LUND UNIVERSITY

System-Level Access to On-Chip Instruments

Larsson, Erik; Kiran Gangaraju, Shashi; Murali, Prathamesh

Published in:
Proceedings of the European Test Symposium

DOI:
10.1109/ETS50041.2021.9465415

2021

Link to publication

Citation for published version (APA):
Larsson, E., Kiran Gangaraju, S., & Murali, P. (2021). System-Level Access to On-Chip Instruments. In *Proceedings of the European Test Symposium* (pp. 1-6). Article 9465415 https://doi.org/10.1109/ETS50041.2021.9465415

Total number of authors:
3

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 07. Dec. 2025

System-Level Access to On-Chip Instruments

Erik Larsson, Shashi Kiran Gangaraju and Prathamesh Murali
Lund University, Lund, Sweden
Email: erik.larsson@eit.lth.se

***Abstract*—Modern integrated circuits (ICs) contain thousands of instruments to enable testing, tuning, monitoring, and so on. These on-chip instruments must be accessed through the ICs' lifetime. However, when ICs are mounted on Printed Circuit Boards (PCBs), access from system-level is challenged due to complex system hierarchies with a multitude of interfaces. In this paper we enable access from system-level to chip-level instruments by proposing hardware, protocol, and communication schemes. We have validated our scheme by implementing a system with two ICs on a Field-Programmable Gate Array (FPGA) where each IC includes an IEEE Std. 1687 network, communication between ICs is with Serial Peripheral Interface (SPI) and communication with the outside is with Universal Asynchronous Receiver Transmitter (UART). In experiments we evaluate communication based on software (polling) and hardware (interrupt) as well as overhead in terms of transported data and needed area.**

***Keywords*—IEEE Std. P1687.1, IEEE Std. P2654, IEEE Std. 1687**

I. Introduction

While the semiconductor technology development towards integrated circuits (ICs) with smaller, faster and more transistors gives advantages such as more functionality, better performance, and lower power consumption, it is becoming increasingly challenging to avoid various types of malfunctioning. Smaller and faster transistors lead to tighter margins, which in combination with more transistors increase the risk of malfunctioning. To avoid malfunctioning, ICs are increasingly equipped with embedded (on-chip) instruments for testing, tuning, trimming, configuration and so on [1]. These instruments, which for a single IC can be in the range of thousands, must be accessed throughout the life cycle of the system: from prototype, debug, test and validation to in-field monitoring and test [2]. However, access from system-level to chip-level instruments when ICs are mounted on printed circuit boards (PCBs) is challenging due to complex system hierarchies often with a multitude of interfaces.

Figure 1 shows three scenarios to access on-chip instruments. In the first scenario, IEEE Std. 1149.1 [3] and IEEE Std. 1687 [4] are used. The test access port (TAP) of IEEE Std. 1149.1 is the interface between the outside and the on-chip IEEE Std. 1687 network and IEEE Std. 1687 is the infrastructure connecting instruments. IEEE Std. 1687 includes two description languages, instrument connectivity language (ICL) and procedural description language (PDL). ICL describes how instruments are interconnected and PDL describes how to operate on instruments. To access instruments, an Electronic design automation (EDA) tool takes PDL and ICL as inputs and forms in a retargeting process access (test) patterns. Several research works exist on analysis and design aspects of IEEE Std. 1687 networks [5]–[11]. Second scenario shows the scope of the on-going work of IEEE Std. P1687.1 [12]. A

