2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Unified Firmware Debug throughout SoC Development Lifecycle

Dimitri Ciaglia, ams-OSRAM, Pavia, Italy (*dimitri.ciaglia@ams-osram.com*)

Thomas Winkler, ams-OSRAM, Premstaetten, Austria (*thomas.winkler@ams-osram.com*)

Jurica Kundrata, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia (*jurica.kundrata@fer.hr*)

***Abstract*— The consumer and automotive market segments are driving an increasing demand of sensing and visualization systems. Advanced solutions are fostering the requirement to make these systems smart by adding several components to the core device. Achieving the best integration of sensors and related components is key for ams-OSRAM. With these additional features, the sensor system becomes more complex constituting effectively a system-on-chip with microcontroller/microprocessor, memories, and peripherals. The presence of a microprocessor, such as an ARM Cortex-M device for deeply embedded applications, introduces additional effort of designing the firmware and the co-verification environment for testing the firmware. In ams-OSRAM industry, standard debug probes and software development tools and debuggers are used for both FPGA and silicon-based software development and testing. In this paper, we propose a method to extend this tool flow also to pre-FPGA software development and hardware/software co-verification activities such that FW debug can start as soon as the initial RTL and preliminary firmware are available. Additionally, we present a case of study in which the hardware/software co-verification is performed with the aim to demonstrate the re-usability of the co-verification environment for both FPGA prototyping and silicon bring-up. Finally, the actual implementation and results obtained with this methodology are reported, paving the path for future work and improvements.**

Keywords—Co-verification; on-chip debug; Cortex-M; automotive; firmware; DPI; DOTT;

I. INTRODUCTION

Nowadays the complexity of the designs makes nearly impossible to obtain a bug-free design in the first attempt. This poses a challenge to the pre-silicon verification stage, which needs to ensure error-free designs while taking into account the stringent time-to-market. The pre-silicon verification must ensure that the system does not have hardware or software bugs. For this reason, it is of utmost importance to provide an environment in which hardware and software co-verification can happen as early as possible during the design phase.

Traditional software development and debugging tools are bound to the availability of an FPGA-based prototype of the target design. To facilitate early firmware development and firmware-based IC validation, it is common practice to simulate the IC together with firmware and use System Verilog test benches to provide stimuli to, e.g., the peripherals of the IC as well as to monitor device state defined by firmware actions. With this methodology, the FPGA-based prototypes or silicon cannot directly re-use these test benches. This leads to an additional implementation effort to port the tests into the established software test flows.

To summarize, the state of the art has two major shortcomings as shown in Figure 1. First, there is a considerable time gap between the availability of an IC design in simulation versus its availability in FPGA or silicon, which use the standard software development and debug tools. Second, there is a tool boundary because System Verilog test benches, developed for early simulation-based firmware development and hardware/software co-verification, cannot be directly re-used in the tool flows used for FPGA or silicon-based firmware development and testing.

In this work, we propose a methodology based on the On-chip debugging in conjunction with System Verilog DPI for firmware development, testing and debugging. The methodology is adopted in the case of study in which the software porting and testing done in the hardware/software co-verification environment can be directly re-used for FPGA prototype and silicon bring-up. In addition to that, we introduce the ams Debugger-based On-Target Testing (DOTT), a framework with the goal to simplify the on-target testing of firmware developed for ARM Cortex-M microcontrollers [1].

