UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Ville Hokkinen

RAPID HIGH-LEVEL BEHAVIORAL MODELLING OF SOC COMMUNICATION INTERFACES

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
February 2022

Hokkinen V. (2022) Järjestelmäpiirien tiedonsiirtoyhteyksien nopea korkean tason käytösmallinnus. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 22 s.

TIIVISTELMÄ

Laitteistojen ja systeemien kasvava kompleksisuus lisää niiden mallinnukseen vaadittua työmäärää ja resursseja. Erityisesti järjestelmäpiireissä (SoC) mallinnuksen monimutkaistuminen näkyy. Elektronisen järjestelmätason (ESL) mallinnus käyttämällä tapahtumatason mallinnuksen (TLM) hyötyjä vähentää vaadittuja resursseja ja nopeuttaa mallinnukseen kuluvaa aikaa.

Tässä opinnäytetyössä tarkastellaan likimääräisesti ajastettua mallinnusta käytösmallinnuksen kontekstissa abstrakteilla prosessointi elementeillä ja abstrakteilla sovittelu proseduureilla.

Tässä työssä kuvataan myös järjestelmäpiirin ja TLM:n perusperiaatteet, jonka jälkeen tutkitaan erään yrityksen SoC:in tiedonsiirtoväylän mallinnuksen ongelmaa, ja tutkitaan TLM ratkaisua mallinnusongelmalle.

Avainsanat: järjestelmäpiiri, tiedonsiirtoväylä, elektroninen järjestelmätaso, tapahtumatason mallinnus

Hokkinen V. (2022) Rapid High-Level Behavioral Modelling of SoC Communication Interfaces. University of Oulu, Degree Programme in Computer Science and Engineering, 22 p.

ABSTRACT

The increasing complexity of hardware and software systems increases the amount of labour and resources required to model them. Especially in system-on-chips (SoC), the complexity of modeling is evident. Electronic system-level (ESL) modeling using the benefits of transaction level modeling (TLM) reduces the required resources and speeds up the modeling time.

This thesis examines approximately timed modeling in the context of behavioral modeling with abstract processing elements and abstract arbitration procedures.

This thesis describes the basic principles of SoC:s and TLM, and then a problem of modeling a SoC communication interface for a company is investigated, and a TLM solution to the modeling problem is explored.

Keywords: system-on-chip, communication interface, electronic system-level, transaction level modeling

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

SoC	System on Chip
ESL	Electronic System-Level
TLM	Transaction Level Modelling
DSE	Design Space Exploration
IC	Integrated Circuit
HW	Hardware
FIFO	First In, First Out
IP	Intellectual Property
VHDL	Very high-speed integrated circuit Hardware Description Language
SysML	Systems Modeling Language
UML	Unified Modeling Language

1. INTRODUCTION

The increasing complexity of system-on-chips (SoC), combined with the drive for more sophisticated applications utilizing the available resources smarter and more energy-efficient, has driven designers around the world to increase the level of abstraction in system modelling. This has been pursued by adopting electronic system-level (ESL) design methodologies to lower simulation times and reduce the labour intensity of the modeling. Modeling systems early gives the engineers a chance to catch issues earlier in the development cycle, making correcting the issues cheaper and easier. [1]

Transaction level modelling (TLM) is a set of techniques that enable ESL and make it practical. Implementing these methodologies in system design has several application areas: architectural modeling, algorithmic modeling, virtual software development platforms, functional verification and hardware refinement. [2 Section 1.3]

