Accessing general IEEE Std. 1687 networks via functional ports

Larsson, Erik; Murali, Prathamesh ; Zhang, Zilin

Published in:
International Test Conference

DOI:
10.1109/ITC50571.2021.00051

2021

Accessing general IEEE Std. 1687 networks via functional ports

Erik Larsson, Prathamesh Murali, and Ziling Zhang
Lund University
Lund, Sweden
Email: erik.larsson@eit.lth.se

***Abstract*—Reconfigurable scan networks (RSNs), like IEEE Std. 1687 networks, offer flexible and scalable access to embedded (on-chip) instruments. These networks are typically accessed from the outside via a dedicated test port, like the test access port (TAP) of IEEE Std. 1149.1. As not all integrated circuits have a dedicated test port, the IEEE Std. P1687.1 working group is exploring how existing functional ports can be used. Fundamental challenges are to determine what hardware to include in the component translating information between a functional port and an IEEE Std. 1687 network and to describe a protocol for the data transported over a functional interface. We have previously shown hardware and protocol to access a limited type of IEEE Std. 1687 networks, known as flat segment insertion bit (SIB)-based networks. In this paper, we present a solution to handle general IEEE Std. 1687 networks. We have made a number of implementations with various benchmarks on an FPGA to evaluate the data overhead and the area usage.**

Keywords*—*IEEE Std. P1687.1, IEEE Std. 1687, IEEE Std. 1149.1, functional port, embedded instruments

I. Introduction

The semiconductor technology development makes it possible to constantly design and manufacture integrated circuits (ICs) with smaller, faster and more transistors. While the development gives many advantages, there are also many challenges. Smaller and faster transistors lead to tighter margins, both in device sizes and timing. Tighter margins lead to higher risks of wear-out effects and larger sensitivity to process variations (transistors length, widths, oxide thickness). Tighter margins, in combination with higher transistor count, increase the risks of malfunctioning, which lead to an increased need of possibilities to test, tune, re-configure, and so on. To accurately perform measurements, there is a need of on-chip, instead of off-chip (external), instrumentation [1]. Embedded (on-chip) instruments are increasingly used at different stages through the life cycle of ICs: from prototype debug, test and validation to in-field monitoring and test. A modern IC can include thousands of instruments [2].

IEEE Std. 1687 [3] was developed to offer flexible and scalable access to embedded instruments. The flexibility is achieved by the possibility to dynamically configuring the active scan-path so that only desired instruments are included, for example by means of segment insertion bits (SIBs). The standard 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. The access to the outside is typically via a dedicated test port, like the test access port of IEEE Std. 1149.1. To operate on instruments, an Electronic Design Automation (EDA) tool takes PDL and ICL as inputs and generates access patterns. These access patterns are transported to and from the IC using the protocol of IEEE Std. 1149.1 and translated by the TAP of IEEE Std. 1149.1 to the IEEE Std. 1687 network.

Rearick *et al.* described early in the development of IEEE Std. 1687 problems, activities and instruments [4]. Portolan described test design-flows with IEEE Std. 1687 [5]. Zadegan *et al.* made access time analysis [6] and explored design methods for IEEE Std. 1687 networks [7]. Baranowski *et al.* investigated minimization of reconfiguration steps of IEEE Std. 1687 networks [8]. Jutman *et al.* proposed the first work with IEEE Std. 1687 for fault management where the network was complemented with an additional infrastructure to speed-up fault detection [9]. A similar infrastructure as proposed by Jutman *et al.* [9] was used by Petersen *et al.* [10]. Zadegan *et al.* proposed self-configuring SIBs to speed-up fault localization [11]. Cantoro *et al.* proposed techniques to test the IEEE Std. 1687 network [12].

As not all ICs are equipped with an IEEE Std. 1149.1 TAP, the IEEE Std. P1687.1 [13] working group is currently exploring the use of functional interfaces, like serial peripheral interface (SPI), inter-integrated circuit (I2C), universal serial bus (USB), and advanced microcontroller bus architecture (AMBA) to enable access to IEEE Std. 1687 networks. The fundamental question is what hardware is needed instead of the IEEE Std. 1149.1 TAP and what protocol is needed to describe data transportation over a functional interface. There are some work on IEEE Std. P1687.1 [14]–[16]. We have previously proposed and explored protocols and hardware components such that a functional port can be used to access flat SIB-based IEEE Std. 1687 networks [15]. In this paper, we present a solution to handle general IEEE Std. 1687 networks.

