The design could contain the information on the used internal sub-components and their properties. The information about the hierarchy can be extended with the design configuration document type which can be used to detail exact information about the sub-component instances. [13,14,15] The first three document types are for describing the structure of the IP from

top-level to internal structure. The level of detail can then be chosen by using only the component and not the other two types as they are extensions for the component. The last two document types in the list are for describing the communication properties of the interconnect. The bus definition type can be used to describe the direct connections in the design. For interconnects these could be the connections between the masters and the slaves. The bus definition contains also the protocol used for the connection and ports. The bus definition can be extended by using the abstraction definition type which defines the exact information about the buses such as logical ports and their properties. Like with the structure, the level of detail for the communication properties can be chosen. [13,14] The combination of the different document types enables the IP-XACT description to have the preferred amount of detail.

One of the important features enabled by using IP-XACT for interconnects is the possibility to include address mapping information. In the IP-XACT description, the address mapping is enabled with using bridges in the component. The bridge works as a connection between the master and the slave bus interfaces. In IP-XACT, there are two different types of bridges and they define how the address mapping is linked between the master and the slave. [14] The following sub-chapters explain the two bridge types: transparent bridge and opaque bridge.

2.4.1 Transparent bridge properties

Transparent bridging utilizes direct mapping. This means that the address space of the bridges master interface is one-to-one mapped to the address space of the slave interface. An example with a connected component is shown in the figure 5 below.

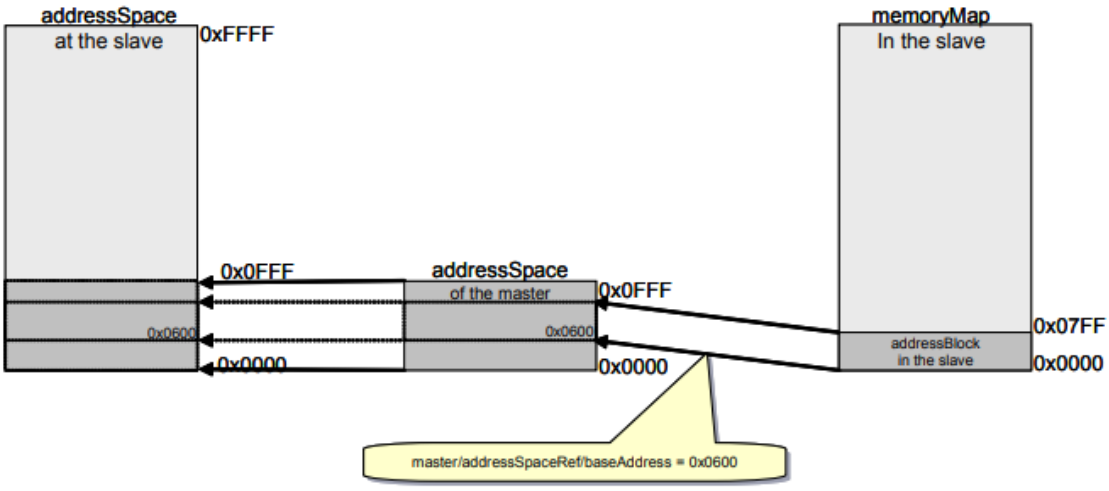


Figure 5. *An example of the address mapping with transparent bridges [14]*

As shown in the figure 5, the address spaces of the master and the slave interfaces start from the same position (0x0000). The master address space range is also mirrored to the slave interfaces address space. When a component with its own address block (0x0000-0x07FF) is connected to the master interface, the address block can be offset into the master address space. However, the offset does not change the direct mapping and only offsets the location in the master address space. The offsetting can be done in both directions. Multiple bridges to different masters can be made for the same slave in which multiple master address spaces are mapped to the same slave interface. The transparent bridging does however have a restriction regarding the master interface address spaces. The address spaces are declared with only a base address and a range. Therefore, the address space must be continuous. [14] In some systems, this restriction could cause overlap which might cause errors in the functionality. The transparent mapping is simple and efficient in most interconnect designs.

2.4.2 Opaque bridge properties

Opaque bridging is used when the address maps should not be directly mapped. Using the opaque bridges requires the usage of subspace maps in the slave interface. The subspace maps are a way to divide the slave interface address map into smaller sections. The subspace maps are then used to determine how the master address space is mapped to the slave interface. An example with a connected component is shown in the figure 6 below.

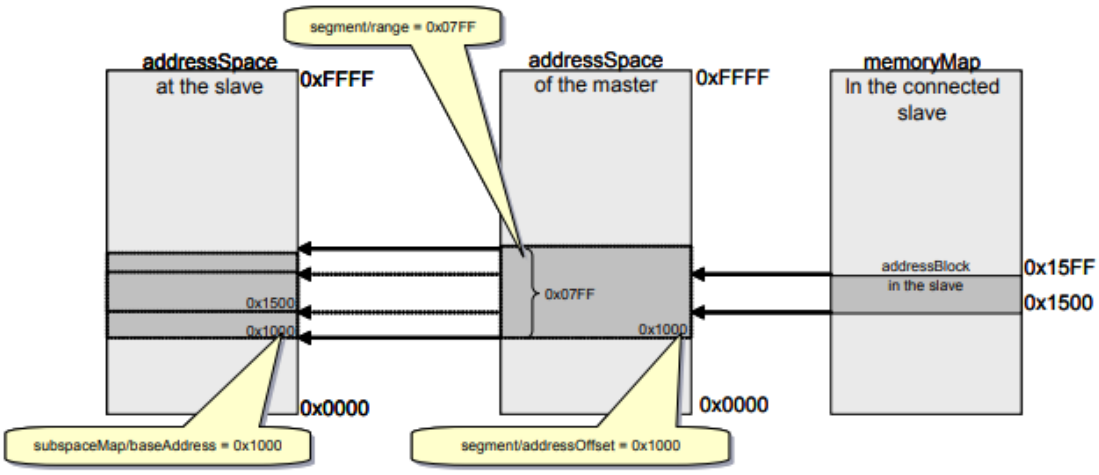


Figure 6. *An example of the address mapping with opaque bridges [14]*

In addition of the requirement for subspace maps, the master interface address space is often divided to address space segments as shown in the figure 6. The opaque bridging then maps the segment of the master address space to the subspace map of the slave address space. The location of the segment does not affect the location of the subspace map; thus, the segment offset can be freely changed. Only the range of the segment is mirrored to the subspace map. For example, if the segments offset in the figure 6 would be changed from 0x1000 to 0x3000, the segment would still be mapped to the subspace map starting from 0x1000. This enables the option to choose which segment is mapped to which subspace map. With multiple segments and subspace maps, the address mapping is flexible and can be for example shuffled if necessary. Connected address blocks are mapped like with transparent mapping in which the address block can be offset. [14] Care must be taken when connecting components to ensure that the address block is connected inside some segment in the master address space.

With the ever-increasing complexity of systems, the interconnects will become more complex as well. Utilizing the opaque bridging offers the tools to modify the interconnect address mapping rather freely in the later steps of the design process.

3. INTERCONNECT TOOL

This chapter introduces the interconnect automation tool used in the company. First the current features of the implementation are listed. The features are separated to generator and protocol specific features. Lastly the working principle and the flow of the tool is explained thoroughly.

3.1 Current features

The interconnect automation tool is an RTL (Register Transfer Level) generator which can be used to generate three different interconnect designs. These designs are generated based on communication protocols which can be chosen by the user. The tool includes an interface for setting various design parameters and information sharing. Interface inserted data is then used for the generation of the interconnect RTL and necessary files for later work steps such as packaging and verification. The following subchapters introduce the generator specific and the protocol specific features.

3.1.1 Generator specific features

The generator features a single interface for everything to allow easy and fast usage. The interface is used for setting the protocol for the generation and there are three protocols to choose from. The protocols supported are AXI4, AXI4-Lite and modified AXI4-Stream protocol. The design parameters for the interconnect can be set after the protocol is chosen. There are universal parameters such as the number of masters and slaves. Protocol specific parameters such as protocol signal widths can be set as well. The generator supports the addition of address mapping for the interconnect RTL and the packaging. The address mapping is inserted as address regions. Multiple address regions can be assigned for the same master. The slave and master interfaces can be named by the user. The generation is started from the interface and the generation is done locally without the need for separate setup for the tool. The tool includes a status window and the ability to point out the erroneous inputs from the user. The shared features for every protocol are shown in the table 4 below.

Table 4. *Shared generator specific features*

Feature	Options	Constraints	Limit
Protocol	AXI4, AXI4-Lite, modified AXI4-Stream	-	-
Slaves	-	-	max=32
Masters	-	-	max=32
Address regions	multiple regions can be assigned to single master	start address<=given address size, no overlap, continuous space for a single master, alignment, range, destination	max=32
Port names	-	no duplicates, one word, first character=letter	-

As can be seen from the table 4, there are constraints and limits for the shared features. Multiple constraints are placed for the address regions since the address mapping is affected by the interconnect RTL implementation heavily. The limits are present due to the implementation of the interconnect tool interface thus they are not RTL specific limitations.

3.1.2 AXI4/AXI4-Lite specific features

The protocol specific features for the AXI4 and AXI4-Lite protocols in the generator are the possibility to modify the signal widths of the data and address bus sizes as well as the protocol specific signal widths. The modifiable signal widths for the protocols are collected in the table 5 below.

Table 5. *Modifiable signal widths for AXI4 and AXI4-Lite protocols*

Protocol parameter	AXI4	AXI4-Lite	Constraint
Datasize	supported	supported	log2 format of bytes
Address bus size	supported	supported	max=39bits
ID size	supported	not supported	-
Len size	supported	not supported	max=8bits
User size	supported	supported	-

From the table 5, the datasize and the address bus size are the same in both protocols and they have the same constraints. The constraint for the datasize indicates that the data size must be given as binary logarithm of bytes. For example, value 2 would indicate

32 bits wide data bus. The maximum address bus size is 39 bits which depends on the current RTL implementation. The protocol specific signals available for modification for the AXI4 protocol are ID, Len and User signals. Only the user signal width is modifiable in the AXI4-Lite protocol since the ID signals are not supported and the Len signals are defined to be 1.

3.1.3 Ring interconnect specific features

When generating a ring interconnect, the datasize and the address bus size can be set same as with the other protocols. The modified AXI4-Stream protocol does not however have the same signals. Ring interconnect generation also has extra features for the individual nodes regarding for example CDC (Clock Domain Crossing). The modifiable signals and extra features are collected in the table 6 below.