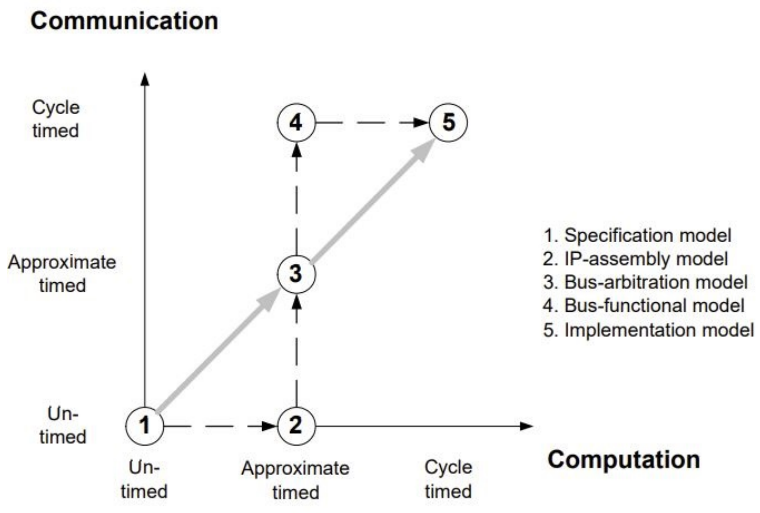

Figure 1. System modeling graph

In the context of behavioral modelling of SoC communication interfaces, this thesis looks into approximately timed modelling with abstract arbitration procedures and abstract processing elements. From the system modeling graph introduced by Cai et al. in Figure 1 [3], that would correspond to bus-arbitration modelling. This allows the model accurate design space exploration (DSE), while decreasing the complexity of the model and the needed labour to produce the model.

This thesis will look into different options for rapid TLM and explore a solution for a company with a DSE problem in their communication interface under development. The problem also includes the need for the solution to be applicable to future similar modeling DSE problems,

2. THEORY

2.1. System on Chip

SoC is, by definition, an integrated circuit (IC) system with all of the components of a specified electronic system integrated on the chip. In this context, system includes a microprocessor, memory and peripherals. One kind of memory is First In, First Out (FIFO) memory, found in SoC:s.

SoC:s are found in every consumer product area, and semiconductor companies are moving towards SoC designs, due to their numerous benefits. SoC design provides lower system cost, compact system size, reduced power consumption, increased performance and intellectual property (IP) blocks, which can be reused, customized and optimized.

Even though moving to SoC provides many benefits, it also has its disadvantages. For example, in big designs the whole logic may not fit on the chip. Also the manufacturing becomes a challenge, as now the whole chip needs to be manufactured instead of modules. Same problems arise in design verification and validation, as verifying and validating individual modules becomes hard or impossible, due to the system being monolithic. [4, 5 Section 1]

2.2. Data Communication Interface

Communication interfaces in computer science consist of data lines on which data is sent and received. This data transmission is done in a diverse set of protocols and arrangements, such as serial and parallel protocols, depending on where the interface is used. They are used to make other systems available to communicate with. The data in this communication consists of either address, control and payload type items. Address is used to communicate a location in which some operation is performed, control describes what kind of operations are performed and payload information which the communication system transmitting or receiving.

The place of the interface and type of data handled in the interfaces poses some indirect requirements on the system. An interface used for transmitting controls typically is more vulnerable to errors, as the tolerances are smaller and resources for highly accurate error correction are not always available. Interfaces used for transmitting payloads typically have more redundancy in the form of system level repetition and error correction procedures. [6]

The lines between systems are most typically electrical wires, but optical cabling can also be used, both having their advantages and disadvantages. Electrical signals are transmitted as a difference in voltage in the wire, and optical signals as fluctuations in the light inside the cable.

2.2.1. Parallel and Serial

In parallel communication, bits are sent on parallel data lines, with a package consisting of the state of the lines at the corresponding moment in time, as per Figure

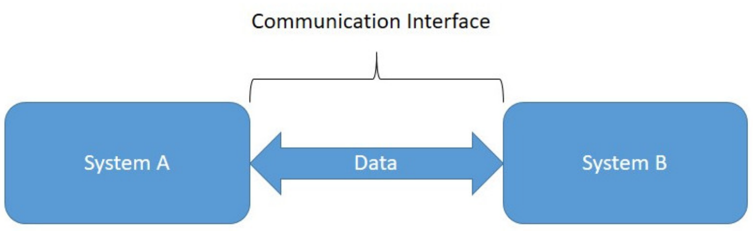

Figure 2. Communication interface.