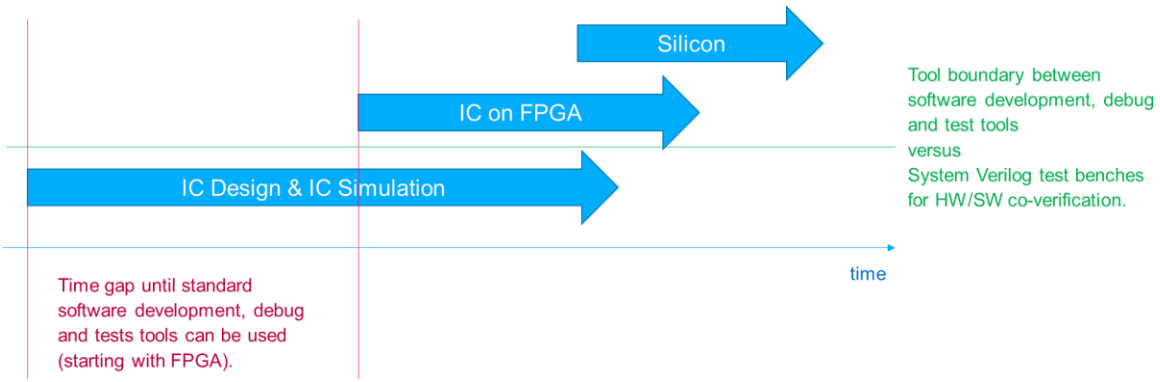

Figure 1 State of the art limitations for software developments and debug tools

II. RELATED WORK

Different techniques exist to verify the hardware design and the related software concurrently [2]. Each technique have advantages and disadvantages as shown in Table 1. In this paper, we will focus on the software debugging using the HW/SW co-verification technique and the on-chip debug methodology. The typical co-verification environment, as shown in Figure 2, consists of the hardware (processor, system interface, and peripherals), the hardware simulator or engine, the debug interface, and the software framework where the software and the debugger are sitting.

Table 1 Comparison of co-verification techniques [2]

Technique	Speed	Timing	Software	Debug	Cost
Instruction set simulation	Medium	No	C Model	Algorithm	Low
HW/SW co-verification	Slow	Yes	Real	HW/SW	High
Rapid prototype	Fast	Yes	Real	Low	Medium
Emulation	Very fast	Yes	Real	Low	Very high

The SW interacts directly with the HW, translating the SW calls into test vectors to stimulate the hardware simulator. Therefore, the HW simulator responds and outputs the results back to the SW environment. Full observability of both the inputs/output and the internal signals is guaranteed by the HW simulator which is usually shipped with a waveform viewer. It is important to notice that the debug interface requires the hardware to have dedicated debug logic (usually called design for debug or DFD). The standard debug interface for ARM processors can be either JTAG [3] or Serial Wire Debug (SWD) [4], which allow for serial low pin-count debug connections.

An approach similar to what this paper discusses is used in OpenTitan [5], an open source silicon Root of Trust project. In this project the co-verification environment uses the open source on-chip debugging also called OpenOCD [6] in conjunction with the System Verilog Direct Programming Interface (SV-DPI) [7] for the JTAG protocol.

While the OpenTitan approach may be directly re-used, an alternative approach based on the SWD debug protocol seemed more appropriate for ARM-based SOC. Finally, we have analyzed the benefits and implemented an I2C-DPI (not used in OpenTitan) for direct command interface with the peripherals.

