Ruprecht-Karls-Universität Heidelberg

Department of Physics and Astronomy

Kirchhoff-Institute for Physics

Project Internship Report

Verification of a Parameterizable JTAG Driver Module

Marco Rettig

supervised by
Vitali Karasenko

January 2019

Abstract

The Electronic Vision(s) Group at the Kirchhoff-Institute for Physics uses JTAG Test Access Ports in different parameterizations for the communication with the integrated circuits. For the control of these Test Access Ports, an equally parameterizable driver module was designed in SystemVerilog by a former group member. This driver module was only used with one specific configuration, since all other configurations had not yet been tested. The goal of this project was therefore to verify the correct functionality of the driver module for all permitted configurations. For the verification, a parameterizable test bench was designed. With the help of the test bench, different tests were carried out which revealed various malfunctions of the driver module. These malfunctions could be fixed in the course of a comprehensive debugging process. The correct functionality of the driver module is now ensured for all possible configurations. It thus can be used for different projects such as the development of the HICANN-X chip.

Contents

List of Figures

1 Introduction

1.1 Problem Description and Goal Setting

The Electronic Vision(s) Group at the Kirchhoff-Institute for Physics deals with the development of an adaptive neuromorphic computing system. This system is based on a physical model of a neural network consisting of neurons and synapses. The networks are implemented in analog microelectronics which are placed on computer chips. Communication with the analog part of those chips requires additional digital integrated circuits. The proper functionality of these integrated circuits can be verified in the installed state via a parameterizable so-called JTAG Test Access Port (wherein JTAG stands for Joint Test Action Group). For the control of the Test Access Port, an equally parameterizable driver module was designed by a former group member using the hardware description language SystemVerilog. This driver module is currently used with only one specific configuration, since all other configurations have not yet been tested. The aim of this thesis is therefore to verify the correct functionality of the driver module for all permitted configurations. This includes the identification of possible errors and their elimination.

1.2 Structure of the Project Thesis

In chapter 2, the structure and operating principle of the Test Access Port and the driver module and their interaction are outlined. In chapter 3, a test bench for verification of the proper functionality of the driver module is designed. For this purpose, the criteria to be tested is firstly defined and afterwards implemented in the form of a VHDL file. In chapter 4, the test of the driver module is carried out using the previously designed test bench. In an initial test phase, the module is tested in the state in which it is already being used. It is then modified with regard to the elimination of the detected errors before undergoing a final test phase. In chapter 5, the achieved results are summarized and an outlook on further possible projects is given.

2 Interaction of the Driver Module and the JTAG Test Access Port

2.1 Structure and Operating Principle of the JTAG Test Access Port

The primary purpose of a JTAG Test Access Port (TAP) is to test the structure and functionality of complex integrated circuits (ICs) that are already installed in their working environment. All flip-flops of a JTAG compatible IC are equipped with an additional multiplexer at the input. In this way, it is possible to monitor and control the state of each flip-flop externally [2]. The TAP mainly consists of [2]:

- The control lines.
- The *TAP Controller*, a state machine that controls the test logic.
- The Instruction Register (IR) and other shift registers that are subsumed under "Data Register" (DR).

The control lines comprise [2]:

1. The Test Data Input (*TDI*): Serial input of the shift registers.
2. The Test Data Output (*TDO*): Serial output of the shift registers.
3. The Test Clock (*TCK*): The clock signal for the entire test logic.
4. The Test Mode Select (*TMS*): It controls the *TAP Controller*.
5. The Test Reset (*TRST*): Reset of the test logic.

The *TAP Controller* is a state machine being clocked by TCK and controlled by the TMS line. All possible states and state transitions are illustrated in figure 1. In each state, certain control functions are triggered. Transitions to other states are made depending on the current state and the value of the TMS line (0 or 1). There are states in which the controller can remain for only one test clock cycle and so-called stable states that can be maintained for several consecutive test clock cycles. In the *Test Logic Reset* state, the test logic is reset. The *Run Test/Idle* state is chosen for waiting times after the reset or between two successive complete shift phases. In the two *Shift* states, the respective register is shifted by exactly one bit per test clock cycle. With the help of the two *Pause* states, shift operations can be interrupted.

According to the IEEE 1149.1 standard, the TAP contains an instruction register and further shift registers which are classified as data registers. These are [1]:

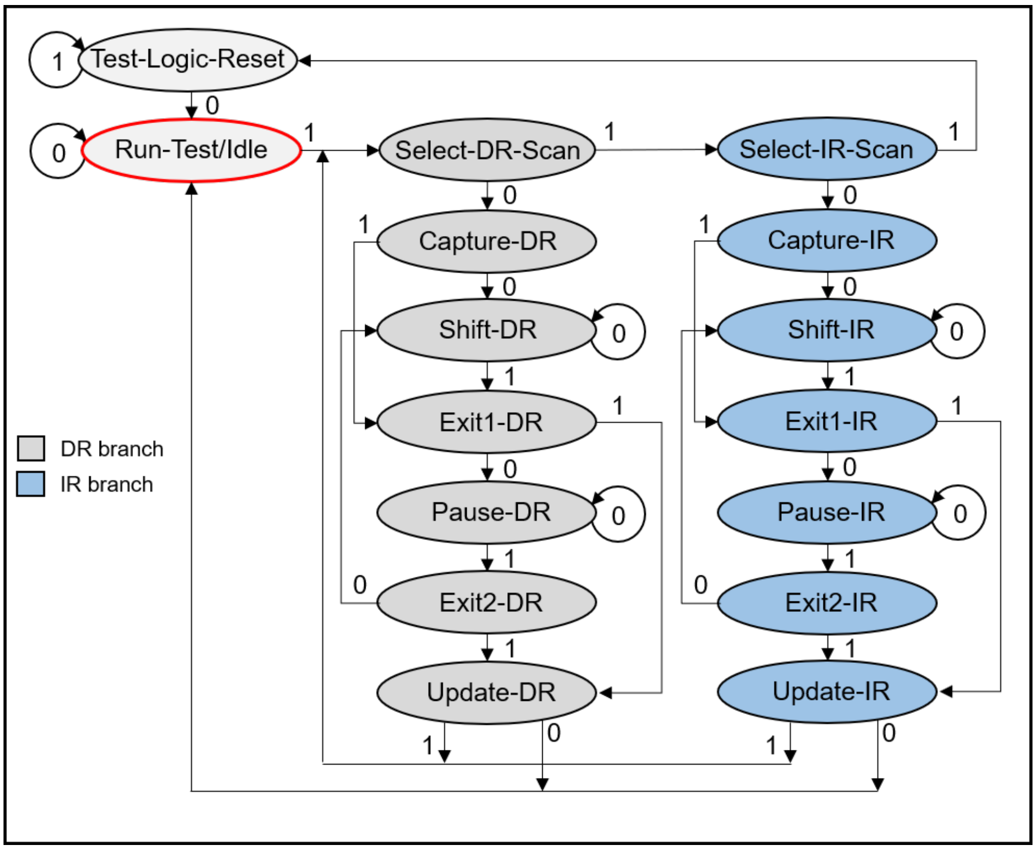

Figure 1: State diagram of the used JTAG *TAP Controller* (Adapted from [1]).