3. A data line is required for each bit and an additional data line for synchronization. Parallel data transmission is competent over small distances, but over longer distances, the cost of multiple data lines becomes a problem, and signals in the cables may get misaligned, causing timing skew and packet corruption on the receiving end. [7 Section 7.5]

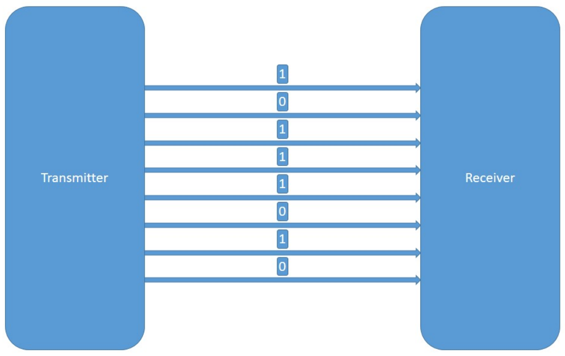

Figure 3. Parallel interface.

Serial communication on the other is carried out on only one data line, where packets are sent in subsequent bits as in Figure 4. The advantage of this is that bits arrive in the right order in the receiver. Also, due to the decrease in the required number of wires, circuitry and cabling become simpler and more affordable. Serial communication has become more practical over the years compared to its earlier days, due to the increase in circuit speeds and the development of a wide variety of protocols. These protocols aim to work at different transfer speeds, as the number of errors and disturbances in data transmission grows with speed. [8]

An example package from universal asynchronous receiver-transmitter protocol is shown in Figure 5, which shows how in addition to data payload, other types of bits than data bits are added to a packet. Here one start bit is added, to mark the start of the package to the receiver. The start bit is followed by the payload. After the payload, parity bit or bits are added, to check the validity of data. This is done by using some algorithm on the data, which outputs the parity bits. This algorithm is also performed on the payload in the receiving end, providing the same parity bits if the data is the same. If one bit of the data is different, the algorithm aims to change the parity bit or

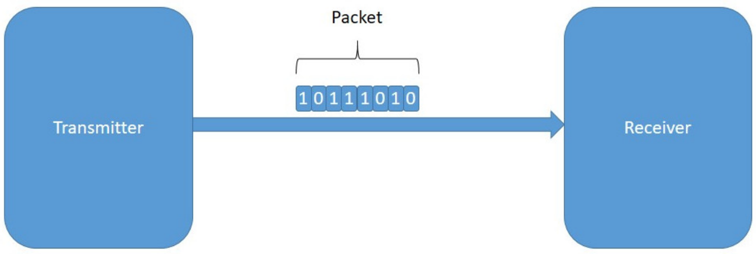

Figure 4. Serial interface.

bits so that the receiver notices the data corruption, and can inform the sender, that the data was corrupted and a resend is required.

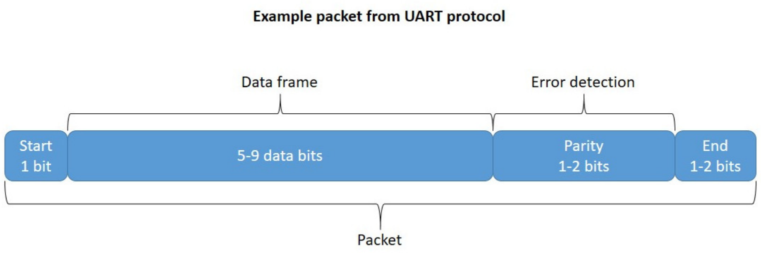

Figure 5. Example packet from universal asynchronous receiver-transmitter protocol.

