2021 IEEE Latin America Electron Devices Conference (LAEDC). April 19-21, 2021

2021 IEEE Latin America Electron Devices Conference (LAEDC) | 978-1-6654-1510-1/20/$31.00 ©2021 IEEE | DOI: 10.1109/LAEDC51812.2021.9437939

An affordable post-silicon testing framework applied to a RISC-V based microcontroller

Roberto Molina-Robles*, Ronny García-Ramírez*,
Alfonso Chacón-Rodríguez*, Renato Rimolo-Donadio* and Alfredo Arnaud†
*Escuela de Ingeniería Electrónica, Tecnológico de Costa Rica
†Depto. de Ingenieria Electrica, Universidad Catolica del Uruguay
{rmolina, rgarcia, alchacon, rrimolo}@tec.ac.cr
aarnaud,@ucu.edu.uy

***Abstract*—The RISC-V architecture is a very attractive option for developing application specific systems needing an affordable yet efficient central processing unit. Post-silicon validation on RISC-V applications has been done in industry for a while, however documentation is scarce. This paper proposes a practical low-cost post-silicon testing framework applied to a RISC-V RV32I based microcontroller. The framework uses FPGA-based emulation as a cornerstone to test the microcontroller before and after its fabrication. The platform only requires a handful of elements like the FPGA, a PC, the fabricated chip and some discrete components, without losing the capacity to functionally validate the design under test and save development testing time by using a re-utilize philosophy.**

***Index Terms*—Post-silicon validation, testing, FPGA, RISC-V, microcontroller, EDA tools, architecture, test generation, I/O protocols, testing-platforms, SPI.**

I. Introduction and related work

The RISC-V architecture has become quite popular due to its open-source nature and high configurability, which allows it to meet the specifications of a wide range of applications. Such features are particularly appealing to a growing number of small design groups using the RISC-V architecture as a stepping stone for their particular ventures. However, developing a RISC-V core can still be challenging when trying to assure proper testing and validation on a budget.

It is well known that the increasing complexity of digital systems and their tight development schedules place strict demands on proper design validation. Moreover, simulating extracted post-place-and-route models can not always detect timing related structural errors. Industry's post-silicon validation environments are often custom-based to the characteristics of their already closed design-under-test (DUT), and most of them are very expensive to replicate. FPGA verification of RTL designs provides a faster avenue for the debugging of architectural problems, particularly in microprocessor-based designs, when thorough testing implies the execution of software tests on them as well. Short on financial and human resources, we decided to focus the post-silicon validation efforts of our RISC-V core employing FPGA as the testing platform. Additionally, we decided to re-utilize what we could of the functional verification environments' elements used on previous steps of the design flow to save time and resources.

There are several publications related to the design and use of a RISC-V based processors for specific applications. For instance, [1], [2], [3], [4], [5] and [6], describe their designs or how they used their respective RISC-V-based processors or microcontrollers to address a particular problem. Information about pre-silicon functional verification also exists, yet it is scarcer. Publications like [7], [8], [9] and [10] share some of their functional verification flows and techniques applied to RISC-V based processors. Finding publications about specific post-silicon testing strategies of any RISC-V based microcontroller was more difficult. For instance, authors of [11], focus their paper on the integration and evaluation of DFT modules on a RISC-V SOC, but without addressing the general post-silicon testing framework.

Regarding FPGA for emulation and post-silicon validation, there is plenty of literature about alternative approaches, such as [12], where a test generator for a microcontroller on-silicon validation was proposed. In [13], a post-silicon method to validate memory consistency in multiprocessor systems was presented. Authors of [14] implemented an interesting framework on a FPGA that tried to reach functional verification levels of observability by adding some scan cells to a specific design. Other works, like [15], [16] and [17], emulated different types of hardware pieces on FPGAs for testing purposes too, while [18] and [19] proposed using a FPGA as a low-cost ATE tool. In [20], a FPGA was used as a mean to capture data from a DUT, and at [21] an automated test methodology to test a FPGA was proposed.