Table 6. *Ring interconnect specific features*

Protocol parameter	Options	Restriction	Constraint
Datasize	-	value=2 only supported (32bit datasize)	log2 format
Address bus size	-	-	max=39bits
Source and destination signal size	-	-	-
Protocol for individual node	AXI4-Lite, modified AXI4-Stream	-	-
Clock bridge addition	full bridge, half bridge	-	-
Node sequence	-	no duplicates	must be a running number from 0

As seen from the table 6, the only supported datasize with modified AXI4-Stream currently is 32 bits. The modifiable signal widths are the source and the destination signal widths, which are not present in the other protocols. The extra features shown in the table 6 are the ability to set individual properties for the nodes. Firstly, the input signal interface can be chosen between two protocols for the node which are the AXI4-Lite and the modified AXI4-Stream. Then CDC components, as in full or half clock bridges, can be added into the design for the desired nodes. Lastly, the node sequence for the ring can be changed to be preferred while following the constraints and restrictions.

3.2 Working principles

The current implementation of the interconnect automation tool is based on Microsoft Excel -spreadsheet software. Excel is a suitable software for the interconnect tool since the tool should be easy and informative to use. Often multiple different interconnects are needed thus the modification of the settings or the parameters should be fast while also allowing the separation between the interconnects. Fast modification is possible since the interface of the tool is a predesigned spreadsheet in which the user can alter various settings for the generation and the parameters for the generated interconnect. Separation, for example different input files, is also possible since the spreadsheet acts as the input file for the generation. The generation itself is behind a macro which contains multiple different functions such as checking functions or VHDL generation functions. The macro uses Microsoft Visual Basic for Applications (VBA) -programming language. VBA is suitable as the programming language for the generation since it has the necessary functionalities [17,18]. With VBA, a separate tool or environment is not needed for the generation. By using the spreadsheet as the interface and the macro for the generation, the complexity of using the tool is reduced since the user does not need the knowledge on how to setup multiple tools.

There are two workflows which are executed in the normal operation of the interconnect automation tool. The first workflow is for the user and the second is for the generation. Both workflows have been designed to be straightforward and fast. The user operates only with the interface of the tool and there should be no reason to do otherwise. The predesigned user interface consists of 4 different areas, which are introduced in the following figures. The user workflow starts by setting the parameters in the interface area shown in figure 7 below.

	A	B	C	D	E	F
1			**Parameters**			
2						
3			Interconnect type (0/1/2)	1		
4			Number of inputs (slave IFs)	6		
5			Number of outputs (master IFs)	6		
6			Number of addr regions	6		
7			Log2NumOfBytes	2		
8			Address bus size	32	bits	
9			Source and Dest signal size	4	bits	
10			ID size (AXI)			
11			Len size (AXI)			
12			User size (AXI/Lite)		bits	
13			XML version	1.0		
14			VHDL entity name	example_entity		

Figure 7. *Modifiable parameters for the interconnect tool*

As seen from the figure 7, the user can alter various parameters such as the interconnect protocol used, the number of I/O ports, and the properties of the interconnect signal widths. The blackened cells in the figure 7 indicate parameters which cannot be modified with the current interconnect type. The same indicator is used in other parts of the interface as well. The next step in the workflow is setting up the memory address mapping for the interconnect which's interface area is introduced in the figure 8 below.

		Address mapping				
18	Addr Region #	Start address (HEX)	Length in Kbytes	Destination (Master interface)	End address	Overlap (row)
19	0	0	1024	0	FFFFF	
20	1	100000	1024	1	1FFFFF	
21	2	200000	1024	2	2FFFFF	
22	3	300000	1024	3	3FFFFF	
23	4	400000	1024	4	4FFFFF	
24	5	500000	1024	5	5FFFFF	
25						
26						

Figure 8. *Interconnect memory address mapping interface*

The memory address mapping consists of address regions which are indicated by a start address and a length as shown in figure 8. The routing for the address region is indicated with a number belonging to the destination which in this interconnect tool is the master. The amount of address regions visually available to modify is carried over from the value set in the parameters area. Regarding the possible start addresses and ranges, some rules and restrictions have been made for the address mapping which are checked in the second workflow later. The next step in the user workflow is defining the master and slave interface names. In the case of ring interconnect generation, the properties of the

nodes are also defined in this step. The interface for defining the individual properties for the slaves and masters is shown in the figure 9 below.

	SLAVE INTERFACE				
#	PROTOC 0=AXIL 1=NODE	CB 0= No CB 1= Incl CB	CBR 0= No CBR 1= Incl CBR	NODESEQ	PREFIX
0	0	0	0	6	slaveif0
1	0	1	0	7	slaveif1
2	0	0	1	8	slaveif2
3	1	0	0	9	slaveif3
4	1	1	0	10	slaveif4
5	1	0	1	11	slaveif5
6					
7					

	MASTER INTERFACE				
#	PROTOC 0=AXIL 1=NODE	CB 0= No CB 1= Incl CB	CBT 0= No CBT 1= Incl CBT	NODESEQ	PREFIX
0	0	0	0	0	masterif0
1	0	1	0	1	masterif1
2	0	0	1	2	masterif2
3	1	0	0	3	masterif3
4	1	1	0	4	masterif4
5	1	0	1	5	masterif5
6					
7					

Figure 9. *The individual properties of the slaves and masters*

As seen from the figure 9, the number of slaves and masters visually available is carried over from the user set parameters similarly like in the address mapping. For the ring interconnect type, there are various settings for the user to individually set for each of the slave and master nodes. As shown in the figure 9, the user can alter the protocol adaptation used for the node input side. The protocols supported are the AXI4-Lite protocol or the modified AXI4-Stream protocol. The addition of clock domain crossing components can also be set by the user. There can be full clock bridges indicated by CB (Clock Bridge) or half bridges indicated as CBR (Clock Bridge Receiver) and CBT (Clock Bridge Transmitter). Lastly for the ring interconnect, the user must set the order in which the nodes are connected in the generated component. The tool interface is designed to be less prone to user error, thus only the names can be modified if the interconnect type is for crossbar generation. All Excel cells affiliated with the ring interconnect generation are blackened to indicate that they are not used if it is not used as the interconnect type. The last step in the user workflow is to start the generation workflow by using the ROUHI! button of the interface. The last area of the user interface includes some quality-of-life features as well which are shown in the figure 10 below.

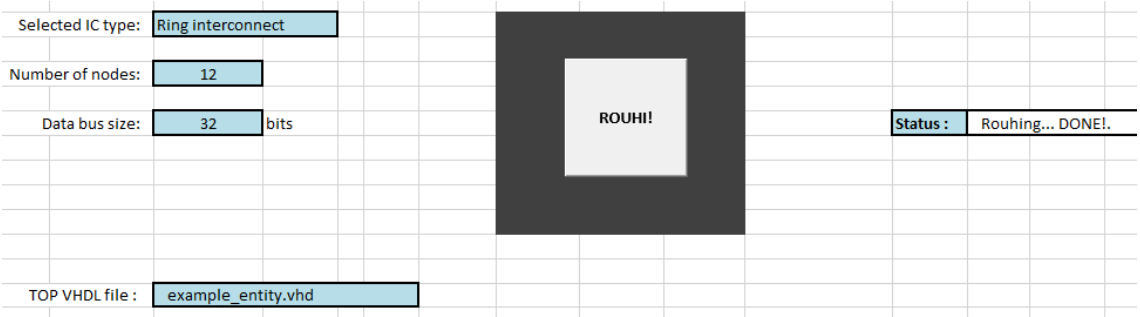


Figure 10. *Info and status interface*

The button seen in the figure 10 starts the Excel macro which is responsible for the generation flow. The generation flow includes checks for input errors which are then

indicated for the user using various visual methods such as coloring the cells and message windows. The user must correct the possible errors and run the generation flow again until a successful generation is indicated by the tool. As seen in the figure 10, the user interface also includes some info bars for the parameters and a status bar about the current progress of the generation workflow.

In a similar way to the user workflow division into different input areas, the generation workflow can be divided into different tasks. These tasks consist of a series of functions inside the Excel macro. Like a usual simple software program, the macro also includes a main function which works as the task manager for the generation flow. The main function called Generate is introduced in the program 1 below.

```
    Sub Generate()
2           ErrFound = 0
            IncMemMap = 1
4
            Do While True
6                   SetVariables
                            If ErrFound <> 0 Then Exit Do
8                   CheckAddresses
                            If ErrFound <> 0 Then Exit Do
10                  CheckOverlap
                            If ErrFound <> 0 Then Exit Do
12                  GenVectors
                            If ErrFound <> 0 Then Exit Do
14                  CheckPrefix
                            If ErrFound <> 0 Then Exit Do
16                  If NodeMode <> 0 Then
                            CheckNodeSequence_and_protoc
18                  End If
                    Exit Do
20          Loop

22          If ErrFound = 0 Then
                    MakeDirectory
24                  PrintTBSV
                    Print_packager_csh
26
                    If ICTypeValue = 0 Then
28                          PrintVHDL_wrapper
                            Print_axl_packager_tcl
30                          Print_sanity_tb_wrapper
                            Print_dut_sv_wrapper
32                  ElseIf ICTypeValue = 1 Then
                            PrintVHDL_wrapper_node
34                          Print_node_packager_tcl
                            Print_sanity_tb_wrapper_node
36                          Print_dut_sv_wrapper
                    ElseIf ICTypeValue = 2 Then
38                          PrintVHDL_wrapper_AXI
                            Print_axi_packager_tcl
40                          Print_sanity_tb_wrapper_AXI
                            Print_dut_sv_wrapper_AXI
42                  End If

44                  Cells(StatusRow, StatusCol).Value = "Generating... DONE!."
                    MsgBox "Generation completed w/o errors!"
46          Else
                    Cells(StatusRow, StatusCol).Value = "Generating... ERRORS FOUND!."
48                  MsgBox "Generation completed w/ errors!"
            End If
50  End Sub
```

Program 1. *The main function Generate of the generation workflow*

There are three tasks in the generation workflow which can be seen in the code structure of the main function in the program 1. The first task is responsible for checking the values inputted in the Excel spreadsheet and generating the necessary variables and vectors used later in the other tasks. If there are any errors made by the user, the error information is relayed by notifications and the generation workflow is terminated. The functions included in the first task are listed in the table 7 below.

Table 7. *Functions and properties of the first task in the generation flow*

Order of execution	Function name	Main feature	Error check
1	SetVariables	generate variables from user inputs	-
2	CheckAddresses	check if the addresses are inside the inputted address size	start address
3	CheckOverlap	check if the address regions overlap	overlap
4	GenVectors	generate mapping vectors	address region parameters
5	CheckPrefix	check if the master and slave names are according to rules	master and slave name
6	CheckNodeSequen-ce_and_protoc	check if the node sequence and protocol settings follow the rules	sequence and protocol

As seen from the table 7, the order of execution follows the user workflow order of input data since the address mapping and the interface inputs are dependent in the same order. Five of the functions are executed always and the sixth function is protocol specific which is when ring interconnect is generated. With every protocol the functions 2-4 in the table 7 are responsible for checking the address mapping of the interconnect. The checking logic structure is bottom up which is starting from the start addresses and working towards the destination inputs. The checks include bit lengths, any kind of region overlap, alignment to region range and destination checks. When the address mapping checks are passed, the GenVectors function generates mapping vectors for a mapper component used in the RTL design. The mapping vectors include the mapping data in binary format which is then used to determine the destination for input addresses. The last part of the first task is the checks for individual master and slave parameters. The function 5 in the table 7 checks the names inputted and if ring interconnect is generated the function 6 checks the node sequence and protocols for the nodes. After the first task, the input data is valid, and everything is ready for the generation of various files.