- The Boundary Scan Register.
- The Bypass Register.
- An optional Device Identification Register.
- Further test data registers that can be defined by the user for test purposes [1].

The registers are serially accessed via the TDI and TDO lines. During a shift operation of a particular register, the TDI input is pushed into the foremost bit of the register while its rearmost bit is shifted out to the TDO output. Whether the instruction register or one of the data registers is shifted depends on the branch of the state diagram in which the TAP Controller is currently located. The instruction register is used to shift in instructions, which select and operate the data registers. Which data register is selected and what operation is performed during a particular instruction, has to be implemented in user-defined instruction decode logic [1]. Furthermore, the IEEE 1149.1 standard prescribes a few mandatory instructions for the Boundary Scan Register which are already implemented. The latter provides access to the logic signals of the IC, containing a shift register cell for each signal pin. The connected cells represent a certain path around the IC's boundary which allows for detailed visibility and controllability of the signals [3]. The Bypass Register is a 1-bit register enabling a direct connection between TDI and TDO. If several TAPs are serially concatenated to a so-called "JTAG-Chain" and only the data register of one single TAP is to be read or written, the Bypass Register of all other TAPs can be selected via the corresponding instruction. In this way, the desired register can be accessed directly instead of shifting values through all registers of the chain. This minimizes the latency [2]. The optional Device Identification Register can be used for loading a 32-bit number to identify the TAP.

2.2 Structure and Operating Principle of the Driver Module

The driver module is connected to the TAP via the control lines. It generates the TCK signal, controls the TAP Controller via the TMS line and writes or reads the TAP registers by shifting in instructions and data. These functions are performed by different shifter modules being implemented within the driver module:

- The TCK shifter.
- The TMS shifter.

- The IR shifter.
- The DR shifter.

Since in synchronous mode all TAP operations depend on the test clock signal generated by the TCK shifter, all other shifters work with the same clocking as the TCK shifter. This clocking usually corresponds to the system clock of the driver module itself. However, it is also possible to set lower frequencies than that of the system clock by changing the so-called scaler value. This value is transferred to a counter module also implemented within the driver module. The counter module increments an internal variable at every positive edge of the system clock and generates a hit signal if the variable value corresponds to the scaler value. A shift operation of a certain shifter is therefore only triggered when the hit signal appears and the shifter is enabled. Which shifter is enabled is controlled by an internal state machine.

The TCK shifter always alternately shifts ones and zeros into the TCK control line and thus requires no data dependent input from outside the driver module. Since the TMS sequences for the required state transitions of the TAP Controller are also independent of the data to be transmitted, they can also be generated within the driver module. The instructions and data to be shifted by the IR and DR shifter, on the other hand, are transmitted to the module via two separate input buses (*jtag_ir* and *jtag_dr*). The outputs of both shifters are connected to the TDI control line feeding the currently selected TAP register. Which of the shifters is supposed to shift data bits into the line is again controlled by the state machine of the driver module. Furthermore, one of the input lines of the DR shifter is connected to the TDO control line. In this way the data bits being shifted out of the selected TAP data register are captured and can be read out via an additional output bus of the DR shifter (*data_out*). The driver module also provides the possibility to shift only a certain number of the transmitted data bits into the selected TAP data register. The bit vectors containing the number of bits to be shifted are transmitted to the module via an additional input bus (*num_shift_data_bits*). There are further input lines and buses for the transmission of the scaler value (*clk_scaler*) and the signals which trigger TAP IR or DR shift operations (*start_instruction* or *start_data*) or the reset of the test logic (*start_reset*). The *reset* input line transmits the signal which triggers the reset of the internal state machine. The signal of the system clock of the driver module is transmitted via the *clk* input line. Apart from the TCK, TMS and TDI control line, the module is equipped with further output lines which are used to send various status messages (*processing*, *cmd_accept*, *error* and *read_data_available*). The *jtag_read_data* bus forwards the *data_out* signal from the DR shifter.

The structure described above represents the core module of the driver module and is referred to as *jtag_driver*. Due to the number of different inputs, control of the *jtag_driver* is relatively complex. For this reason, it was integrated into another module which is denoted as *ut_jtag_driver* where *ut* stands for *universal translator*. The *ut_jtag_driver* acts as an interface between the external user and the *jtag_driver* enabling an easier control through fewer inputs. The *write_data* input bus is used for transmission of:

- The instructions for the IR shifter.
- The data for the DR shifter including the number of bits to be shifted and an additional so-called *keep_response* bit influencing the start of DR shift operations.
- The scaler value.

The signals which trigger shift operations, the reset of the test logic or changes of the TCK frequency, are transmitted by the *write_idx* bus. For this purpose, the *ut_jtag_driver* encodes each trigger signal with a certain value. Depending on the value received via the *write_idx* bus, it then initiates the corresponding operation unless the *jtag_driver* is currently performing other operations. Its availability is signalled via the *processing* line. If the TCK frequency is supposed to be changed, the corresponding scaler value transmitted by the *write_data* bus is forwarded to the *clk_scaler* input of the *jtag_driver*. For shift operations of the DR shifter, the part of the received bit vector containing the number of bits to be shifted is forwarded to the *num_shift_data_bits* input while the part containing the actual data is forwarded to the *jtag_dr* input. Whether a DR shift operation is initiated, does not only depend on the availability of the *jtag_driver* but also on the *keep_response* bit and the signal received by the *read_rdy* input line. The latter indicates whether the user is ready to read the data which is sent back to him. If the *write_idx* bus transmits the signal for the reset of the test logic, the *start_reset* line of the *jtag_driver* is simply fed with a high bit triggering the TMS shifter to shift the appropriate bit sequence into the TMS control line. The *write_valid* input line indicates whether the user is currently sending data to the driver module. The TDO input line forwards the data bits being shifted out of the selected TAP data register to the *jtag_driver* TDO input. The *read_data* output is connected to the *jtag_read_data* output of the *jtag_driver*. The *write_next* line sends a high bit when a new shift operation, the reset of the test logic or a change of the TCK frequency is initiated. The end of a DR shift operation is communicated via the *read_next* output line. The signal of the system clock is received via the *clock* input line and directly forwarded to the *clk* input of the *jtag_driver*. The *reset* input analogously forwards the reset signal to the corresponding *jtag_driver* input. A simplified schema of the structure of the entire driver module is illustrated in figure 2.

