A Chisel Generator of JTAG to Memory-Mapped Bus Master Bridge for Agile Slave Peripherals Configuration, Testing and Validation

Vukan D. Damnjanović, *Student Member, IEEE*, and Vladimir M. Milovanović, *Senior Member, IEEE*

***Abstract*—This paper presents a design of a JTAG to memory-mapped bus master bridge generator implemented using Chisel hardware design language. This type of digital module can provide convenient and practical means of configuring a wide range of peripheral circuitry with a memory-mapped slave interface attached to a bus interconnection, as well as of their testing and debugging. The peripherals can be configured by driving the input signals of the JTAG interface with the values that represent the previously defined instruction codes, thus initiating write or read data transactions on the interconnect bus through the master interface to their memory-mapped registers. The master interface can be either AXI4 or TileLink, depending on the characteristics of the whole system which the depicted bridge is a part of. The proposed generator offers the ability of creating slightly different modules by using different parameter selections. The implemented design has been extensively tested using various software simulations with a number of different slave peripherals and mapped and tested onto a commercial FPGA platform. These actions experimentally confirmed the previously made assumption of the utility and convenience of the proposed generator.**

***Index Terms*—JTAG to memory-mapped bus master bridge, AXI4 and TileLink protocols, memory-mapped interface, peripherals testing and debugging, Chisel hardware design language, design generator.**

I. Introduction

From the very beginnings of the integrated circuits and the emergence of the first microprocessors, memories and data storing were emphasized as one of the most important and vital parts of its structure because of a vast number of possibilities and numerous functions it provided. Almost all digital applications and devices were able to develop and operate on its basis. As the time passed by, with the improvement of the existing technologies and the emergence of the new ones, along with the development of microprocessors, those applications and devices started to become more and more complex and sophisticated as well. Therefore, not only that the limited capacities of the devices' memories appeared to be the major problem, but the ways of accessing their data were too, usually due to a need for the standardized methods or high performance criteria of the systems. Several ways of a microprocessor data access were developed over the years, with the usage of the port-mapped input/output (PMIO) and the memory-mapped input/output (MMIO) interfaces [1] being the most common ones.

The main characteristic of the port-mapped input/output data access interface is the presence of special address space outside the common system memory for every included peripheral. Usually, that implied the existence of special, dedicated instruction set for data access, such as "IN" and "OUT" instructions in x86 architectures [2]. The PMIO was more extensively utilized in earlier digital systems with less developed microprocessors with small address spaces, since the valuable resources were not consumed by the input/output (IO) devices. However, sometimes it is not convenient to use this kind of data access because of the possible frequent context switching or the need for the manipulation of IO devices using only standardized memory access instructions. Those features are delivered by using the memory-mapped input/output interface.

As mentioned before, systems with the MMIO have a shared virtual address space, along with the program memory or user memory, with the same instruction set for accessing it. All the devices are attached to an interconnect bus and from the perspective of the microprocessor, there is no difference whether it manipulates with the peripheral I/O device, or some internal data. Therefore, a wide range of different peripherals with the memory-mapped registers can be integrated in the system without almost any additional logic, thus allowing it to grow plentifully in terms of its functionality. Having this in mind, it is no wonder that modern-day systems more and more rely on this version of peripheral device's data access.

In the previous couple of paragraphs, the characteristics and the importance of the MMIO interface within the microprocessor-based systems were elaborated. For each one of those systems, manipulating the peripheral devices by accessing their data should be well-explained and straightforward. However, sometimes there is a need to manipulate or test those devices without implicating the microprocessor. In those cases, accessing the interconnect bus and initiating data transactions could be very challenging and complex. For that particular reason, the JTAG to memory-mapped bus master bridge generator from this paper's topic was designed and created. It allows a user to access the peripheral device connected to either AXI4 [3] or TileLink [4] bus without the engagement of any kind of processing unit, but by using the standardized and quite popular JTAG interface [5], [6].

Vukan D. Damnjanović is with NOVELIC d.o.o., Veljka Dugoševića 54/B5, 11060 Belgrade, Serbia (e-mail: vukan.damnjanovic@novelic.com).

Vladimir M. Milovanović is with the Department of Electrical Engineering, Faculty of Engineering, University of Kragujevac, Sestre Janjić 6, 34000 Kragujevac, Serbia (e-mail: vlada@kg.ac.rs).

