ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA023

CO-SIMULATION OF HDL USING PYTHON AND MATLAB OVER Tcl TCP/IP SOCKET IN XILINX VIVADO AND MODELSIM TOOLS

Ł. Butkowski*, B. Dursun, C. Gumus, M. K. Karakurt
Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

This paper presents the solution, which helps in the simulation and verification of the implementation of the Digital Signal Processing (DSP) algorithms written in hardware description language (HDL). Many vendor tools such as Xilinx ISE/Vivado or Mentor Graphics ModelSim are using Tcl as an application programming interface. The main idea of the co-simulation is to use the Tcl TCP/IP socket, which is Tcl build in feature, as the interface to the simulation tool. Over this interface the simulation is driven by the external tool. The stimulus vectors as well as the model and verification are implemented in Python or MATLAB and the data with simulator is exchanged over dedicated protocol. The tool, which was called cosimtcp, was developed in Deutsches Elektronen-Synchrotron (DESY). The tool is a set of scripts that provide a set of functions. This tool has been successfully used to verify many DSP algorithms implemented in the FPGA chips of the Low Level Radio Frequency (LLRF) and synchronization systems of the European X-Ray Free Electron Laser (E-XFEL) accelerator. Cosimtcp is an open source available tool.

INTRODUCTION

The correct operation of the Low Level Radio Frequency (LLRF) [1] and synchronization systems of the European X-Ray Free Electron Laser (E-XFEL) [2] accelerator requires a huge amount of Digital Signal Processing (DSP) calculations in real time. This task is mainly handled by FPGAs. In the high level design stage, the first step is to determine the exact algorithms that has to be implemented in FPGAs so the machine can operate. This work is executed with the help of tools such as MATLAB with Simulink or Python. Solutions are simulated and the exact formula is derived. Next the implementation process starts in which the code written in hardware description language (HDL) is developed. During this process the following problem always appear: how to verify functionality of written HDL code.

There are a few ways we can solve this problem. There are available methodologies and libraries which can help go trough this process. It is very common to write the testbench using HDL first. Next phase is to write the model of the component under test in order to go through the verification process. However, writing the DSP algorithm model in HDL is really difficult and time consuming. Additionally there is risk of introducing new errors in the model, which fail our verification. Most wanted in the verification process would be to use directly the same tools as the one in the high level design in a co-simulation process. This can improve significantly verification [3].

There are already available methods of HDL co-simulation with the use of high level programming languages. One of the way to use high level programming languages like C in a verification process is to use Direct Programming Language Interface (DPI) of the System Verilog [4] or Foreign Language Interface (FLI) [5] of the simulation tool. The simulation is fast but this method is not so straight forward and easy to use. It is platform depended or tool depended and requires recompilation of the code every change. One of the another way is to use Programming Language Interface (PLI) like the file input/output interface using the VHDL textio library [6]. This method has limitation in interactive simulations and reuseability.

We came with and another idea. Almost all of the HDL simulator tools support Tcl script language as an application programming interface, enabling control of the simulation using TCL commands. It also means that all the default Tcl features are available in the tool. The idea proposed was to use these features as the communication layer between HDL simulation tool and an external tool. The Tcl socket function has been found as a perfect candidate for this role. It is a build in command which opens a TCP network connection. The communication between simulation tool and high level language is done over TCP/IP socket. This is a good separation between these two. The solution gives a good balance between reuseability, easy to use and simulation speed. The TCP/IP socket protocol is well-know, widely used and has a build-in support in many tools.

The solution we came up with has been called cosimtcp [7]. It is a set of script in the form of libraries, which can be easily added to current or new simulation flow. Currently the solution is used with Xilinx Vivado, ModelSim, Matlab and Python tools.

In the following chapters the steps that has to be done to run simulation using cosimtcp libraries will be presented. Also the examples of the usage will be given. In the end prons and cons of this solution will be discussed.

CO-SIMULATION ARCHITECTURE

The block diagram of the co-simulation is presented in Figure 1. The simulation is divided into two main blocks. The server side: responsible for the HDL simulation of Unit Under Test (UUT) and the client side: responsible for the generation of stimulus and verification. Between them the data are exchanged using a dedicated protocol over a socket connection. There is one protocol used among all the tools.