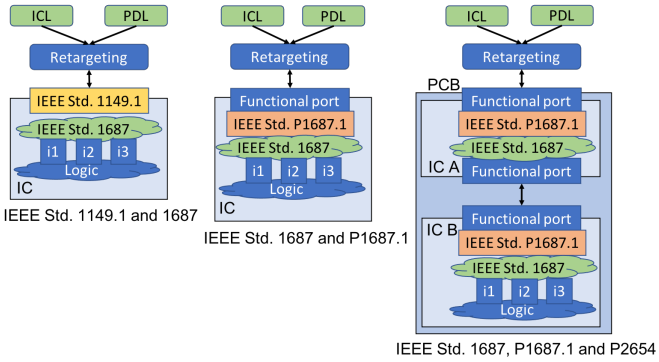


Fig. 1. Access to instrument using interplay with different IEEE standards

limitation of the first scenario is that all ICs must be equipped with an IEEE Std. 1149.1 TAP. Instead of being dependent on an IEEE Std. 1149.1 TAP, which due to pin limitation may not be available on all ICs, IEEE Std. P1687.1 proposes to make use of existing functional ports, like Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C) and so, to interface IEEE Std. 1687 networks. There are some prior work on IEEE Std. P1687.1 [13]–[15]. The third scenario shows the scope of the on-going work in IEEE Std. P2654 [16]. The objective is to enable access when there is a multitude of interfaces in a system. For example in Figure 1, one interface, let say SPI, is used to connect IC A to the outside, and another, let say I2C, is used to connect IC A and IC B. If one wants to make access to instrument *i1* and *i3* in IC B there is a need to use the path: SPI $\leftrightarrow$ IEEE Std. 1687 in IC A $\leftrightarrow$ I2C $\leftrightarrow$ IEEE Std. 1687 in IC B. This access path crosses different interfaces. There are some overview paper [17].

A key to enable access from system-level to on-chip instruments is to enable communication passing through one IC with different interfaces, for example IC A in Figure 1. When this access is solved, the solution can be scaled to hierarchies with several ICs. However, it is important to ensure that individual ICs can be designed independently from each other, for example in respect to protocol (the way transported data is formatted). In this respect, it should also be noted that current line of thinking in the IEEE Std. P1687.1 working group is to allow flexibility in what hardware that is implemented between the functional port and the IEEE Std. 1687 network. A result of this flexibilityy is that the protocol might differ between IEEE Std. P1687.1 compliant ICs. In this paper we propose hardware, protocol, and communication schemes meeting these requirements and we explore both software-based (polling) and hardware-based (interrupt) communication at an individual IC.

The paper is organized as follows. In Section II we give a brief background to hardware and protocol where a functional

TABLE I. TRANSPORTED DATA FOR SOME COMBINATIONS OF HARDWARE AND PROTOCOL [14]

Instruments	Full-featured	No dummy	No ILM	Naive	Bit-banging
50	4832	7122	6432	29302	222128
100	9632	14218	12832	96060	765232
150	14432	21336	19232	201110	1628848

port, instead of the TAP of IEEE Std. 1149.1, is used to interface IEEE Std. 1687 networks [14]. In Section III we detail proposed hardware, protocol and communication schemes, that is polling and interrupt, needed to enable exchange of information between ICs. We have validated proposed scheme by implementing a system with two ICs on a Field-Programmable Gate Array (FPGA) where each IC includes an IEEE Std. 1687 network of instruments, communication between ICs is with SPI and communication with the outside is with Universal Asynchronous Receiver Transmitter (UART), Section IV. For experiments, we make use of IEEE Std 1687 networks with 50, 100, and 150 instruments. We evaluate the overhead in terms of data and area. We compare proposed system hierarchy (two levels - two ICs) against single level (one IC) [14] and we compare software-based (polling) against hardware-based (interrupt) for communication. The paper is concluded in Section V.

II. BACKGROUND

IEEE Std. 1149.1 specifies the interface from the outside to the TAP, including protocol for communication. A major advantage is that ICs from different vendors compliant to IEEE Std. 1149.1 easily can be integrated into the same system (PCB). However, as not all ICs have an IEEE Std. 1149.1 TAP, the IEEE Std. P1687.1 is working towards a standard where functional ports can be used to access IEEE Std. 1687 networks. The current idea is to propose a more flexible solution in respect to hardware than the one used in IEEE Std. 1149.1. We have for an IEEE Std. P1687.1 environment investigated the impact of flexible hardware and protocol in respect to the amount of data that is needed to be transported and the area of the hardware [14]. Table I shows for some different combinations of hardware and protocol the amount of data that needs to be transported. Important to note is that different hardware impacts what is included in a protocol.

