True Interactive Testing based on IJTAG

Michele Portolan

True Interactive Testing based on IJTAG

Michele Portolan

Univ Grenoble Alpes, CNRS, Grenoble INP[1], TIMA, 38000 Grenoble, France
michele.portolan@univ-grenoble-alpes.fr

Abstract— The IEEE 1687 standard, commonly called ITAG, introduced several innovations. While the hardware-related ones, most notably mux-enabled dynamic topologies, are clearly successful and are widely adopted, several pans of the standard are still unsupported by the EDA vendors and are little known to the general public. In particular, the Procedural Description Language (PDL) theoretically allows for interactive routines whose outputs and control flow can be modified by the data retrieved from the System Under Test. However, the traditional Test Flow and Execution backend are not able to really support such features. In this paper, we present an in-depth analysis of these limitations and propose a fully-functional solution able to support true interactive behavior, whose features are demonstrated through a Proof-of-Concept.

KEYWORDS—FUNCTIONAL TEST, IEEE 1687, PDL, INTERACTIVE TEST, AUTOMATED TEST ENVIRONMENTS, MAST, PROOF-OF-CONCEPT

INTRODUCTION

The IEEE 1687-2014 [1] standard, commonly known as IJTAG, is often referred to as a "paradigm shift": for the first time, the focus on of Design-for-Test (DfT) architectures is shifted from the Top to the Instrument level: instead of prescribing a fixed top-level architecture as in JTAG [2], IEEE 1687 allows the description of a rich, dynamic topology and its operations inside of the Chip, and then "retarget" it (i.e. translate and adapt) to the top Level. The hugely successful IEEE 1500[3] standard did not go this far, and just proposed to replicate the JTAG DfT setup at Chip-level.

This shift is done in two steps: the first, and more visible, is the possibility of describing complex and dynamic DfT topologies by using the Instrument Connectivity Language (ICL). This allow designers to propose optimized access networks, being sure that the Electronic Design Automation (EDA) tools will be able to correctly support them. The second, more subtle innovation, is the possibility of describing Operations directly at the instrument level, thanks to the Procedural Description Language (PDL). In its simplest form, the "PDL Level 0" (PDL-0) the language allows users to describe vector operations at the Instrument level in the same way that this is done at the top-level: data can be written (iWrite) to a register, expected data can be set (iRead) and scan operations can be performed (iApply). A PDL-0 procedure can therefore be "retargeted" through the ICL topology in order to obtain the corresponding top-level vectors. Most, if not all, EDA vendors focused on these features in order to support IP-level ATPG and 1687 retargeting to the Top-level.

On top of it, "PDL Level 1" (PDL-1) is probably the one feature with the most disrupting potential: algorithmic capabilities are added, allowing the definition of complex interactive routines at the Instrument level that can then be retargeted and executed from the top-level. By the author's best knowledge, these features have been seldom supported by industrial EDA tools, at least in their fullest potential. In particular, no real work has been done on the what "Interactive Execution" actually entails. The only exceptions are academia-driven tools like the "Manager for SoC Test" (MAST) [4], which focuses on the Functional application of the IEEE 1687 standard.

In this paper, we present complete and operation flow for dynamic and interactive execution of IJTAG procedures against real hardware. First, Section 1 provides a State of the Art of IEEE 1687, focusing on the capabilities of PDL-1 and the limitation of current approaches when faced with interactive execution, while Section 3 introduces the new Interactive Test Flow able to overcome them, as well as its implementation exploiting the MAST tool, which is then used in Section 4 to obtain a complete Proof-of-Concept. Lastly, Section 5 draws some conclusions and points out the perspectives opened by this work.

1 STATE OF THE ART

In this Section, we will focus on the interactive elements of IEEE 1687 and on the limitations of the legacy execution flows.

1.1 PDL-1 primer

In the Standard document [1], the main focus is put on the ICL language (~100 pages), roughly the double of the space dedicated to PDL (~50 pages). Of these, PDL-1 is limited to a mere 10 pages. However, these few pages contain probably the most disruptive innovation: interactive behavior. Rather than defining yet another programming language, the Standard decided to allow PDL to be executed as a subset of TCL language, which is widely used in EDA shells. In this paper we focus exclusively on the PDL commands that have a direct effect on the vectors exchanged with the System-Under-Test (SUT), and not on the other commands that are rather used for configuration or other "housekeeping" operations.