* lukasz.butkowski@desy.de

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA023

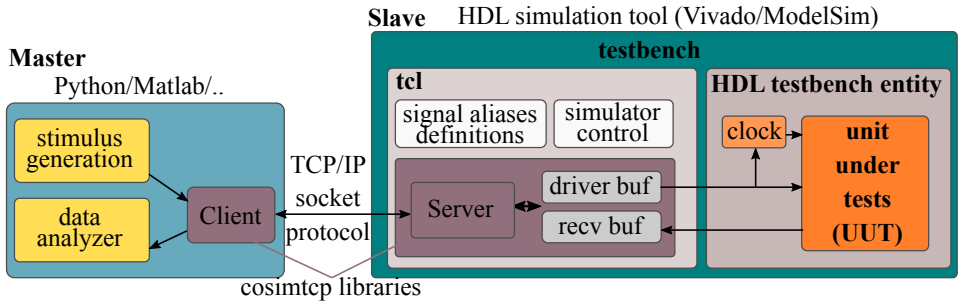

Figure 1: VHDL Co-Simulation block diagram using cosimtcp.

The server is a slave that executes the instructions of the master, which is the client.

Protocol

The communication protocol between the client and the server is based on string commands, which are send over established TCP/IP socket connection. There are a few main commands:

- **set <object name> <buffer offset> <data vector>**
  client request: fill the input buffer with the data vector of the specified object starting with the given buffer offset
  server response: buffer fill done
- **get <object name> <buffer offset>**
  client request: send the data from the output buffer of the specified object
  server response: returns data vector from the buffer
- **sim run <steps> <step time> <time unit> <mode>**
  client request: run simulation in the number of steps with the specified time step,
  simulation can be run in various modes: do not record data in buffers, start from the beginning, continue
  server response: simulation steps done
- **cmd <command>**
  client request: execute system command like restart or quit simulation
  server response: command executed

Server

The server side is implemented on the HDL simulator tool. The tool executes a Tcl script in which the server function is run. The main task of the server is to:

- listen for and accept incoming connections on socket, server can accept only one client at a time
- translation of incoming commands into executable functions of HDL simulator
- run and control simulation
- send back requested data

The flow diagram of the server operation is presented in Figure 2. After accepting connection, the server goes to the idle state where it waits for incoming commands. The server has two buffers. The first one is used to store the value of the input objects. This buffer is filled with data by the client. The second one is used to store the value of the output objects. This buffer is filled with the values during the simulation. The simulation is executed in steps. In each step, the values from the input buffer are set to input objects, and the values of output objects are saved in output buffers. Next the pointer of the data buffers is incremented and the simulation run for one step. Only the values of the objects defined in the list are set or examined. The simulation is controlled with the Tcl commands available in the tool. In the case of ModelSim these are: *force* to set object value, *examine* to read object value and *run* to run simulation step. In the case of Vivado they are respectively: *add_force*, *get_value*, *run*.

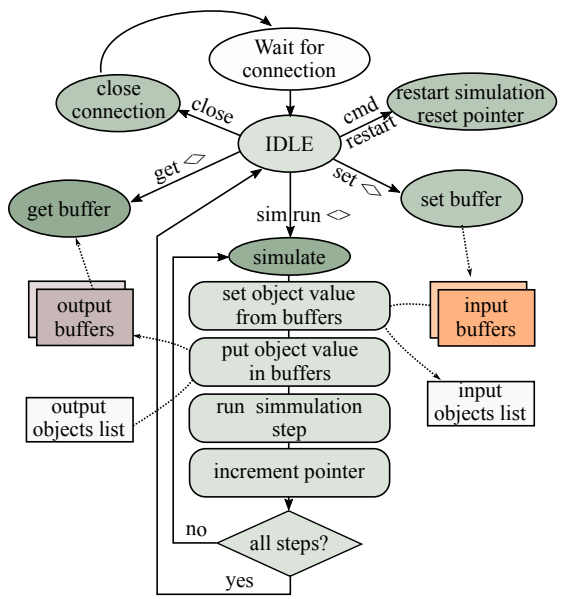

Figure 2: Server side flow diagram.