But none of the former publications directly address the particularities of RISC-V post-silicon validation, this may be because of the still ongoing maturation process of the RISC-V toolchains and the standard itself. On the RISC-V community websites you can find some information for pre-silicon verification, however, RISC-V post-silicon validation documentation is rare. The objective of this paper is to propose a post-silicon validation framework amenable to other small design groups with limited resources, using a RISC-V core as a case study. A secondary objective, maybe not so technical but still critical for the authors, is to promote a larger discussion on the issue of post-silicon validation in the RISC-V community. Especially as more and more RISC-V designs by small and not so small teams move into the commercial arena, with the consequential need for stronger, standardized post-silicon validation flows.

II. The RISC-V core interfaces overview

Figure 1 describes the interfaces of the RISC-V based microcontroller that is to be tested.

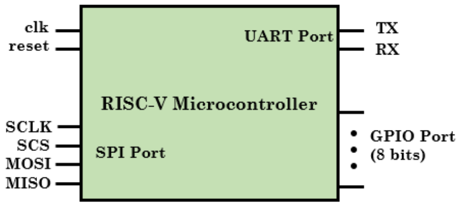

Figure 1. Chip-level description of the RISC-V microcontroller's interfaces.

As illustrated, the microcontroller has three main interfaces: a Serial Peripheral Interface (SPI) port [22], a Universal Asynchronous Receiver-Transmitter (UART) port and a General Purpose Input/Output (GPIO) port. The design intent is to connect the microcontroller to a flash memory through the SPI port.

From those three, the SPI port is in charge of handling an external flash memory, where the microcontroller's bootstrap routine is stored. After bootloading, the flash memory serves as a secondary data memory. The UART interface is a simple port that uses a standard 19200 baud rate, with 8 data bits, 1 start bit, 1 stop bit and even parity. The main purpose of the UART is to be able to communicate with the processor from a PC. The GPIO ports are used to control displays connected to the processor or to receive data from sensors.

III. FPGA emulation Phase: the bootloading architecture

The testing process of the microcontroller was divided into phases. Each phase represents a specific moment inside the project development schedule. Figure 2 depicts the order in which the proposed phases were executed.

The "Functional/Formal Verification" and the "Post-Layout Validation" phases are considered pre-silicon testing, and as such, they will not be covered in this article.

Now, because testing time is one of the bottlenecks inside the VLSI design flow, we strove to re-utilize code and implementations as much as possible between steps, while trying to debug one feature at a time to try to isolate hidden bugs.

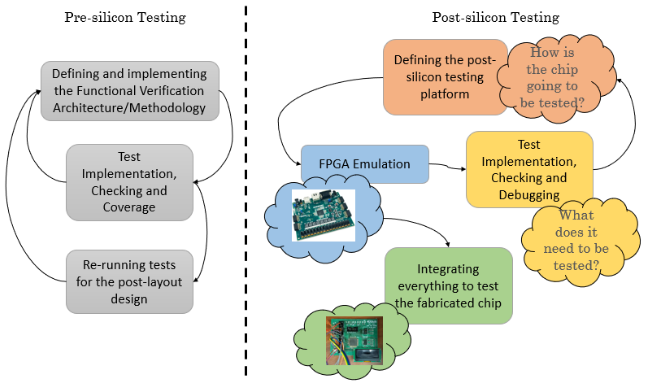
Pre-silicon Testing
Post-silicon Testing
Defining and implementing the Functional Verification Architecture/Methodology
Test Implementation, Checking and Coverage
Re-running tests for the post-layout design
Defining the post-silicon testing platform
How is the chip going to be tested?
FPGA Emulation
Test Implementation, Checking and Debugging
What does it need to be tested?
Integrating everything to test the fabricated chip

Figure 2. Testing methodology used to test the RISC-V microcontroller.

The first step on our post-silicon testing was to define "how" were we going to test the microcontroller. The following phases can vary depending on the needs of the project. For our case, we decided to emulate the microcontroller and the flash memory on a FPGA, using it as support to test the final chip that was going to be integrated along with some displays over a PCB.