The paper is organized as follows. In Section II we give a brief background to IEEE Std. 1687, how to access IEEE Std. 1687 networks via a dedicated test port, like IEEE Std. 1149.1, and via a functional port. We also describe our previously proposed hardware component and protocol to access flat SIB-based IEEE Std. 1687 networks via a functional port [15]. The proposed solution to handle general IEEE Std. 1687 networks accessed via a functional port is described in Section III. We provide an illustrative example of our solution in Section IV. Experimental results are in Section V and the paper is concluded in Section VI.

II. Background

In this section we provide a brief background to IEEE Std. 1687, how to access IEEE Std. 1687 networks via a dedicated test port like IEEE 1149.1 and via a functional port, including

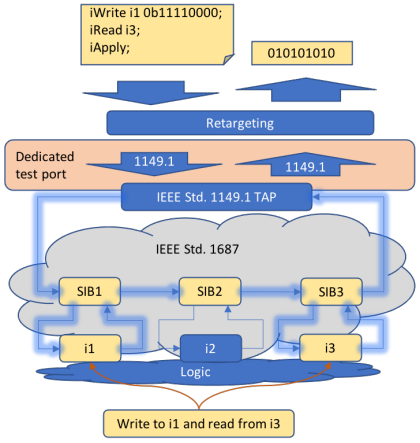


Fig. 1. Accessing reconfigurable scan networks using IEEE Std. 1687 via a dedicated test port, that is normally IEEE Std. 1149.1 TAP

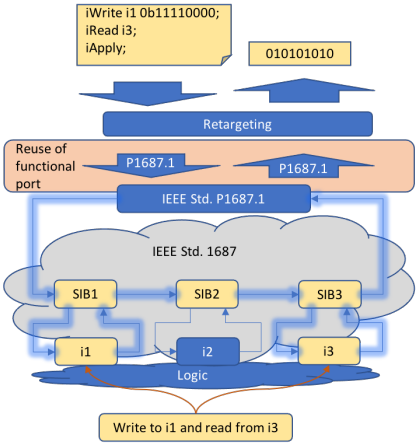


Fig. 2. Accessing reconfigurable scan networks using IEEE Std. 1687 via a functional port, the scope of IEEE Std. P1687.1

our previous work on using a functional port to access IEEE Std. 1687 networks.

Figure 1 shows an IEEE Std. 1687 network with three instruments, *i1*, *i2* and *i3*, connected in a flat manner with one SIB per instrument. The SIBs provide a possibility to dynamically configure the active scan-path by including or excluding instruments. The IEEE Std. 1687 network in Figure 1 is accessed from the outside via a dedicated test port, the IEEE Std. 1149.1 test access port (TAP).

Figure 1 shows some PDL with an iApply group to simultaneously write to instrument *i1* and read from *i3*. The retargeting process forms access patterns according to the protocol of IEEE Std. 1149.1. When applied, the access patterns will be received by the IEEE Std. 1149.1 TAP, which transforms the data to fit the IEEE Std. 1687 network. Smart retargeting includes only needed instruments in the active scan-path as any additional instruments lead to a need to transport more data and longer time for the shifting process. For the example in Figure 1, a smart active scan-path sets SIBs such that *i1* and *i3* are included but *i2* is excluded.

Figure 2 shows the same IEEE Std. 1687 network as in Figure 1 with the difference that a functional port is used instead of a dedicated test port for access with the outside. In the retargeting process, the data (access patterns) for the IEEE Std. 1687 network is formed to be transported over the functional port, instead of using IEEE Std. 1149.1, and some hardware translates data transported on the functional port such that it fits the IEEE Std. 1687 network. The topic of specifying the protocol for how data should be transported over a functional interface and what hardware to include in the component placed between the functional port and the IEEE Std. 1687 is in the scope of the on-going IEEE Std. P1687.1 working group.