Client

Cosimtcp client run on MATLAB or Python using simulation scripts which can be added as a library. The main responsibilities of the client are:

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA023

- open and keep connection with the server
- translation of simulation data into the defined communication protocol
- control the data flow

DESIGN FLOW

The preparation for the verification of UUT starts with writing a HDL testbench entity with the instance of the UUT and a simple clock process in minimum. Next there are the following steps to be done on the slave side:

- write source compilation steps for the HDL simulator – this is a common to all simulation strategies, HDL project sources and libraries has to be added to the project and next compiled. Makefile or tcl do file can be used
- create Tcl .do file if not done in the first step
- add cosimtcp libraries path to Tcl .do file
- define inputs and outputs of UUT by creating aliases to the paths of HDL objects in Tcl .do file
- run the server, can be done in Tcl .do file or from the simulator console

The example Tcl code of slave side is presented in Figure 3.

```
# ...
# source compilation script
# ...
global input
global output
# list input objects
# input(<alias_name>,path) <object path>
set  input(A,path)       UUT\/pi_data_a
set  input(B,path)       UUT\/pi_data_b
set  input(VALID,path)   UUT\/pi_valid
# list output objects
# output(<alias_name>,path) <object path>
set  output(C,path)      UUT\/po_data_c
set  output(VALID,path)  UUT\/po_valid
# add cosimtcp library
source ../cosimtcp/server/Server.tcl
# start listning at port 1234
cosimtcpServer 1234
```

Figure 3: Example Tcl code of the slave side for the 3-input and 2-output UUT.

While the slave side awaits for the connection and instructions there are the following steps to be done on the master side:

- create cosimtcp object which will connect to the slave side
- create stimulus and send to the master side
- run simulation steps
- request output data from the slave
- verify result
- run next step or close simulation

The example Matlab code of the master side is presented in Figure 4.

```
% add library path
addpath('../cosimtcp/client/matlab')
% create cosimtcp object and connect to slave
cosim = cosimtcp('localhost',1234,1000000,60);
% create stimulus
dataA = (1:1:300) * (-1);
dataB = (1:1:300) * 2;
valid_in = ones(1,300);
% fill the data buffers
cosim.send_data('A', dataA);
cosim.send_data('B', dataB);
cosim.send_data('VALID', valid_in);
% run simulation with the buffered data and record result
% 300 clock cycles of 10ns
cosim.run_sim(300, 10, 'ns');
% get result from the data buffers
dataC = cosim.get_data('C');
valid_out = cosim.get_data('VALID');
% close simulator
cosim.quit()
```

Figure 4: Example Matlab code of the slave side simulating UUT with the 3 inputs and 2 outputs.

CASE STUDY: STATE SPACE CONTROLLER

The solution presented in this paper has been used to validate the design of a state space controller used in one of the synchronization subsystems. Stable operation of the controller was tested under different conditions. There was a hundreds of iteration runs, each with a different coefficients set. The result of the one of the iterations is presented in Figure 5.

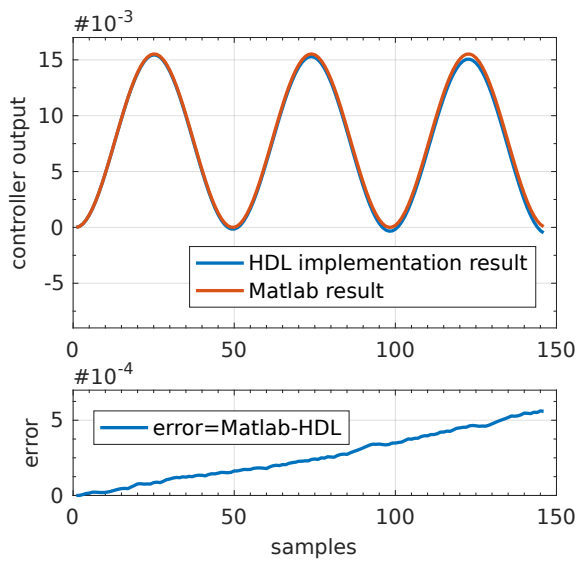

Figure 5: Matlab result of the simulation of the state space controller for one coefficients set.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA023

CASE STUDY: RF FIELD DETECTION

Another case in which described co-simulation method has been used is verification of the amplitude and phase detection of the RF signal. The component has been driven with a long data trains in a range of 60000-200000 of the ADC samples. The result of this simulation visible in ModelSim is presented in Figure 6.