This paper, along with the quick overview of the used protocols and detailed description of the design of the JTAG to memory-mapped master bus bridge and its implementation with the used Chisel libraries, also serves as the user manual of the module and depicts the obtained results through simulations and hardware implementation.

II. A JTAG to Memory-Mapped Bus Master Bridge, Its Interfaces and Instructions

A JTAG to memory-mapped bus master bridge [7], [8] is a digital component which initiates data transactions with the peripheral devices possessing the memory-mapped input/output interface, attached to an interconnect bus. This module communicates with the outer world through two ends and three possible different interfaces. The user sends the desired instruction by driving the signals on the user-end JTAG interface, present in all the variants of the module. The instruction, if performed correctly, then initiates the appropriate transaction on the bus-end interface, which can be either AXI4 or TileLink, depending on the user's preferences or the system requirements. In the next few paragraphs, a brief overview of these interfaces is provided.

A. JTAG Interface

The Joint Test Action Group (JTAG) is a standardized four-wire serial protocol usually used for testing and debugging integrated circuits through a JTAG port. It consists of three input signals and an output signal: Test Clock (TCK), which is used as a clock signal for a JTAG controller and is independent of system clock signal; Test Mode Select (TMS), an input signal which serves as a control signal for a JTAG finite state machine (FSM), which will be discussed later; Test Data Input (TDI), an input signal that represents the input serial data for JTAG instruction and data registers; Test Data Output (TDO), the serial output data for JTAG instruction and data registers (an additional output signal exists to express the validity of the output data on the TDO pin). The on-chip JTAG Test Access Port (TAP) implements the mentioned FSM, which is used for the realization of the JTAG subpart of the module.

The JTAG FSM is used to correctly accept the JTAG interface signals and to recognize whether the arrived data values are the instruction values or the data values itself. It is driven by the rising edge of the TCK signal and the state changing is controlled by the TMS signal. In certain FSM states, data from the TDI port is captured. From the idle state, by driving the TMS signal in the appropriate way, the user chooses to enter either the select data or the select instruction state and then the data/instruction capture state. From that point on, the procedure is identical for both instruction and data capturing. The only difference is that the captured values are stored in different registers. In the data/instruction shift state, values from the TDI port are stored in a shift register as long as the FSM stays in that state. Afterwards, the FSM enters the update data/instruction state and then returns to the idle state, with the complete data/instruction value stored in the appropriate register. A block diagram of the complete JTAG FSM with all the state changes and the TMS signal values is shown in Fig. 1.

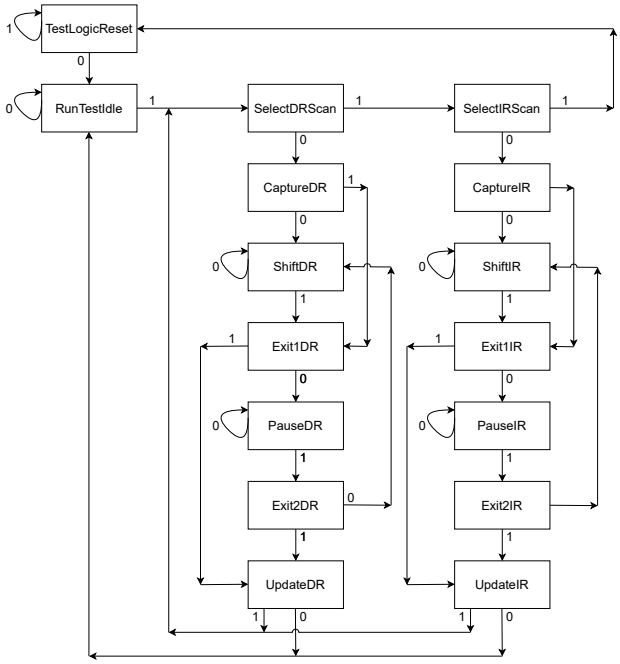

Fig. 1. A block diagram of the complete JTAG finite-state machine with all the state transitions and the TMS signal values.