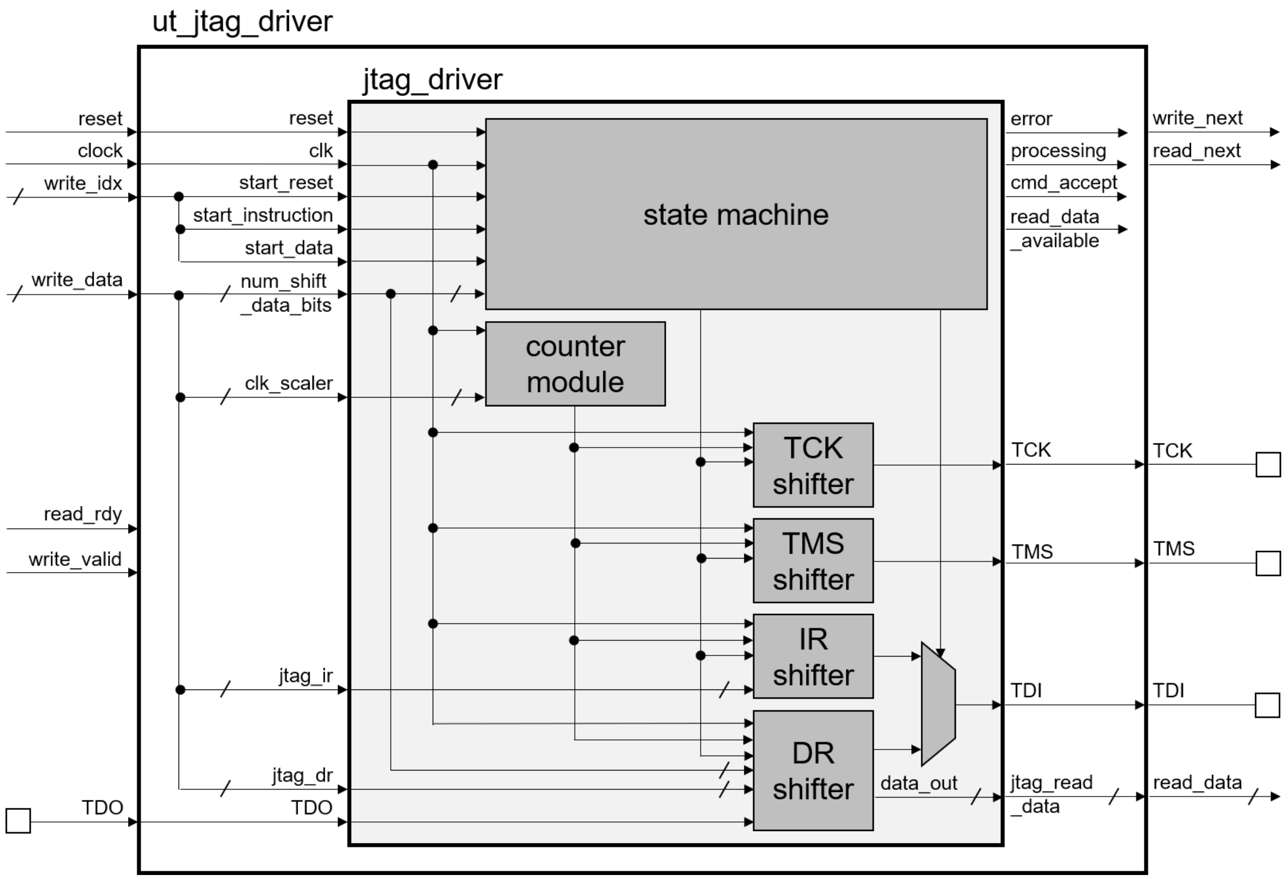

Figure 2: Simplified schema of the driver module structure (own presentation).

3 Design of a Test Bench for the Driver Module

3.1 Definition of Test Criteria

The purpose of the test bench is to verify the correct interaction between the driver module and the JTAG TAP and to identify any malfunctions that occur. The primary test criterion is the proper writing and reading of the TAP registers. This includes:

- Selection of the appropriate TAP register (IR or DR).
- Shifting of the specified number of bits of the desired bit sequence into the selected register while maintaining the initial bit order.
- Correct display of the data bits being shifted out of the TAP DR.

Furthermore, it is necessary to check whether the TAP Controller can be reset and the TCK frequency can be changed. The proper functionality of the driver module must be guaranteed for different IR and DR widths as well as for different data alignments and TCK frequency settings. It is thereby assumed that the JTAG TAP itself works error-free with every configuration.

3.2 Design of the Test Bench

The test bench must be suitable for testing the functionality of the driver module for all possible configurations. It is therefore designed in such a way that the width of all data buses and the length of the bit vectors to be transmitted are adapted according to the specified parameters. The completed test bench will be composed of the following sections:

1. Parameter specification: IR and DR width, the scaler value and other parameters are determined.
2. Data structure and function definition: Different data structures for the communication with the driver module and various functions that are required for the state machine are defined.
3. Driver module and JTAG TAP integration: The internal signal lines of the test bench are connected to the appropriate input and output ports of the driver module and the TAP.
4. Test program: The system clock is generated for a specified test runtime and the reset times are defined. Display of information on the test configuration during a test run is initiated.
5. State machine: Different states, state transitions and the data to be generated and sent in each state are defined.
6. Functional check: The proper functionality of the driver module is verified by comparing the data which was sent to the module and the TAP with the data which was written into and read out from the TAP registers.

The parameters can be specified by indicating concrete values or by randomly choosing a value from the permitted range. For the final test of the driver module, random parameter generation will be used. The parameter values are thereby only pseudo random numbers, as they are generated based on a certain seed value. In this way, it is possible to reproduce a random test run with a specific parameter configuration simply by indicating the same seed value as used for the initial test run. The possible values for the IR width are specified in the TAP data sheet and range from 2 to 32 [1]. In order to avoid the random generation of a mandatory instruction which would trigger an unintended TAP behaviour, the permitted range for the IR width is reduced to 3 to 31. The DR width is determined by the IC to be analyzed. For the test of the driver module, a test data register as mentioned in section 2.1 will be defined instead of accessing real physical boundary scan circuitry. Although the width of such a test data register can be arbitrarily selected, a permissible range from 3 to 64 is defined. The scaler value can range from 0 to 2^{scaler_width} - 1 where *scaler_width* is the width of the data bus transmitting the scaler value to the *clk_scaler* input of the *jtag_driver*. This width is set to 8, so that the scaler value can adopt all values between 0 and 255. For the data alignment, a range from 1 to 32 is defined. In addition to these parameters, a so-called *active_probability* and a *ready_probability* are randomly generated. They will be used by the state machine for generating idle periods of random length between consecutive sending procedures of the test bench.