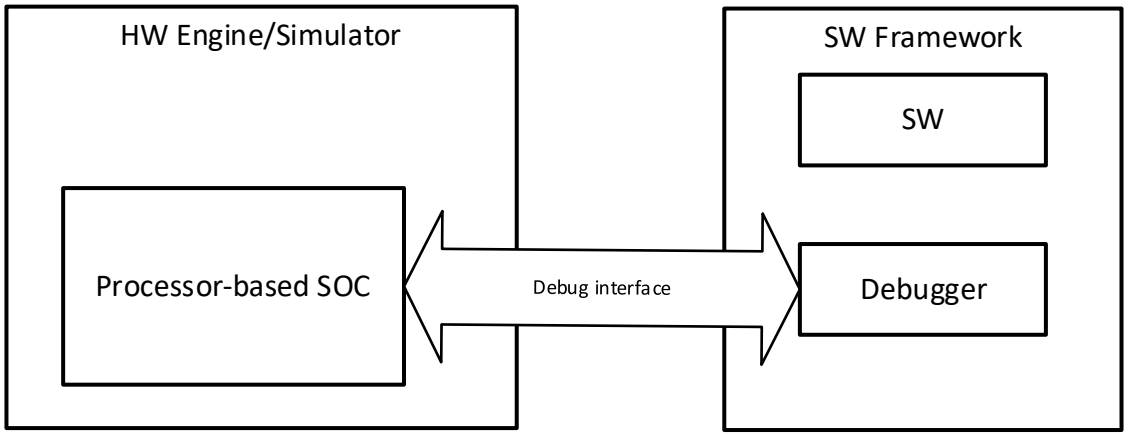

Figure 2 Typical hardware/software co-verification environment

III. PROPOASED APPROACH

Figure 3 shows the co-verification environment proposed in this paper. The architecture consists of: (1) the System-on-chip under test, (2) OpenOCD and SWD SV-DPI for firmware debugging, (3) I2C SV-DPI to provide stimuli to the IC, (4) the SW framework for tests development. With this infrastructure the simulation environment can be used to verify the hardware functionality while the firmware can be tested using the standard software tool flow. The major benefit is that this approach closes the tip gap for the involvement of firmware in the development process. In addition, established software toolchain can be used earlier in the project, allowing to discover and to fix bugs early on reducing the risk of going into the ECO theatre.

This system implementation uses the custom tool DOTT (publicly available on GitHub) for firmware module and integration tests. DOTT enables developers to implement test for firmware executed on-target. In contrast to other approaches, the firmware is not modified for the sake of testing (i.e., tests are not compiled into the firmware), but the firmware is exactly in the same form it would be shipped with the device. Additionally, DOTT leverages the debug probe and the GDB debugger [8] to call on-target firmware functions for unit and integration testing. Test implementation itself is purely on the host system in Python. With OpenOCD and the SWD DPI interface, these tests can now also be executed against the IC running in the hardware simulation engine. The I2C DPI component (similar as SPI DPI or GPIO DPI blocks) is used by the Python-based tests to provide external stimuli to the simulated IC.

Once an FPGA or IC silicon is available, as shown in Figure 4, a standard debug probe (e.g., Segger's J-LINK) will replace both OpenOCD and SWD DPI and the same test suite is used, achieving the goal of this approach to maximize re-usability of the developed tests across the different verification and validation phases. Likewise, an off-the-shelf USB to I2C Bridge will replace the I2C DPI component.