B. AXI4 and TileLink Interfaces

Advanced eXtensible Interface 4 (AXI4) protocol is a parallel, synchronous, high-frequency multi-master and multi-slave communication interface. It is tailored mainly for the on-chip communication, which makes it suitable for the systems mentioned above. AXI4 interfaces consists of a vast number of different signals, with many of them optional, making it a versatile interface applicable to various different systems and applications. Even though it is described as multi-master and multi-slave interface, in every transaction only a single master and single slave communicate with each other. All the masters and the slaves are mutually connected through an interconnect bus. In the JTAG to memory-mapped bus master bridge module, this interface is used as a master interface and in most applications that include it, because of its sole purpose, the only active master is the module itself, whereas one or more slaves could exist. A simplified block diagram of an example of such a system is shown in Fig. 2. AXI4 interface protocol consists of 5 different channels: write address (AW), read address (AR), write data (W), read data (R) and response (B). Every one of those channels work on the handshaking principles and thus contain the pair of ready/valid signals. All the channels except the response channel have data signal among the various others signal whose function is to additionally describe and secure successful data transactions. Apart from the single read and single write data transactions, AXI4 interface supports the burst read and burst write data

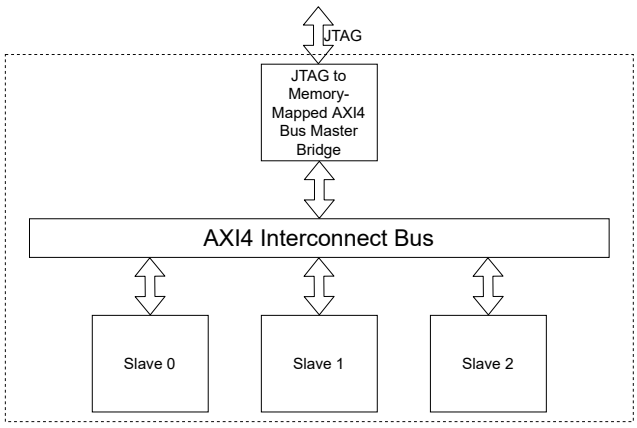

Fig. 2. A simplified block diagram of a system with a JTAG to memory-mapped bus master bridge and three slave peripherals attached to an AXI4 interconnect bus.

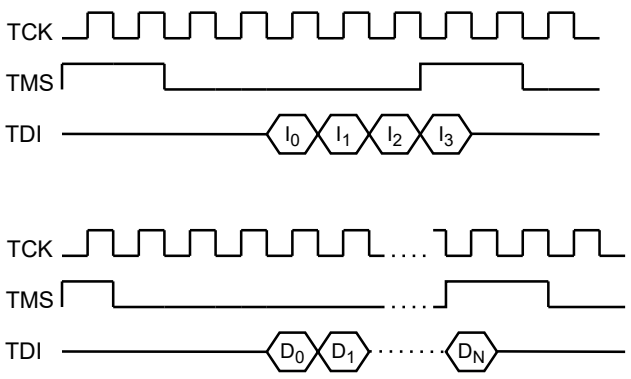

Fig. 3. JTAG input signal diagrams for correctly sending an instruction code (upper diagram) or a data value (bottom diagram) when the JTAG finite-state machine is in the idle state.

transactions.

TileLink is a parallel, synchronous, high-frequency multi-master and multi-slave communication protocol, in some extent similar to the AXI4 protocol. It is also designed mostly for the on-chip communication, with a special emphasis on the cache coherence transactions. The communication between a master and a slave (sometimes called a client and a manager in terms of this interface) is also performed based on the handshaking protocol. Overall, five communication channels can exist, with only two mandatory: channels A and D are mandatory, while channels B, C and E are optional. Each of the channels consists of several signals, including a data signal, a couple of ready/valid signals and some other signals used to describe and control transactions. The mandatory channel A flows from master interface to slave interface, carrying request messages sent to a particular address. Then, the slave responds to the master's request through the mandatory channel D. Other channels B, C and E are optional, and they are utilized for complete TileLink cached protocol, where channels B and C have similar functions as channels A and D respectively, whereas channel E is used as a final acknowledgment channel. In the module from the topic of this paper, however, only mandatory channels are used, and therefore, not much attention is provided to other three channels. TileLink protocol interface can be used as the master interface of the module instead of the AXI4 interface.