The FPGA emulation not only is useful to design the post-silicon tests for when the resulting fabricated chip comes back from the foundry, but it also helps as a complemented to the functional verification phase. As a matter of fact, this was particularly helpful to identify structural bugs that have been overlooked in previous phases of our project and they were spotted when implementing the design on the FPGA.

Our goal with FPGA emulation was to try to replicate the whole testing environment as if the chip has been already fabricated and was ready for testing. In consequence, it was necessary to emulate the flash memory and find a way to ensure that tests were working as intended. However, observability during post-silicon phases is minimal, even more so when no DFT modules are included inside the design. Hence, the only way to see whats happening inside was using peripherals connected to the UART and GPIO ports as indicators. Figure 3 shows the general idea of this strategy.

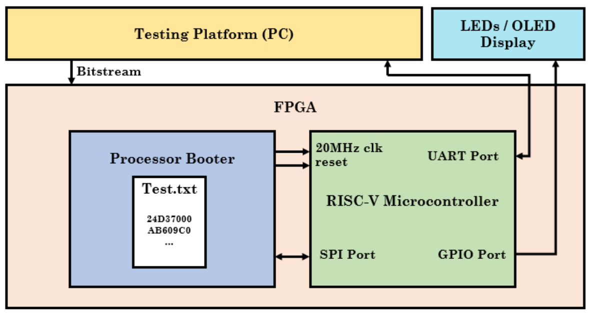

Figure 3. Testing concept for the developed RISC-V microcontroller synthesized inside a Nexys 4.

As depicted, a standard PC was enough for the testing platform. The idea was to employ the PC as a USB/COM port receptor for the UART after implementing to design inside the FPGA.

The microcontroller was an independent, easy to find, module inside the top level of the testing architecture. This way, the FPGA can easily be incorporated as a support driver/receiver into the final testing framework by removing only the microcontroller module from the FPGA implementation, and setting the SPI Port as an FPGA I/O connected to the real fabricated chip, thus, saving testing development time.

To emulate the flash memory, two major RTL blocks were proposed to implement its functionality as shown in figure 4. The described boot loader module must emulate the flash memory, generate the microcontroller's clock and control the bootstrap routine of the processor by clearing its reset input signal.

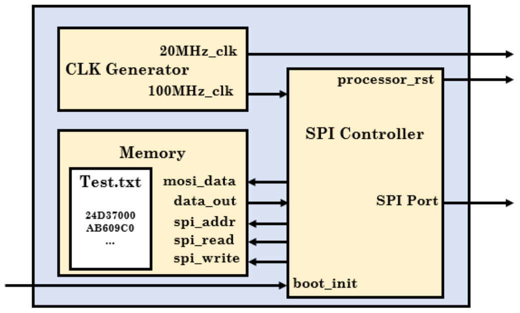

Figure 4. Block-level micro-architecture of the boot loader module.

The test programs used on post-silicon phases were previously validated, compiled and linked in the needed HEX format on the microcontroller's functional verification environment, saving more development time. This programs were loaded into the instantiated flash memory emulator during the FPGA synthesis stage.

The SPI interface controller is a synchronous block that acts as "slave" and receives commands from its "master", the microcontroller. After that, it performs an action according to the code received. This communication must match, the functionality of the selected commercial flash memory. The commands we needed were those related to read and write operations, which can be found here [23]. Figure 5 illustrates a command sent by the RISC-V microcontroller as it starts booting.

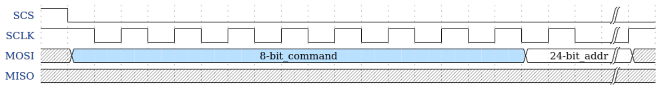

Figure 5. Timing diagram of a command sent by the microcontroller through the SPI port.

The SPI interface module had to capture this information. If a read command came in, it should prepare the corresponding data to be sent to the core, as depicted in Fig. 6.