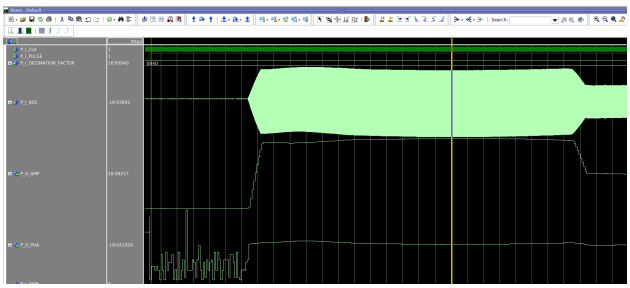

Figure 6: ModelSim window during field detection algorithm simulation, presents the ADC data input and the outputs with the amplitude and phase values of the signal

RESULTS

The co-simulation technique described in this paper has been used to test and verify DSP functionality of many HDL components. Simulation tools, such as Xilinx Vivado and ModelSim, were used alternatively, as was the case with Matlab and Python on the client side. All combinations of the tools are evaluated and the simulation time is observed to be comparable. The best performance is achieved when the HDL simulators run in batch mode. The best result is achieved when tools are run in a batch mode. The simulation time is around 20 times slower when Graphical User Interface (GUI) is opened.

The simulation times for the case studies given above are presented in Table 1 and Table 2. Tests were performed on the machine equipped with the Intel i5 2500K CPU type and 24GB of RAM memory.

Table 1: State Space Controller Simulation Results with ModelSim and Matlab, Components Written in VHDL Code

Resources	DSP:15, LUTs:3085, Reg:4177
Objects	40 inputs, 2 outputs
Iterations	100
Steps	512
Total Time	5.2 min

CONCLUSION AND OUTLOOK

This paper describes a concept of functional co-simulation of the HDL with the use of TCP/IP socket function available in Tcl. The basic idea and architecture of this solution has been given. It has been shown that cosimtcp can be successfully used to verify DSP algorithms written in HDL. Its simple syntax enables rapid simulation preparation. Simulation time using this solution is within the acceptable range. This method can be used with a various types of HDL languages as it does not depend on the HDL language.

Table 2: Field Detection Simulation Results with ModelSim and Matlab, Components Written in VHDL Code

Resources	DSP:8, LUTs:767, Reg:665
Objects	2 inputs, 3 outputs
Iterations	1
Steps	60000
Total Time	3.5 min

At the beginning, Xilinx ISE HDL simulator was successfully usedfor the conceptual evaluation of the first implementation of cosimtcp. Later we focused on Xilinx ISE/Vivado, Mentor Graphics ModelSim/Questa, Mathworks MATLAB/Simulink and Python due to the intensive usage of these tools within our group. Therefore we believe this solution could be adapt to any other tool that supports Tcl.

REFERENCES

[1] J. Branlard *et al.*, "The European XFEL LLRF System," in *Proc. IPAC'12*, New Orleans, LA, USA, May 2012, paper MOOAC01, pp. 55–57.

[2] "The European X-Ray Free Electron Laser Technical Design Report," http://xfel.desy.de

[3] M. N. Wageeh, A. M. Wahba, A. M. Salem and M. A. Sheirah, "FPGA based accelerator for functional simulation," *2004 IEEE International Symposium on Circuits and Systems* (ISCAS), Vancouver, BC, September 2004. doi:10.1109/ISCAS.2004.1329526

[4] S. Jain, P. Govani, K. B. Poddar, A. K. Lal and R. M. Parmar, "Functional verification of DSP based on-board VLSI designs," *2016 International Conference on VLSI Systems, Architectures, Technology and Applications* (VLSI-SATA), Bangalore, India, October 2016, pp. 1-4. doi:10.1109/VLSI-SATA.2016.7593030

[5] U. Hatnik and S. Altmann, "Using ModelSim, Matlab/Simulink and NS for Simulation of Distributed Systems," in *Parallel Computing in Electrical Engineering, 2004. International Conference on*, Parallel Computing in Electrical Engineering (PARLEC), Dresden, Germany, 2004, pp. 114-119. doi:10.1109/PCEE.2004.74

[6] N. Canellas and J. M. Moreno, "Speeding up hardware prototyping by incremental simulation/emulation," in *Proceedings 11th International Workshop on Rapid System Prototyping. RSP 2000. Shortening the Path from Specification to Prototype (Cat. No.PR00668)*, 11th International Workshop on Rapid System Prototyping, Paris, France, 2000, pp. 98-102. doi:10.1109/IWRSP.2000.855203

[7] Co-Simulation cosimtcp repository on GitHub, https://github.com/mskfw/cosimtcp