The second task in the program 1 is focused on generating the files which are not dependent on the protocol used for the interconnect. These files are necessary setup

files for the IP-XACT packaging and the sanity testbench. The functions in the second task are listed in the table 8 below.

Table 8. *Functions and properties of the second task in the generation flow*

Order of execution	Function name	Main feature	Other feature
7	MakeDirectory	generate necessary directory structure	-
8	PrintTBSV	generate parameter and definition files for sanity testbench	find a free address segment for error response testing
9	Print_packager_csh	generate setup packaging script	-

The necessary files for the sanity testbench in the second task are parameter and definition files which are generated in the function 8 seen in the table 8. These files contain the same input values for the parameter section and the memory mapping section from the spreadsheet. They are needed in the testbench environment thus they are exported from the spreadsheet as SystemVerilog -files. An error response area is also generated within the memory mapping for the error response testing. The other part of the second task is the generation of the setup script file for the packaging flow. The script file is an existing setup script used by the company and its main purpose is to setup the packaging environment and running the packaging flow. Automating the packaging with scripts allows better clarity and less complicated approach for the IP-XACT description generation [19]. The function 9 in the table 8 is responsible for modifying the necessary parts of the script for the entity generated later in the third task.

The last task shown in the program 1 is the generation of protocol specific files. The functions are responsible for the generation of the interconnect itself and the packaging script for it. The required testbench files are generated as well after the interconnect generation. Each protocol has its own functions for RTL, packaging and testbench files which are listed in the table 9 below.

Table 9. *Functions and properties of the third task in the generation flow*

Order of execution	Function name	Main feature
10	PrintVHDL_wrapper PrintVHDL_wrapper_node PrintVHDL_wrapper_AXI	generate top-level VHDL wrapper
11	Print_axl_packager_tcl Print_node_packager_tcl Print_axi_packager_tcl	generate IP-XACT packaging script
12	Print_sanity_tb_wrapper Print_sanity_tb_wrapper_node Print_sanity_tb_wrapper_AXI	generate sanity testbench wrapper
13	Print_dut_sv_wrapper Print_dut_sv_wrapper_AXI	generate setup files for sanity testbench

Appropriate function for the specific protocol is executed from the table 9 which can be also seen in the program 1. The first function in the third task generates the top-level VHDL wrapper which is the interconnect design. Even though there are three different protocols and therefore the wrapper designs are different, for example in the components used or interface, the basic principle and structure for the generation is similar. The pseudo code for the AXI4 and AXI4-Lite protocol VHDL wrappers are shown in the program 2 below.

```
entity xbar_entity is
        generic (
                ------------------------------------------------------------------------
                -- Various generics
                ------------------------------------------------------------------------
                );
        port (
                ------------------------------------------------------------------------
                -- Clock and reset
                ------------------------------------------------------------------------
                        Asynchronous clock input
                        Asynchronous reset input
                ------------------------------------------------------------------------
                -- Status output
                ------------------------------------------------------------------------
                        Pipeline status output
                        Mapping error output
                ------------------------------------------------------------------------
                -- Priority, arbitration quantum
                ------------------------------------------------------------------------
                        Priority input port
                ------------------------------------------------------------------------
                -- AXI4/AXI4-Lite SLAVE/MASTER interfaces
                ------------------------------------------------------------------------
                        Slave/Master interface #X to #Y, PREFIX = spreadsheet input
                        Slave/Master interface ports for different channels
                        for every slave/master interface );
end xbar_entity;

architecture rtl of xbar_entity is
        ------------------------------------------------------------------------
        -- Constants
        ------------------------------------------------------------------------
                Constants from the inputted spreadsheet parameters
        ------------------------------------------------------------------------
        -- Components
        ------------------------------------------------------------------------
        component crossbar_component
                generic (
                        Input parameter generics
                        Generics from top entity );
                port (
                        Protocol specific ports );
        end component;
        ------------------------------------------------------------------------
        -- Signals
        ------------------------------------------------------------------------
                Internal signals for address mapping
                Internal input/output signals for the crossbar instance
begin
        ------------------------------------------------------------------------
        -- Crossbar instance
        ------------------------------------------------------------------------
        I_Crossbar : crossbar_component
                generic map (
                        Generic mapping )
                port map (
                        Port mapping to internal signals );
        ------------------------------------------------------------------------
        -- Mapping vectors
        ------------------------------------------------------------------------
        ------------------------------------------------------------------------
        -- Slave/Master IF mapping
        ------------------------------------------------------------------------
                Slave/Master interface #X to #Y, PREFIX = spreadsheet input
                Slave/Master interface ports are mapped to respective
                internal slave/master signals for every slave/master interface
end rtl;
```

Program 2. *Pseudo code of the generated AXI4/AXI4-Lite wrapper*

The VHDL wrapper in the program 2 is generated using VHDL template structures which occur regardless of the set parameters. These templates are modified according to the parameters and the calculated variables during the earlier stages in the generation. The modification is done by inserting Visual Basic variables into the template which is then written to the output file. As seen from the program 2, the wrapper structure is as follows. First there is the entity declaration in which there are various generics and ports. The ports of the entity are dependent on the protocol used and the number of masters and slaves. These ports are generated using the respective protocol templates for the master and slave interfaces. User set number of ports are generated, and the port declarations are separated from other masters/slaves using the user set port names as a prefix in the port declaration. Then in the wrapper structure there are the constants which are a combination of user set and calculated variables. The constants are used later in the component instantiation which's component declaration is under the constants. Then there are the internal signals which are used to contain the address mapping vectors and the data from the masters and the slaves. Next in the wrapper structure, there is the component instantiation. With AXI4 and AXI4-Lite protocols only a single crossbar instance is used. The single crossbar is responsible for the operation of all the masters and slaves. Lastly in the structure the internal signals are mapped to the crossbar instance and the wrapper input/output ports. The mapping is done by slicing the individual master and slave wrapper ports into the internal signals using the prefixes set by the user.

When generating a ring interconnect there are differences in the function used for the generation compared to the AXI4 or AXI4-Lite functions. The differences are due to the nature of the interconnect. The master and slave nodes are in a ring as the name suggests. There is also the feature for the user to modify these individual nodes. These alter both the structure of the VHDL wrapper and the generation logic of the function. The pseudo code of the wrapper for the ring interconnect is shown in the program 3 below.

```
entity example_entity is
        generic (
                Storage size generics
                Clock bridge fifo size if used
                Various generics for the slave and master node instances );
        port (
                ------------------------------------------------------------------------
                -- Clock and reset
                ------------------------------------------------------------------------
                ------------------------------------------------------------------------
                -- SLAVE/MASTER INTERFACES
                ------------------------------------------------------------------------
                        Slave and master interface ports for the set protocol
                        Generated interface depends on the user set parameters
                        For example if the protocol for the node is set to
                        node or AXI4-Lite. The inclusion of half or full clock
                        bridges also adds port.
                ------------------------------------------------------------------------
                -- Status outputs
                ------------------------------------------------------------------------
        );
end example_entity;

architecture rtl of example_entity is
        ------------------------------------------------------------------------
        -- Constants
        ------------------------------------------------------------------------
        ------------------------------------------------------------------------
        -- Components
        ------------------------------------------------------------------------
                Components declared is a separate package
                These include the node component, clock bridges,
                protocol converters and queue components.
        ------------------------------------------------------------------------
        -- Signals
        ------------------------------------------------------------------------
                Mapping signals
                Node specific signals for every node instance
                Slave/master signals for AXI4Lite/modified AXI4-Stream protocol
                Status signals
begin
        ------------------------------------------------------------------------
        -- Delta balancing.
        -- Domain clock mapping.
        ------------------------------------------------------------------------
        ------------------------------------------------------------------------
        -- Mapping vectors
        -- Values imported from Excel.
        ------------------------------------------------------------------------
        ------------------------------------------------------------------------
        -- Node instances
        ------------------------------------------------------------------------
                Node ID vector used for user set sequence
                Separate node instance for every slave and master
        ------------------------------------------------------------------------
        -- Slave/master interfaces
        ------------------------------------------------------------------------
                Clock bridge and converter instantiations if necessary
                Slave/master interface mapping to signals and nodes
        ------------------------------------------------------------------------
        -- Status output retiming
        ------------------------------------------------------------------------
end rtl;
```

Program 3. *Pseudo code of the generated ring interconnect wrapper*

As seen from the program 3, the structure of the ring interconnect wrapper is like the other protocols. The user set parameters for the nodes are the varying parameters which must be considered in the generation. These parameters are the chosen protocol for the node, inclusion of clock bridges and the node sequence. First difference seen in the program 3 is the wrapper port declarations. These are dependent on the individual settings for the node. Different templates are used according to the settings for the generation. In AXI4 and AXI4-Lite protocols the templates were always the same for each of the master and slave ports. The second major difference is the usage of VHDL package for the component declarations. This is due to the number of necessary components. The package allows better clarity for the wrapper and the generation logic. As in the logic for deciding the necessary component declarations is not needed. Due to the nature of the ring interconnect, the signals and the instantiations are the third major difference. The signal declarations are separate for every node and not like in AXI4 and AXI4-Lite protocols where a single signal was used for every master and slave. This is seen also in the instantiations where every master and slave have its own node instance instead of a single instance for them all. The user set parameters also add signals and instantiations for the added clock bridges and the necessary protocol converters between the nodes. Lastly the signal and the component mappings are different since the nodes must be connected in the user set sequence.