To achieve the SPI functionality, a finite state machine was implemented. This FSM was created under the intention that

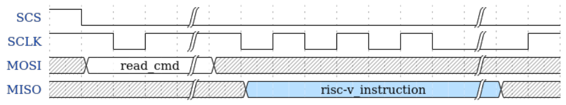

Figure 6. Timing diagram of the boot loader response to a read command.

the SPI interface module could have two purposes: to monitor and decode received comands through the MOSI input, and to execute the captured command using the MISO output. Figure 7 shows the state diagram of this FSM. Inside the MOSI_Mode state, the SPI controller waits for the microcontroller's command. Otherwise, inside the MISO_Mode state, the SPI controller should resolve the command previously received.

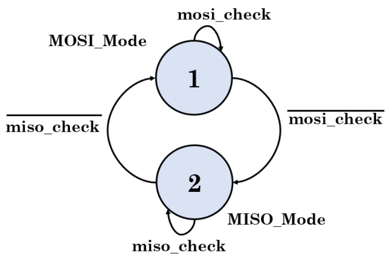
mosi_check
MOSI_Mode
1
$\overline{\text{miso\_check}}$
$\overline{\text{mosi\_check}}$
2
MISO_Mode
miso_check

Figure 7. SPI Controller FSM. State change evaluation is triggered with SCLK positive and negative edges.

Auxiliary counters and shift registers to capture the command, a combinational module to decode the command, a counter to update the internal memory address, a clock generator using the FPGA 100 MHz clock as source, etc; were ther digital blocks needed to achieve the emulation. Other important note, is that The SPI Controller's frequency was set at 100 MHz, while the microcontroller's frequency was set at 20 MHz by our specification. As a result, the FSM and counters are triggered with the slower SCLK line generated by the microcontroller. Because of this, both positive and negative edges were needed inside the FSM functionality to avoid metastability. Figure 8 depicts the way both edges of the SCLK were interpreted. During the positive edges, the master reads the current data on the MISO line and, at same time, the following data bit was requested to the internal memory. On negative edges, the requested bit is set on the MISO line, ready for the next positive edge. Edge detectors where needed to identify the corresponding edge.

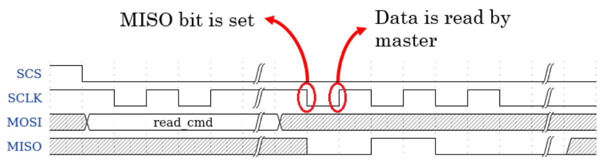

Figure 8. FSM SPI Controller functionality triggered by positive and negative edges.

IV. Post-Silicon Test Elaboration and Chip Testing Phases

The most fundamental features we considered to be tested were: the characteristics related with the compliance of the microcontroller with the chosen RISC-V instruction subset (RISC-V RV32I), that interruptions behaved as intended by our specification, that all the I/O ports worked as expected by their respective protocols, that data could be stored and loaded from the internal memory and registers, and to ascertain how much power consumption did the chip need. Most of the required assembly code for this tests were already available from functional verification.

Because of observability, the first thing that had to be tested were the needed I/O ports working as specification checkers. For instance, developing programs that, turn on and off LEDs and may use the internal timer to toggle, and ”Hello World” programs that sent a message through the UART port to the PC. For UART monitoring we used RealTerm or LabVIEW on the PC.

Once the I/O ports were tested to some degree, we were ready to validate features that without DFT were difficult to test, like, register bank checking, memory checking, interruptions checking, etc. We also converted some of the pre-silicon tests into post-silicon self-checking tests. This way, the same test program checks its internal results and uses a LED or a generated message as indicators to the user that the test has passed or not. This way, we tested that that the microcontroller could run the complete RISC-V RV32I instruction subset and other internal features already mentioned.

Once the chip was fabricated, it was integrated into a PCB subsystem with LEDs, an OLED display, an oscillator, the flash memory and circuitry for voltage supply. Separate supply rails allowed for the independent power measurement of each voltage domain inside the microcontroller: the processor core (1.8 V), memory (1.8 V) and pads (3.3 and 1.8 V). Multiplexers were added to allow the use of reset, clock and bootstrap signals either from the FPGA or from the discrete components in the PCB.

