International Journal of Engineering &Technology, 7 (4) (2018) 6649-6651

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.28509
Research paper

ScalabletTest Framework for device driver validation

P. Nagabhushan Reddy[1]*, Dr. T. Bhaskara Reddy[2]

[1]P. Nagabhushan Reddy, Research Student, RU, Kurnool – 518002
[2]Professor, Dept. of Computer Science, S.K. University, Anantapur-515003
**Corresponding author E-mail:bhaskarareddysku@gmail.com*

Abstract

Real-time embedded systems are increasingly being implemented on system-on-chip (SoC) devices in order to take advantage of lower power consumption, lower unit cost, and higher integration. These SoCs have processors, memory, peripherals, controllers, and other subsystems on the same silicon die. Based on the different business requirements wide variety of SoCs are designed and manufactured. The application development on these SoCs requires device drivers for communicating with the peripherals. Some of the most common device driver categories are: Serial (I2C, SPI, UART), Storage (MMC/SD, NAND etc), Audio, Connectivity (USB), Networking and Video. The interfaces and implementation of these device drivers vary with different SoCs and RTOS (e.g. DSP/BIOS, PrOSetc).
With increasing number of SoCs and less time to market, the device driver validation needs to be efficient and reliable. Typically test benches or test suites are developed to carry out the validation. Such test benches should have an architecture that helps plugging in new testcases easily, provide test case portability across SoCs and RTOS, enable ease of use, help in regression tests, provide better maintainability and improved time to market.
The purpose of this paper is to explain the architecture of the test bench that we have developed. We also want to share the details on prior work/experience, what motivated us to develop the test bench. We will also cover the impact of these test benches across different teams within our organization and our customers. Finally we will conclude with the status of current work and future plans.

Keywords: *Device Drivers; Framework; Test Bench; Test Case.*

1. Introduction

1.1. System on chip

Typically a System on chip has a processor, peripherals and other resources on the same silicon die.

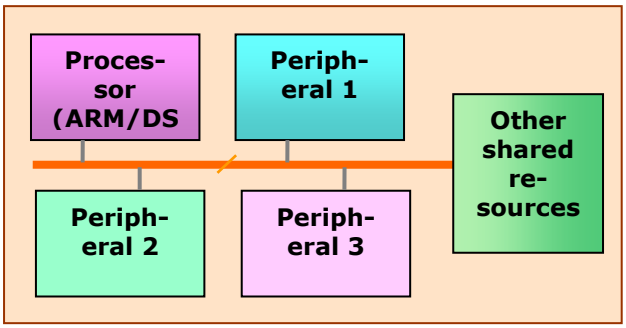

Fig. 1: Block Diagram of A Soc (Example).

1.2. Device driver

Applications use device drivers to communicate with hardware peripherals and devices present in the system. A device driver communicates with the peripheral using the underlying hardware mechanism. There are different categories of device drivers like serial, networking, display, audio, video and storage drivers. The implementation/ interfaces of each driver vary with different platforms and operating systems.

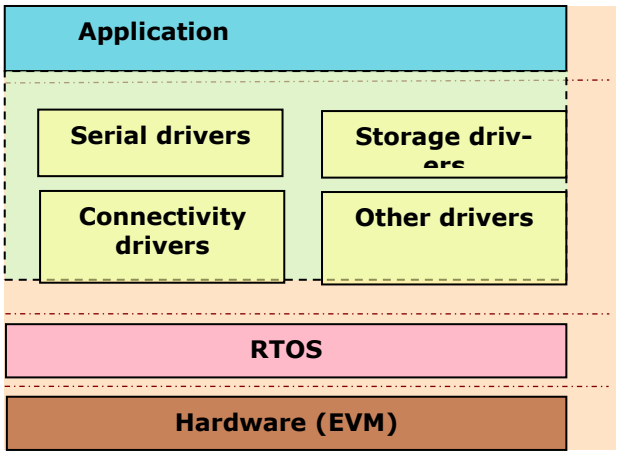

Fig. 2: Software Stack with Device Driver Layer.

1.3. Device driver validation

Validation of device driver includes creating different categories of test cases which can be run on the required platform and operating system.