For the communication with the driver module, the following data structures are defined:

- *data*: This data structure consists of the components *keep_response*, *len* and *payload*. The *keep_response* component always comprises a single bit indicating whether the data being sent back to the test bench is to be read or not. The *len* component specifies the number of bits to be shifted into the TAP DR. The minimum number of bits to be shifted is 3, which is due to the structure of the driver module. The largest possible number corresponds to the selected width of the test data register. The *payload* component finally contains the actual bit sequence to be shifted. The length of this sequence is always equal to the selected DR width.
- *transaction*: This data structure contains all information for a complete transaction, comprising an instruction for the TAP IR (included in the *ins* component), the data for performing a DR shift operation (in the form of the *data* data structure) as well as the data which is loaded into the test data register and supposed to be sent back to the test bench (included in the *response* component). The instruction length thereby corresponds to the selected IR width whereas the length of the *response* component is determined by the DR width.
- *raw_data*: This data structure contains the same information as *data*, but combined to one bit vector taking the selected data alignment into account.
- *write_data*: This data structure is used for transmitting data to the driver module. *write_data* is a bit vector like *raw_data* and contains either an instruction, a *raw_data* vector or a scaler value. Consequently, it has the length of the longest bit vector to be transmitted.

As with the parameters, the transaction components are randomly generated on the basis of the selected seed value. For this purpose, a *randomize* function is defined. The range for the permitted instruction values is thereby again limited to 3 to 2^{IR_width} - 2 in order to avoid the random generation of a mandatory instruction. For the generation of the *raw_data* vector, a *serialize* function is defined.

The control and monitoring of the driver module and the TAP is only possible by connecting their input and output ports to the corresponding internal signal lines of the test bench. In addition to the control line ports, the TAP, like the driver module, is equipped with further ports that provide information on the current TAP status. This includes an *instructions* port displaying the current content of the TAP IR and other ports which indicate the current state of the TAP Controller as well as the imminent beginning or the end of a DR shift operation. They will be used in the course of the functional check of the driver module.

The test runtime, the reset times and the information displayed during a test run are specified in the *test program* section of the test bench. The runtime can be set to arbitrary values. The test is designed so that the internal state machine of the *jtag_ driver* is reset directly at the start of each test run.

The state machine of the test bench comprises 4 states: *scaler*, *init*, *ins* and *data*. Depending on the current state, different data is generated and sent to the driver module. In the *scaler* state, the *write_ idx* bus is fed with a binary 1, signaling the transmission of a scaler value to the driver module. At the same time, the randomly determined scaler value is sent to the module via the *write_ data* bus. In the *init* state, a binary 0 is sent to the module via *write_ idx* what is supposed to trigger the transition of the TAP Controller into the *Test Logic Reset* state. In the *ins* state, a random transaction is generated by calling the *randomize* function. While a binary 2 is transmitted via *write_ idx*, *write_ data* is fed with the instruction of the generated transaction. In the *data* state, *write_ idx* sends a binary 3 announcing data for a DR shift operation. The corresponding *data* component of the transaction is converted to a *raw_ data* vector by calling the *serialize* function and then transmitted to the driver module via *write_ data*. Test runs always start in the *scaler* state in order to set the TCK frequency for the run. From there, the state machine passes through all other states in the abovementioned order up to the *data* state. It then switches periodically between the *ins* and the *data* state, thus alternately initiating TAP IR and DR shift operations with randomly varying transactions. For more realistic testing, an *active* signal is defined triggering the generation and transmission of new data. This signal is generated randomly, whereby the probability of generation is given by the *active_probability* which was also determined randomly based on the selected seed value. In this way, idle periods of random length between consecutive sending procedures of the test bench are created. An additional variation factor is given by the randomly selected *keep_ response* bit of the *raw_ data* vector which indicates whether the data being sent back from the TAP DR via the driver module is to be read or discarded. If the data is supposed to be read (*keep_response* = 1), it must not be sent back by the module until the test bench is ready to read it. This decision is also made at random (based on the *ready_probability*) and communicated to the driver module via the *read_ rdy* line. If the data is supposed to be discarded (*keep_response* = 0), it is immediately shifted out of the TAP DR and sent to the test bench by the module by shifting new data into the DR. Once, the test bench is ready to read, this readiness is maintained until it has received data.

For the functional check, a test data register with the randomly determined DR width is defined. Its foremost bit is connected to the TDI input of the TAP while its rearmost bit is connected to the TDO output of the TAP. Since, apart from the mandatory decoded instructions, no instruction decode logic is defined and no boundary scan circuitry is connected for the tests, the test data register is automatically accessed when the TAP Controller is located in the DR branch. In this case, the bits entering the TAP via the TDI input are directly forwarded to the test data register while those being shifted out of it leave the TAP via the TDO output. If, however, the TAP Controller is located in the IR branch, the TDI bits are forwarded to the instruction register instead. Accidental access to another data register than the test data register (e.g. to the Bypass Register) caused by the driver module sending a mandatory instruction can be excluded, as the specified ranges for the IR width and the instruction value do not allow the random generation of these instructions. The connection of the test data register to the TDI and TDO port of the TAP makes it behave like a normal data register contained within the TAP. However, since the register is defined in the test bench, it is also possible to access all register cells simultaneously from the test bench. This feature enables the implementation of the following functional checks:

- Check of correct TAP IR write operations: The transactions which are randomly generated in the *ins* state, comprise an instruction, the data for a DR shift operation and a *response* component. The instruction of a newly generated transaction is always directly sent to the driver module before in the following state, which is always the *data* state, the corresponding data for the DR shift operation is sent. Consequently, shortly before the start of a DR shift operation, the TAP IR must contain the instruction of the last transaction that was generated. At this moment, which is signaled to the test bench via the corresponding TAP status port, the content of the TAP IR displayed via the *instructions* port is therefore compared to the instruction of the lastly generated transaction. If the driver module has worked correctly, the comparison must be positive. At the same time of the comparison, the *response* component of the transaction is written into the test data register. This step is necessary in order to be able to check for correct TAP DR read operations.

