


Choc: A Communication Toolkit to Help Hardware Module Drivers Navigate SoC Architectures

Mattis Hasler 
Barkhausen Institut
Dresden, Germany

Abstract

Testing may be the most important part of hardware design. Test early, test often, test in every possible stage. The basis for testing a hardware unit is a driver. It abstracts the exact interfacing with the unit and presents the user with an easy-to-use interface. While the driver stays constant, the communication channel between the driver and the unit may change drastically, depending on the scenario. When embedding a unit into a system-on-chip, reaching the unit might need to tunnel the communication through various communication fabrics and/or protocols. This work presents a data flow toolkit that abstracts from the communication channel thus allowing easy modeling of connection interfaces. This means the unit driver can be written early in development and tested in a unit test case setup. The driver stays unmodified throughout the development process until a fabricated chip is tested in the lab. Each communication fabric or bridge also defines and tests its drivers in a unit test environment. The composition of the final communication pipeline is constructed as part of the top-level architecture design by plugging together the building blocks.

ACM Reference Format:

Mattis Hasler . 2025. Choc: A Communication Toolkit to Help Hardware Module Drivers Navigate SoC Architectures. In *Proceedings of 4th Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE '25)*. ACM, New York, NY, USA, 3 pages.

1 Introduction

The development path of an accelerator consists of many steps. Throughout this path, several different testing setups will be used. In the ideal case, the driver used to access the hardware module can stay constant the whole time. In the first phase of development, the Device under Test (DUT) is very close to the driver. It can directly send command words to the DUT, for example through a register interface. Later in development, an accelerator may be included into a bigger system to—as the name suggests—accelerate special-purpose computation. Immediately, the distance between the accelerator and its driver increases. The command words have to be passed through the system’s infrastructure to reach the DUT. As the project matures into a system-on-chip (SoC), the distance between the DUT and the driver keeps increasing. Commands may have to be passed through one or more system buses or network-on-chips. While doing RTL simulation there may be a way to directly

inject signals into the system bus, but at some point (e. g. netlist simulation) that possibility also closes. Later access may only be possible through the SoC (physical) pins using some communication protocol like JTAG or UART for example. Ultimately, when the chip is fabricated and arrives in the lab on a PCB, access to the pins again gets more complicated. Access to the physical JTAG interface from a PC needs a JTAG-to-USB converter and on the controlling PC software, for example, OpenOCD [4]. The commands are transferred over a couple of communication fabrics and have been wrapped in equally many protocols, that the developer of the accelerator should not be forced to care about.

We propose a communication toolkit, named Choc, to easily represent the communication structure of the hardware designs on the test/software side. The communication is modularized, to match the structure in hardware as a digital twin. Each hardware module that implements a piece of communication fabric or a bridge between two pieces also receives a software representation. Users plug the software drivers into a pipeline that matches the hardware structure simply and intuitively. The toolkit enables the development of modularized hardware that is easy to use/test, and reusable.

2 Related Work

Putting test cases into a Python environment and connecting a simulator with the DUT has found its way into the mainstream with cocotb [6]. Also other projects like pyvhdl [1] and cosimtcp [2] present similar solutions. The Universal Verification Methodology (UVM) register abstraction layer (RAL) defines standardized ways of describing register files and tests for them. In [7] UVM RAL is utilized to create reusable verification functions from unit tests to system-level tests. Similar to this [5] shows a way to reuse cocotb-based test cases in different chip design stages. Focusing on the implementation of different bus protocols to get access to the DUT, it does not cover environments up to the lab setup.

3 The Choc Communication Toolkit

The core functionality of Choc does not contain any implementations of protocols like JTAG or UART, but provides the basic mechanics for building a typed dataflow pipeline. Choc is written in Python with asyncio. Like cocotb, Choc uses Python as the language for test cases and for the same reasons: Tests are software and the DUT is hardware and both should be written in a language that is designed for it [6] e. g. Python for software and Verilog for Hardware. In addition, Choc also targets setups that are not based on simulated but real hardware, and an HDL would be out of scope.

Choc is a data-flow toolkit, so the basic building blocks are data processing blocks, called “Items”. Items define sockets, that have a name, a direction (i. e. input or output), and a type, which can be any Python type. Sockets can be hooked up to each other to create

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

LATTE '25, March 30, 2025, Rotterdam, The Netherlands

© 2025 Copyright held by the owner/author(s).

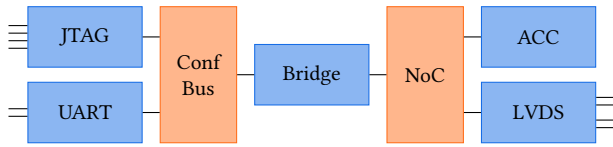


Figure 1: Example SoC Communication Structure. Modules are in blue, fabrics in orange.

data flows between Items. Crafting connections between Items can easily be done using shift operators:

```
sender = RandInt() #sockets: out(int)
transformer = Int2Str() #sockets: in(int), out(str)
sender >> transformer
```

Because sockets are typed, the `>>` and `<<` operators work flawlessly, even with multiple outputs and inputs. The type of a socket is not restricted. It can easily be a class describing a complex request. For example, a `MemoryRequest` will typically contain at least an address, a data word and a mode (read/write). When replicating hardware 1-to-1 any request type would need an additional response type that is transferred over a second socket in the opposite direction. Matching the response to the correct request requires extra effort and is error-prone, especially when multiple parallel requests are allowed and out-of-order processing of requests may be supported.