The microcontroller was successfully tested on the PCB re-using the post-silicon tests developed so far on the FPGA emulation, with little adjustments on the testing environment (just removing the DUT from the FPGA implementation). Nonetheless, two more types of tests were developed for this paper: a test for the emulation of a I^2C interface via the GPIO ports, and a bank of continuous tests to measure the microcontroller’s power consumption. For the I^2C test, a ”Hello World” assembly program writes on the OLED display. The program emulates the signaling required to handle the OLED’s I^2C protocol, using around 4 KB of the internal memory (our internal memory size was 8 KB). Internal interrupts and timers were used to generate the I^2C clock, and its functionality was emulated with an algorithm following a designed state diagram. Figure 9 shows the microcontroller writing on the OLED display.

The last group of tests were designed to loop the microcontroller, while performing specific operations to measure the average consumed current. Estimates of power consumption per instruction or per type of instruction were generated, as a result, we obtained an average power consumption of 1,67 mW. Idle current consumption was also obtained by measuring current while maintaining the boot signal high, so that the microcontroller could not boot up, on this state it consumed 1,3 mW. Finally, static power consumption was estimated measuring the current when the microcontroller was powered on, without an active clock, resulting in 36 nW power consumption. From simulations we have identified unexpected power consumption on certain modules inside the microcontroller, said that, we strive to reduce significantly the power consumption when computing on subsequent versions of the microcontroller.

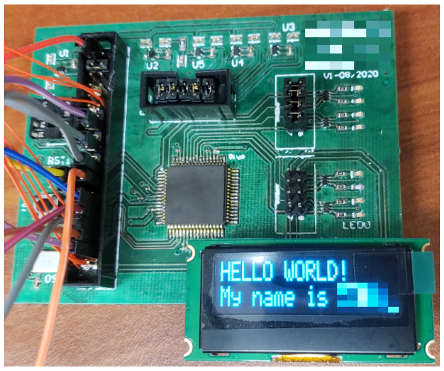

Figure 9. ”Hello World” program on a OLED display, emulating the required I^2C protocol via software routines and the GPIO ports.

Summarizing the resources used in this framework, the only physical components needed to replicate it are a FPGA, a PC for monitoring, the fabricated chip (and PCB if wanted) and some lesser display/support components. On the software side, most of the employed software is free. Also, tests development time can be greatly reduced if a recycling code approach like the one used in this article is used. All, without losing the capacity to test exhaustively the RISC-V microcontroller.

V. Conclusions

This paper proposed a low-cost post-silicon validation framework to test a RISC-V RV32I microcontroller using a FPGA for emulation and support as a study case. This framework shows that is possible for small design teams in the RISC-V community to take advantage of modern, standardized verification and post-silicon techniques at a low cost. The paper has included a detailed description of the employed testing methodology, some architectural information about design and implementation of the testing framework and integration of all the components along with experimental results, using the process of testing this DUT as a concept example to validate the methodology. Our roadmap for future works includes: adaptation of pre-silicon and post-silicon testing environments to newer versions of the microcontroller, to develop software applications to command on-the-fly the microcontroller from a PC and polishing the current framework to improve delivery times.

References

[1] D. K. Dennis, A. Priyam, S. S. Virk, S. Agrawal, T. Sharma, A. Mondal, and K. C. Ray, "Single cycle risc-v micro architecture processor and its fpga prototype," in *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, Dec 2017, pp. 1–5.

[2] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer, and M. Linauer, "Open-source risc-v processor ip cores for fpgas — overview and evaluation," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, June 2019, pp. 1–6.

[3] G. Zhang, K. Zhao, B. Wu, Y. Sun, L. Sun, and F. Liang, "A risc-v based hardware accelerator designed for yolo object detection system," in *2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE)*, 2019, pp. 9–11.