2.3. Modeling

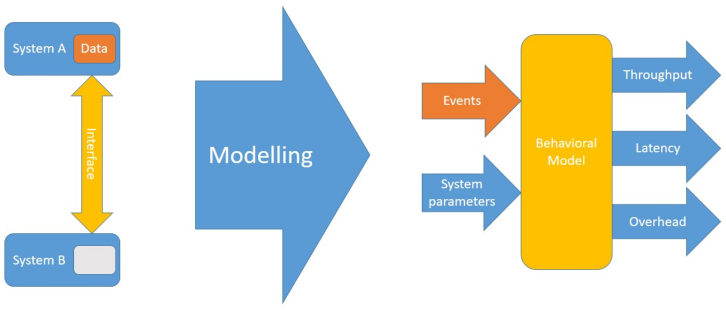

Figure 6. Behavioral model use case example.

TLM has been widely adopted in different fields and several tools have been introduced for modeling tasks. In general, TLM:s impose a trade off between simulation speed and accuracy loss. The problems designers face now relate to re-usability of modelled system blocks and capturing the right amount of complexity in the model. The models need to also have support for being flexible and quickly adaptable, due to rapidly changing design needs. The modelling and eventual refinement of Intellectual Property (IP) blocks becomes a priority as block like interface-based design allows for the needed flexibility. [9]

TLM modelling can in theory be done with almost all high level languages. Still some specific versions have emerged which offer better support for different aspects of modelling. [2 Section 1.4]

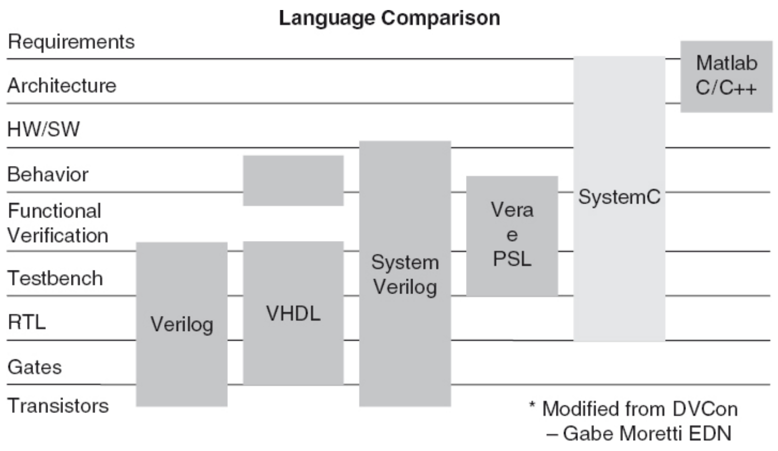

Figure 7. Use of languages.

2.3.1. Existing Tools

You can see in Figure 7 [2 Section 1.4] some of the general use cases for different modeling coding languages. The most suitable of these are described in more detail below, in addition to some unorthodox, but still possible options.

SystemC

SystemC is a C++ language extension with class libraries used for system design and verification. It is especially used for modeling system partitioning, assessing and establishing hardware or software block assignment, and designing and quantifying communication between and amid operative blocks. SystemC is used for architectural exploration by well-known companies in the electronic design automation, intellectual property, embedded software, electronic systems, and semiconductor industries [10]. SystemC has been criticized for lacking support for verification and debugging, being expensive on resources to run and being difficult to debug. [11]

VHDL

Very High-Speed Integrated Circuit Hardware Description Language (VHDL) is a formal notation that is designed to be used in every phase of electronic systems design. According to the IEEE standard, VHDL has support for "the design, development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware." [12 p.14]. As a hardware description language, the main focus for VHDL behavioral modeling is in modeling low level hardware circuits. This falls out of the scope of this thesis.

SysML

Systems Modeling Language (SysML) is to be used as a commonplace modeling language for use cases in Systems Engineering. It has support for the analysis, specification, validation, and verification of systems such as hardware, software, processes, information, facilities, and personnel. It is built upon Unified Modeling language (UML) as an UML 2 profile, meaning it reshapes UML by three structures, which are tagged values, stereotypes, and constraints. SysML has support for SysML-as-model-simulation, where the modeled system can be run on a behavioral modeling engine. [13]