We have previously investigated some protocols and corresponding hardware components such that a functional port can be used to access IEEE Std. 1687 networks [15]. Figure 3 shows the alternative that gave least need of transported data. In respect to protocol, a given iApply group is translated into one or more control commands followed by one or more data commands. Figure 3 shows the protocol and the hardware component for the same iApply group as in Figure 1. We see in Figure 3 that the first control command is for *i1*. There is one bit indicating that this is a control command, one bit to indicate that a write operation should be performed and an address to instrument *i1*. The second control command is for *i3*. There is one bit indicating that the command is a control command, one bit indicating that a read operation should be performed and an address to instrument *i3*. The third command is a data command, indicated by the most significant bit in the first byte is set to 1. The following 15 bits specify the number of bytes of data that will follow. In this example, these bits specify the value one, which means that one byte of data follows. This is the data (`0b1111000`) that should be written to *i1*.

The hardware component is based on a finite state machine (FSM) complemented with an instrument control register (ICR), a SIB control register (SCR), and an instrument length memory (ILM). ICR keeps a position for each instrument to indicate the type of operation, read or write, and SCR keeps the value for each SIB, active or inactive. Control commands for a given iApply group set suitable values of ICR and SCR. ILM, which contains the length in bits of each instrument, is fixed at design time. The control commands for the iApply group in Figure 3 set SCR such that *SIB1* and *SIB3* are active and set ICR such that a write is performed on instrument *i1* and a read is performed on instrument *i3*.

The FSM operates as follows. First, when the first control command arrives, which indicates the beginning of a new iApply group, the FSM resets the IEEE Std. 1687 network such that only SIBs are included in the active scan-path.

Second, when the first data command arrives, indicating that ICR and SCR are fully configured according to the iApply group, the FSM sets the active scan-path by traversing the SCR and shifting in corresponding bits. For the example in Figure 4, the first bit to be shifted is the value of *SIB3*, which is 1, then 0 for *SIB2*, and finally 1 for *SIB1*. The values that are shifted out during this process are dropped by the FSM; hence, not sent back via the functional port.

Third, the FSM starts to operate on the active scan-path. The shift-in sequence is constructed as follows. The first bit in the shift-in sequence is the value for *SIB3*, which the FSM gets from SCR at position *SIB3*. As this value is 1 the FSM knows that corresponding instrument (*i3*) should be included in the active scan-path. Hence, the FSM checks ICR to learn that a read operation should be performed, which means that the shift-out values should be returned over the functional port. The

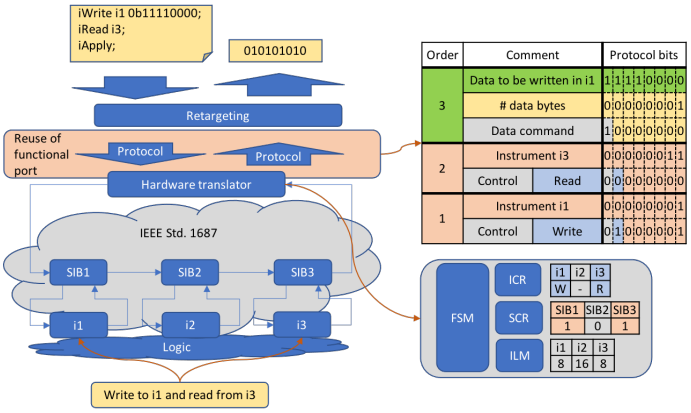


Fig. 3. Protocol and hardware translator to access IEEE Std. 1687 networks via a functional port [15]

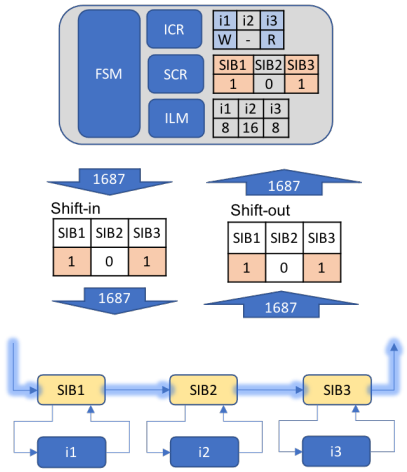


Fig. 4. Setting the active scan-path [15]