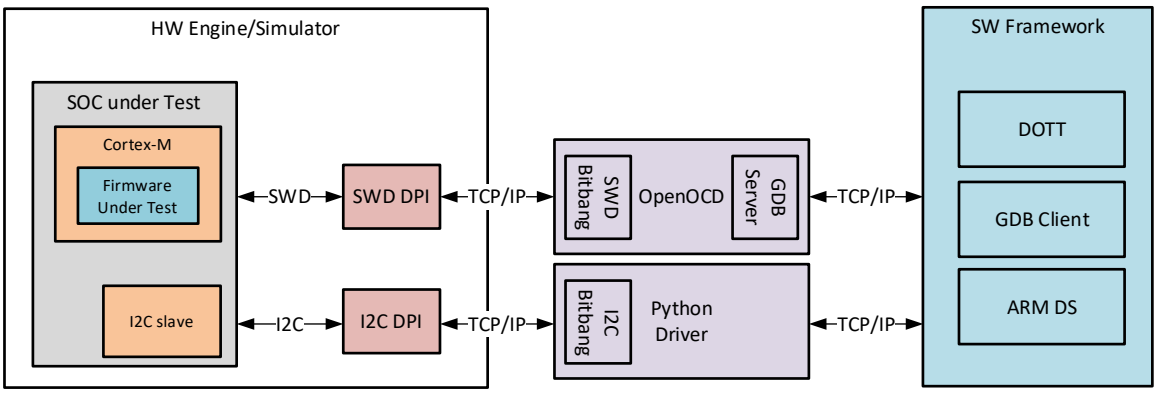

Figure 3 Architecture of the proposed co-verification environment

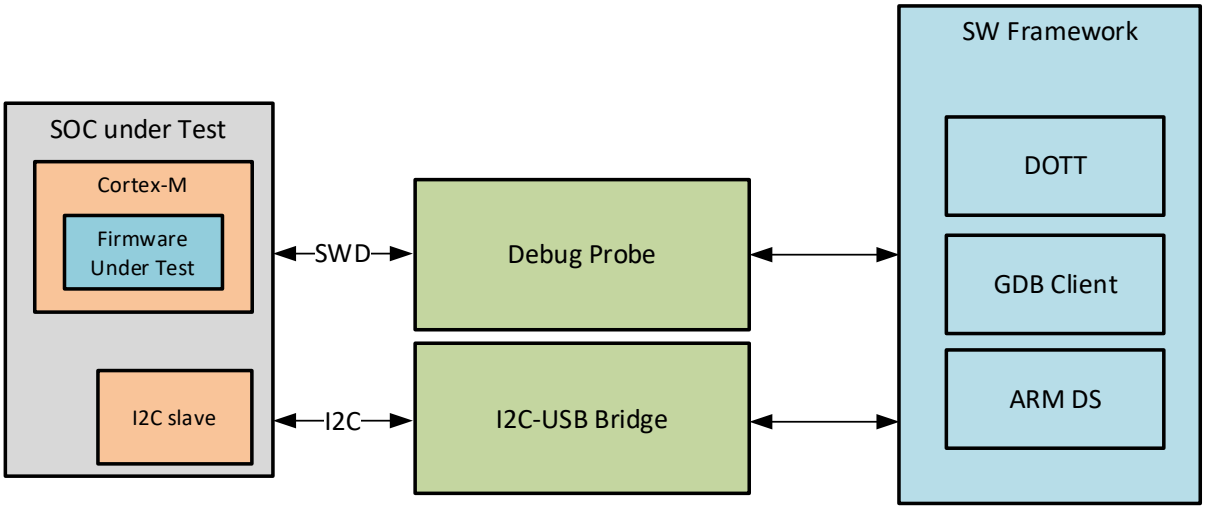

Figure 4 FPGA-based or post-silicon debug environment

IV. IMPLEMENTATION OF THE DPI-BASED SOC SIMULATOR

A. Simulation environment

As already shown in Figure 3, the simulation environment consists of a System Verilog test-bench that instantiates the System-on-Chip under Test and the Direct Programming Interfaces (DPI) connected to the SoC under Test. The SOC under Test is based on the ARM Cortex-M processor, which features the Serial Wire Debug interface used to debug the processor core. Additionally, the SOC instantiates the Inter-Integrated Circuit (I2C) communication interface. Both the SWD and the I2C interfaces are exposed outside the test-bench simulation via the corresponding DPIs. The SWD DPI represents an SWD debugger probe controlled via TCP port using a bitbang protocol. Specifically, the SWD DPI is controlled using the open-source On-Chip Debugger (OpenOCD) tool. The I2C DPI represents an I2C master core controlled via TCP port using a bitbang protocol and a Python driver module.

B. Implementation of the SWD-DPI

The SWD-DPI enables communication via the SWD interface of the HW simulator and the external software. This communication is achieved using a TCP server. This implementation of the SWD DPI is based on the JTAG DPI implementation used in the OpenTitan project [9]. The JTAG DPI implementation uses a remote bitbang protocol, to maximize re-use the SWD DPI implementation also uses a remote bitbang protocol based on the commands and their descriptions as shown in Table 2.

Table 2 Commands and their descriptions used in the SWD DPI remote bitbang protocol.

Command	Description
O	Set the SWDIO line to output mode
o	Set the SWDIO line to input mode
c	Read the SWDIO line
d	Set SWCLK = 0 and SWDIO = 0
e	Set SWCLK = 0 and SWDIO = 1
f	Set SWCLK = 1 and SWDIO = 0
g	Set SWCLK = 1 and SWDIO = 1

The source code of the DPI implementation consists of a System Verilog module and a C-language functions definitions. The System Verilog module defines the digital module ports and uses the imported C function calls. The C-language functions implement the TCP server and the functions used by the System Verilog module.