C. Defined Instructions

As it can be concluded from the previous paragraphs and sections, the JTAG to memory-mapped bus master bridge is a module that integrates two different interfaces with its dedicated controllers: JTAG and AXI4/TileLink. Those two controllers operate independently, they are even driven by different clock signals (TCK for JTAG controller and system clock signal for the AXI4/TileLink controller), but they are mutually synchronized through the internal signals. The JTAG controller works on the basis of the previously described JTAG FSM, while AXI4/TileLink has its own FSM. The whole module has the following, rather simple, signal flow. User calls an instruction by driving the serial JTAG input signals. The JTAG controller accepts the data, converts it to the parallel form and sends it to the AXI4/TileLink controller who, if it is recognized as a write instructions or a read instruction or any subvariant of them, initiates the transaction between the module and the appropriate slave. JTAG input signal values for correctly sending an instruction code or a data value when the JTAG FSM is in the idle state are shown on the timing diagrams in Fig. 3. In order to send the data value to the serial TDI input correctly, the least significant bit of the data should be sent first. It is strongly recommended that, prior to using the module, the user ensures that the JTAG FSM enters the reset state. It is achieved by driving the TMS signal with the active high value for five straight TCK cycles. By doing so, no matter what state was the JTAG FSM in, it will enter the desired reset state.

Total of four types of data transactions can be initiated by the AXI4/TileLink master interface on the interconnect bus: write, read, burst write and burst read. Also, nine instructions that the user can call through the JTAG interface are defined. The purpose of them is to enable the data transactions to be performed and to pass all the information needed for it. Each instruction can require either both instruction code and data itself to be provided, or just the instruction code. Both of them are captured in their dedicated JTAG FSM state. After the input serial data arrived entirely to the JTAG controller, it sends both the instruction code and the data to the AXI4/TileLink controller. A list of defined instructions, along with their codes and descriptions, is the following:

- 0x01 - Write instruction, initiates the AXI4/TL controller to begin writing the acquired data to the acquired address.
- 0x02 - Address acquire instruction, accepts the serial data as the address for the read/write instruction.
- 0x03 - Data acquire instruction, accepts the serial data as the data for the read/write instruction.
- 0x04 - Read instruction, initiates the AXI4/TL controller

to begin reading data from the acquired address.

- 0x08 - Number of burst transactions acquire instruction, accepts the serial data as the number of the read/write instructions during one burst transfer cycle.
- 0x09 - Burst write instruction, initiates the AXI4/TL controller to begin performing acquired number of the write transactions. Data is written to the consecutive addresses.
- 0x0A - Data index number acquire instruction, accepts the serial data as the index number of data to be acquired using the following instruction for the burst read/write transfer.
- 0x0B - Indexed data acquire instruction, accepts the serial data as the data at the acquired index number for the burst read/write transfer.
- 0x0C - Burst read instruction, initiates the AXI4/TL controller to begin performing acquired number of the read transactions. Data is read from the consecutive addresses.

Before the write instruction, both the address acquire and the data acquire instructions must be performed. Before the read instruction, the address acquire instruction must be performed. For the burst write instruction, the data for every single transaction must be acquired beforehand, as well as the total number of burst transactions for both the burst write and the burst read instructions. Two read/write/burst read/burst write instructions of the same type cannot appear sequentially one right after another, there must be at least one other instruction between the two. After performing the read or burst read instruction, read data appear on the serial output JTAG TDO data port, with the TDO driven signal having the active high value. All the instruction codes that are not mentioned in this paper can be assumed to be the no-operation (NOP) instructions.

III. A Design Generator and Its Implementation

Previously depicted JTAG to memory-mapped bus master bridge have been captured inside Chisel 3 hardware design generator. Both solely and in a combination with numerous slave peripheral modules, the generator has been thoroughly tested using standard Chisel verification and implementation paths for FPGA design flow. The design generator is made available [9] for public use as a free and open-source hardware library.

The generated module itself consists of two main subparts: JTAG controller and AXI4/TileLink controller, with their interfaces and internal communication signals. A block diagram of the JTAG to memory-mapped bus master bridge with all its interfaces and two main submodules is depicted in Fig. 4.