- Check of correct TAP DR write operations: During the DR shift operation, the number of shifts is checked by incrementing a count variable at every positive edge of the TCK. The end of the operation is signaled via another TAP status port. The count variable should now correspond to the number of bits to be shifted which was specified by the *len* component of the *data* part of the generated transaction. Furthermore, this number of bits of the *payload* component of the *data* part should be contained in the left section of the test data register. These two aspects are checked for verifying the correct writing of the TAP DR.
- Check of correct TAP DR read operations: While a certain part of the *payload* component is shifted into the test data register during the DR shift operation, an equally large part of the *response* component, which was written into the register shortly before the start of the operation, is shifted out via TDO. At the end of the operation, which is signaled by the *read_next* port of the *ut_jtag_driver*, these *response* bits should be displayed at its *read_data* port. If yes, the DR read operation was correct.

The described verification principle is visualized in figure 3. The figure shows the register contents after a DR shift operation. The transaction containing all relevant data is thereby abbreviated with *trans*.

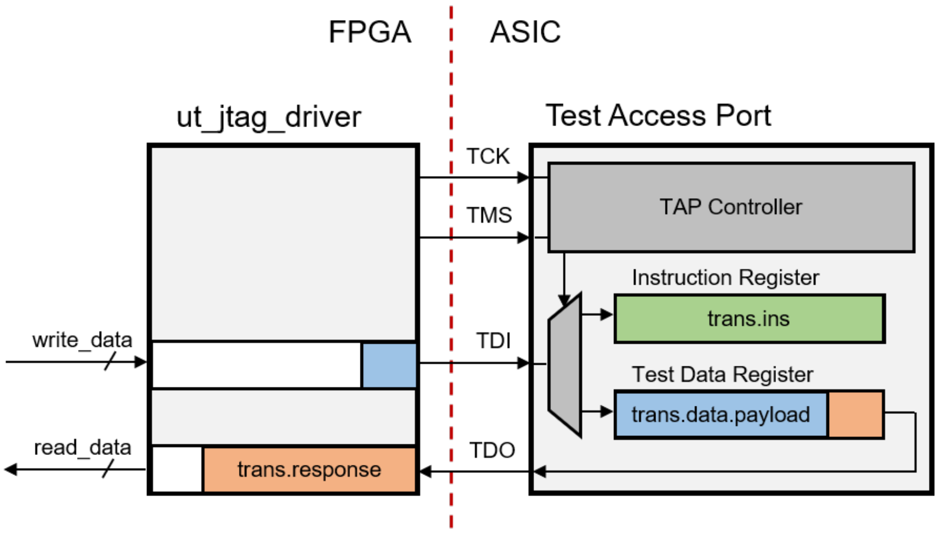

Figure 3: Test bench verification principle. Shown are the register contents after a DR shift operation (own presentation).

4 Test and Debugging of the Driver Module

4.1 Test Results Before Debugging

With the help of the designed test bench, the proper functionality of the driver module can be verified for one configuration (i.e. for one specific combination of IR width, DR width and scaler value) per test run. In order to verify its correct functionality for all possible configurations, a corresponding number of test runs was performed. In the first part of the test phase, the IR and DR width were randomly varied within the permitted ranges while the scaler value was permanently set to 0 thus generating a TCK signal with half the frequency of the system clock of the driver module. The results of 10.000 test runs are illustrated in figure 4. The color hereby represents the ratio of correct to faulty runs for a specific configuration. Furthermore, the intensity of a color indicates how many test runs were performed with a certain configuration related to the most frequently tested configuration. A weakly colored square thus indicates that the corresponding configuration was tested much fewer times than at least one other configuration. Whereas the majority of configurations produced correct results, there is also a triangular area of adjacent configurations which almost exclusively exhibit faulty test runs. Only the columns with a DR width of 4, 8 and 16 seem to show partly correct and partly faulty runs.

In the course of the debugging process, a seemingly working configuration with a conspicuously low number of transactions compared to the duration of the test run was detected. Therefore, the number of transactions per test run dependent on the IR and DR width was also examined. In order to guarantee a constant number of transactions for a specific configuration, the *active_probability* and the *ready_probability* were permanently set to 1. The results of this analysis are illustrated in figure 5. The diagram confirms the detected abnormality showing irregularities for all configurations whose DR width corresponds to a power of 2. These are the same

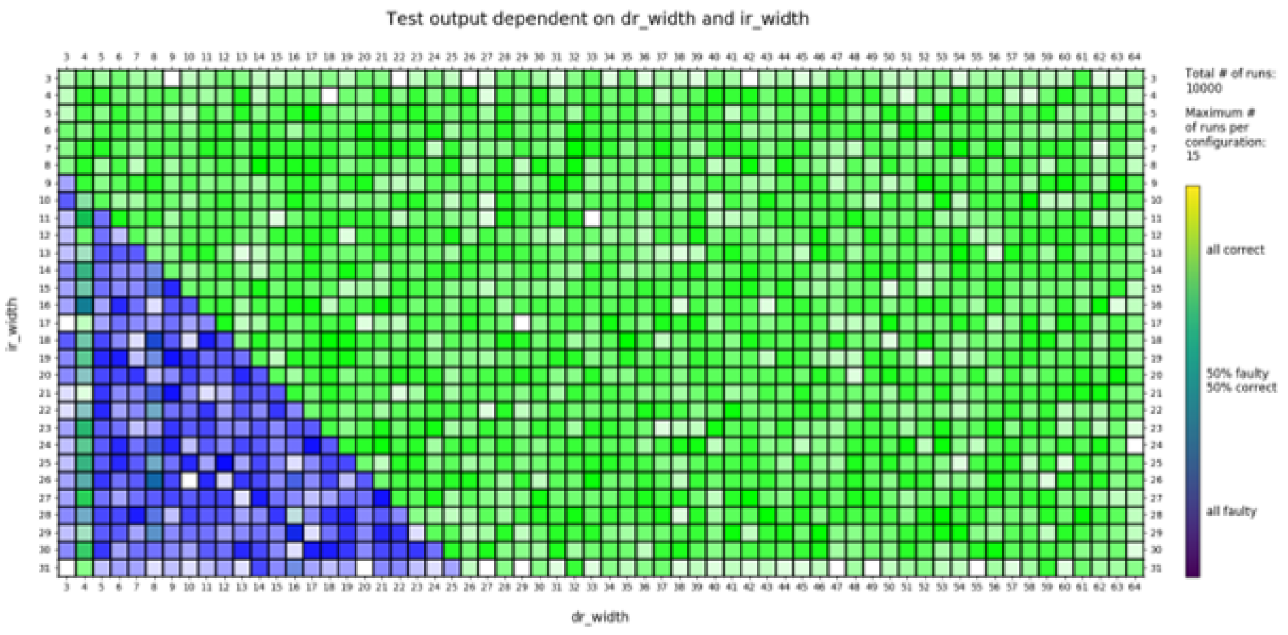

Figure 4: Ratio of correct to faulty test runs dependent on the DR and IR width before debugging (own presentation).