DIPLODOCUS

DIPLODOCUS is a UML profile run on an open-source toolkit TTool for modeling and designing data-flow embedded systems from UML and SysML diagrams. It is based on four principles: using high-level and well-known language, abstracting data, using simulation and static formal analysis techniques on models, and separation of architecture and application. Because of the underlying transformation rules, DIPLODOCUS design can be used to generate a SystemC executable that generates visualizable traces or to generate LOTOS code, a formal description technique, which is supported and standardized by formal validation tools.[14]

Universal Verification Methodology

Universal Verification Methodology (UVM) is a methodology, which is used to verify IC designs [15]. It is a class library to SystemVerilog, which is an hardware verification and description language [16]. The focus of UVM is in helping companies develop testbench structures, which are modular, reusable and scalable. UVM has been criticized for having a steep learning curve [17], and for this reason will not be considered for later modeling efforts, as it falls out of the scope of this thesis.

JavaScript

JavaScript is a popular scripting language, which can be used together with Node.js, an asynchronous event-driven runtime environment for JavaScript code, to create executable applications. Together JavaScript and Node.js can be used to create easy to maintain, high-level, operating system independent and fast applications. JavaScript also supports importing C/C++ native addons, allowing for integrating existing real system code to the model, bringing the model closer to reality. JavaScript also has an extensive libraries containing tools for many kinds of data manipulation as well as tools for assisting development. [18, 19]

While any high-level language can be used for TLM modeling, no proof of JavaScript being used for modeling was found. This hampers modeling, as no proof of concept is available as an example.

Python

Python is a high-level programming language, used for general-purpose programming. Main focus of the language is in readability and producing clear and logical code for both small and large projects. It has extensive libraries for various data manipulation techniques and it has support for asynchronous operations in the form of coroutines. [20]

3. REQUIREMENTS AND DESIGN

3.1. The Problem

There are a good deal of different communication interfaces in the company's product, and an example interface is provided in Figure 8. One of the key design factors for optimized systems in the product is finding the optimal parameters for all the interfaces. This optimization needs a robust and a reliable way to model different design options. As a part of this thesis work a solution in the form of a general model is explored and an example design is described.

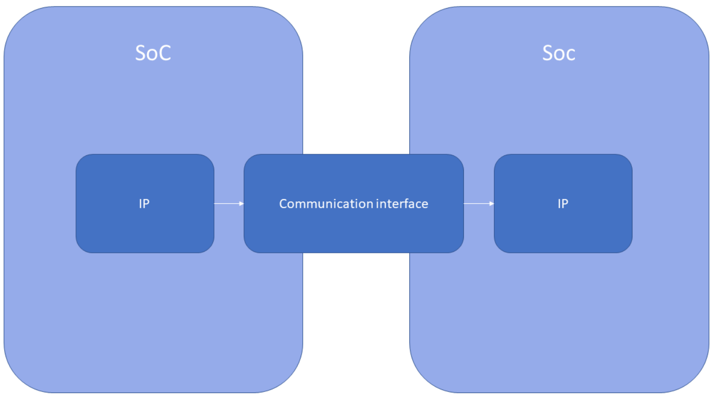

Figure 8. Problem description.

3.2. Requirements

The development environment in the company combined with the use case for the model impose some requirements on the model. In this chapter the requirements are highlighted and introduced under two topics: one for bringing forth the functional requirements for the modeling system and one for highlighting the architectural requirements for the modeling system architecture.

3.2.1. General Functional Requirements

The model needs to provide certain output parameters to measure the functionality of the link. It needs to offer throughput for different use cases, suitable FIFO size for the data flow, approximate power consumption of the system and latency for each control passed through the interface. The model also needs to have certain static parameters, which can be profiled using the model, such as the number of parallel data lines and FIFO:s, control command sizes and the composition of the higher level controls, clock frequency and data encoding.

In addition, the model has needs to have an intuitive input and output interface which can be handled in a standard way. This way different use cases can be crafted to suit the need of the use case that is investigated. The input data flow needs to be in high enough abstraction that creating new use cases is easy and efficient. To accomplish this, the model needs to have a parser to break down high-level controls into low-level control commands to pass through the system. Using the predetermined output the model should also have a standard logging and visual representation of the logs. The model should also have a built-in tracing and debugging to find possible problems with the design. These should also be plotted for more accessible debug data.