[4] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Castro-Gonzalez, A. Arnaud, M. Miguez, J. Gak, R. Molina-Robles, G. Madrigal-Boza, M. Oviedo-Hernandez, E. Solera-Bolanos, D. Salazar-Sibaja, D. Sanchez-Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, and R. Rimolo-Donadio, "Siwa: a risc-v rv32i based micro-controller for implantable medical applications," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[5] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Molina-Robles, R. Castro-Gonzalez, E. Solera-Bolanos, G. Madrigal-Boza, M. Oviedo-Hernandez, D. Salazar-Sibaja, D. Sanchez-Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, R. Rimolo-Donadio, A. Arnaud, M. Miguez, and J. Gak, "Siwa: A custom risc-v based system on chip (soc) for low power medical applications," *Microelectronics Journal*, vol. 98, p. 104753, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269219303787

[6] A. Arnaud, M. Miguez, J. Gak, R. Puyol, R. Garcia-Ramirez, E. Solera-Bolanos, R. Castro-Gonzalez, R. Molina-Robles, A. Chacon-Rodriguez, and R. Rimolo-Donadio, "A risc-v based medical implantable soc for high voltage and current tissue stimulus," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[7] M. Chupilko, A. Kamkin, and A. Protsenko, "Open-source validation suite for risc-v," in *2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV)*, 2019, pp. 7–12.

[8] R. Molina-Robles, E. Solera-Bolanos, R. García-Ramírez, A. Chacón-Rodríguez, A. Arnaud, and R. Rimolo-Donadio, "A compact functional verification flow for a risc-v 32i based core," in *2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA)*, 2020, pp. 1–4.

[9] A. Munir, M. Magdy, S. Ahmed, S. Nasr, S. El-Ashry, and A. Shalaby, "Fast reliable verification methodology for risc-v without a reference model," in *2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, 2018, pp. 12–17.

[10] A. Oleksiak, S. Cieślak, K. Marcinek, and W. A. Pleskacz, "Design and verification environment for risc-v processor cores," in *2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems"*, 2019, pp. 206–209.

[11] W. Ramirez, M. Sarmiento, and E. Roa, "A flexible debugger for a risc-v based 32-bit system-on-chip," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[12] P. Moharikar and J. Guddeti, "Automated test generation for post silicon microcontroller validation," in *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, 2017, pp. 45–52.

[13] B. W. Mammo, V. Bertacco, A. DeOrio, and I. Wagner, "Post-silicon validation of multiprocessor memory consistency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 1027–1037, 2015.

[14] F. Moraes, M. Moreira, C. Lucas, D. Corrêa, D. Cardoso, M. Magnaguagno, G. Castilhos, and N. Calazans, "A generic fpga emulation framework," in *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, 2012, pp. 233–236.

[15] A. Koczor, Matoga, P. Penkala, and A. Pawlak, "Verification approach based on emulation technology," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2016, pp. 1–6.

[16] K. Hayama, T. Matsumoto, and Y. Shimada, "Design and implementation of emulator on fpga," in *2007 IEEE International Conference on Mechatronics*, 2007, pp. 1–5.

[17] W. Y. Lo, C. S. Choy, and C. F. Chan, "Hardware emulation board based on fpgas and programmable interconnections," in *Proceedings of IEEE 5th International Workshop on Rapid System Prototyping*, 1994, pp. 126–130.

[18] A. A. Bayrakci, "Elate: Embedded low cost automatic test equipment for fpga based testing of digital circuits," in *2017 10th International Conference on Electrical and Electronics Engineering (ELECO)*, 2017, pp. 1281–1285.

[19] D. Jadaan, K. Gonsholt, and A. Skavhaug, "Low-cost platform-independent fpga based real-time systems tester," in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016, pp. 686–689.

[20] W. Huang, G. Hou, J. Huang, and T. Kuo, "An fpga-based data receiver for digital ic testing," in *2019 IEEE International Test Conference in Asia (ITC-Asia)*, 2019, pp. 25–30.

[21] A. W. Ruan, Y. B. Liao, P. Li, W. Li, and W. C. Li, "An automatic test approach for field programmable gate array (fpga)," in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, Dec 2009, pp. 474–477.

[22] F. Leens, "An introduction to i2c and spi protocols," *IEEE Instrumentation Measurement Magazine*, vol. 12, no. 1, pp. 8–13, February 2009.

[23] I. S. S. Inc., "32mb serial flash memory with 133mhz multi i/o spi quad i/o qpi dtr interface," 2019, datasheet. [Online]. Available: http://www.issi.com/WW/pdf/25LP-WP032D.pdf