DR widths which had already shown partly correct and partly faulty test runs in the previous analysis. Obviously, the correct test runs were hereby produced by signal paths which had ended before they could reach the functionality checks of the test bench. Apart from the irregularities and the triangular faulty region, the diagram shows the expected behaviour, meaning that the average number of transactions per test run decreases with increasing IR and DR width.

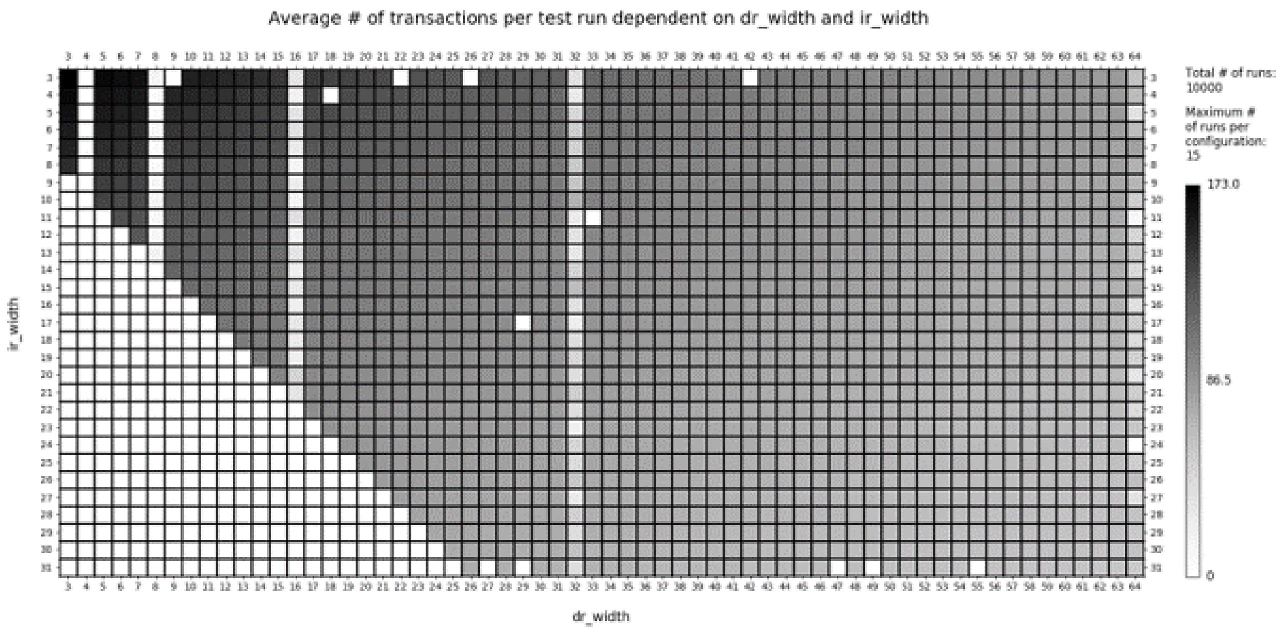

Figure 5: Average number of transactions per test run dependent on the DR and IR width before debugging (own presentation).

In the second part of the test phase, the scaler value was randomly varied within the permitted range while the IR and DR width were constantly set to 3. The results of 10.000 test runs are illustrated in figure 6. The diagram shows the number of correct and faulty test runs dependent on the scaler value. As can be seen, only a value of 0 or greater than approximately 195 produced exclusively correct results.

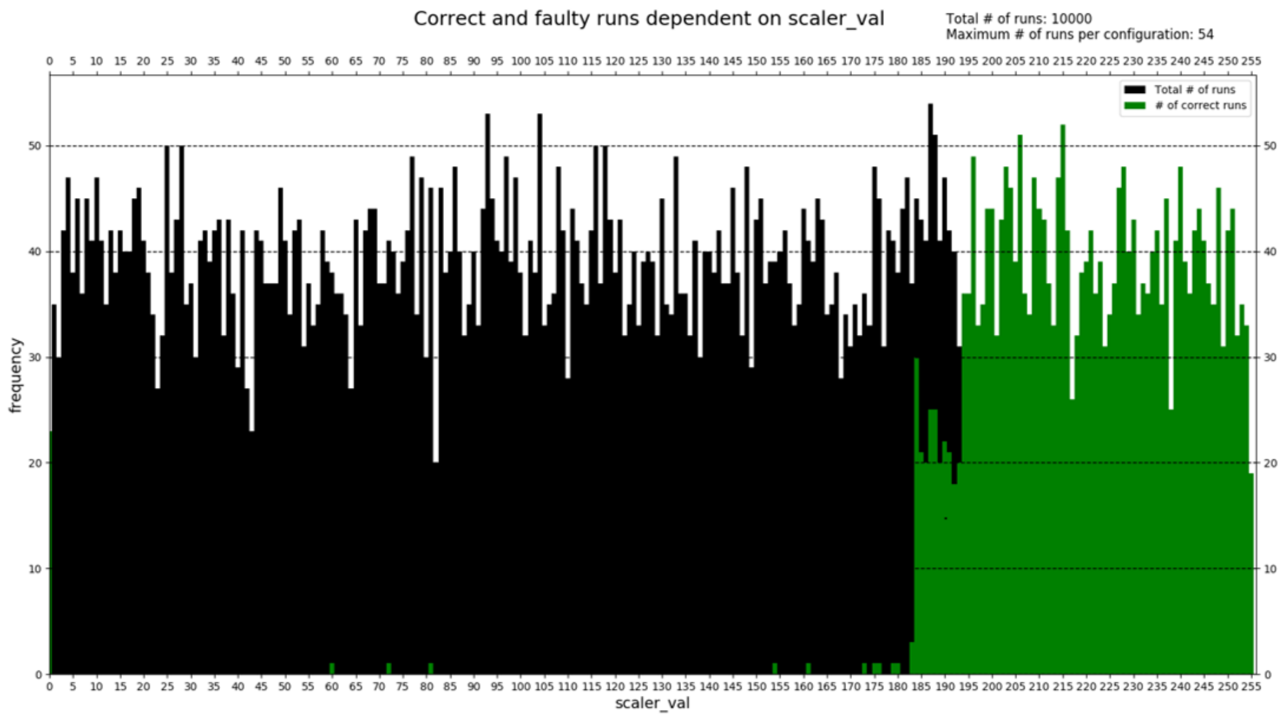

Figure 6: Number of correct and faulty test runs dependent on the scaler value before debugging (own presentation).