As mentioned in the Introduction, PDL Level 0 (PDL-0) has been designed to replicate at the Instrument level the "Capture-Shift-Update" cycle typical of Scan operations. This is done through what is called the iApply cycle, which works in three main steps:

- "iRead" marks data to be captured, and can also set Expected values for mismatch check;
- "iWrite" queues data that needs to be Updated to the

[1]Institute of Engineering Univ. Grenoble Alpes

Instrument;

- "iApply" acts as a synchronization barrier: the Retargeter takes all queued iRead and iWrites and generates the set of scan operations that can realize them.

The parallel with Automated Test Pattern Generation (ATPG) is immediate and completely intentional: the aim of the Standard is to provide a seamless transition from IP to Top-Level for vectors operations. In this, it is undoubtedly successful.

PDL Level 1 (PLD-1) on the other hand has been designed to provide the user with the opportunity of retrieving data from the SUT and act over it. It is only composed of 4 commands explained in 5 pages in [1], most of which (iGetMiscompares, iGetStatus, iSetFail) are in fact "house-keeping" commands aimed at TCL scritping. It does anyway introduce one really new and disruptive command: "iGetReadData". It allows a TCL shell to access the data Captured during the last iApply cycle and use it inside its own procedures, introducing for the very first time interactive capabilities at the very heart of the standard. However, no clear indication is given in the document on how this interaction is going to be implemented.

1.2 Limitations of the Legacy Test Flow

The problem with vector operations is that they are completely static: their aim is to provide precise instruction and data to an Automated Test Equipment (ATE) to perform test as fast as possible. The classical Test Automation flow is based in the "Generation vs Application" duality [4] : a big EDA tool takes care of solving all the constraints of performing a given operation and generate a set of static patterns that can then be applied to the SUT to detect the "bad" circuits, as depicted in Figure 1. As the Generation happens only once, it can be heavy and Slow if it allows Application, which is performed for each chip, to be Fast.

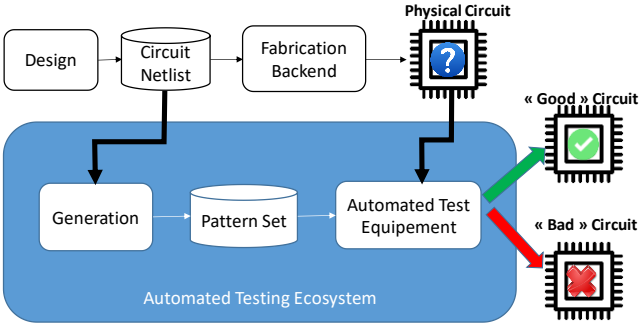


Figure 1 The Automated Test Flow [4]

Unfortunately, "Slow/Fast" can also be dubbed as "Smart Generation vs Dumb Application": all intelligence is in the Generation phase, while Application can only push and compare pre-computed vectors.

2 Algoritmhic Execution

As stated in the introduction, one of the most disruptive innovations of IJTAG is the introduction of interactive algorithmic execution thanks to PDL-1. The ability of modifying the control flow during execution is critical to react to the state of the SUT. For instance, depending on configuration options the size of a register in an IP might change, or the number of times a given set of operations needs to be applied might need to be adapted. PDL-1 was introduced to provide such flexibility, but the legacy Automated Test Flow is not adapted to algorithmic execution [4].

The Execution model was not directly mandated into the Standard document, but is it implicitly referred to what is usually called "ATE Bring-Up" or "ATE Debug", depicted in Figure 2 : the TCL script containing the PDL-1 code is executed by the shell of an EDA tool, which is able to communicate to an ATE in order to push to the SUT the vectors computed from the iApply operations, and provide to the iGetReadData commands the data read captured from it. The EDA Tool in the middle plays an essential role: it is responsible for both sending and receiving data with the SUT, therefore enabling an interactive execution loop.