The JTAG controller is the submodule that communicates with the user. Its main purpose is to accept the serial data from the JTAG user interface, pack it in the appropriate format and send it to the AXI4/TileLink controller, as well as to accept the parallel data read from the slave peripherals and to put it on the JTAG serial data output. The JTAG controller is based on the previously depicted, standard JTAG FSM. Besides the

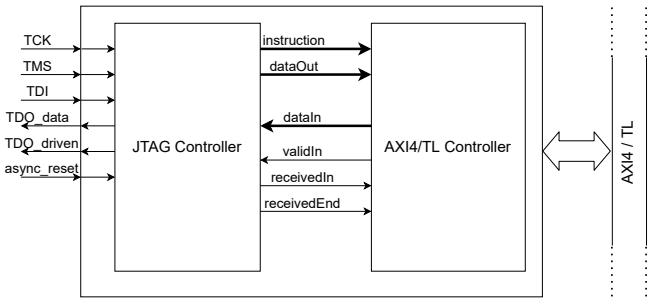

Fig. 4. A block diagram of the JTAG to memory-mapped bus master bridge with all its interfaces and the two main submodules.

common JTAG interface signals TCK, TMS, TDI and TDO, several more I/O signals exist. To begin with, TDO signal is divided into the one-bit-wide output signals: TDO data, which represents the serial output data, and TDO driven, which serves as the data valid signal. Those two signals are active at the falling edge of the TCK clock. There is also an asynchronous reset input signal which transits the JTAG FSM current state to the reset state.

Through the TDI JTAG serial input pin, the user can send either an instruction code or the data value itself. The JTAG can distinguish between those two thanks to the TMS control signals. Arrived data is translated from serial to parallel data format by using two shift registers, one for both instruction and data values. Data from the shift registers are stored into the two data registers, one at a time, when the JTAG FSM enters the appropriate data/instruction capture states. Data from those two registers are sent separately to the AXI4/TileLink controller.

A. The AXI4 and TileLink Controllers

The AXI4 and TileLink controllers are similar to one another, with the obvious difference in the master interface signals. They accept the instruction and data values from the JTAG controller, recognize the instruction code and take the action correspondingly. If the instruction code suits the either read or write instruction code, the controller initiates the communication with the appropriate slave peripheral through the AXI4/TileLink interconnect bus. The instruction code and data values are stored into the two separate registers. The value from the instruction register is constantly checked and compared to the instruction code of each of the four data transfer instructions (write, read, burst write, burst read). When those two values match, the appropriate flag value is set to the active high (a flag exist for every one of those four instructions) and that signifies that the appropriate instruction is set to be executed. Apart from signalizing that the instruction should be performed, the flag signals are used to prevent other instructions to be executed until the end of the current instruction. For the purpose of the burst transfers, a counter that counts the number of performed transfers is implemented inside the controller. Both controllers rely on their own FSMs

which secures the correctness of the communication with the peripherals.

The AXI4 and TileLink controllers also send the data read from the peripherals to the JTAG controller. For that purpose, several more internal signals exist. Apart from the one that carries data values to the JTAG controller, there are signal that marks the validity of the arrived data and two signals that mark that the JTAG controller has received the data and that the all bits of the data were sent to the output TDO pin.

The AXI4 controller FSM has the task to control the communication with the slave peripherals. State transitions are realized thanks to either flag values mentioned above, or the AXI4 signal values from the slave, such as ready signal for the handshaking protocol. Following states exist:

- `sIdle` - The idle state, the FSM stays in this state until the write instruction or the read instruction flag is set.
- `sSetDataAndAddress` - The state in which address is set on the AW channel, data is set on the W channel and valid signals are set on both the AW and W channels. The FSM stays in this state until the ready signals are not set on both the W and AW channels or until a counter which ensures that the FSM isn't stuck in this state counts out.
- `sResetCounterW` - The state in which the mentioned counter is reset. Stays in this state for exactly one clock cycle.
- `sSetReadyB` - The state in which the ready signal is set on the acknowledgement B channel. The FSM stays in this state until the B channel valid signal is not set or until the counter which ensures that the FSM is not stuck in this state counts out. The write instruction flag is reset in this state.
- `sSetReadAddress` - The state in which the address and the valid signals are set on the AR channel. The FSM stays in this state until the AR channel ready signal is not set or until the counter which ensures that the FSM is not stuck in this state counts out.
- `sResetCounterR` - The state in which the mentioned counter is reset. The FSM stays in this state for exactly one clock cycle.
- `sSetReadyR` - The state in which the ready signal is set on the R channel and data is read from the same channel. The FSM stays in this state until the R channel valid signal is not set or until the counter which ensures that the FSM is not stuck in this state counts out.
- `sDataForward` - The state in which the read data is forwarded to the JTAG controller, along with the active valid signal. The FSM stays in this state until the JTAG controller does not confirm that the data is received. The read instruction flag is reset in this state.