We will briefly describe our full-featured solution, which results in least amount of transported data [14]. For the protocol, each iApply group becomes retargeted (translated) into one or more control commands, which resembles a header, followed by one or more data commands, like a payload. The hardware is based on a Finite State Machine (FSM) that is complemented with a segment insertion bit (SIB) control register (SCR), an instrument control register (ICR), and an instrument length memory (ILM). Control commands set current values for SIBs in SCR and current operation for each instrument in ICR. The length in bits of each instrument is stored in ILM, which is fixed at design time.

Figure 2 shows an example with an iApply group to write data to instrument *i1* and to read[1] data from instrument *i3*. The iApply group is retargeted into two control commands and one data command, in total 7 bytes. The control commands set

[1]iGetReadData (iGet) reads information from an instrument

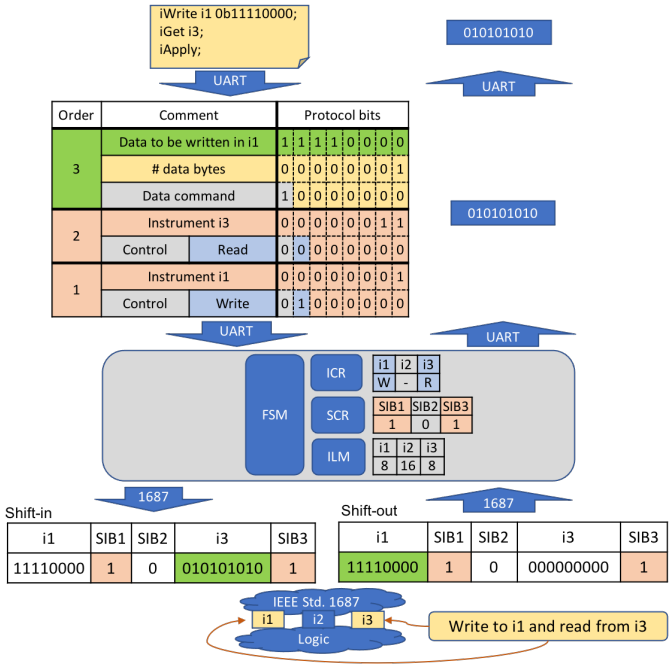


Fig. 2. Hardware translator and protocol [14]

required values in SCR and ICR. The first control command makes SIB 1 active and sets instrument *i1* in write mode, that is $SCR(1) = 1$ and $ICR(1) = W$ (index is instrument number). The details are as follows. Bit $b_7 = 0$ in the first byte indicates that current byte and the following byte form a control command. Bit $b_6 = 1$ indicates that a *write* operation should be performed. The following 14 bits, which hold the value 1, indicate that SIB 1 should be active so that instrument *i1* is included in the active scan-path. The next two bytes are also forming a control command, which will make SIB 3 active and set instrument *i3* in read mode, that is $SCR(3) = 1$ and $ICR(3) = R$. The following 3 bytes form a data command as $b_7 = 1$ in byte 5. The remaining 15 bits are used to specify the number of bytes with data that follows. In this example, the 15 bits specify the value 1, meaning that one byte of data follows. Byte 7 holds the data that should be written to instrument *i1*.

When the first control command arrives, the FSM is resetting the registers and setting SCR and ICR according to control commands. When data commands arrive, the FSM begins operating the IEEE Std. 1687 network. First, the active scan path is set by traversing SCR and shifting the content to the IEEE Std. 1687 network. Second, the shift sequence for the active scan path is created, see Figure 2. The FSM begins checking SCR at its highest value, in this example 3 ($SCR(3)$), and includes that bit in the shift-in sequence. As $SCR(3) = 1$ instrument *i3* is included and corresponding command ($ICR(3)$) is checked to learn that a *read* operation should be performed, which means data needs to be shifted in such that the content of instrument *i3* is shifted-out. This additional (dummy) shift-in data is created by the FSM. The number of bits to shift for an instrument is given by the instrument length memory ($ILM(3)$). Then, the FSM proceeds with $SCR(2)$. As $SCR(2) = 0$ instrument *i2* is not in the active scan-path, the FSM adds a 0 to the shift-in sequence and focuses on next bit in SCR, which is $SCR(1)$. $SCR(1) = 1$,

which means instrument *i1* is included in the active scan-path and as $ICR(1) = W$ a *write* operation should be performed. The FSM finds the length of instrument *i1* from $ILM(1)$ and takes data from the UART buffer and adds it to the shift-in sequence. Figure 2 shows the created shift-in sequence.