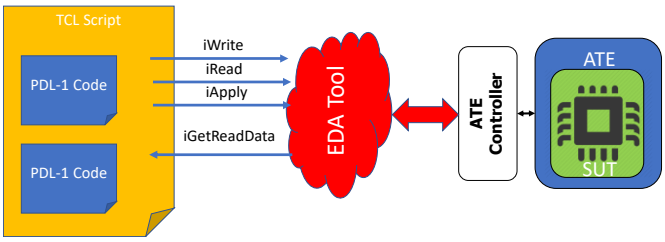


Figure 2 Implicit PDL-1 Execution Model

This is a complex setup, which is proposed by some industrial vendors [6], but its main aim is not really interactive execution but rather run-time parametrization. It is possible, for instance, to read an Identification register and choose a certain subset of tests to be execute depending on the version of the Chip or IP. Another example is to measure some reference value/voltage and use it to calibrate one or more routines. For these Use Cases the "parameter substitution" features of languages such as TCL or STIL [9] are more than enough: once the substitution has taken place, the Vector File becomes static and be Applied as usual.

However, this is not true for a really interactive setup, where both the Flow Control and the Applied Data depend on the status and data read from the System Under Test. Access to an EDA tool might not always possible during Test Application (for instance, if testing is done by a third-party company) or its execution might be impossible by the target platform (for instance, when performing online testing through an embedded controller). These use cases are not really addressed by the existing 1687 tool suites.

2.1 New Usages

The success of IJTAG brought also some complications: new usages, not foreseen by the Working Group have started implementing the Standard and trying to use it for application that are far from traditional ATPG-based testing. Most of these modes exploit the Functional access to embedded instruments to retrieve information about the SUT and interactively act on it.

An example application is analog signal-processing: for instance, in [7] the authors propose a BIST (Built-In Self-Test)

[1]Institute of Engineering Univ. Grenoble Alpes

for an ADC that makes a spectral analysis (Fast Fourier Transform, FFT) from sampled data, and uses it to finely-tune the converter. This kind of component is implemented as a self-enclosed and independent BIST not only to guarantee good signal quality, but also because of the difficulty of expressing complex algorithms in vector terms. Modern ATEs are powerful machines providing programming language capabilities and could easily execute such algorithms, but those are ad-hoc solutions, different for each ATE family and sometimes even between models. There is no direct way for DfT Designer to transmit such algorithms to the final Test Engineer, so this solution even though theoretically possible is actually seldom used. In this context IJTAG is a compelling alternative: leverage PDL-1 to extract data from the SUT, and use the overlay TCL to express the processing needs. Even if this opportunity is promising, no real solution has been proposed yet: as explained in the previous section, the Execution Model of Figure 2 does not really target applications such as Signal Processing applications which are eminently interactive. Moreover, these types of applications are often used outside of a pure testing framework, where the usage of a full-fledged Test Generation Tool might not be possible.

3 A New Execution Model

The best way to overcome the limitations of the legacy Automated Test Flow is to break the "Smart Generation/ Dumb Application" duality and implement an hybrid flow where intelligence is shared among the two phases, as first presented in [4], to obtain a Fully Interactive Flow as depicted inFigure 3. The principle is that instead of resolving all retargeting steps during the Generation phase to obtain a static set of vectors, the information collected from ICL and PDL (most noticeably the algorithmic operation and the calls to PDL) is converted into a Text Executable, which is then processed at runtime. Vectors are therefore generated dynamically depending on both the state of SUT and the results of the algorithmic execution. This way instead of having an extremely fast but simplistic execution backend, the flow leverages its computational resources.

At runtime, MAST loads the executable together with Configuration information (i.e. the ICL file), which is used to build a Model of the System Under Test. The MAST kernel then leverages the Operating System running on the Test Host (Linux or Windows) to take care of launching the Test Executable. Instead of unrolling and processing the TCL+PDL procedures offline to obtain a top-level set of vectors that can be agnostically pushed to the SUT, MAST rather packs the Procedures code in a binary format, mixing PDL and algorithms (compiled into assembler instructions) and executes them against the real SUT [4].

This Flow pushed interaction with the SUT at its very heart: the MAST Kernel only cares about retargeting and is extremely lightweight and streamlined (especially if compared to a traditional EDA Test Generation Tool) and can be executed even in resource-constrained execution backends. The User, in the left-hand side, prepares his PDL-1 files and compiles them thanks to the provided Headers. MAST made the choice to use C/C++ as an overlay language instead of TCL for performance reasons, even though this is not strictly IEEE 1687-2014 compliant. However, discussions are ongoing in the P1687 Refresh Working Group [5] that might open the Standard to alternative languages than TCL.