FSM checks ILM for *i3* to know the length of the instrument, the number of shifts. As the operation on *i3* is read, the FSM creates dummy bits that are shifted in to push out the content of *i3*. The FSM proceeds to *SIB2* and finds by checking SCR that the value is 0, which means corresponding instrument (*i2*) is not in the active scan-path, the FSM shift in a 0 for *SIB2*. And then the FSM proceeds with *SIB1*. *SIB1* holds the value 1, which means that *i1* is included in the active scan-path. The FSM checks ICR to learn that a write operation should be applied. This means that data transported on the functional bus should be shifted in, which arrives in a data command (`0b1111000`). The FSM checks ILM for *i1* to learn how much data to shift-in. The data that is shifted out is dropped by the FSM. In general, the FSM drops all shift-out data expect the data that is read from instrument *i3*.

The protocol and hardware component we developed are general and can be applied using any functional interface. The fact that key information, basically the complete ICL describing the IEEE Std. 1687 is included in the FSM leads to that the amount of data that needs to be transported is very low. However, there are some important shortcomings. The scheme is only applicable to SIB-based IEEE Std. 1687 networks, the structure of the IEEE Std. 1687 network must be flat, that is, SIBs can only be on the top-level, and there can only be one instruments, same operation, under each SIB.

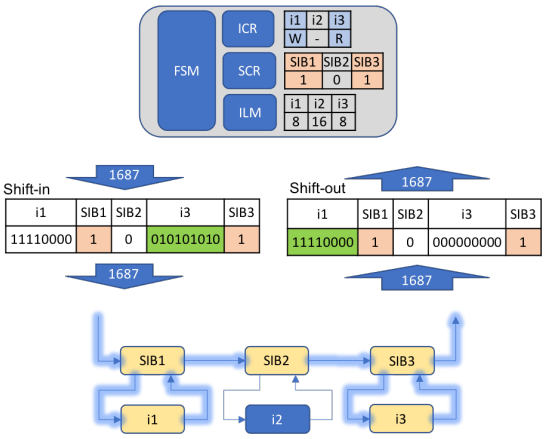


Fig. 5. Applying operation to the active scan-path [15]

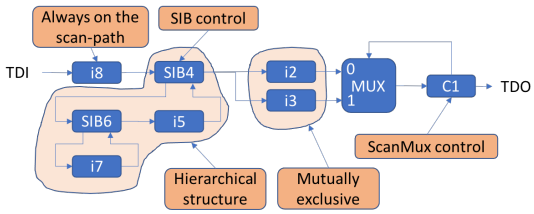


Fig. 6. A general IEEE Std. 1687 network

III. Translator and Protocol

In this section we describe our approach to access and operate general IEEE Std. 1687 networks via a functional port. Figure 6 shows a general IEEE Std. 1687 network with sequential elements, that is instruments, SIBs and ScanMux control. These elements are connected in a general (non-flat) manner. We make the following observations:

- There are several constructs to define the active scan-path, for example SIBs, like *SIB4*, and ScanMuxes, like *C1*
- Some instruments are always on the scan-path and cannot be excluded, for example instrument *i8*.
- Some instruments must be on the scan-path under some conditions, for example if instrument *i7* is on the active scan-path, instrument *i5* is also included.
- There is a hierarchy which leads to that several configuration steps may be needed to include some instruments, for example if *SIB4* is not on the active scan-path, there is a need to first include *SIB4* and then *SIB6* to access instrument *i7*
- Some instruments are mutually exclusive such that they cannot be on the active scan-path at the same time, for example instruments *i2* and *i3*

Our scheme to operate general IEEE Std 1687 networks is as follows.

First, we number the sequential elements in a given IEEE Std. 1687 network in the order they can appear on an active scan-path. We will use the general IEEE Std. 1687 network in Figure 8 for illustration. First bit to be set in a shift-in sequence (or the first bit that is shifted out) is that for *C1*. We therefore assign *C1* with number 1. The second group of bits to shift in