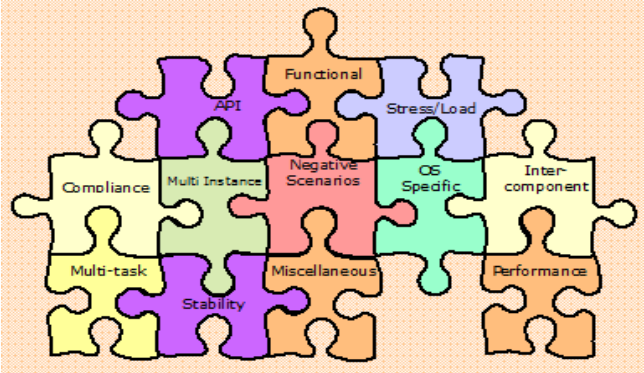

Fig. 3: Categories of Tests.

2. Problem statement

Semiconductor companies produces various SoCs having different set of peripherals and configurations.The device drivers are delivered independently for each platform and operating system. The driver validation activity specifically the test framework is dependent on following attributes.

2.1. Maintainability

The test bench architecture should define a standard way to add test cases for a new driver, new test cases for existing drivers. This will enable the test developers to develop the test cases in a well defined way.

2.2. Portability and reusability

The APIs and parameters of device driver for a particular peripheral could vary across platforms and RTOS. Test cases developed for a device driver should be portable across multiple platforms and RTOS to enable reuse of test cases.

2.3. Test case configurability

All the test parameters should be configurable at run time. The user should be able to execute a test case with varying parameters through a test script or user interface. Varying of test parameters should not require changing the test code. This will increase the robustness of test code and increase the flexibility to add new test cases on the fly.

2.4. Test execution and analysis

The test bench needs to be executed on the target. The test bench should provide a user interface through which the user should be able to run all the predefined tests at one shot using a test script. It also should give the flexibility to execute a particular set of tests and on the fly configuration of tests during execution.
The test logs should provide very clear details regarding the test case that is executed, test parameters, the intermediate logs and result. The log should help in further analysis and debugging.

2.5. Test bench as a product

The test bench should be productized and releases should be made available to the developer as well as the customer.
Developers can use the test bench for unit testing and regression testing before the release to the validation.
Customers can use the test bench and can execute the tests on their custom hardware.

3. Literature review

The main idea of this thesis is to introduce a scalable test bench architecture to validate device drivers for an IP on various SoCs and Platforms. The test architects can re-use 98% of the code using this architecture when moving into new platform. Only the Platform Abstraction layer and OS abstraction layer need to change and all the test cases can as it is be run with this.

4. Proposed work

4.1. Test bench architecture

The following figure 1 represents different modules of the test bench.

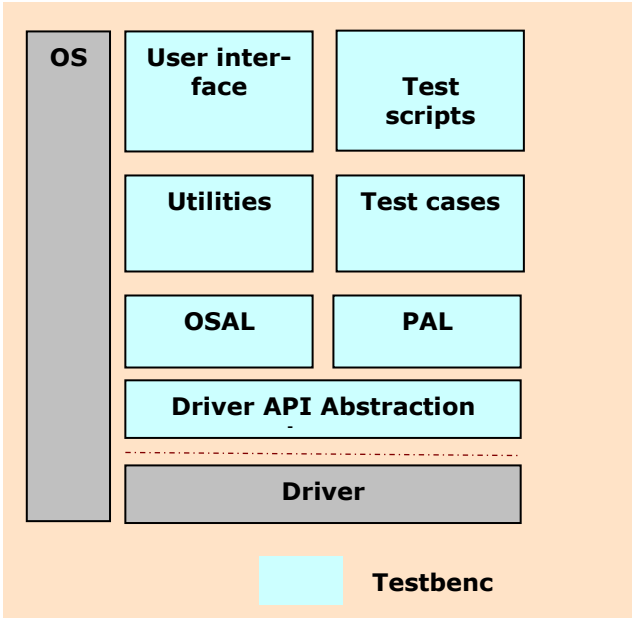

Fig. 4: Test-Bench Architecture.