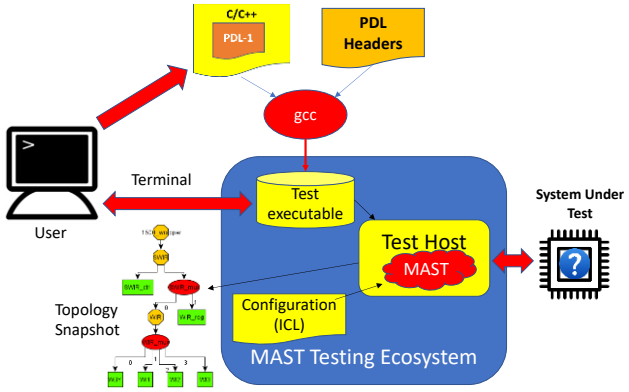


Figure 3 True Interactive Flow with MAST

At Runtime, the Test Executable is loaded by MAST together with the ICL description, which allows the Kernel to build is internal System Model representing the actual SUT. When Execution begins the User can monitor the results through a terminal, observing the output of the Test Program and, if needed, extract snapshots of the SUT Topology configuration. It is a standard computer Science execution and debug flow, so the User can leverage existing strategies and tools (such as, for instance, GDB, Eclipses, etc.). The absence of a big EDA tool in loop, as in Figure 2, makes this his solution much more streamlined and optimized as legacy setups. As a result, performances are much higher on traditional desktop setup, while the setup is completely portable to constrained execution backends such as ATEs or Embedded Processors [4].

4 Experimental Demonstrator

To prove the novelty and performances of this new setup, and its capability to provide true interactive execution, we developed a Proof-of-Concept Prototype that is at the same time representative of an interactive signal processing setup as the one described in [7] and visually compelling: music volume bars. The setup, depicted in Figure 4, is functionally quite simple: a 1687 Instrument samples music coming from a stereo audio source, and the digital samples are collected by a PDL-1 function which performs an FFT to extract the Volume level of the right and left channels, which are then sent to a visualization 1687 Instrument.

[1]Institute of Engineering Univ. Grenoble Alpes

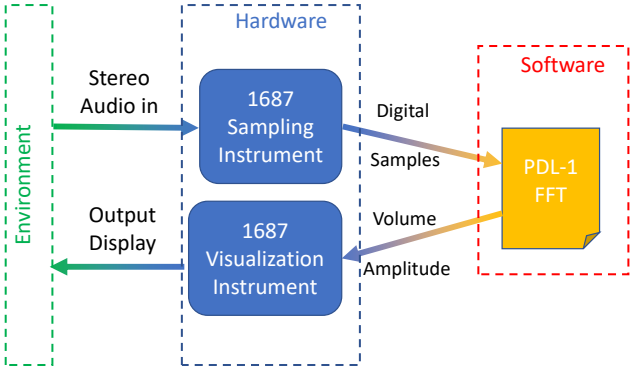


Figure 4 Functional Specification of the Demo Setup

The key of the PoC is that functionality is split between hardware and software, rather than being a fully enclosed in hardware like in traditional BIST. Such a setup is only possible with a true run-time interaction between the 1687 Software and the SUT.

4.1 Top-Level Implementation

We implemented the demo using a Xilinx ML505L card [8], a popular, if somewhat dated, FPGA card that presents several interesting features:

- Stereo AC97 Audio codec (Analog Devices AD1981) for input sampling;
- 16-Character x 2-Line LCD Screen (Tianma TM162VBA6) as output display;
- Fairly-sized FPGA fabric (XC5VLX50T)
- 30+ GPIO for interface and debug
- CompactFlash to program the FPGA at power-on;

The Prototype schematics specification is shown in Figure 5. The system-Under-Test (SUT) is programmed inside the PFGA and its composed by the Input / Output Instruments accessed through the 1687 Network, whose TAP pins, connected to the GPIO, are controlled by an FTDI chip providing USB-to-JTAG conversion [11]. This is used in the Host PC by the MAST kernel [4] for the communication and synchronization between the System-Under-Test programmed in the FPGA and the PDL-1 algorithm.