4.2 Debugging

The low number of transactions for DR widths corresponding to a power of 2 was caused by a false definition of an input port and a register of the *jtag_driver*. On the one hand, the width of its *num_shift_data_bits* input port was chosen one bit too small, so that the most significant bit of the vector sent by the *ut_jtag_driver* was always discarded. Whenever the latter sent a bit vector whose value corresponded to a power of 2, the *jtag_driver* only received a bit vector consisting of zeros. This caused the internal state machine of the *jtag_driver* to stop in an intermediate state in which it remained until the end of the test run. Consequently, no further transactions were executed. On the other hand, the width of an internal register used for counting the number of DR shifts was also one bit too small, so that values corresponding to a power of 2 could never be reached. This also prevented the state machine from getting back into the idle state and therewith from executing further transactions. The width of the *num_shift_data_bits* port and the internal register could easily be corrected.

The analysis of the triangular faulty region revealed that tests were failing whenever the inequation

$$IR\_width > DR\_width + 5 \tag{1}$$

applied. This information could be used to find the source of error which was a wrong width of the *jtag_driver* registers used for generating the TCK signal. This signal is generated by the registers alternately shifting out ones and zeros until the ones are consumed. The width of these registers was only dependent on the DR width, assuming that a DR shift operation would always take longer than an IR shift operation. If, however, the IR width is larger than the DR width, so that the abovementioned inequation applies, the number of TCK cycles required to traverse the IR branch of the TAP Controller is larger than the number required to traverse its DR branch. In this case, the TCK stopped before the IR branch was left, causing a transition into the *Pause-IR* state instead of the *Update-IR* state. The following TMS sequence of the *jtag_driver* then moved the TAP Controller back into the *Shift-IR* state instead of the *Shift-DR* state. While the state machine of the *jtag_driver* then started a DR shift operation, the TAP Controller still accessed the IR. Consequently, the TAP IR was filled with the content which was intended for the DR whereas the DR content was not changed. The subsequent DR write and read checks thus returned faulty results. This problem was solved by implementing an additional dependence of the register widths from the IR width.

The malfunctions that occurred due to changes of the scaler value were caused by insufficient conditions for some state transitions of the state machine of the *jtag_driver*. Due to these insufficient conditions, the *jtag_driver* stopped its IR and DR shift operations before all relevant data bits were shifted into the corresponding register. This problem could be fixed by an appropriate adjustment of the transition conditions and the introduction of an additional counter.

4.3 Test Results After Debugging

After the debugging of the *jtag_ driver* code, the functionality of the driver module was tested again for different settings of the IR and DR width and the scaler value. The test results are shown in the following figures. The yellow squares in figure 8 hereby represent configurations which have not been tested. As can be seen, the driver module now works properly for all tested configurations.

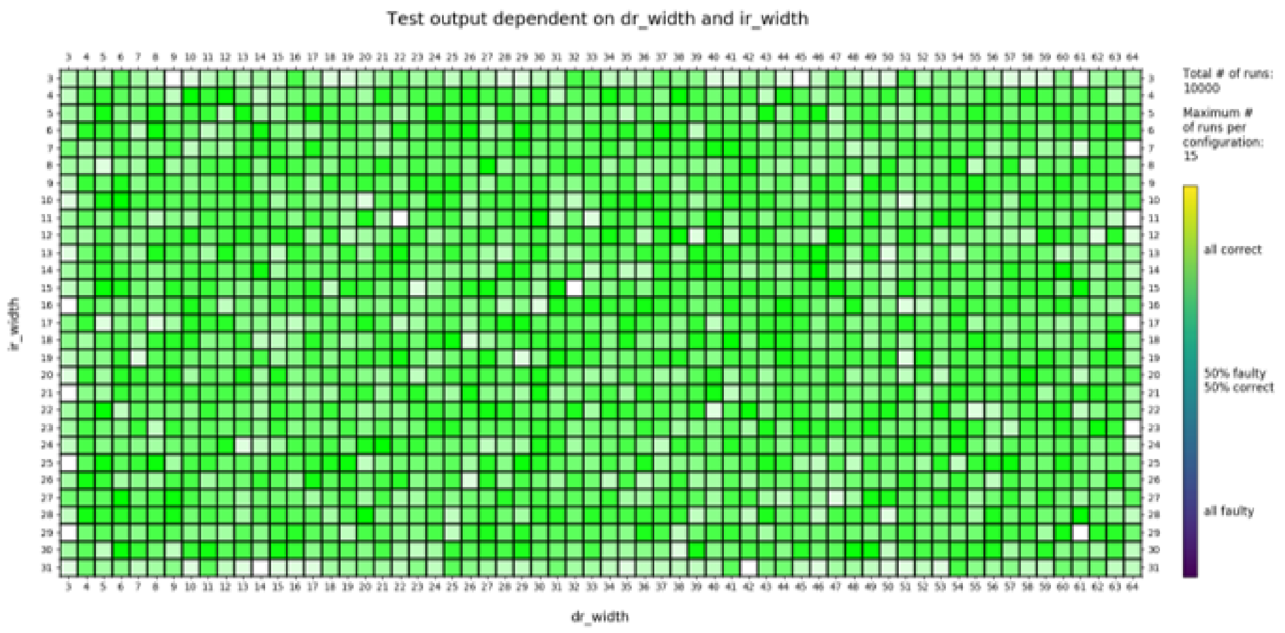

Figure 7: Ratio of correct to faulty test runs dependent on the DR and IR width after debugging (own presentation).