The FSM can with help of SCR, ICR, and ILM (in a similar process as for the shift-in sequence) remove all unwanted (dummy) bits from the shift-out sequence such that only useful information is transported out from the IC. Applying the PDL in Figure 2 results in that the only information returned is the eight bits from the read of instrument *i3*.

III. Proposed scheme

In this section we detail proposed scheme, which includes hardware, protocol and communication schemes to handle the transfer of information through an IC (crossing interfaces). We include two alternatives to handle communication; software-based (polling-driven) and a hardware-based (interrupt-driven). We use the system in Figure 3 as a working example.

A. Protocol

The protocol in Section II is extended to make it possible to pass information through a chain of ICs connected to each other with different interfaces. The PDL in Figure 3, same as in Figure 2, should set the active scan-path of the IEEE Std. 1687 network in IC B (level *i+1*) such that data is written to instrument *i1* and data is read from instrument *i3*. Different from Figure 2 is the need to pass data through IC A (level *i*). The principle is that data sent to IC A is formed as control commands (similar to a header) followed by data commands (similar to payload). In IC A (level *i*), the data (payload) is sent to IC B (level *i+1*) where the payload is interpreted as control commands (header) and data commands (payload).

The PDL (iApply group) is formed as control and data commands as follows. One control command is needed to set up the IEEE Std. 1687 network in IC A (level *i*). That is to include instrument *iA* in our hardware (the hardware is described below). After the control command, there is one data command. In this example, the data command specifies that there are seven bytes of data. These seven bytes of data are sent to IC B (level *i+1*) (see Figure 2). When IC B receives these seven bytes of data, IC B interprets them as control and data commands. The first control command is to include *i1* and make a write. The second control command is to include *i3* and make a read. Then, a data command follows, indicating that one byte of data follows, which is the data to be written in *i1*. While the example shows a case with two ICs, the scheme is general such than an arbitrary number of ICs can be passed.

B. Hardware

The hardware to control communication between two IEEE Std. 1687 networks at different ICs is formed as IEEE Std. 1687 compliant instruments, see Figure 4. The hardware consists of two parts; one for sending data and one for receiving data. The hardware for sending data from IC *i* to IC *i+1* consists of an instrument of length 9 bits where 8 bits are for the data that is to be sent and one control bit to inform the functional port when data is available such that sending can be initiated. The instrument for sending data from IC *i* to IC *i+1* is named *iA* in Figure 4. The hardware to receive data from IC *i+1*, that is to move data from IC *i+1* to IC *i*, consists of three instruments, named *iB*, *iC*, and *iD* in Figure 4. The actual data is kept in *iB*, and *iC* is a single flag bit. The flag is set by the functional inteface when data has arrived completely in instrument *iB*. Instrument *iD* is a single bit to acknowledge to the functional interface when data in *iB* has been read (consumed).

C. Communication

The controller in IC B is described in Section II, marked as ITC'19 in Figure 3. The controller in IC A is based on the one described in Section II but extended with two alternatives to handle communication; polling-based (software-based) and interrupt-driven (hardware-based), marked as ITC'19+ in Figure 3.

The software-based scheme uses polling initiated from the outside to check *iC*, which is set when data is available in *iB*, see Figure 4. The PDL to perform polling of *iC* is:

```
iGet iC;
iApply;
```

The polled bit is packaged into a byte, the smallest unit to be transported. As soon as *iC* indicates that data is available, the data can be read from instrument *iB* by using:

```
iGet iB;
iApply;
```