C. CONFIGURATION OF THE OPENOCD TOOL

The SWD remote bitbang protocol is currently not used in the main line OpenOCD tool distribution. This SWD remote bitbang protocol is implemented in the proposed change number 6044 [10]. Consequently, the OpenOCD tool used in this testing framework is compiled using this proposed patch.

The OpenOCD tool compiled with the SWD remote bitbang protocol patch allows for selection of either JTAG or SWD as the transport type. This is reflected in the configuration file used to configure the OpenOCD tool when used in this testing framework. Figure 5 shows the configuration of the OpenOCD tool used to connect to the SWD DPI. The first part of the configuration file defines the interface which uses the remote bitbang driver on the TCP server (localhost:44853) and the SWD transport type. The second part of the configuration file defines the debugging target, i.e., the Cortex-M processor core in the SOC under Test.

```
1 adapter driver remote_bitbang
2 remote_bitbang_host localhost
3 remote_bitbang_port 44853
4 transport select swd
5
6 swd newdap cortex_m0p_sim cpu -dp-id 0x0BC11477
7 dap create cortex_m0p_sim.dap -chain-position cortex_m0p_sim.cpu
8 target create cortex_m0p_sim cortex_m -endian little -dap cortex_m0p_sim.dap
```

Figure 5 Configuration file of the OpenOCD tool used to connect to the SWD DPI.

D. IMPLEMENTATION OF THE I2C DPI

The I2C DPI implementation uses a remote bitbang protocol just like the SWD DPI implementation. The I2C DPI remote bitbang protocol is based on the commands and their descriptions as shown in Table 3.

Table 3 Commands and their descriptions used in the I2C DPI remote bitbang protocol.

Command	Description
O	Set the SDA line to output mode
o	Set the SDA line to input mode
c	Read the SDA line
d	Set SCL = 0 and SDA = 0
e	Set SCL = 0 and SDA = 1
f	Set SCL = 1 and SDA = 0
g	Set SCL = 1 and SDA = 1

E. The I2C DPI remote bitbang driver

The I2C DPI remote bitbang driver is implemented in the Python language due to its ease of use and scripting capabilities, which are very useful in creating testing sequences. Figure 6 shows the organization of the I2C DPI remote bitbang driver module. Starting from the bottom, the first group of module functions are used to initialize and close the TCP connection to the I2C DPI TCP server. The second group of modules implement the remote bitbang protocol, which is used to read and determine the direction of the SDA line, as well as write to the SCL line. Then, the third group of module functions implement the atomic steps in I2C communication, e.g., the START and STOP conditions, acknowledgments from/to slave, sending the I2C address, requesting read/write access, reading/writing data. These module functions are used to implement singular writes and reads from the I2C DPI to the I2C slave interface on the tested System-on-Chip. Finally, the last module function implements the I2C test sequence.