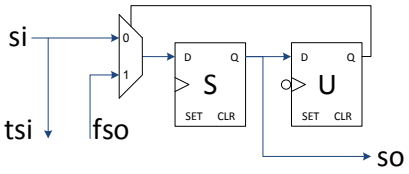


Fig. 7. Illustration of a SIB

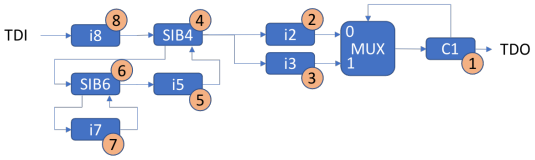


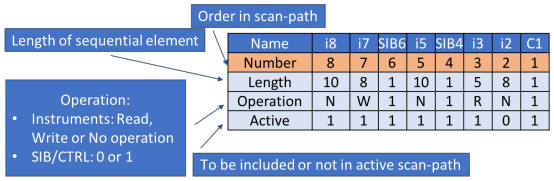
Fig. 8. Numbering of sequential elements in an IEEE Std. 1687 network

are determined by the value of *C1*. If *C1* = 0 the multiplexor selects *i2* and if *C1* = 1 the multiplexor selects *i3*. These two instruments are mutually exclusive, which means they cannot be on the active scan-path at the same time. Hence, the individual numbering of them is not important so we number them in an arbitrary way. In this example, we give *i2* number 2 and *i3* number 3. Next bit to consider is related to *SIB4*. The numbering here depends on the SIB implementation. Let us assume a SIB implementation is as in Figure 7, which leads to that the content of a SIB appears before anything controlled by the SIB. Hence, we give *SIB4* number 4. With an implementation of SIBs where shift-out from the SIB appears after anything controlled by the SIB, the SIB gets a higher number as it, if included in the active scan-path, appears later. Next groups of bits to consider are related to instruments *i8* and *i5*. We make the following observation; there is no possibility to include *i5* and *i8* in the an active scan-path such that *i8* comes before *i5*. Hence, our numbering scheme gives instrument *i5* number 5. For the rest of the sequential elements, the numbering is straight forward; *SIB6* gets number 6, *i7* number 7, and finally, *i8* gets number 8.

Second, for the numbered elements, we create a table with one column per element, see Figure 9. Each column specifies for a given element; length in bits, current operation, and if the element is included in the active scan-path or not. The length is the number of bits, which for a SIB is one and for an instrument, like *i5* is 10. This information is fixed at design time. Operations for elements like SIBs and ScanMuxes are 0 or 1 and for other instruments, like *i7*, read (R), write (W), and No operation (N). A read operation means that data should be taken from the particular instrument and reported and a write operation means that data from the outside should be placed in the instrument. An instrument marked with no operation means that the instrument is on the active scan-path but no operation should be performed on it. The operation is needed for instruments that appears on the active scan-path but there is no operation specified for them in the iApply group. For example, instrument *i8* is always on the active scan-path but there might not always be an operation for it.

Given a table for an IEEE Std. 1687 network, like the one in Figure 9, we create an FSM complemented with registers storing data in the table. We have:

- *Length* of sequential elements, which is fixed at design time, are stored in Length Memory (LM).



Name	i8	i7	SIB6	i5	SIB4	i3	i2	C1
Number	8	7	6	5	4	3	2	1
Length	10	8	1	10	1	5	8	1
Operation	N	W	1	N	1	R	N	1
Active	1	1	1	1	1	1	0	1

Fig. 9. Numbering and information of sequential elements in an IEEE Std. 1687 network

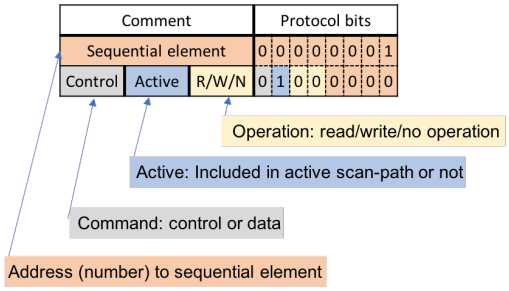


Fig. 10. Our numbering of sequential elements in an IEEE Std. 1687 network

- Current *Operation* on a given sequential element is stored in Operation Register (OR). We have three types of operations: *Read*, *Write* and *No operation*.
- *Active* is a flag to determine if an element should be included in the active scan-path or not, which is stored in Active Register (AR)

The protocol we use to set OR and AR is based on control commands and data commands, as proposed by Larsson *et al.* [15]. These commands where introduced for flat SIB-based IEEE Std. 1687 networks with one SIB associated with each instrument. To handle general IEEE Std. 1687 networks, we updated the control command to separate the control of SIBs and instruments, which means that each sequential element in an IEEE Std. 1687 network can be individually controlled. And, to be able to control if a sequential element should be included in the scan-path or not we added one bit to indicate if an element should be active or not (in the active scan-path). The updated control command is shown in Figure 10.