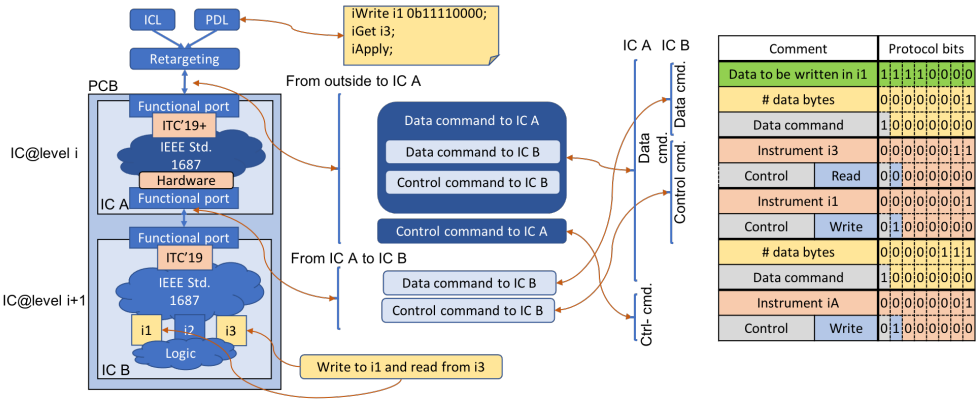
As soon as data has been read, *iD* is automatically set by the controller to indicate that data has been consumed. Each of the iApply groups above translates into one control command of 2 bytes and one data command of 2 bytes.

Instead of having externally initiated polling, hardware in the on-chip controller can implement the checking (polling) function in an interrupt-driven manner. The hardware in IC *i* automatically checks *iC* and when new data is available, data is read from *iB* and sent (returned) to the outside and *iD* is set to inform the functional interface that data is consumed.

IV. Experimental Results

The objective of the experiments is to validate our scheme, evaluate the impact of system hierarchy and compare software-based (polling) with hardware-based (interrupt) communication in terms of transported data and area utilization.

We used an FPGA (Nexys 4 DDR with an Artix-7 (XC7A100T-1CSG324C)) to implement a system with two ICs, IC A and IC B. The interface between IC A and IC B is with SPI and the interface between IC A and the outside is with UART. We set SPI and UART to transport data in sizes (packets) of one byte at a time. There is one IEEE Std. 1687 network in IC A and one in IC B. The IEEE Std. 1687 network in IC A consists of proposed hardware and the IEEE Std. 1687 network in IC B contains 50, 100, and 150 instruments. The instruments in IC B are connected in a flat manner with one SIB per instrument. Instruments are of length of 8, 16, and 32 bits. That is, instrument 1 is of length 8 bits, instrument 2 is of length 16 bits, and instrument 3 is of length 32, and



Comment		Protocol bits
Data to be written in i1		1 1 1 1 0 0 0 0
# data bytes		0 0 0 0 0 0 0 1
Data command		1 0 0 0 0 0 0 0
Instrument i3		0 0 0 0 0 0 1 1
Control	Read	0 0 0 0 0 0 0 0
Instrument i1		0 0 0 0 0 0 0 1
Control	Write	0 1 0 0 0 0 0 0
# data bytes		0 0 0 0 0 1 1 1
Data command		1 0 0 0 0 0 0 0
Instrument iA		0 0 0 0 0 0 0 1
Control	Write	0 1 0 0 0 0 0 0

Fig. 3. A system with two ICs (IC A and IC B) and corresponding hardware and protocol for system-level access to on-chip instruments

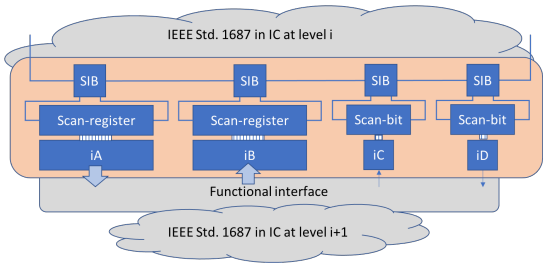


Fig. 4. The IEEE Std. 1687 compliant hardware to interface IC *i* and IC *i+1*

instrument 4 is of length 8, and so on. We made use of four trivial PDL descriptions with one *iApply* group each, *iGet* from instrument 1, *iWrite* to instrument 1, *iGet* from all instruments, and *iWrite* to all instruments. We also made use of the PDL scheme used in the BASTION benchmarks [18]. The PDL scheme in BASTION is to first perform one *iApply* group with write to all instruments, followed by one *iApply* group with read from all instruments, and finally, for each individual instrument, an *iApply* group with a write followed by an *iApply* group with a read. For the benchmark with 50 instruments the PDL scheme results in 102 iApply groups.