After the VHDL wrapper is generated the next function executed in the third task is the second script for the packaging. This script is generated in the appropriate function for the protocol from the table 9. The second script for the packaging is launched from the setup script and is responsible for running the actual packaging commands in the packaging environment. The script is generated by modifying a template script used in the company and adding the user inserted memory address mapping data in the script. The resulting XML (Extensible Markup Language) file from the execution of the packaging script follows the IP-XACT standard. Thus, the memory mapping commands are generated in the script by using the IP-XACT transparent mapping which is simple for the current implementation of the interconnect tool. The commands used in the packaging environment depend on the packaging tool used. The current packaging approach uses commands from the TGI (Tight Generator Interface) API (Application Programming Interface) [20]. The commands used in the script are abstracted even further into functions containing the necessary TGI commands to generate building blocks for the address mapping. An example of the mapping commands used in the packaging script is shown in the program 4 below.

```
    ##############################################################################
2   # Bus definition commands for interfaces
    ##############################################################################
4
    #### SLAVE INTERFACES ####
6
    # Busdef(s) for slave interface prefix
8   <cmd_lib>::automapBusInterface -name <prefix>_<protocol>_Slave -mode slave

10  #### MASTER INTERFACES ####

12  # Busdef(s) for master interface prefix
    <cmd_lib>::automapBusInterface -name <prefix>_<protocol>_Master -mode master
14
    # Add BusIf mapping for Clock
16  <cmd_lib>::automapBusInterface -name Clock -mode slave

18  # Add BusIf mapping for Reset
    <cmd_lib>::automapBusInterface -name RESETn -mode slave
20
    ##############################################################################
22  # Memory map commands
    ##############################################################################
24
    # Base address and range for master interface prefix
26  <cmd_lib>::setBaseAddr <prefix>_<protocol>_Master <base address> <range>

28  # Bridges for slave interface prefix
    <cmd_lib>::addBridge <prefix>_<protocol>_Slave <prefix>_<protocol>_Master
```

Program 4. *An example of the address mapping commands used*

There are three building blocks in the transparent mapping which can be seen in the program 4. The first building block is the bus definitions for the masters and slaves. The second is setting the base address and range for the master bus interfaces which are taken and calculated from the spreadsheet. If there are multiple address regions assigned to a single master, the lowest address is the base address, and the range is calculated during the generation as the accumulation of the assigned address region lengths. Lastly the third building block is making the connection between the masters and slaves with a transparent bridge. In the current interconnect tool implementation, all the slaves are connected to each master.

The last functions executed in the generation workflow generate a testbench wrapper of the interconnect wrapper and a setup file for the testbench environment. The sanity testbench wrapper generated in the function 12 of the table 9 instantiates the earlier generated interconnect wrapper and adds testbench related signals. The generation structure and the principle follow the same structure as in the previous functions for every protocol. A SystemVerilog setup file is generated in the function 13 and its purpose is to be the connection between the VHDL and SystemVerilog. The function generates a waveform script for the EDA (Electronic Design Automation) tool used as well. A

testbench environment and tests are not generated in the interconnect tool. A sanity test, which tests that the protocol is working correctly when exchanging data between the masters and slaves, is simple and same regardless of different user parameters. Thus, a universal test and testbench made for the interconnects are used instead.

After every function in every task is generated successfully the user is informed about the completion and the files can be found in the folder containing the spreadsheet. As a summary of the files generated, the interconnect tool generates the interconnect VHDL wrapper, the packaging scripts and lastly the sanity testbench files.

4. IMPROVEMENTS

This chapter introduces the limitations surfaced during the investigation and then explains the progress done to improve the interconnect tool during the thesis work. The limitations are listed while giving insight to their cause and the benefits of improvement. Then the improvements done during the thesis are described thoroughly. The methods used and the steps of the implementation process are explained, and insight is given to the questions and decisions made during the implementation.

4.1 Limitations

The limitations discussed in this chapter are the base for the improvements of the interconnect tool implementation. They are limitations in a way which limit the use cases for the tool. For example, extra work must be done to get the interconnect which is needed since the tool can only provide interconnects with certain settings or which follow certain rules. The extra work could be for example, modifying the generated interconnect to be suitable or generate additional interconnects as a workaround. Both examples are not practical in the long run; thus, the interconnect tool improvements are essential. The limitations were scouted by requesting improvement ideas from other employees which were familiar with using the tool in projects. This kind of approach for the improvements was used to get more generic improvement ideas. Thus, the improvements would be beneficial in larger number of projects. Multiple improvement requests were received, and the most prominent requests are collected in the table 10 below.

Table 10. *Interconnect tool improvement requests*

Improvement	Limitation	Scale (major, minor)
multiple non-continuous address spaces for master interfaces	only continuous address space supported	major
add support for master-slave visibility	all slaves connected to all masters by default	major
add flexibility for address region range definition	range must be 2^N	minor
improve generic features	variables are set during generation	minor
add self-documenting features	self-documentation not implemented	major
add default master interface	masters must be set by the user	major

As can be seen from the table 10, most of the improvement requests are based on missing features in the interconnect tool. The design choices made for the interconnect tool do not allow generating preferred designs or doing useful modifications. The improvement requests were also given a scale at the start of the improvement planning phase which was determined by the amount of expected work hours for possible implementation phase.

The first request in the table 10 is to allow non-continuous address mapping for masters. The address space for the master could then contain empty spaces or possibly address regions of other masters inside the address space. This is currently not supported due to a constraint for the address mapping which defines that the address mapping must be continuous in the case when multiple address regions are assigned to the same master. The constraint is due to design choices for the generation related to the packaging. The non-continuous address mapping would be useful for example in designs where there would be a master for debugging in the middle of some other master's address space. With the current implementation non-continuous address space could be implemented by using a workaround interconnect design which combines masters from another interconnect. This way there would be a single master which can access multiple address spaces which are not continuous. Workaround would be always present which is not preferrable since it always adds extra work for the designers and complexity in the systems. The workaround would also increase the logic needed and thus increase the area of the interconnect and decrease the performance.

The second request is improving the visibility of masters and slaves. Visibility in this case means the accesses from slaves to masters. The request is thus to allow the user to decide which slaves have access to which masters. With the current implementation every slave has access to every master. This limitation is due to the interconnect RTL design and the packaging. The RTL for the AXI4 and AXI4-Lite protocols include only a single crossbar component which connects every slave to every master. The packaging script also adds bridges similarly. User defined accesses could then reduce the routing logic since often the slaves need to have access only to certain masters. The reduction in routing logic could possibly increase the maximum operating frequency of the interconnect as well.

The address region range definition improvement is to remove the limitation which is that the range must be inputted in the power of 2. Often the address region widths do not follow this rule, thus the interconnect tool supports assigning multiple address regions to a single master. This way the address region range for the master can be variable if the range can be divided to sections which are in the power of 2. This limitation forces the

designs to only have suitable address spaces and causes extra manual work for the tool user. Removing the limitation all together or even easing it would improve the usability of the tool and the interconnect development.

The rest of the requests in the table 10 are more akin to quality-of-life improvements. They are not crucial but would be helpful in the future or when making more generic interconnects. The generic features are currently limited to adding queue and pipeline sizes. The parameters set in the interface of the tool cannot be altered as generics after the generation. The parameters are either constants or fixed values in the generated RTL code. The generic features improvement would be to change the RTL to add the parameters such as datasize, address size or the extra signals for example ID size as modifiable generics. This would be useful in situations where there are multiple similar interconnects which differ mostly in the signal widths. With the improvement the signal widths could be altered with generic values rather than using the tool multiple times for every similar interconnect. The tool could be used only once, thus reducing the workload and waiting time of the generation.

Another helpful feature would be to add self-documentation for the interconnect tool. The proposed feature in the request is to add a visual representation of the generated interconnect. Currently the user must use another tool to visually see what was generated with the tool. For example, run the packaging scripts to get the XML file of the generated interconnect and then import the XML to an EDA tool. The EDA tool could then present a block diagram of the interconnect. A visual representation of the interconnect would be useful especially for the ring interconnect where the order of the nodes might need changing after the generation. If the tool could represent a block diagram of the interconnect straight after the generation workflow, the extra steps would not be needed and the dependency in other tools would reduce.

The last improvement request in the table 10 is to alter the interconnects to have a default master interface. Default interface in this case would be a master interface which is automatically generated. With the current implementation, every master must be inputted by the user in the address mapping section of the tool interface. The default master interface improvement would be useful in designs which have one or more masters with smaller sections of the full address space and the rest of the address space would always be assigned for a single master. For example, the full address space available would be 4 gigabytes and there are three masters. Two of the masters are assigned 1-megabyte sections and the rest of the address space is for the third master. The default master interface would be in this example the third master. The usability of the tool would improve since the user could set the total address space width and the smaller sections

for the masters. The tool would then generate another master interface automatically with the rest of the address space.

The improvement requests discussed in this chapter were tackled in no specific order during this thesis work. The workflow for the improvement followed a simple template. First the feasibility of the improvement was studied. Then the possible solutions were investigated. The decision to implement the solution was based on multiple constraints such as skill level, time and severity of the changes. The following chapters discuss thoroughly the progress done for the improvement requests during the thesis work.

4.2 Non-continuous address space improvement

As stated above, the improvement process started with a feasibility study. The current implementation and the work principle were studied for the continuous address mapping while figuring out the root cause for the limitation. During the study, the root cause was found which was a design choice made for the packaging. The packaging script used the IP-XACT transparent bridges. This limited the address space to be continuous since the transparent bridges do not support non-continuous address space defines for a single master. The RTL of the interconnect was studied, and it already supported non-continuous address mapping, thus the IP-XACT restriction was the only root cause. The choice to use the transparent bridges was made during the making of the interconnect tool, and it was then deemed to be a simple and efficient method for most of the interconnect requirements. There were in-house commands based on the TGI API already available as well, which used the transparent bridges. The result for the feasibility study was that the improvement is feasible using available tools and resources, and the new feature would be worth the effort. The next step in the improvement process was finding out the possible solutions for the limitation.

While investigating the improvement possibilities, there were some restrictions present. The solution should not need the modification of the RTL components used in the wrapper design. Modification in the components would then most likely affect many other designs which use the same components. Thus, the component changes would need to support the other designs as well. The restriction was set to avoid the possible problems arising from the changes. Keeping the restriction in mind, the solutions were investigated based on whether the packaging would use the same transparent bridges or some other method. One solution for each case was found for the AXI4 and AXI4-Lite protocols. Both would have their own pros and cons. During the investigation, it was decided that the improvement will not be made for the ring interconnect as the request was directed at the AXI4 and AXI4-Lite protocols, and there were no foreseeable use cases in which the

improvement would be useful with ring interconnects. The solutions found for AXI4 and AXI4-Lite protocols are presented in the figure 11 below.