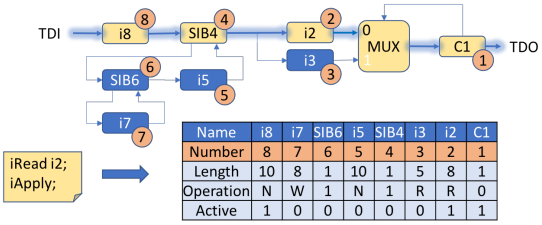
The data command is as proposed by Larsson *et al.* [15]. The data command consists of 2 bytes where the most significant bit in the first byte that arrives to the FSM determines if the command is a data command or a control command, see Figure 11. The remaining bits are used to specify the number of bytes of data needed for the active scan-path. In the example in Figure 11, the fifteen bits specifies the value one, meaning that one byte of data follows. It should be noted that the active scan-path is rarely an exact multiple of 8 bits (a byte). However, the FSM knowns the length of the active scan-path and knows exactly how many bits that are needed. Hence, the additional bits needed to fill-up the last byte are simply dropped. The worst that can occur is that one bit is to be sent, which mean that seven out of eight bits in the byte are dropped. However, in practice, huge amount of data will be transported, which means that losing a couple of bits in the last byte has a minor impact on data overhead.

The FSM includes three main states:

1) The FSM waits while control commands set OR and AR. When the first data command arrives, the FSM

Comment	Protocol bits							
Data for the active scan-path	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

Fig. 11. Our numbering of sequential elements in an IEEE Std. 1687 network



Name	i8	i7	SIB6	i5	SIB4	i3	i2	C1
Number	8	7	6	5	4	3	2	1
Length	10	8	1	10	1	5	8	1
Operation	N	W	1	N	1	R	R	0
Active	1	0	0	0	0	0	1	1

Fig. 12. Illustration of active scan-path

generates a *capture_en* signal to capture instrument data in their corresponding shift register.

2) Once data has been captured, the shift process begins. The FSM traverses OR, AR and LM for each sequential element to learn what action to take. For a given sequential element, a column in Figure 9, there are several possible combination. First, FSM checks AR to learn if current sequential element is included in active scan-path. If AR stores the value 0, current sequential element is not part of the active scan-path; hence, no shift-in data is created for this element. The FSM moves to next sequential element. If AR, on the other hand, stores the value 1, current sequential element is part of the active scan-path. The FSM checks OR to find the type of operation and LM to find the length in bits of the element. The possible operations on an element are:
   - *Read*. The FSM performs two operations on the number of bits shifted out from the element. First, the FSM returns them out of the system as this is requested by the iGet in the iApply group. Second, the FSM feeds back the shift-out bits to the shift-in such that the instrument maintains the same value as before the shift process.
   - *Write*. The FSM shifts from the data command the number of bits given by LM.
   - *No operation*. If there is no operation specified in the iApply group for an element but the element is on the active scan-path, the FSM takes the shift out bits for the element and feeds them directly back to scan in to maintain current value in the elements shift register.
3) Once the FSM has traversed all elements, the table in Figure 9, an *update_en* signal is generated to make data go from shift registers to instruments.

IV. Example

In this section we illustrate with examples some aspects on how our scheme handles general IEEE Std. 1687 networks.

We begin with an iApply group to read from instrument *i2*:

```
iRead i2;
iApply;
```

in an IEEE Std. 1687 network as given in Figure 12.