To overcome this problem Choc defines an awaitable `Request` class, that can be sent over sockets. The receiver of a `Request` can `commit()` a result that unblocks the waiting initiator without sending the result through sockets. It is common to inherit from `Request` to create special-purpose requests like the `MemoryRequest` or a `ConfigRequest` for the config bus. Like hardware modules converting signals from one interface to another, the corresponding driver “Item” can create a request chain. It can convert each incoming request of one type to a request of another type. When the derived request is committed, the result will be back-converted, committing it to the original request. For example, a `MemoryRequest` could be serviced by synthesizing it to one or more `ConfigRequest` to access a memory that is only reachable through the config bus.

4 SiCo: Connecting Simulators

The Item-socket-connection system works well for high-level data structs like integers, strings, or requests. But at some point things have to be converted to hardware signals, to “1” and “0”, maybe even also “Z” and “X” and a few more. This is where the SiCo (Simulation Controller) module comes into play. Using the standard DPI and VPI simulator interfaces, it allows the connection of a Choc environment to any simulator supporting one of the interfaces. It provides Choc Items and corresponding SystemVerilog modules to stream hardware signals both into and out of simulators. Signals may be based on simulation time, clock cycles, or requests. For request signals the Verilog module follows a simple ready-valid interface to stream signals to/from the DUT.

5 Application

Choc and SiCo has been developed and refined over the course of three MPSoC tape-outs, of which one is published [3], one is currently measured in the lab, and one is being fabricated. It is intended

to simplify the software stack to operate a chip from the module design phase until testing the fabricated chips in the lab. The MPSoCs share a common architecture, making the communication structure similar, allowing the reuse of many communication modules. The following will describe the communication pipeline for a custom Accelerator (ACC) to showcase Choc. The ACC is controlled using a driver class that accesses the hardware using `RegisterRequests` to read and write registers. Originally it was planned that access to the chip is mainly done over a FPGA evaluation board through the chip’s LVDS link. Unfortunately, the LVDS-FPGA link proved to be unusable, so that all communication had to be funneled through the chip’s config bus. As displayed in Fig. 1, the ACC can be configured using the JTAG controller, going through the config bus, the NoC bridge, the NoC, and the NoC interface unit of the ACC. To create a communication pipeline in Choc that follows this path, we backtrack the connections and replace them by appropriate drivers:

```
acc = AccDriver()
nocif = nocIf(noc_addr=NOC_ADDR_ACC)
bridge = ConfToNoc(bus_addr=BUS_ADDR_BRIDGE)
jtag = JTAGCtrl()
acc >> nocif >> bridge >> jtag
acc.start()
```

With this pipeline a `RegisterRequest` from the `AccDriver` is first converted to a `NocRequest` by the `nocIf`. The `ConfToNoc` item runs an independent asynchronous task to emit `ConfRequests` to drive the bridging hardware. That includes polling for new messages because the bridge is not a config bus master and cannot forward received NoC messages into the config bus. The `ConfRequests` are forwarded to the `JTAGCtrl` driver that generates the appropriate JTAG commands. In our lab setup, JTAG commands are processed by an `OOCDriver` that is connected to a running `OpenOCD` instance. `OpenOCD` drives the USB-JTAG dongle that produces the signals on the PCB.

6 Summary

Choc is a versatile communication toolkit that eases the process of building communication pipelines for module/accelerator access at any stage of an SoC project. In a Choc-enabled project, each hardware module also defines a Choc “Item” that acts as a driver for the software/test part of an SoC development project. Items are independent data-flow processing units that can be connected to create pipelines. Using pipelines wraps the communication protocols needed to connect to the DUT from the driver. SiCo is a part of Choc and is the bridge between the controlling software and a hardware design, simulated by any simulator software that supports a VPI or DPI interface. Choc and SiCo are open source as part of the “bilib”, the Barkhausen Instituts hardware building block library¹.

Acknowledgments

The project on which this report is based was funded by the German Federal Ministry of Education and Research under grant number 16ME0527. The author is responsible for the content of this publication.

¹<https://github.com/Barkhausen-Institut/bilib>

References

- [1] Alfredo Benso, Stefano Di Carlo, Paolo Prinetto, and Y. Zorian. 2008. IEEE Standard 1500 Compliance Verification for Embedded Cores. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16(4) (05 2008), 397 – 407. <https://doi.org/10.1109/TVLSI.2008.917412>
- [2] Lukasz Butkowski and Koray Karakurt. 2019. CO-SIMULATION OF HDL USING PYTHON AND MATLAB OVER TCL TCP/IP SOCKET IN XILINX VIVADO AND MODELSIM TOOLS.
- [3] Sebastian Haas, Christopher Dunkel, Friedrich Pauls, Mattis Hasler, and Yogesh Verma. 2024. Trustworthy Silicon: An MPSoC for a Secure Operating System. In *2024 IEEE Nordic Circuits and Systems Conference (NorCAS)*. 1–7. <https://doi.org/10.1109/NorCAS64408.2024.10752473>
- [4] Hubert Högl and Dominic Rath. 2006. Open on-chip debugger–openocd-. *Fakultat für Informatik, Tech. Rep* (2006).
- [5] Marcin Ludwiniak and Arkadiusz W. Luczyk. 2024. Using Cocotb Framework During Different Stages of IC Design. In *2024 31st International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*. 154–157. <https://doi.org/10.23919/MIXDES62605.2024.10613994>
- [6] Benjamin John Rosser. 2018. Cocotb: a Python-based digital logic verification framework. In *Micro-electronics Section seminar. CERN, Geneva, Switzerland*.
- [7] Tudor Timisescu and Uwe Simm. 2015. Leveraging the UVM Register Abstraction Layer for Memory Sub-System Verification. *Proceedings of DVCon (2015)*.