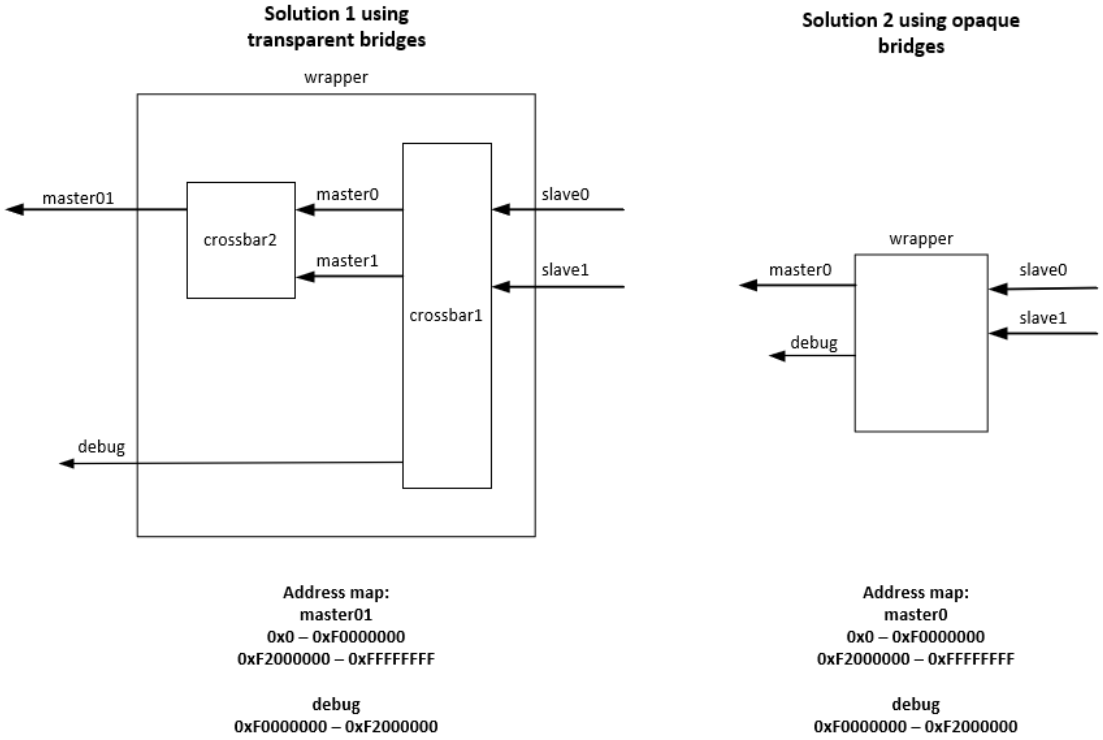


Figure 11. *Possible solutions for non-continuous address space mapping*

The solutions in the figure 11 are shown as block diagrams of an example interconnect which contains two master interfaces, master and debug, and two slaves. The address mapping for the master interface is non-continuous where it contains an empty space inside its address space reserved for the debug interface.

The solution 1 is based on the case where transparent bridges are still used. As the transparent bridges do not directly support the non-continuous addressing, the solution is to change the RTL of the interconnect to be suitable. The RTL would then allow transparent mapping while still having the non-continuous address space inside the interconnect. As shown in solution 1 in the figure 11, the RTL change is like the workaround previously mentioned. The basic principle is to add a second layer of crossbar component instances, which combine the separate address spaces into a single master interface. In the example the second layer crossbar, named crossbar2, would combine the address spaces of the master0 and the master1 interfaces from the crossbar1. The address space for the master0 is 0x0-0xF0000000, and the address space for the master1 is 0xF2000000-0xFFFFFFFF. The address mapping of the second layer crossbars would be set continuous and thus the packaging is possible with the transparent bridges. A benefit of keeping the transparent mapping would be reliability as

the method is well known and much used in the company. There would be less problems for example due to compatibility with tools and the arising problems would be faster to debug as the transparent mapping is a simple method. The drawback of transparent mapping would be the extra layer of RTL code solely due to the method used for packaging. As mentioned earlier, the RTL of the interconnect already supports non-continuous mapping as well, thus the extra layer might not be needed with other methods. The extra logic needed for the second layer would possibly induce latency and performance issues which are not preferrable. The scalability of the tool could also suffer when there would be multiple non-continuous master interfaces as they would need their own crossbar instances for the combining. The issues with latency or performance arising from the second layer could then grow large enough where the amount of non-continuous address spaces must be restricted.

The solution 2 shown in the figure 11 was found by seeking alternate methods to define the address mapping for the packaging script. The basic idea was to find a way to keep the RTL unchanged and only the contents of the packaging script would be changed. With the help of colleagues, the IP-XACT opaque bridges were proposed for the address mapping. With opaque address mapping the address regions assigned for a master could be defined as segments, thus the address space of a master would be a collection of segments instead of a continuous address block like in transparent mapping. The segments could be in any order, thus allowing the occupation of the empty address spaces by other masters. In the example shown in the figure 11 the address ranges are each defined as a segment which has its own start address and range. These segments are then assigned to the respective master. The order of the segments would then in this example be according to the start addresses. With opaque address mapping the RTL could remain the same as it already supports non-continuous address spaces which is a major benefit. Another benefit would be the adaptiveness of the mapping since it is not restricted as much as transparent mapping. The opaque bridges could be used with continuous address spaces as well which would possibly allow better features for the interconnect tool in the future development and better support for fine tuning the interconnect if necessary. Drawbacks of using opaque bridges would be related to compatibility with tools since the opaque mapping is less often used currently. The problems due to opaque bridges would then be more time consuming. The tools and resources used in the company would perhaps also need updates or modification to support the opaque mapping. These drawbacks fortunately would reduce in significance the more the method is used and the experience with it would accumulate.

The solutions found were then evaluated in terms of approximated effort, complexity and time needed for the improvement. They were compared to each other since the better would be chosen for implementation. The solution 1 would be mostly programming the second stage into the RTL and adding logic to the generation functions. The logic would need to be more complex than it is currently, but there probably would not be logic structures which are too complex or time consuming to implement. The solution 2 would be a more experimental approach since the opaque bridges are less known in the company. The working principle would need to be studied and then figure out their correct usage. The programming would be possibly trial and error as there would only be a few examples to study from or no examples at all. The approximated overall effort would be smaller for the solution 1 due to simpler nature even though the amount of programming would be much larger than with the solution 2. The complexity of the solutions would be on par with each other as the solution 1 would have logic structures based on the input values and the solution 2 would have more complex packaging command structures. The approximated time needed would be smaller for the solution 1 as the implementation would be rather straightforward. With these evaluations, the implementation decision leaned towards the solution 1 and the decision was finalized after comparing the pros and cons of each solution. The solution 1 would be the option less prone to deadlock as there would be better support in case of problems which cannot be solved without help of colleagues. The solution 1 would also have better reliability in the future due to high usage levels. Therefore, the implementation would be based on the solution 1 at first.

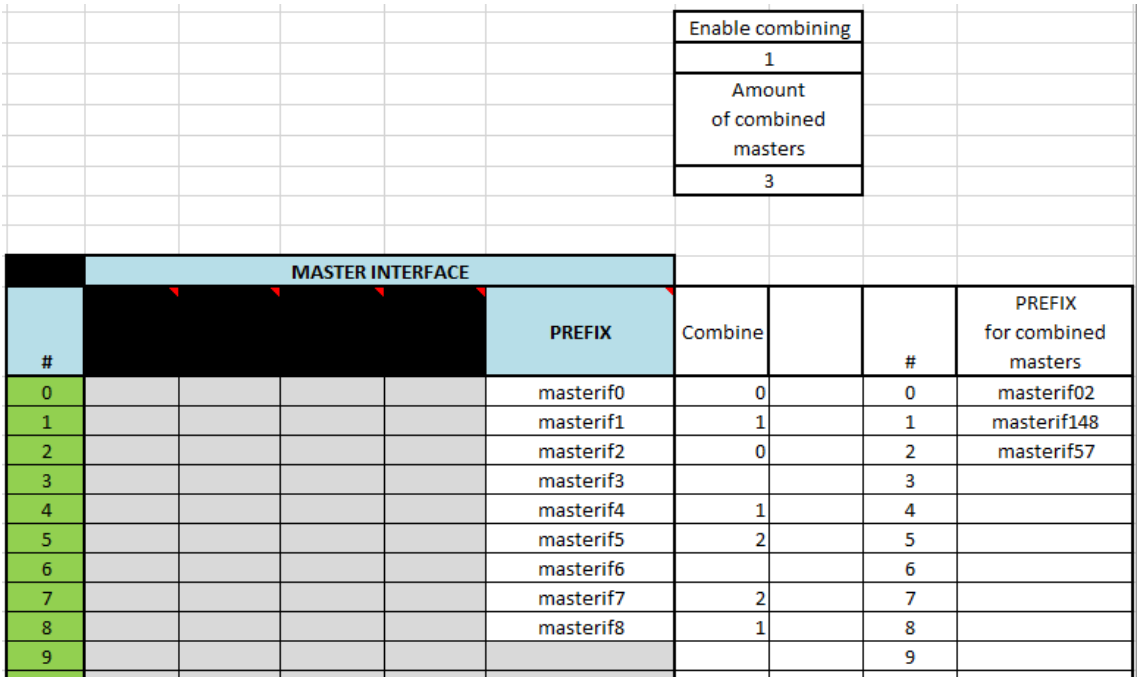
4.2.1 Non-continuous address space with transparent bridges

The implementation process can be categorized into three sections: the RTL related, the packaging related and the verification related modifications. The implementation process started with the RTL modification as the other sections are dependent on it. Before any modifications could be made, the exact operation of the interconnect tool functions was studied for the AXI4 protocol. The modifications could be replicated to the AXI4-Lite protocol easily, thus the AXI4 protocol was worked on first. The study of the functions was to investigate the interactions with the spreadsheet interface and the interactions between the functions themselves. Thus, pinpointing which functions need modification and if there is a need to alter the operation of the current functions or even create new functions with new tasks.

For the RTL section, the modifications were done mostly to the following functions: PrintVHDL_wrapper_AXI and GenVectors. The first function is responsible for the

wrapper generation and the latter generates the address mapping vectors. The wrapper modification started with designing the basic VHDL structure which contains the second layer crossbar instances. The resulting design is based on the original design, depicted previously in the program 2, with the necessary additions and alterations to the RTL to support the second layer. The planned changes to the original RTL code were modifying the wrapper ports, adding the second stage signals and instances, and modifying the signal mappings to the second stage instances and wrapper ports. A major design choice was done regarding the VHDL coding style. With the original RTL design, the code is static as it is generated with templates in the generation functions. The additional code from the second stage could be programmed to utilize the VHDL generate statements. The code would have better clarity and the ports, signals, instances and mappings would not have to be programmed separately. However, the programming was deemed to be more complex, thus the generate statements were not utilized in the new RTL design.

After the non-continuous address space design was planned, the interconnect tool functions modification was started. The basic generation principle was preserved, thus new templates were created for the additional second layer code parts. The first modification was the wrapper port definition when there are non-continuous address spaces present. The port definition should consist of the second stage ports and the uncombined ports. The first stage ports which are combined in the second stage crossbars should not be included in the wrapper ports. The logic to use the templates for VHDL generation is based on simple loops. The user interface prefix cells are looped in order and the modified templates are used for each port. This kind of approach would not work as the combined ports should not be generated. The non-continuous address space is also an additional feature and the tool should preserve the previous features. Therefore, the new templates utilized to create the new design should only be used when wanted by the user. From these requirements, it was apparent that there is a need to add logical structures to the generation function. To make the logic simple while preserving the previous features, a new interface area was added to the user interface. The added interface is an extension to the existing interface area. The updated interface is shown in the figure 12 below.