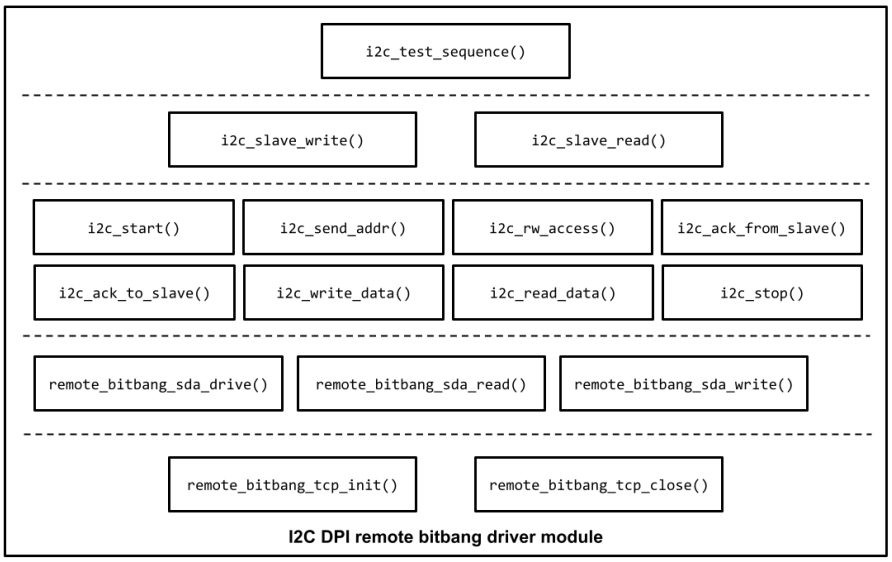

Figure 6 Organization of the I2C DPI remote bitbang driver module

F. Compilation of the DPI library

This DPI-based System-on-Chip simulation environment uses two separate DPIs - the SWD and the I2C DPI. The simulator is provided with a shared object library to make the required DPI function calls. This implementation uses a single shared object library to provide the DPI functions. Figure 7 shows the file organization of the SWD/I2C DPI compilation. The SWD and the I2C specific functions are defined in the swddpi.c and i2cdpi.c files. These definitions make use of the System Verilog Direct Programming Interface header file and they use the TCP server functions defined in tcp_server.c/.h. The SWD and the I2C DPI, as well as the TCP server functions are compiled as Position Independent Code. Finally, they are linked into a shared object library - libdpi.so, using the script shown in Figure 8.

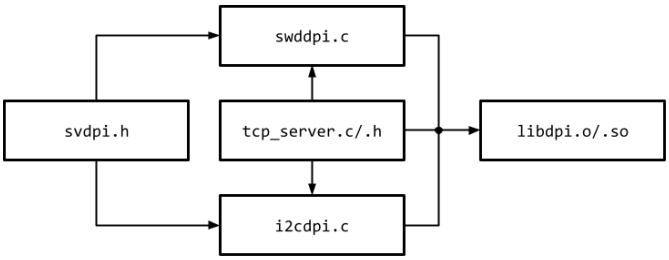

Figure 7 File organization of the SWD/I2C DPI compilation.

```
1 #!/bin/sh
2 g++ -c -fPIC tcp_server.c -o libdpi.o -include swddpi.c -include i2cdpi.c
3 g++ -shared -Wl,-soname,libdpi.so -o libdpi.so libdpi.o
```

Figure 8 Script used to compile and link the SWD/I2C DPI into the shared DPI library

G. Testing results

The DPI-based System-on-Chip simulation environment is tested on an example System-on-Chip that includes an ARM Cortex-M processor core with an SWD debugging port and an I2C communication interface. The firmware of the processor core is preloaded and consists of processing the data values received via I2C interface, that are then read back by the I2C master. The test bench simulation is run in the Xcelium simulator. The developed I2C DPI driver module is used to execute a test sequence, which consists of two I2C slave writes, followed by two I2C reads. The SWD remote bitbang-patched OpenOCD tool is used to connect to the SWD DPI. Telnet [11] is used to connect to the OpenOCD tool server. The test sequence used on SWD DPI consists of halting the Cortex-M