A state transition diagram for the AXI4 FSM is depicted in Fig. 5. Note that the states for the write and the read instructions only are shown. The reason for the deficiency of the other two is simply the similarity to the depicted ones. The only novelty for the burst transfers is the fact that after the completed single data transfer, FSM enters `sIdle` state only if the burst

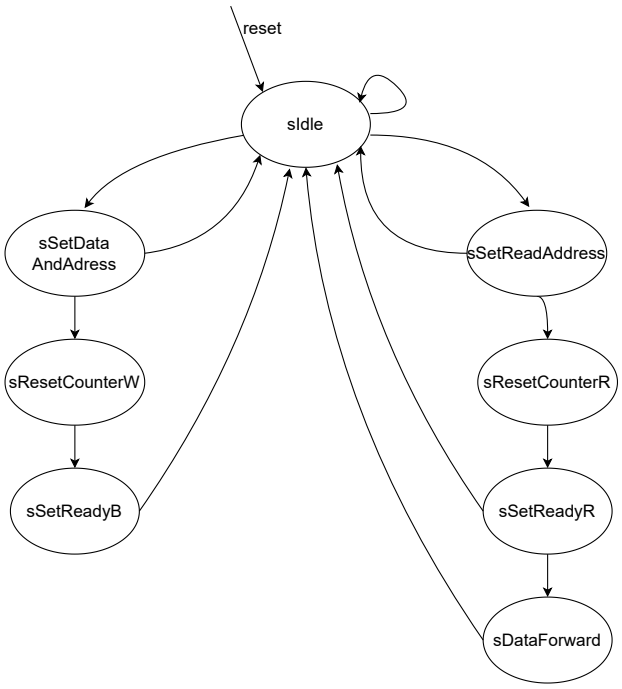

Fig. 5. A state transition diagram for the read and the write instructions of the AXI4 FSM.

transfers counter has counted out. Otherwise, the AXI4 FSM enters `sSetDataAndAddress/sSetReadAddress` state to perform another transfer.

The TileLink controller FSM has the same task as the AXI4 FSM. Its state transitions are also realized thanks to either the flag values or the signal values received from the slave peripheral. Although the states themselves are not identical to the ones from the AXI4 FSM, mostly because of the differences between the interfaces, the overall principles are the same. Therefore, they will not be elaborated in this paper.

B. The Chisel Generator

The Chisel generator of the JTAG to memory-mapped bus master bridge has few parameters that can impact the characteristics of the generated instances. Data and address buses widths for all the AXI4/TileLink channels can differ between 32 and 64. The instruction code width is also changeable. As the current number of instructions is nine, the width of four bits is sufficient for all the instruction codes. Another parameter represents the code for the initial instruction. It is strongly recommended that any code of the NOP instruction is provided as this parameter. Maximum number of transfers in a burst cycle can also take different values, as well as the set of the addresses that the module's master interface can access. Even though the Chisel language is extremely suitable for parameterization of the modules, this capability is not exploited a lot in this case due to the nature of the proposed module itself.

For the implementation of the generator, several exploited open-source Chisel libraries worth mentioning exist. Chip-

salliance's Rocketchip library [10] is extensively used. It provided the extremely valuable classes for the implementation of both AXI4 and TileLink master interfaces, as well as of the interconnect bus and memory-mapped address space. Also, the Ucb-art's Chisel-JTAG library [11] was beneficial for the realization of the module. Its JTAG FSM design with some other modules, such as shift registers and I/O bundles, were utilized. The generator itself was integrated into the Ucb-bar's Dsptools library [12].

IV. Implementation and Verification Results

There are several stages of the JTAG to memory-mapped bus master bridge testing. The first one represents the usage of the software simulations. For the performance of these tests, Chisel testers are utilized to drive the JTAG input signals. Apart from testing the module solely, it was also verified experimentally using various other modules with memory-mapped control and status registers, from the simple ones, such as a streaming multiplexer, to more complicated ones, such as a parameterizable numerically-controlled oscillator or run-time configurable fast Fourier transformation module. The tested module was also verified within the simulation environments with multiple slave modules.