3.2.2. Architecture Requirements

The interface model needs to be re-usable as many different use cases within the company exist for such a communication interface model. Re-usability greatly lowers the required design time in future applications, but increases the current design time. To achieve re-usability the model should be modular, with thought put into possible future extensions.

The model also needs to have some kind of abstract definition for communication system components, with additional details being added on per model basis. This also supports the re-usability aspect of the design, as it is relatively easy to build a new system model with generic components.

The model should provide easy debugging for finding out faults in the system. The model needs to also support parallelism and data priority, as data is coming in to the system from multiple sources and being fed through multiple FIFO:s, with different priority associated with different data.

The requirements have been summarized in Table 1, where some examples have been provided and the effect of the requirement has been evaluated along with where the requirement is targeted at.

3.3. Chosen Tool

The major options for modeling are JavaScript, SystemC and Python. All other tools get pruned out by either being unsuitable for this task or unsupported for this kind of modeling. From these options the JavaScript tool run on Node.js was chosen for its high-level coding language properties and built-in support for being used in web-based applications. Using TypeScript, a superset for JavaScript, we can also achieve strict type checking, easing future development. The Node.js runtime offers a built-in asynchronous high-performance JavaScript engine for fast modeling. It has an event loop which can be used in TLM for accurate and efficient model execution. For plotting the modeling data a Python tool, Bokeh, will be used as a working framework for similar plotting has already been completed in another project.

Possible challenges might appear in building the needed code library for successful modeling, as no such library exists to the best of authors knowledge. Other challenges might appear in debugging the model while it is still in production, as the event loop is not the easiest to keep track of.

Table 1. Model requirements and their effects.

Requirement	Effect	Influences
Support for web applications	Easier integration into existing web framework	Modeling language
Strict types	Simplifies model design	Modeling language
Parallelism	Support for parallel data input from multiple sources	Architecture
Re-usable	Assists future development	Model
Abstract component modules	Assists scalability	Model
Static parameters	Provides changeable system parameters such as clock frequency, assists DSE	Model
Power consumption	Estimates system power consumption, assists in DSE	Model
Throughput	Estimates system throughput, assists in DSE	Model
Latency	Estimates data latency through the link, assists in DSE	Model
Priority handling	Support for prioritized data in FIFO:s	FIFO
Intuitive input and output	Assists future developers in using the model	Input and output
High abstraction for input	Assists crafting different use cases	Input
Plotting for output	Visualizes output, easier interpretation of output data	Output
Built in tracing to modules	Module states at given times known, easier debugging	Every module
Plotting the traces	Visualizes module states at given times, easier interpretation.	Every module

3.4. High-Level Architecture and Design of the Tool

3.4.1. Architecture Overview

The model contains several general modules: source, FIFO, sink, element and plotting. Source is a general module for incoming data, which represents data coming in from other blocks. It feeds data into system at times defined in the parsed version of use case data. General FIFO module mimics the nature of real FIFO memory elements and it is used to handle different priority data coming from different instances of source elements, to control and regulate the data flow through the system. Sink module is an endpoint, where data is accumulated to store in an output file. The general element module is for adding different behavioral properties to the system. It provides the actual functionality of the behavior from the communication interface. Plotting module provides visual output of its input.

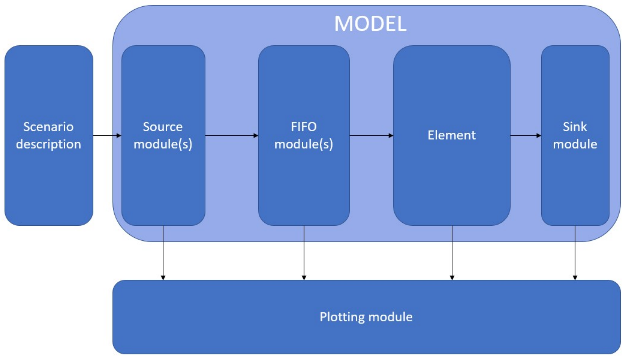

Figure 9. Model description.