processor core and then reading out the core registers. Figure 9 shows an example run of the DPI-based SoC simulation environment.

```
xcelium> run

swd: Virtual SWD interface swd0 is listening on port 44853. Use OpenOCD and the following configuration to connect:
  interface remote_bitbang
  remote_bitbang_host localhost
  remote_bitbang_port 44853

I2C: Virtual I2C interface i2c0 is listening on port 44855.

i2c0: Accepted client connection
I2C DPI: Remote disconnected.
swd0: Accepted client connection
```

```
[user@linux python]$ ./run_i2c_dpi_remote_test.py
========== I2C DPI Remote Bitbang Test =========
Write access ACKnowledged.
I2C write @ 0x12 = 0xba.
Write access ACKnowledged.
I2C write @ 0x12 = 0xdc.
Read access ACKnowledged.
I2C read @ 0x12 = 0x47.
Read access ACKnowledged.
I2C read @ 0x12 = 0x23.
```

```
[user@linux openocd]$ telnet localhost 4444
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x20000180 msp: 0x20000508
> reg
===== arm v7m registers
(0) r0 (/32): 0xe000e100
(1) r1 (/32): 0x00000007
(2) r2 (/32): 0x00000000
(3) r3 (/32): 0x40003048
(4) r4 (/32): 0x200001c4
(5) r5 (/32): 0x00000001
(6) r6 (/32): 0x200001c4
. . . . . .
(13) sp (/32): 0x20000508
(14) lr (/32): 0x200000e9
(15) pc (/32): 0x20000180
(16) xPSR (/32): 0x01000000
(17) msp (/32): 0x20000508
. . . . . .
===== Cortex-M DWT registers
```

```
[user@linux openocd]$ ./openocd
Open On-Chip Debugger 0.11.0-rc2+dev-00003-gb5563b75d-dirty (2022-09-02-17:21)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : Initializing remote_bitbang driver
Info : Connecting to localhost:44853
Info : remote_bitbang driver initialized
Info : This adapter doesn't support configurable speed
Info : SWD DPIDR 0x0bc11477
Info : cortex_m0p_sim: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for cortex_m0p_sim on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'telnet' connection on tcp/4444
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x20000180 msp: 0x20000508
```

Figure 9 An example run of the DPI-based SoC simulation environment.

V. CONCLUSIONS

In modern complex SOC based on microcontroller, the possibility of introducing a methodology to start debugging before the FPGA prototype is ready, allows to reduce the effort and the number of potential ECOs in the SOC design flow. The methodology proposed in this paper demonstrates how to design the architecture for the hardware/software co-verification environment. In addition to that, a simple SOC example was developed and integrated into the co-verification infrastructure to demonstrate the testing and debugging features of the proposed approach. Finally, by eliminating the tools boundary, the same set of DOTT-based tests can be used for the IC simulation, the FPGA prototype, and the post-silicon validation.

The overall development process and hardware/software co-verification benefits from the proposed approach since software development becomes involved much earlier, and software development expertise and resources can be leveraged, reducing the risks to find both HW and SW bugs in later design stages.

REFERENCES

[1] T. Winkler, Debugger-based On-Target Testing, [Online], Available:https://twinkler-ams-osram.github.io/dott_docu/index.html

[2] P. Rashinkar, P. Paterson, and L. Singh, Cadence Design Systems Inc., “Hardware/Software Co-verfication,” in System-on-a-chip verification, methodology and techniques. USA: Kuwer Academic Publishers, 2001, pp. 295-296.

[3] IEEE Standard for Test Access Port and Boundary-Scan Architecture, IEEE 1149.1-2013, 2013.

[4] ARM Limited, ARM Debug Interface v5, [Online], Available: https://developer.arm.com/documentation/ihi0031/latest/

[5] OpenTitan, [Online], Available: https://docs.opentitan.org/

[6] Dominic Rath. "Open On-Chip Debugger an On-Chip Debug Solution for Embedded Target Systems based on the ARM7 and ARM9 Family," Diploma Thesis, Dept. Computer Science, Univ. of Applied Sciences, Augsburg, Germany, 2005. [Online], Available: https://openocd.org/files/thesis.pdf

[7] C. Spear and G. Tumbush, "Interfacing with C/C++," in SystemVerilog for Verification, a guide to learning the testbench language features, 3rd ed., USA: Springer, 2012, pp. 415-453.

[8] GDB developers, GDB Remote serial protocol, [Online], Available: http://sources. redhat.com/gdb/current/onlinedocs/gdb_33.html

[9] OpenTitan JTAG-DPI, [Online], Available: https://github.com/lowRISC/opentitan

[10] OpenOCD remote bitbang SWD support, [Online], Available: https://review.openocd.org/c/openocd/+/6044

[11] J. Postel, J. Reynolds, RFC 854 – Telnet protocol specification, 1983, [Online], Available: http://www.faqs.org/rfcs/rfc854.html