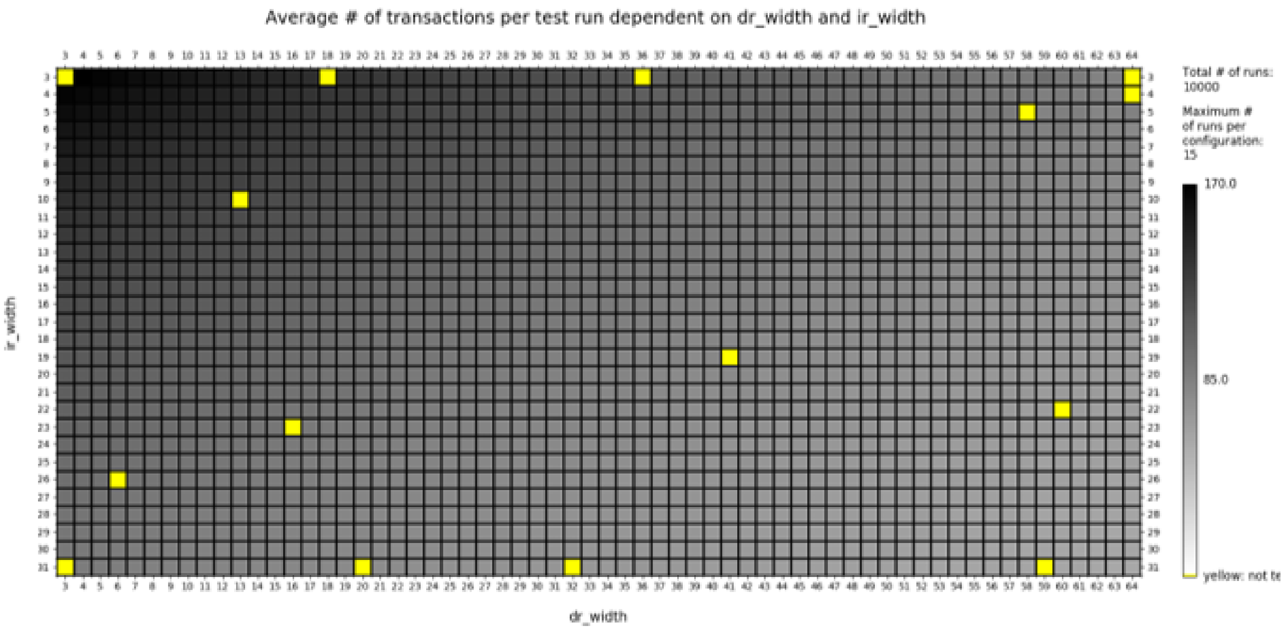

Figure 8: Average number of transactions per test run dependent on the DR and IR width after debugging (own presentation).

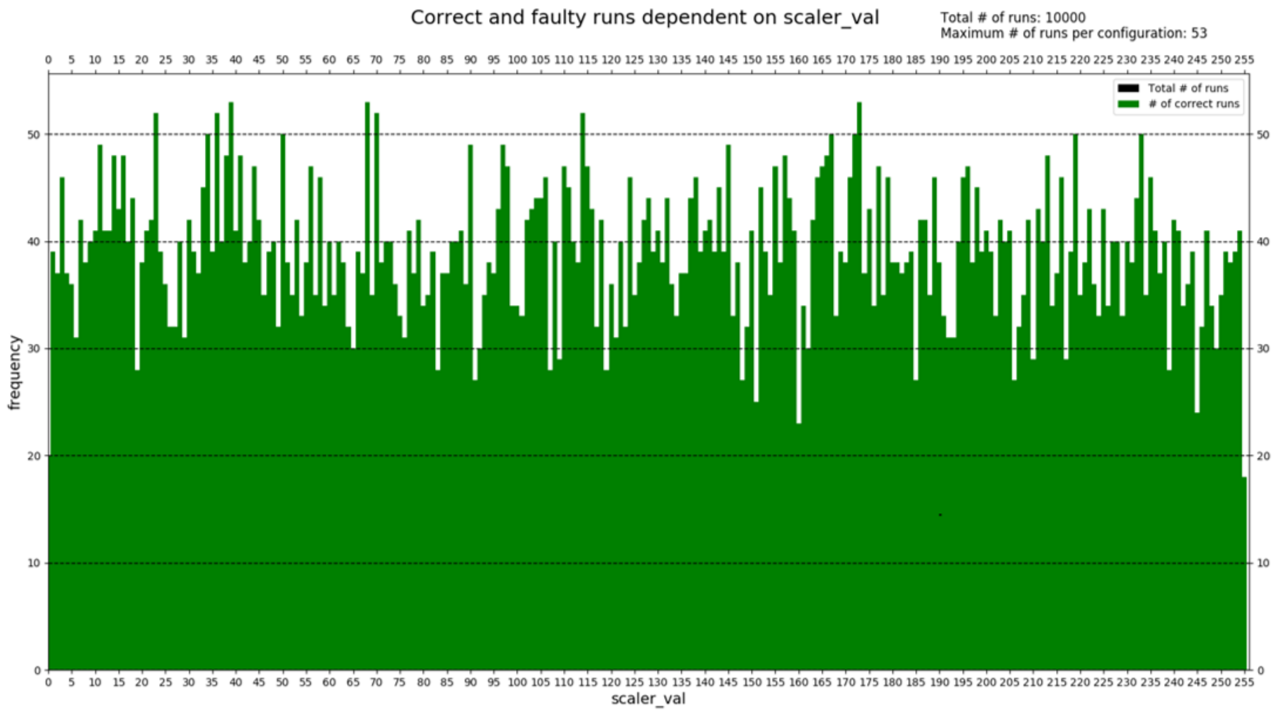

Figure 9: Number of correct and faulty test runs dependent on the scaler value after debugging (own presentation).

5 Conclusion and Outlook

The goal of the project was to verify the correct functionality of a parameterizable driver module for a JTAG Test Access Port at all permitted configurations. For the verification, a parameterizable test bench was designed. With the help of the test bench, different tests were then carried out which revealed various malfunctions of the driver module. These malfunctions could be fixed in the course of a comprehensive debugging process. Whereas the driver module was only used in a specific configuration before the project, its correct functionality is now guaranteed for every possible configuration. It thus can be used for different projects such as the development of the HICANN-X chip.

However, the driver module currently doesn't provide the option to enter the *Pause-DR* state between consecutive DR shift operations. Consequently, it is not possible to read out the states of all logic signals of an IC at a given moment by initiating various consecutive DR shift operations, since the initial signals will be overwritten by new ones whenever the TAP Controller traverses the *Capture-DR* state while reentering the *Shift-DR* state. It is therefore recommended to extend the driver module in such a way that a transition from the *Shift-DR* into the *Pause-DR* state and vice versa can be made.

References

[1] Synopsis, Inc. *DW_tap: TAP Controller*, June 2018.

[2] Wikipedia. *Joint Test Action Group.* https://de.wikipedia.org/wiki/Joint_Test_Action_Group, August 2018.

[3] Wikipedia. *JTAG.* https://en.wikipedia.org/wiki/JTAG, October 2018.