The experimental results on transported data are collected in Table III, which is organized as follows. The number of instruments is in column one, the applied PDL is in column two and the amount of useful data bits is in column three. Column four lists the types of data, including total overhead and the amount of useful data in relation to the total amount of data. The three approaches we compare are listed in columns five, six and seven. Column five includes the approach from ITC'19 (system hierarchy with one IC, one-level IC), column six includes proposed scheme (system hierarchy with two ICs, two-level IC) with hardware-based (interrupt-driven) communication, and column seven includes proposed two-level IC with software-based (polling-driven) communication.

Transported data is divided into the following categories; useful bits and overhead bits where overhead is divided into dummy, control and data. These categories are illustrated with two PDL examples:

```
iWrite i1 0b11110000;
iApply;
```

and

```
iGet i1;
iApply;
```

For a one-IC setup (Section II [14]), in the case of *iWrite i1*, eight bits will be written to *i1*, this is *useful data*. And in the case of *iGet i1*, eight bits will be read from *i1*, this is *useful data*. For both cases, there will be one active instruments, so there will be one control command, which is of size 16 bits. Hence, *control overhead* is 16 bits. There will be one data command, size 16 bits. Hence, *data overhead* is 16 bits. There is no *dummy overhead*. Total overhead becomes 32 bits. In total 40 bits are transported, overhead plus useful data, which gives that the ratio of useful data over total data is 20%.

For a two-IC setup (IC A and IC B) there is in addition to the transported data from the one-IC case, additional data overhead due to the path between the two ICs. We have two cases. First, sending data from the outside via IC A to IC B and, second, sending data from IC B via IC A to the outside.

For the first case, sending data from the outside, the effort is the same for both cases (*iWrite i1* and *iGet i1*). The path is set by one control command to *iA* in IC A, size 16 bits, which means *control overhead* is 16 bits. One data command, which includes control and data commands for IC B, adds another 16 bits, *data overhead*. For both PDL cases, 32 additional bits are needed, giving that the overhead becomes 64 bits. In total 72 bits are transported, overhead plus useful data, which gives that the ratio of useful data over total data 11%.

The second case, sending data from IC B via IC A to the outside, applies only to *iGet i1* as then there is data (bits) from instrument *i1* that should be transported to the outside. Instrument *iC* (in IC A) needs to be checked to determine if data is ready/available and if data is available the data should taken from *iB* (in IC A). In the case of a software-based solution (polling), the activity is handled from the outside by using one control command to set the active scan-path to include *iC* and one data command to apply the check (poll). In total, 16 additional bits of *control overhead* and another 16 additional bits for *data overhead*. To report the status bit, 8 bits are needed as 8 bits is the smallest unit to be transported. This is called *dummy overhead*. If the status bit indicates that information is not available, a new poll must be made. If

TABLE II. AREA IN CONFIGURABLE LOGIC BLOCKS (CLBS)

Instruments	Software CTRL@IC A	Hardware CTRL@IC A	CTRL@IC B	IEEE Std. 1687 network@IC B
50	120	158	90	371
100	120	158	113	749
150	120	158	134	1123

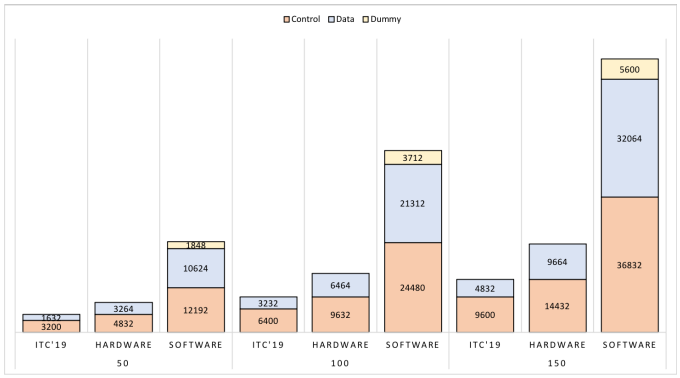


Fig. 5. Data overhead for the BASTION benchmark for 1-level (one IC) and 2-level (two ICs) with hardware (interrupt-driven) communication and software (polling-driven) communication

information is available, a read operation can take place, which means *iB* should be on the active scan-path, which is achieved by using one control command, 16 bits *control overhead*, and one data command, 16 bits of *data overhead*. This gives that the total overhead is 136 (64+64+8) bits. In total 144 bits are transported, overhead plus useful data, which gives that the ratio of useful data over total data is 6%. In the experiments we report the best polling case, which means that data is available at the first poll attempt. In the case of hardware-based handling, our controller in IC A performs the checking activity without any involvement from the outside; hence, no additional overhead.