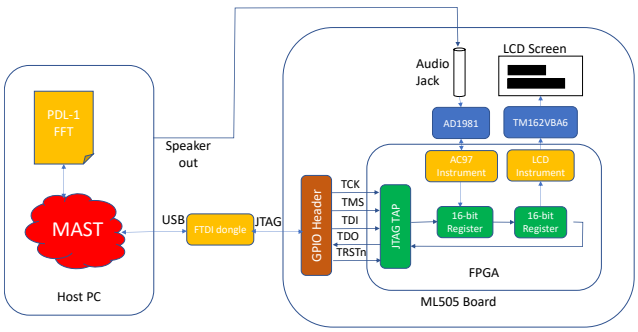


Figure 5 Prototype Schematics Specification

The aim of this demo is to showcase the dynamic capabilities of the Standard in terms of software, so the hardware part is extremely simple: an 1149.1 Test Access Port (TAP) driving two daisy-chained registers. The first register is read-only and is connected to the "AC97 Instrument", which is in fact an FSM implementing the AC97 Codec to collect the audio samples in 2 times 8-bit values (one per channel) and making them available as a 16-bit Scan Register. The second register is write-only and receives 2x8-bit values representing the volume amplitude level of each channel, which the "LCD Instrument" uses to visualize two black bars.

4.2 Software Setup: Signal Processing in PDL-1

The Software side of the Demo exploits MAST's capability of dynamically interacting with the SUT: instead of generating a set of static vectors from an analysis of the PDL inputs as done by legacy EDA solution, MAST executes the PDL-1 at runtime, using iApply as synchronization barriers where the data exchange with the SUT happen. This way, "to SUT" vectors are generated only when needed and "from SUT" vectors are dynamically analyzed and their data directly re-injected in the PDL-1 routines as return data from the iGetReadData commands. To obtain this result, MAST uses C++ as the PDL-1 overlay language: the choice of a compiled rather and an interpreted language both boosts performances and allows the reuse of existing software libraries.

In this demo we chose to use KissFFT [12] to estimate the volume level for each channel in a Time Window of 256 samples. To avoid glitches on the output, the Amplitude displayed an any given cycle is the maximum value of the amplitudes of each sample in the Time Windows, calculated using KissFFT.

The pseudo-code is as follows:

```
1.  While (1)
2.  {
3.  iGetReadData(AC97_Instrument, new_Sample);
4.  Extract RightChannelSample and LeftChannelSample
    from new_Sample;
5.  add RightChannelSample to RightTimeWindow;
6.  add LeftChannelSample to LeftTimeWindow;
7.  For each Time Window:
8.      Use KissFFT to compute amplitude of each sample;
9.      Volume = max(Amplitude(TimeWindow)
10.     Convert Volume to an 8-bit integer value Volume_8
11. BarValue= Volume_8_left || Volume_8_right
12. iWrite(LCD_Instrument, BarValue);
13. iApply();
14. }
```

In its simplicity, the proposed algorithm is still representative of a generic signal processing loop: first data is retrieved from the system (line 3), then its is processed (lines 4 to 10) and finally action is taken on the result (lines 11 and 12). The interleaving of PDL-1 operations and C++ is possible thanks to 1687's decision of presenting PDL as an API for an overlay language: a TCL script would look similar, but the execution would be much slower (and the Signal Processing part much more cumbersome).

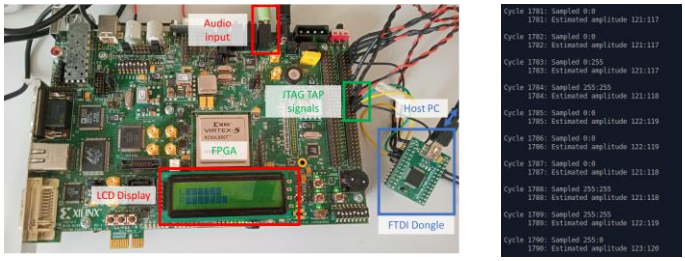
The MAST Kernel uses the ICL description of the SUT to build its own internal model, which is then used to handle the execution of the PDL-1 code and generate the necessary SVF operations [13], which are then passed to a Callback using the OpenOCD library [14] to control the FTDI USB dongle, following the principles being developed by the IEEE P1687.1

[1]Institute of Engineering Univ. Grenoble Alpes

Working Group [15]. All this heavy-lifting is transparent to the user, who simply has to provide the PDL and ICL files.