Enable combining
1
Amount of combined masters
3

	MASTER INTERFACE								
#					PREFIX	Combine		#	PREFIX for combined masters
0					masterif0	0		0	masterif02
1					masterif1	1		1	masterif148
2					masterif2	0		2	masterif57
3					masterif3			3	
4					masterif4	1		4	
5					masterif5	2		5	
6					masterif6			6	
7					masterif7	2		7	
8					masterif8	1		8	
9								9	

Figure 12. *Updated user interface for non-continuous address space feature*

As can be seen from the figure 12, the continuous address space feature was added into the interface as an opt-in feature. The user must enable the feature by setting the enable combining cell to value 1. The number of combined masters must be assigned which in this case means the amount separate combined master ports. In the figure 12 for example there are three combined masters which can be seen from the combine column and the prefix column for combined masters. The combining is done by assigning the same number to the combine cell next to the first stage master prefix cell. Those masters which have the same number are combined to the same second stage crossbar. The blank cells will be uncombined and will be in the port definition as is. The prefix for the second stage crossbar master port can be assigned in the prefix column for combined masters. The numbering must be continuous starting from zero to the maximum number of combined masters minus 1. This restriction is set to make the logic needed in the functions simple. The name of the combined masters can be anything with the same ruleset as the first stage prefixes.

With the added interface inputs, the modification of the generation function continued. The new input values were pulled from the interface to the macro as new variables in the SetVariables function. To preserve the previous features the port definition was modified from the sequential loop structure to branching loop structures. The major branch division was set to continuous branch and the non-continuous branch. The continuous branch contains the previous templates, and the non-continuous branch contains the new templates. The enable combining setting works as the value for the branch division. In

the non-continuous branch, there is another branching loop structure. The task for the loop is to create the combined master ports or the uncombined master ports. The logic for creating the ports is as follows. The combine column is looped in order and the value of the cell is checked. If the cell is blank, a first stage port is created. In other cases, on the first appearance of a number starting from zero, a combined port is created. When coming across an already created port, the port is not duplicated, and the next iteration of the loop is started. The names of the ports are pulled from the interface during the execution. For example, the order of the ports created with settings in the figure 12 is masterif02, masterif148, masterif3, masterif57 and lastly masterif6.

The next modification in the RTL was to add the second stage signals and crossbar instances. These are added only if there are combined masters set in the user interface. The second stage signals differ from the first stage signals which contain all the slave and master port slices. For every combined master, slave signals are created which contain only the assigned slave port slices. The number of slaves connected to a single second stage crossbar is calculated from the combine column. The master signals for the combined master only contain one master port slice as there is always only one master port in the second stage crossbars. After the additional signals are created, a crossbar instance is created for every combined master. The port map for the instances links their respective additional master and slave signals.

The last modification was the signal mapping. The signal mapping shares the similar continuous and non-continuous branches described previously. The signal mapping differs from the continuous case where the signals are mapped one-to-one from the wrapper slave ports through the crossbar to the wrapper master ports. With non-continuous case, the signal mapping differs after the first stage crossbar. The first stage master signals which are set to combine are mapped to the respective second stage crossbar slave signals. After the second stage crossbar inputs are mapped, the wrapper master ports are mapped. The mapping is like the port definition as the logic for the mapping is the same. The second stage master signals are mapped to the respective combined master ports and the uncombined master ports are mapped from the first stage master signals.

The other function which needed modification due to the second stage crossbars was the GenVectors function. Like the first stage crossbar, the address mapping vectors must be generated for the added crossbars as well. There are three different mapping vectors: address, mask and target vectors. The address vector contains all the address region start addresses combined into a single vector. The mask vector contains the ranges of the regions, modified into a format which can be used as a mask for incoming addresses.

And lastly the target vector contains the destination masters for the regions. All the values set in the spreadsheet are converted to binary format and the order of the regions is preserved in the vectors. These vectors are then used for routing incoming transactions to the correct master based on the address of the transaction. The generation of the first stage vectors follows the same simple approach in which the regions are looped through while doing the necessary conversions and formatting. The addition of the second stage vectors follows the same principle with modification to the contents of the vectors.

Before the modification was started, there were some planning and information sharing regarding the second stage vector contents. The address mapping could be done multiple different ways which would be functionally the same. Would there be only one address region which starts from the first combined master and has wide enough range for all the combined masters. Or perhaps the address mapping would be duplicated from the starting master to the last master. Another major question was if the address regions between the combined masters would be included in the second stage address mapping. If so, the second stage mapping would contain regions which are not mapped in the RTL. This could cause problems in a case where addresses not belonging to the master are used. Another way could be to remove the middle regions and shift the combined regions together. This could cause problems in the first stage addressing and there could be a need to add offsets to the address regions. After discussion, it was decided that the duplication method was the simplest and the least prone to error.

The logic for the second stage vector generation is as follows. Firstly, the starting point and the ending point of the combined master is determined utilizing the combine column of the interface. The logic includes the possibilities that the combined masters are not in the same order as in the first stage address mapping. For example, the combined master could be the masters 3 and 7, but the address mapping could have the opposite order. The duplication is from the starting address region to the last address region regardless of the order of the masters. The vectors are then generated using the same principle as with the first stage vectors. The difference is the regions which are looped through. As there can be multiple second stage crossbars, the same generation is done for every crossbar. The generated vectors are kept in a container from which the respective vectors are mapped to the second stage address mapping signals in the generation function.

The RTL related modifications were verified during and after the programming by running trial runs of the interconnect tool. Multiple different designs were tested including known edge cases. The operation of the tool and the resulting files were verified with internal

checks and code review. The functionality of the interconnect would be verified later with the sanity testbench after verification related modifications.

The next part of the improvement was the packaging related modifications. The modification would be solely to the Print_axi_packager_tcl function. Broadly the modification would be to modify the script commands in the program 4 to mirror the new wrapper design. The bus definitions would be changed to add the new combined ports and remove the old ports. The slave bridge definition would also follow the same procedure. However, the base address and range definition would be problematic even with the new RTL design. The second stage master ports would have a range which includes all the regions and thus the resulting IP-XACT description would have overlaps in the address mapping. The overlap would be caused by the limitation of the transparent bridges. The overlaps could cause problems in the usage of the interconnect after the packaging. This problem would not be present in a workaround since it would be a separate component with its own IP-XACT description.

A fix for the packaging script was not found in a timely manner, therefore the further investigation of the solution 2 was started and the solution 1 implementation was put on hold. The solution 2 could avoid the problems in the packaging script and could be the overall better solution if the opaque bridges are suitable. The remaining modifications for the solution 1 would have been to mirror the RTL modification to the verification related functions and add checks for the new interface options. After modification, the interconnect tool functionality would be checked with the sanity testbench. The same procedure would be replicated for the AXI4-Lite protocol as well.

4.2.2 Non-continuous address space with opaque bridges

The goal of the further investigation was to find certainty that the opaque address mapping could be used like proposed in the solution 2. The main question was if the mapping could be done in a script like the transparent mapping. By inquiring colleagues, opaque bridges had been used before, but not with a script. Instead, the opaque bridges had been done only in an EDA tool GUI (Graphical User Interface). As the transparent bridge scripting was based on TGI commands, the command reference was checked to find out if opaque bridges are supported. TGI based scripting would be beneficial since the scripting environment already supports it. At first glance, the reference had at least some commands for segments. Thus, a further study on the IP-XACT standard and the exact working principle of the opaque bridges was carried out to figure out the building blocks. The exact commands for the building blocks would then be figured out later. The

further investigation ascertained the feasibility of the solution 2 and the implementation process was started.

As a script had not been used in the company for opaque bridges, an experimental approach was chosen for the implementation. The implementation plan was to try to replicate the process of using the opaque bridges with a GUI within a script. An example design done with a GUI, which was provided from a colleague, acted as the reference. The example was a simple design which had two masters. The first master had a non-continuous address map with two segments. The second master had one segment between the first master's segments. After the necessary commands to replicate the example would be figured out, a generic flow of the commands would be created for the interconnect tool functions.

With trial and error -based approach, the script commands were figured out to make the necessary building blocks for the example. The basic principle was to try different TGI commands within an already generated script. Then running the script to generate the IP-XACT description which was imported to the EDA tool GUI. In the GUI, the block created from the IP-XACT description was placed into a design and its address map was checked. The address map was then compared to the reference and the differences were checked. This cycle was repeated until the generated file and the reference were the same. After the successful replication, the necessary commands for the building blocks were known. The next part of the implementation process was to modify the building blocks to be generic in the tool generation functions.

Before the programming started, some design choices were made regarding the implementation. The opaque bridges would be only used with non-continuous address space even though they could be used in all cases. This was to avoid changing the interconnect tool too much in one go. The non-continuous feature would be a great trial run for the opaque bridges. Another design choice was made for the script implementation itself. The earlier used abstracted TGI based functions could not be utilized fully with opaque bridges. The choices would have been to add abstracted functions to the imported function library or use the new TGI commands as is in the script. The latter was decided to be the approach as the commands would be used only in the interconnect tool specific feature.