Figure 5 shows the overhead for the BASTION benchmark. The ITC'19 results for a system with one IC serves as a reference. The overhead increase for systems with two ICs as more work is needed to set the longer access pass. One should note that the overhead for the hardware-based (interrupt-driven) case is rather modest.

Table II reports the area of the controller in the software-based (polling-driven) case (IC A), the controller in the hardware-based (interrupt-driven) case (IC A), the controller in IC B (which is the same as in ITC'19) and the IEEE Std. 1687 network in IC B. As the controllers in the software-based case (IC A) and the hardware-based (IC A) case are extensions of the controller in IC B (ITC'19) their area is higher. The hardware-based controller is larger than the software-based controller for the reason that the work corresponding to polling is completely handled. The size of the hardware-based controller and the software-based controller in IC B remains constant when the number of instruments increase in IC A.

V. CONCLUSIONS

Access to on-chip instruments is difficult when ICs are mounted on PCBs due to complex system hierarchies with a multitude of interfaces. We developed protocol, hardware and communication to handle communication between ICs crossing different interfaces. There are two key learnings. First, protocols describing data must be formed such that ICs can be developed independently from each other. Second, communication (data transport) crossing different interfaces can result in high data overhead if a software-based (polling) scheme is used instead of a hardware-based (interrupt) scheme.

REFERENCES

[1] "Embedded Instrumentation: Its Importance and Adoption in the Test and Measurement Marketplace, Frost and Sullivan, Whitepaper, 2010."

[2] K. Posse, "Component manufacturer perspective," in *2015 International Test Conference*, 2015, pp. 1–10.

[3] "IEEE standard test access port and boundary-scan architecture," *IEEE Std 1149.1-2001*, 2001.

[4] "IEEE standard for access and control of instrumentation embedded within a semiconductor device," *IEEE Std 1687-2014*, 2014.

[5] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Scan pattern retargeting and merging with reduced access time," in *European Test Symposium (ETS)*, 2013, pp. 39–45.

[6] M. Portolan, B. Van Treuren, and S. Goyal, "Executing IJTAG: are vectors enough?" *IEEE Design & Test*, vol. 30, no. 5, pp. 15–25, Oct 2013.

[7] J. Rearick and A. Volz, "A case study of using IEEE P1687 (IJTAG) for high-speed serial I/O characterization and testing," in *International Test Conference (ITC)*, 2006.

[8] F. G. Zadegan et al., "Reusing and Retargeting On-Chip Instrument Access Procedures in IEEE P1687," *Design & Test of Computers, IEEE*, vol. 29, no. 2, pp. 79 –88, april 2012.

[9] Y. Fkih, P. Vivet, B. Rouzeyre, M.-L. Flottes, G. Di Natale, and J. Schloeffel, "2D to 3D test pattern retargeting using IEEE P1687 based 3D DFT architectures," in *Computer Society Annual Symposium on VLSI (ISVLSI)*, 2014, pp. 386–391.

[10] R. Krenz-Baath, F. Ghani Zadegan, and E. Larsson, "Access time minimization in IEEE 1687 networks," in *International Test Conference (ITC)*, 2015.

[11] Z. Zhong, G. Li, Q. Yang, and K. Chakrabarty, "Access-time minimization for the ijtag network using data broadcast and hardware parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 1, pp. 185–198, 2021.

[12] IEEE P1687.1, "Standard for the Application of Interfaces and Controllers to Access 1687 IJTAG Networks Embedded Within Semiconductor Devices," Dec. 2016.

[13] A. Crouch, M. Laisne, and M. Keim, "Generalizing access to instrumentation embedded in a semiconductor device," *IEEE Computer*, vol. 50, no. 7, pp. 92–95, 2017.