3.4.2. System Interfaces

The model will take in a file where events are defined at the systems required accuracy, and system parameters, which define event sizes and different system properties. The model then provides the needed FIFO depth at different use cases, the throughput of the system, and the total overhead in the system. Event input is implemented as 3 files: One for high-level use case specification with time accurate control specification, one for defining control command contents and priority, and one for defining control command sizes and contents. These are then parsed to create an input event list, with the size of control command inputs and the time of the input. Then the input data is processed through FIFO:s and the link, keeping track of link utilization and contents and FIFO contents. These attributes are mapped into output files and throughput is estimated from this data. The output file will be visualized using the Python plotting library Bokeh.

The model may also interact with external C/C++ addons, which may include code intended to be run on the actual application. This interaction takes place inside the general modules, and is used to bring the model closer to the application.

3.5. Submodule Design

The system will have general hardware (HW) blocks comprising of general code modules with details being added when needed. A more specific look into submodule design is offered below.

3.5.1. Source

Whenever a new control command is due to the system the source submodule handles notifying the central event loop of incoming data, which handles that the transaction begins at the correct time in the model. In the setup phase, source submodule is fed a data object list containing all the control commands it will pass into the system as it progresses. While the system runs, the module passes on data objects containing the control commands, but only when it is the correct time to feed the data. Regarding the source submodule, system parameters are the control command sizes passed in to the source.

3.5.2. FIFO

FIFO submodules task is handling incoming data and feeding it forwards in the right order. It also logs its state. This submodule is fed data objects, which it releases into the system when the system is ready to receive data. The data objects have priority, which affects the order in which control commands are fed into the system. System level parameter inside this submodule is the priority handling, which can be modified for different priority algorithms. The submodule produces a suggested FIFO size based on its utilization.

3.5.3. Sink

Sink submodules task is being a clear endpoint for the data flow and logging the data that reaches the end. This submodule is fed data objects and it accumulates them until the model is finished, then it outputs them into a file for easy handling.

3.5.4. Element

The general element submodule handles the actual behavior of the communication interface part of the system, with payload building and other methods describing the interface. Element submodule is fed different data objects, which it might alter and log. Then it forwards a data object with a form described in the setup phase of the system.

3.5.5. Plotting

Plotting module consists of the Bokeh plotting framework. It visualizes the output files produced by other submodules. The plots can be customized and the framework is scalable to even larger data amounts.

4. DISCUSSION

Using JavaScript with the Node.js runtime allows for more flexible modeling, but lacks some needed infrastructure for modeling. Creating a working arbitrator for the transactions proved to be a challenge, but a proof-of-work for event based modeling on JavaScript was successfully created. The system performance was appropriate with a test scenario and the model was able to produce realistic plots in the scenario.

Still some planned features are missing from the proof-of-work model. The logging from each submodule is yet to be implemented and FIFO priority algorithm has some problems. Difficulties with the arbitrator encouraged searching solutions elsewhere, and the model could have most likely been successfully completed with SystemC if it would have proved to be impossible to be successful with JavaScript.

With the proof-of-work JavaScript model we have proven that more detailed models of the company's SoC communication interfaces can be modeled using JavaScript. This can also be extended and modified to cover other similar modeling use cases with the same DSE problems. The plots created by the proof of work model Figure 10 visualize clearly the utilization of different FIFO:s and the link.