Another stage of the proposed generator's verification is implementing and testing the generated instances on an FPGA-based development board. A Digilent's Arty A7 board with Xilinx Artix-7 FPGA family is used for it. All the generator's instances are synthesized for 100 MHz system clock frequency. The JTAG input signals were driven from the PC using the FTDI's C232HM-DDHSL-0 cable [13]. That cable contains the FT232H integrated circuit [14] and represents the USB 2.0 hi-speed to multi-protocol synchronous serial engine (MPSSE) cable. For the utilization of the FT232H chip and the cable itself, Pyftdi open-source library [15] is used. The JTAG input signals are generated as the general purpose input/output (GPIO) signals. JTAG clock frequency was set to 15 MHz (GPIO pins can work with the frequency up to 30 MHz) which is the convenient speed having in mind that the JTAG clock frequency is obligated to be lower than the system clock frequency in order for the module to work properly. Similar to the verification using software simulations, several additional modules with the memory-mapped registers were used to validate the functional correctness of the proposed generator's module.

The FPGA resource utilization for the JTAG to memory-mapped bus master bridge is not significant due to the lack of the complex arithmetic or logic operations and few used registers. Moreover, it is expected for the peripheral's utilized resources to be drastically more numerous.

V. Conclusion

In this paper, a generator of the JTAG to memory-mapped bus master bridge implemented using Chisel hardware design language is proposed. Mainly, it is used for configuring, testing and debugging the peripheral modules with memory-mapped input/output interface in the systems without the processing core or where the processing core is set to remain inactive in terms of communication with the slave peripherals through the interconnection bus. Even tough it seems to be a complete product right now, theoretically speaking a lot of space was saved for the further upgrade, primarily to the Chisel's parameterizable characteristics.

The generated instances of the JTAG to memory-mapped bus master bridge were tested and verified by both using software simulations and mapping onto a commercial FPGA development board. Numerous additional modules with the memory-mapped slave interface are utilized in the testing process. The module from the topic of this paper proved to be trustworthy regarding its functionality and performance.

Acknowledgements

The authors would like to thank NOVELIC d.o.o. for financially and logistically supporting the work on this project.

References

[1] N. Y. Song, Y. J. Yu, W. Shin, H. Eom, and H. Y. Yeom, "Low-latency memory-mapped I/O for data-intensive applications on fast storage devices," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 766–770.

[2] *Intel 64 and IA-32 Architectures Software Developer's Manual*, Order number: 325383-060us ed., Intel, September 2016.

[3] *AMBA AXI and ACE Protocol Specification*, Arm ihi 0022e (id033013) ed., ARM, February 2013.

[4] *SiFive TileLink Specification*, Tilelink specification, version 1.8.0 ed., SiFive, Inc, August 2019.

[5] B. V. Ngo, P. Law, and A. Sparks, "Use of JTAG boundary-scan for testing electronic circuit boards and systems," in *2008 IEEE AUTOTEST-CON*, 2008, pp. 17–22.

[6] L. Ungar, H. Bleeker, J. McDermid, and H. Hulvershorn, "IEEE-1149.x standards: achievements vs. expectations," in *2001 IEEE Autotestcon Proceedings. IEEE Systems Readiness Technology Conference. (Cat. No.01CH37237)*, 2001, pp. 188–205.

[7] *JTAG to AXI Master v1.2*, Pg174 ed., Xilinx, February 2021.

[8] *Embedded Peripherals IP User Guide*, UG-01085 ed., Intel, March 2021, pages 56-60.

[9] V. D. Damnjanović and V. M. Milovanović, "JTAG to memory master (JTAG2MM) chisel generator," www.github.com/milovanovic/jtag2mm, accessed: 2021/06/21.

[10] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[11] C. M. Richard Lin, Paul Rigge, "chisel-jtag," www.github.com/ucb-art/chisel-jtag, accessed: 2021/06/21.

[12] P. R. Chick Markley, Angie Wang, "dsptools," www.github.com/ucb-bar/dsptools, accessed: 2021/06/21.

[13] *C232HM*, version 1.3 ed., FTDI, document No.: FT000401 Clearance No.: FTDI 214.

[14] *FT232H*, version 2.0 ed., FTDI, document No.: FT000288 Clearance No.: FTDI 199.

[15] E. Blot, "PyFTDI," www.eblot.github.io/pyftdi/, accessed: 2021/06/21.