[14] E. Larsson, P. Murali, and G. Kumisbek, "IEEE Std. P1687.1: Translator and Protocol," in *International Test Conference*, 2019, pp. 1–10.

[15] M. Laisne, H. von Staudt, A. Crouch, M. Portolan, M. Keim, M. Abdalwahab, B. Van Treuren, and J. Rearick, "Modeling Novel Non-JTAG IEEE 1687- Like Architectures," in *International Test Conference (ITC)*, 2020, pp. 1–10.

[16] IEEE P2654, "Standard for System Test Access Management (STAM) to Enable Use of Sub-System Test Capabilities at Higher Architectural Levels," Oct. 2018.

[17] M. Portolan, J. Rearick, and M. Keim, "Linking chip, board, and system test via standards," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–8.

[18] A. Tšertov *et al.*, "A suite of IEEE 1687 benchmark networks," in *International Test Conference (ITC)*, 2016.

TABLE III. AMOUNT OF DATA TRANSPORTED

Benchmark	PDL	Useful data	Type of data	ITC'19	Hardware-based	Software-based
50	iGet 1	8	Control overhead	16	32	64
			Data overhead	16	32	64
			Dummy overhead	0	0	8
			Total overhead	32	64	136
			Useful data (%)	20	11	6
	iWrite 1	8	Control overhead	16	32	32
			Data overhead	16	32	32
			Dummy overhead	0	0	0
			Total overhead	32	64	64
			Useful data (%)	20	11	11
	iGet all	920	Control overhead	800	816	4496
			Data overhead	16	32	3712
			Dummy overhead	0	0	920
			Total overhead	816	848	9128
			Useful data (%)	53	52	9
	iWrite All	920	Control overhead	800	816	0
			Data overhead	16	32	32
			Dummy overhead	0	0	0
			Total overhead	816	848	848
			Useful data (%)	53	52	52
	BASTION	3680	Control overhead	3200	4832	12192
			Data overhead	1632	3264	10624
			Dummy overhead	0	0	1848
			Total overhead	4832	8096	24656
			Useful data (%)	43	31	13
100	iGet 1	8	Control overhead	16	32	64
			Data overhead	16	32	64
			Dummy overhead	0	0	0
			Total overhead	32	64	128
			Useful data (%)	20	11	6
	iWrite 1	8	Control overhead	16	32	32
			Data overhead	16	32	32
			Dummy overhead	0	0	0
			Total overhead	32	64	64
			Useful data (%)	20	11	11
	iGet all	1856	Control overhead	1600	1616	9040
			Data overhead	16	32	7456
			Dummy overhead	0		1856
			Total overhead	1616	1648	18352
			Useful data (%)	53	53	9
	iWrite All	1856	Control overhead	16	32	32
			Data overhead	1600	1616	1616
			Dummy overhead	0	0	0
			Total overhead	1616	1648	1648
			Useful data (%)	53	53	53
	BASTION	7424	Control overhead	6400	9632	24480
			Data overhead	3232	6464	21312
			Dummy overhead	0	0	3712
			Total overhead	9632	16096	49504
			Useful data (%)	43	32	13
150	iGet 1	8	Control overhead	16	32	64
			Data overhead	16	32	64
			Dummy overhead	0	0	8
			Total overhead	32	64	136
			Useful data (%)	20	11	6
	iWrite 1	8	Control overhead	16	32	32
			Data overhead	16	32	32
			Dummy overhead	0	0	0
			Total overhead	32	64	64
			Useful data (%)	20	11	11
	iGet all	2800	Control overhead	2400	2416	13616
			Data overhead	16	32	11232
			Dummy overhead	0	0	2800
			Total overhead	2416	2448	27648
			Useful data (%)	54	53	9
	iWrite All	2800	Control overhead	2400	2416	2416
			Data overhead	16	32	32
			Dummy overhead	0	0	0
			Total overhead	2416	2448	2448
			Useful data (%)	54	53	53
	BASTION	11200	Control overhead	9600	14432	36832
			Data overhead	4832	9664	32064
			Dummy overhead	0	0	5600
			Total overhead	14432	24096	74496
			Useful data (%)	44	32	13