For the opaque bridge approach, only the packaging script functions needed modification. The modifications were done first again for the AXI4 protocol and replicated for the AXI4-Lite protocol later. The functions modified were Print_axi_packager_tcl and Print_axl_packager_tcl. The generic approach in the function follows the same principle

of using the combination of logic and templates. The templates created are the generic version of the previous building blocks. Similar with the solution 1 implementation, the generation in the function was separated into the continuous branch and the non-continuous branch. The non-continuous branch is executed only when the address mapping in the spreadsheet contains at least one non-continuous master. An example of the used address mapping commands with opaque bridges is shown in the program 5 below.

```
##############################################################################################
# Bus definition commands for interfaces
##############################################################################################

#### Same as previous script ####

##############################################################################################
# Memory map commands
##############################################################################################

# Create memory map and variable containing its ID
<cmd_lib>::addComponentMemoryMap $<cmd_lib>::ID MemMap
set MemMapIDs [<cmd_lib>::getComponentMemoryMapIDs $<cmd_lib>::ID]

# Segments and subspace maps for master interface prefix
set busifIDs [<cmd_lib>::getComponentBusInterfaceIDs $<cmd_lib>::ID ]
set master_if <prefix>_<protocol>_Master
foreach busifID $busifIDs {
        if { $master_if == [<cmd_lib>::getName $busifID] } {
        <cmd_lib>::setBusInterfaceMasterBaseAddress $busifID 0x0
        <cmd_lib>::addComponentAddressSpace $<cmd_lib>::ID $master_if\_addrSpace 0x400000
        <cmd_lib>::setBusInterfaceMasterAddressSpaceName $busifID $master_if\_addrSpace
        set addspaceID [<cmd_lib>::getBusInterfaceMasterAddressSpaceID $busifID]
        <cmd_lib>::addAddressSpaceSegment $addspaceID <prefix>_seg0 0x100000 0x80000
        <cmd_lib>::addAddressSpaceSegment $addspaceID <prefix>_seg1 0x300000 0x100000
        }
}

# Variables for subspacemap segment references
set <prefix>_submap0 <prefix>_subspacemap0
set <prefix>_submap1 <prefix>_subspacemap1

# Add subspacemaps to the memorymap
foreach MemMapID $MemMapIDs {
        <cmd_lib>::addMemoryMapSubspaceMap $MemMapID $<prefix>_submap0 $master_if 0x100000
        <cmd_lib>::addMemoryMapSubspaceMap $MemMapID $<prefix>_submap1 $master_if 0x300000
        set submapIDs [<cmd_lib>::getMemoryMapElementIDs $MemMapID]
}

# Set segment references for the subspacemaps
foreach submapID $submapIDs {
        if { $<prefix>_submap0 == [<cmd_lib>::getName $submapID] } {
        <cmd_lib>::setSubspaceMapSegmentRef $submapID <prefix>_seg0
        }
        if { $<prefix>_submap1 == [<cmd_lib>::getName $submapID] } {
        <cmd_lib>::setSubspaceMapSegmentRef $submapID <prefix>_seg1
        }
}

# Bridges for slave interface prefix
set busifIDs [<cmd_lib>::getComponentBusInterfaceIDs $<cmd_lib>::ID ]
set slave_if <prefix>_<protocol>_Slave
set master_if <prefix>_<protocol>_Master

foreach busifID $busifIDs {
        if { $slave_if == [<cmd_lib>::getName $busifID] } {
        <cmd_lib>::addBusInterfaceSlaveBridge $busifID $master_if true
        # Memorymap reference for the slave interface, done once per slave
        <cmd_lib>::setBusInterfaceSlaveMemoryMapName $busifID MemMap
        }
}
```

Program 5. *Address mapping commands used for a non-continuous example design*

The example in the program 5 shows the command flow for a single master. The master has a non-continuous address space which consists of two address regions. The implementation for the address mapping has the same three general building blocks as

the non-continuous script. The first building block is for the bus interface definitions. The commands were reused from the continuous implementation since there are no differences in the bus definitions with different bridges. The second building block is the address mapping data for the master. The address mapping structure for the implementation is as follows. Each master has an address space containing segments. The segments are the address regions assigned for the master in the spreadsheet. The slave has a reference to a separate memory map consisting of subspace maps. There is a subspace map for every segment in a master's address space. The subspace maps are linked to the segments using a segment reference. The logic and the templates used to generate such a structure are opaque bridge specific and they use the TGI commands straight without abstraction to functions. The first step in the memory map commands of the program 5 is to create the memory map. The next step is defining the address map of the master. The address map for the master always starts from the address 0x0 and must be wide enough to contain every assigned address region. The width is calculated from the spreadsheet utilizing the destination column. In the example, the width of the master address space is 0x400000. Segments are then created from the address regions with their own base address and range. These segments are added to the address space of the master. The next step is to create the subspace maps. For every segment, a subspace map starting from the same base address is created. The subspace maps are added to the earlier created memory map. Lastly the corresponding segments are linked to the subspace maps with a reference. To complete the address mapping, the last building block is to create the bridges between the masters and the slaves. Like in the continuous script, every slave is connected to every master. The bridge definition creates an opaque bridge to the master. The last step is to add a reference to the memory map, which is the address map seen by the slave, containing every master's segments in a particular order. The address mapping structure generated by running the script in the program 5 is illustrated in figure 13 below.

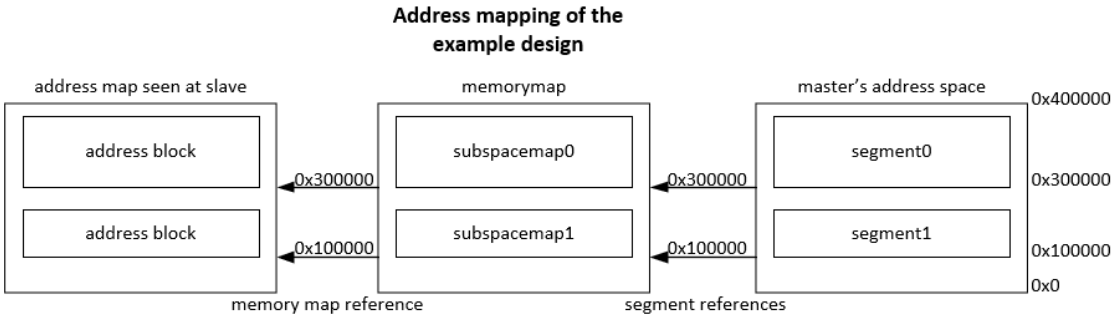


Figure 13. *The address mapping structure of the example design*

As seen from the figure 13, the basic structure of the mapping is simple even though the command flow was more complex than with the transparent bridges. With multiple masters, the building blocks are executed for every master sequentially before moving onto the next building block. The memory map remains the same with multiple masters as well. Thus, the basic structure remains similar as in the memory map will be just wider due to increasing number of subspace maps referencing the segments of every master.

With the modification for the script generation function being ready, the correct functionality of the function was verified similarly as the experimental approach for the script commands. The interconnect tool was run with multiple different non-continuous address mappings and the resulting script was first checked visually by code review. The script was then run to generate the IP-XACT description of the design which was imported to an EDA tool GUI. In the GUI, the structure of the address mapping was checked. The interconnect block was then again inserted to a design, where the address map seen at the slave was verified. After the verification, the modifications were replicated for the AXI4-Lite protocol and the same verification approach was used.

After the feature was finalized for both protocols, the non-continuous address space feature was deemed to be ready. The implementation based on the solution 2 was much better in the end. The only modification necessary were for the packaging functions and the rest of the functions could be kept as is. The implementation based on opaque bridges had more complex packaging command structure which required a substantial study and experimenting on the subject. The overall programming effort was less cumbersome though than with the solution 1. Even if the solution 1 implementation was put on hold, it will be kept as the backup implementation. This is due to the possible unforeseen problems with opaque bridges which could appear in the usage of the generated interconnect.

4.3 Progress on the rest of the improvement requests

After the non-continuous address space improvement was ready, the improvement process for the next improvement from the requests was started. This chapter sums up the progress made for the rest of the requests during the thesis. Since the non-continuous address space improvement turned out to be more time consuming than anticipated, the implementation was not started for some of the requests. It was decided that the feasibility of every request is investigated. If feasible, then suggest solutions and decide the most promising solution for possible future implementation.

4.3.1 Master-slave visibility improvement

The improvement started again with studying the cause for the limitation, which was quickly confirmed to be from the RTL implementation for the crossbar component as earlier discussed. The feasibility of the improvement would then depend on the solution. The priority to keep the utilized components unchanged was present, thus the improvement would have to be made elsewhere.

Two different approaches were discussed. The first approach was to change the packaging for the interconnect to generate the connecting bridges based on user inputs. The required change in the tool would be trivial as in adding visibility options to the user interface and making the bridge commands based on those. The second approach was to modify the wrapper design to adapt to the user designated connections. The design would then have multiple crossbars which connect to each other forming only certain slave-master connections. The packaging would also reflect this. The first approach would be simple, but there could be major problems caused by it. If only the packaging is changed, the generated IP-XACT description would not reflect the internal connections in the RTL. The RTL would have connections from every slave to every master and the IP-XACT description only the ones chosen. This could cause problems in verification as the RTL and the IP-XACT description would clearly describe different designs. Thus, the second approach was chosen as the feasible solution. An example of the solution is introduced in the figure 14 below.

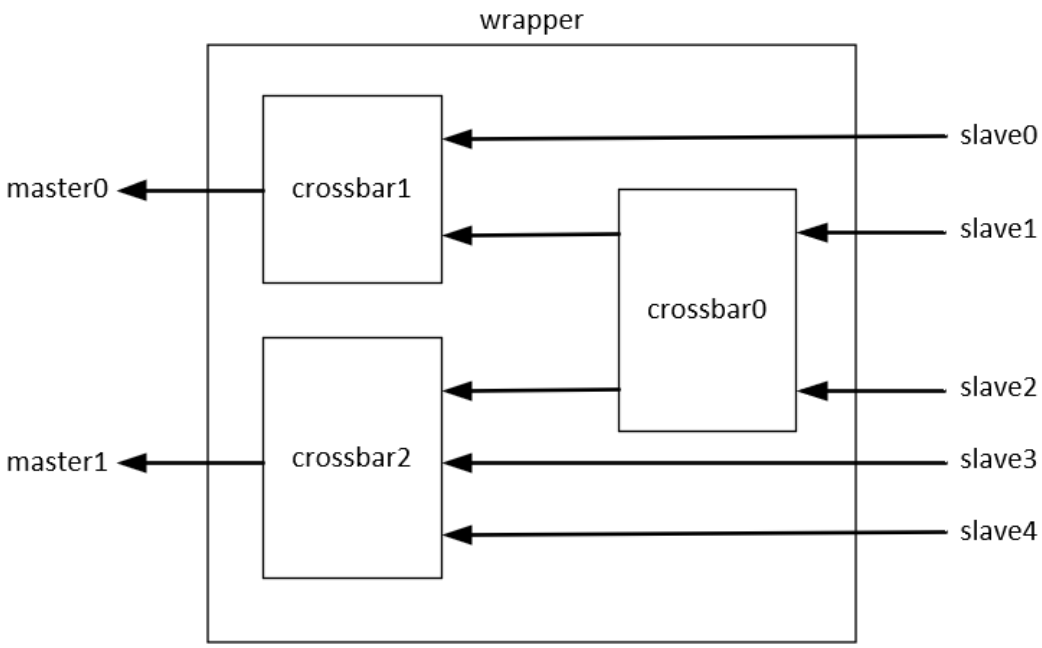


Figure 14. *An example design for visibility improvement*

As shown in the figure 14, the connections are formed by using a certain structure of crossbar instances. The structure is tailored for the user inputs. The basic structure is that there are crossbars which route shared connections and crossbars which form the collection of routes for the specific master. In the example, the slave1 and the slave2 interfaces are routed to both master interfaces and the rest are routed only to a single master interface. Similar structure would be used with each routing.