4.3 Proof-of-Concept Prototype

The final demo is shown in Figure 6-a): the FPGA of the ML505L card is in the middle the picture, the Audio Input receiving the music is in the top half, while in the bottom half it is possible to see the LCD screen displaying the volume bars for the Left and Right channels. The JTAG interface on the right-hand side: the GPIO header is used to connect the TAP signals to the design inside the FPGA with the FTDI dongle. Communication with the Host PC, not displayed, is done through the USB cable.



a) b)

Figure 6 Final prototype

The PDL-1 code is executed like a normal program on the Host PC and can therefore display debug information. Figure 6-b) shows the output of our demo: in each cycle, the value of the sampled input data and the estimate volume amplitude for each channel is displayed on the terminal.

The two outputs demonstrate that our system is effectively performing a true interactive execution of the PDL-1 code against the actual hardware.

5 Conclusions and Perspectives

In this paper, we presented a complete and functional flow able to dynamically execute PDL-1 against real hardware, and we proved it thanks to a Proof-of-Concept for a signal-processing application. It has been realized on a commercial FPGA for the Hardware part and a generic OS (Linux in this case) on the Software side which while being Standard-compliant it still allows free usage of third-party software libraries. To the author's best knowledge, it is the first implementation of this type.

Future evolutions will first focus on leveraging other features of the IEEE 1687 standard, such as exploiting dynamic topologies while maintaining full PDL reuse or explore real-time and concurrency issues. Other directions will be exploiting the new IEEE P1687.1 proposal to extend IJTAG to interfaces other than JTAG (ex: SPI or I2C) or enhance the algorithmic part to cover more complex signal processing problems or adapt it to RF/Mixed Signal testing. A porting of the prototype on a more recent FPGA development board, the Zedboard [16], is also underway.

Acknowledgments

The author would like to acknowledge the work of Niels Grataloup and Clément Tardy for developing and troubleshooting the Proof-of-Concept platform.

References

[1] IEEE Std 1687-2014, "IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device", IEEE, USA, 2014

[2] IEEE Std 1149.1-2001, "IEEE Standard Test Access Port and Boundary-Scan Architecture", IEEE, USA, 2001.

[3] IEEE std 1500 - Standard for Embedded Core Test - http://grouper.ieee.org/groups/1500/.

[4] M. Portolan, "The Automated Test Flow, the Present and the Future", IEEE Transactions on Computer-Aided Design (TCAD), DOI: 10.1109/TCAD.2019.2961328, December 2019

[5] M. Portolan, M. Keim, J. Rearick and H. Ehrenberg, "Refreshing the JTAG Family," 2023 IEEE 41st VLSI Test Symposium (VTS), San Diego, CA, USA, 2023, pp. 1-7, doi: 10.1109/VTS56346.2023.10140015.

[6] Siemens Silicon Insight, https://eda.sw.siemens.com/en-US/ic/tessent/test/siliconinsight/

[7] M. J. Barragan et al., "A Fully-Digital BIST Wrapper Based on Ternary Test Stimuli for the Dynamic Test of a 40 nm CMOS 18-bit Stereo Audio ΣΔ ADC," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 63, no. 11, pp. 1876-1888, Nov. 2016, doi: 10.1109/TCSI.2016.2602387.

[8] "ML505/ML506/ML507 Evaluation Platform User Guide" https://docs.amd.com/v/u/en-US/ug347

[9] "IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450-1999) for Test Flow Specification," in IEEE Std 1450.4-2017 , vol., no., pp.1-190, 9 Feb. 2018, doi: 10.1109/IEEESTD.2018.8283877.

[10] "Standard Test And Programming Language", JEDEC Standard no. 71, 1999

[11] FT4232H Mini Module Evaluation Module Datasheet, Future Technology Devices International Ltd., Reference FT_000115 v 1.8, 2012-08-01

[12] Kiss-FFT homepage, https://github.com/mborgerding/kissfft

[13] "Serial Vector Format Specification", ASSET InterTech Inc. Revision E, 8 March 1999

[14] Open On-Chip Debugger Homepage, https://openocd.org/

[15] M. Laisne et al., "Modeling Novel Non-JTAG IEEE 1687-Like Architectures", 2020 International Test Conference (ITC20), November 2020, Washington DC, US

[16] Zedboard homepage, https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/zedboard-board-family

[1]Institute of Engineering Univ. Grenoble Alpes