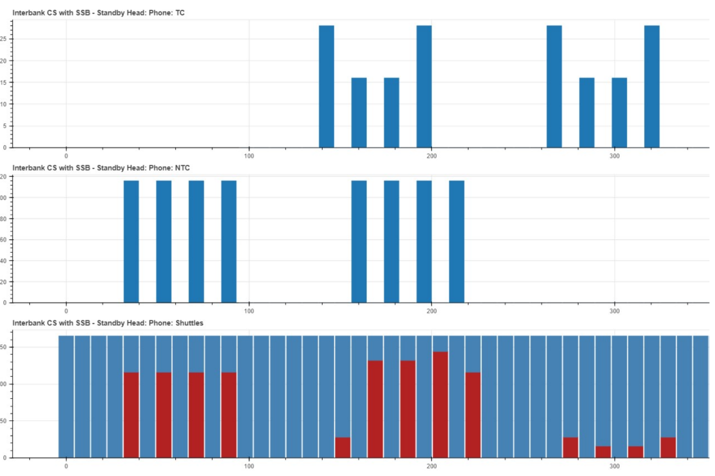

Figure 10. FIFO utilization plots.

5. SUMMARY

This thesis presented the basics of SoC:s and TLM and explored a solution for a DSE problem for a company using TLM. Using TLM reduces the amount of labour required for a realistic model of a communication system, by abstracting the transaction as simple function calls and ignoring the unnecessary details in the system.

The target for this thesis was to study different options and exploring a solution for creating a model to a company's SoC communication interface. The model was implemented with JavaScript and a test scenario was successfully run on the model. Re-usability of the tool in different similar SoC communication interfaces also played a major part in the design, and modeling in the future is moving towards using previous modeling as much as possible and only changing details with minimum modifications.

6. REFERENCES

[1] Grant M., Brian B. & Andrew P. (2007) ESL Design and Verification : A Prescription for Electronic System Level Methodology. The Morgan Kaufmann Series in Systems on Silicon, Morgan Kaufmann.

[2] Black D.C. (2010) Systemc: From the ground up. Springer.

[3] Cai L., Verma S. & Gajski D. (2003), Comparison of specc and systemc languages for system design.

[4] Mishra S., Singh N.K. & Rousseau V. (2016) System on chip interfaces for low power design. Elsevier, Waltham, MA, 410 p. Description based upon print version of record.

[5] Forstner P. (1999), Fifo architecture, functions, and applications.

[6] Da (1984) Chapter 9 data communication interfaces. In: R.R. Smardzewski (ed.) Yes, Data Handling in Science and Technology, vol. 1, Elsevier, pp. 223–246.

[7] Null L. (2015) The essentials of computer organization and architecture, fourth edition. Jones and Bartlett Learning © 2015.

[8] Frenzel L.E. (2016) Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/Output (I/o) Standards. Elsevier.

[9] Rowson J. & Sangiovanni-Vincentelli A. (1997) Interface-based design. Proceedings of the 34th Design Automation Conference .

[10] Systemc. URL: `https://www.accellera.org/community/systemc/about-systemc` About SystemC. Accessed 9.2.2021.

[11] Bailey B. (2015), Is systemc broken? URL: `https://semiengineering.com/is-systemc-broken/`.

[12] (2019) IEEE Standard for VHDL Language Reference Manual. URL: `https://standards.ieee.org/standard/1076-2019.html`.

[13] Sysml. URL: `https://sysml.org/` SysML Open Source Project. Accessed 9.2.2021.

[14] Apvrille L., Muhammad W., Ameur-Boulifa R., Coudert S. & Pacalet R. (2006) A uml-based environment for system design space exploration. In: 2006 13th IEEE International Conference on Electronics, Circuits and Systems, pp. 1272–1275.

[15] (2017), Ieee standard for universal verification methodology language reference manual.

[16] Rich D.I. (2003), The evolution of systemverilog.

[17] Sana S. (2020), The best way to learn systemverilog accelerated verification with uvm. URL: `https://community.cadence.com/cadence_technology_forums/f/functional-verification/46679/the-best-way-to-learn-systemverilog-accelerated-verification-with-uvm-blended-training` UVM. Accessed 16.2.2021.

[18] Jansen R.H., Vane V. & Wolff I.G.d. (2016) TypeScript: Modern JavaScript Development. Packt Publishing.

[19] Javascript. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript` JavaScript. Accessed 9.2.2021.

[20] Python. URL: `https://docs.python.org/` Python. Accessed 16.2.2021.