The main challenge in the implementation would be to add the logic which determines the optimal structure and the connections. The goal would be to have no restrictions for the routing which can be seen from the example as the number of inputs in the crossbars are not limited to multiples of twos. However, the logic increases with the scale of the interconnect. There would possibly be a tipping point where the structure would simply be too large or complex to generate automatically. The benefit gained from the reduced routing would also suffer from the sheer number of crossbar instances. Thus, there would probably be restrictions in the routing. The upper limit would possibly be determined from the usage needs. For example, the visibility feature is tailored towards the common routing structures used in the company. The implementation based on the solution was not started during the thesis.

4.3.2 Range definition improvement

For the improvement, an excessive study of the used RTL components was carried out. This was to determine the root cause for the limitation which was deemed to be from a core design property for an address mapping component. The limitation of the power of 2 ranges stems from the binary address mapping. The mapping vectors must be binary and the ranges in the power of 2 for the mapping component to work properly.

One suggestion was to round up the range to the next power of 2 value for the vector generation. However, this would cause a problem where an incoming address would possibly map to two different regions causing a possible destination error. The error would be caused in a case where the rounded range overlaps with another address region. It was concluded that the range definition improvement is not possible without a major change in the mapping component's functionality. Thus, the improvement is not feasible currently since the component redesign would be out of the scope of this thesis.

4.3.3 Generic features improvement

The generic features improvement request was at first directed to the interconnect tool. The improvement process for the generic features was fast and clear since the design was well known at the point of improving. Generics were added for the various signal widths and the design was changed to use generics instead of constants.

The request was later expanded to more of a project tailored request after the generic address signal width was found out to be unfeasible due to the address mapping vectors. The address mapping for the RTL is generated from the spreadsheet and thus it cannot be changed afterwards to reflect a change in the address width generic. The address mapping vectors would have to be created in the RTL instead of during the generation. The vectors would have to be generated based on the address width generic which would also mean that the address mapping in the spreadsheet could not be static. The start addresses and ranges would need to change accordingly based on the address signal width to avoid out-of-bounds or overlapping.

It was decided that the address width would not be changed to a generic for the interconnects generated with the tool. Generic address width would be better for a separate generic interconnect which has a fixed address space. The further improvement of the generic features was then also forwarded to the requesting project instead.

4.3.4 Self-documentation improvement

As the request was directed towards the ring interconnect generation and the feature would be more useful for it, the feasibility of the improvement was studied with the ring interconnect as the priority. The investigation's main goal was to determine if the Visual Basic for Applications -programming language could be used to create a block diagram based on the input data from the spreadsheet.

The investigation led to a possible solution. At first glance, every action performed in the Excel spreadsheet can be replicated with VBA code. The proposed solution would be to program a macro which automatically generates the block diagram to a separate sheet. The implementation process could be to record the actions for making a simple block diagram. Then modifying the actions to take data from the spreadsheet and using them to generate the necessary blocks and connections. The macro execution would be on-demand to avoid the possible decrease in performance caused by the block diagram generation. For the same reason, the macro should be separated from the interconnect generation macro.

Even though the implementation was not started during the thesis, some priorities were discussed for it. The macro needs to be efficient. The execution should not be tens of minutes since the point of the feature is to save the time from the current process of visually representing the generated interconnect. Other priority is complexity. If the implementation turns out to be more complex than anticipated, the implementation of the feature should be re-evaluated. Other options than VBA, for example Python programming language, should be evaluated when the complexity of VBA code would increase to a certain threshold.

4.3.5 Default master interface improvement

The improvement was quickly determined feasible as the limitation was simply a design choice in the interconnect tool. The example given earlier would also be the proposed solution. The user sets the total address space by the address space width parameter, then defines smaller address regions in the address mapping. The unoccupied address space is calculated and assigned for the default master interface. The user interface would be modified to show the default master interface and the calculated range for it. The solution is simple, but the implementation would have complications.

The major complication would be the range for the default master interface. As earlier discussed in the range definition improvement, values not in the power of 2 are not feasible without a major change in the mapping functionality. This would cause a

restriction for the smaller regions. The user added regions must be also in the power of 2 and their range summed up must be suitable. The range assigned to the default master interface must be in the power of 2 or the range can be divided to multiple address regions which follow the same rule. There would be a need to automatically determine the suitable combination of address region ranges which are assigned to the default master interface. An example of the proposed solution in the user interface is shown in the figure 15 below.

	Address mapping			
Addr Region #	Start address (HEX)	Length in Kbytes	Destination (Master interface)	End address
0	0	2097152	0	7FFFFFFF
1	80000000	1048576	0	BFFFFFFF
2	C0000000	524288	0	DFFFFFFF
3	E0000000	262144	0	EFFFFFFF
4	F0000000	32768	1	F1FFFFFF
5	F2000000	16384	2	F2FFFFFF

Figure 15. *An example of the proposed solution*

In the figure 15, the default master interface would be the master interface indicated by the number 0 in the Destination column. The smaller regions would be the address regions 4 and 5. The address regions from 0 to 3 would be the combination of suitable ranges for the default master interface. The combination would be calculated automatically and assigned to the address mapping interface. The interface would be modified to clarify the user added smaller regions from the default master interface.

This kind of calculation must be already done manually by the user if the wanted address space for a single master is not in the power of 2. Therefore, the logic needed for the calculation could also be utilized in the regular usage. While implementing the default master interface, the feature could be added as opt-in feature and the automatic range calculation as a basic feature. Both would have restrictions for the suitable ranges though. The restriction would be the amount of address regions needed for the automatic calculation. With some ranges, the number of address regions needed to form the wanted address space would be too large for the current implementation of the interconnect tool. This emphasizes the idea of redesigning the address mapping functionality since it would affect multiple parts of the tool and would allow easier improvements to the address mapping. Due to this, it was decided that the solution would not be implemented during the thesis.

5. CONCLUSION

As the number and complexity of the interconnects increases in the SoC devices, the effort required to generate these interconnects becomes larger as well. To reduce the manual effort, automation of interconnect generation is utilized. Currently in the company, the automation is provided in the form of an RTL generator. The generator is built on an Excel spreadsheet which is used to provide the input parameters for the generation. The output files are generated by the spreadsheet as well. The generation is implemented as a macro utilizing Visual Basic for Applications -programming language. The supported communication protocols are the AXI4 protocols introduced in the AMBA 4 specification and the tool can generate crossbars and ring interconnects.

The purpose of this thesis was to improve the interconnect automation tool to provide a more fluid and flexible implementation of the tool. The improvement started with an investigation on the current implementation to define the baseline for the improvements. During the investigation, the limitations of the tool were defined which worked as the improvement proposals. Most of the limitations were regarding the usage of the tool and the possible options provided. The feature worked on the most during the thesis was to add support for non-continuous address spaces. The improvement process followed a simple structure in which the feasibility of the improvement was defined first, and then possible solutions were proposed. The most prominent solution was decided for the implementation depending on multiple factors. One major factor being the decision not to modify the sub-components of the interconnect since the modifications could compromise operation in more places than wanted. Modifications were done in the interface and the generation macro depending on the improvement.

Even though only the non-continuous address space improvement could be implemented, due to it taking much longer than anticipated, the goals of the thesis were met. The feasibility of each improvement proposal was examined and many of them were deemed unfeasible. Proposed solutions for feasible improvements were also investigated even if the implementation for them could not be started. The unfeasible improvements showed an underlying restriction for the tool which affects multiple areas. The implementation for the non-continuous address space provided the possibility to improve the tool further as well which is to move from transparent bridges to opaque bridges in every interconnect in the future. The thesis work clarified the wider scope for the interconnect automation tool and the improvement will continue moving forwards.

REFERENCES

[1] José L. Ayala. Communication Architectures for Systems-on-Chip. CRC Press; 2018.

[2] Pasricha S, Dutt N. On-Chip Communication Architectures: System on Chip Interconnect. San Francisco: Elsevier Science & Technology; 2008.

[3] Jun M, Woo D, Chung E-Y. Partial Connection-Aware Topology Synthesis for On-Chip Cascaded Crossbar Network. IEEE transactions on computers. 2012;61(1):73–86.

[4] Schaumont P. Practical Introduction to Hardware/Software Codesign. Boston: Springer; 2012.

[5] Jerraya A, Wolf W. Multiprocessor Systems-On-Chips. Saint Louis: Elsevier Science & Technology; 2004.

[6] Flynn MJ, Luk W. Computer System Design System-on-Chip. 1st edition. Hoboken: Wiley; 2011.

[7] AMBA 4 Overview. Available: https://developer.arm.com/architectures/system-architectures/amba/amba-4 Accessed on 24.09.2021

[8] Shrivastav A, Tomar GS, Singh AK. Performance Comparison of AMBA Bus-Based System-On-Chip Communication Protocol. In: 2011 International Conference on Communication Systems and Network Technologies. IEEE; 2011. p. 449–54.

[9] Math SS, Manjula RB, Manvi SS, Kaunds P. Data transactions on system-on-chip bus using AXI4 protocol. In: 2011 International conference on recent advancements in electrical, electronics and control engineering. IEEE; 2011. p. 423–7.

[10] AMBA AXI and ACE Protocol Specification. Available: https://developer.arm.com/documentation/ihi0022/hc/?lang=en Accessed on 27.09.2021

[11] An introduction to AMBA AXI. Available: https://developer.arm.com/documentation/102202/0200/AXI-protocol-overview Accessed on 27.09.2021

[12] AMBA AXI-Stream Protocol Specification. Available: https://developer.arm.com/documentation/ihi0051/b/?lang=en Accessed on 28.09.2021

[13] IP-XACT User Guide. Available: https://www.accellera.org/images/downloads/standards/ip-xact/IP-XACT_User_Guide_2018-02-16.pdf Accessed on 30.09.2021

[14] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. IEEE; 2014. 1-510 p.

[15] Shin C, Grun P, Romdhane N, Lennard C, Madl G, Pasricha S, et al. Enabling heterogeneous cycle-based and event-driven simulation in a design flow integrated using the SPIRIT consortium specifications. Design automation for embedded systems. 2007;11(2):119–40.

[16] Kruijtzer W, Vaumorin E, van der Wolf P, de Kock E, Stuyt J, Ecker W, et al. Industrial IP integration flows based on IP-XACT standards. In: 2008 Design, Automation and Test in Europe. IEEE; 2008. p. 32–7.

[17] Morgado F. Programming Excel with VBA: A Practical Real-World Guide. Berkeley, CA: Apress L. P; 2016.

[18] Office VBA Reference. Available: https://docs.microsoft.com/en-us/office/vba/api/overview/ Accessed on 04.08.2021

[19] The Value of High Quality IP-XACT XML. Available: https://www.design-reuse.com/articles/19895/ip-xact-xml.html Accessed on 24.08.2021

[20] TGI API Documentation. Available: http://www.accellera.org/XMLSchema/SPIRIT/1.4/TGI/TGI.html Accessed on 24.08.2021