The functionality of each module is explained below:

- User interface: User can communicate with the test bench using the user interface module. Through this interface the user can choose to execute the test scripts at one go or execute the required tests. The parser which is part of this module would parser the command and invokes the required test.
- Test scripts: Test scripts contain the pre defined test cases. The user can choose to execute the test script so that all the tests are executed at one go.
- Utility:Utility module contains common modules which can be used by different test cases.
  - Log module can be used as a utility for printing the logs and test information while executing the test case.
  - Performance measurement module can be used by test applications that want the CPU load and throughput measurement.
- Test case module contains the set of test cases implemented for running the test.
- OSAL is an abstraction layer for the OS services and will be used by the test cases requiring OS services.
- PAL helps in abstracting the platform specific code.
- Driver API Abstraction Layer is an abstraction to the driver APIs and definitions. The test application will not directly call the driver APIs but instead call into this abstraction layer. The advantage of having this layer is to isolate the changes to the test case layer if the APIs, definitions or parameters of a driver change across the platforms.

5. Result

- The test bench has been developed and productized.
- Test bench supports test suites for various drivers. These include Serial (UART, SPI, I2C), Storage (MMC/SD, NAND, File system, SATA), Audio, Video drivers.
- Test bench has been ported to the following platforms and RTOSs.
  - Platforms: DRx40x, DA830, C6747, DRx45x, C6748 etc.
  - RTOS: BIOS5, BIOS6, PrOS
- Test bench is being used to run all kinds of tests including functional, performance, stress, stability etc.
- Test bench releases are being made regularly adding the new platform support, new test cases and bug fixes.
- Development teams are using test bench for unit testing and regression testing.

Table 1: Differences between Old and New Test Bench

Features	Old Test Bench	New Test Bench
Test case reuse	<50%	>95%
Portableacross OS	No	Yes
Portableacross Platforms	No	Yes
Time to port for new Platform	2 months	2 weeks

6. Conclusion

In this paper we have elaborated the importance of devices drivers and the need for effectively validating them. Generally the device driver validation test codeis designed for specific platform and need to re-written for any new platform (or SoC). This will cause lot of issues in the test case development and also wastes lot of resources to re-do the things.

The solution we have proposed here addresses each of the problems and this new framework not only will scale to multiple hardware platform but also is scalable across various operating systems. This will reduce the development effort, make the test cases robust and thus will reduce the overall project cycle time and enhances the product time to market.

References

[1] Dan Williams, Patrick Reynolds, Kevin Walsh, Emin Gun Sirer, Fred B. Schneider “Device Driver Safety Through a Reference Validation Mechanism”

[2] JianjunDuan, John Regehr, "Correctness Proofs for Device Drivers in Embedded Systems"

[3] Shunan Mu, Guoqing Pan, Zhihao Tian and JianchengFeng“A survey of virtual prototyping Techniques for system development and Validation”

[4] P. Nagabhushan Reddy, Dr. T. Bhaskara Reddy, “Latest Power Management Technologies for Mobile Computing Devices”

[5] P. Rashinkar, P. Paterson, and L. Singh, System-on-a-Chip Verification Methodology and Techniques. Springer US, 2002.

[6] P. Nagabhushan Reddy, Dr. T. Bhaskara Reddy, “Test bench Design for validating Inter Processor Communication (IPC) in a multicore SoC”

[7] Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O’Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez “Moving Fast with Software Verification”

[8] N. Gupta and C. Harakchand, “Embracing the FPGA Challenge for Processor Design Verification,” in 2014 15th International Microprocessor Test and Verification Workshop, Dec 2014, pp. 39–43. [8] K. Kayamuro, T. Sasaki, Y. Fukazawa, and T. Kondo, “A Rapid Verification Framework for Developing Multi-core Processor,” in 2016 Fourth International Symposium on Computing and Networking (CANDAR), Nov 2016, pp. 388–394

[9] C. Spear and G. Tumbush, SystemVerilog for Verification, Third Edition: A Guide to Learning the Testbench Language Features. Springer Publishing Company, Incorporated, 2012.

[10] S. Sarkar, G. S. Chanclar, and S. Shinde, “Effective IP reuse for high quality SOC design,” in Proceedings 2005 IEEE International SOC Conference, Sept 2005, pp. 217–224.

[11] D. Stow, I. Akgun, R. Barnes, P. Gu, and Y. Xie, “Cost Analysis and Cost-driven Ip Reuse Methodology For Soc Design Based On 2.5d/3d Integration,” in 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov 2016, pp. 1–6.https://doi.org/10.1145/2966986.2980095.