We note that the iApply group specifies an operation on instrument *i2* and that for this IEEE Std. 1687 network the active scan-path must include *C1*, *i2*, *SIB4*, and *i8*. Hence, while there is no operation given for instrument *i8*, it is due to the design of the IEEE Std. 1687 network included in the active scan-path. This means that data must be handled for instrument *i8* during the shift process. There are two aspects. First, the number of shifts must consider all instruments on the active scan-path. Second, it is specifically required by IEEE 1687 to maintain the content of their associated shift-registers unless there is a write which explicitly should make a change. While IEEE 1687 indicates that the ”previously written” value must be maintained after a scan shift for an element not currently being written, our scheme will maintain the ”just captured” value. It should be noted that all instruments on the active scan-path will perform capture-shift-update. One alternative is that data is shifted out from the active scan-path and modified externally such that the new shift-in pattern includes the desirable values. Such a scheme leads to additional transportation of data. Our scheme is to directly feedback data from the shift-out to the shift-in. The scheme is used for all instruments in the active scan-path except for those where data should be modified, for example due to a write. In order to implement the scheme, we increase the shift process with one bit such that the output from the active scan-path can be returned to the shift-in.

Let us assume that control commands have been used to setup the data, detailed in the table in Figure 12 and that a data command, which is empty as there is no write operation in the iApply group, arrives to initiate the shift process. The shift process is as follows. First a dummy bit is shift in and concurrently we shift out current value of *C1*. We check the column for *C1* in the table in Figure 12 and learns that the element should be on the active scan path and the operation is 0, so the first real bit (apart from the dummy bit) to be shifted-in is a 0. Instrument *i2* is the second element in the table and it is active, with length 5 bits, and the operation is read, which means that 5 bits should be returned to the outside. To maintain the value in the shift register the shift-out is also used as shift-in. Instrument *i3* is the third element in the table and it is not active, hence no shift occurs for this element. The fourth element is *SIB4* and it is closed and on the active scan-path so a value 0 is shifted in. The output at this shift, current value of the SIB, has no value and is dropped. For elements 5 to 7 in the table, all are inactive, which means they are ignored and no shifts are created for them. The last element on this active scan-path is instrument *i8*. When checking the table in Figure 12, the operation is N (No operation), which means there is no operation for the instrument in the iApply group. The output from the shifting is feed-back so that the instrument contains the same value as before the shifting, and those bits are not sent as part of the output stream.

Our second example illustrates how to reconfigure the IEEE Std. 1687 network, that is to go from accessing one set of instruments to accessing another set of instruments. In previous work [15], which was limited to flat SIB-based IEEE Std. 1687 networks, the FSM performed an automatic reset before operating on a new iApply group. The main advantages are that each individual iApply group can be handled independently as the IEEE Std. 1687 network is always in a reset state and that configuration to set the active scan-path is simple and quick;

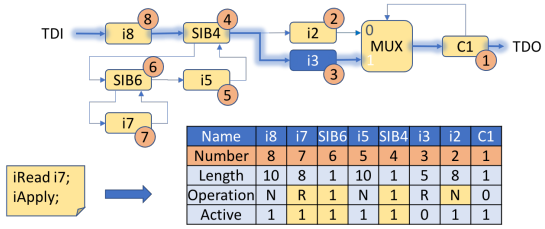


Fig. 13. Reconfiguring going from configuration in Figure 12 to read (iGet) from instrument *i3*

one step is needed to decide which SIBs to open and then the operation on the active scan-path can begin. In current work, where we handle general IEEE Std. 1687 networks, several steps can be needed to go from accessing one set of instruments to accessing another set of instruments. We assume that we have performed the iApply group in Figure 12 and would like to do a read from instrument *i7*, see Figure 13:

```
iRead i7;
iApply;
```

The first step is to indicate set instrument *i2* to invalid as there is no operation for it in the iApply group and change *SIB4* so that it opens up for access. These modifications require two control commands and one data command to execute:

```
Control command: set i2 to no operation
Control command: open SIB4
Data command
```

In the second step, there is a need to include instrument *i5* in the active scan-path and change values for *SIB6* such that it is opened and included in the active scan-path:

```
Control command: include i5
Control command: include SIB6
Control command: open SIB6
Data command
```

In the third step, there is a need to include instrument *i7* and set it for the read operation:

```
Control command: include i7
Control command: set i7 to read
Data command
```

V. Experimental Results

The objective of the experiments is to evaluate overhead in respect to transported data and implementation area. The transported data overhead is computed as described in Section V-A and proposed scheme is compared against so called Bit-banging, which is a straight forward scheme, described in Section V-B for all benchmarks and also against the approach in [15] for the flat SIB-based benchmark. The three benchmarks that have been used are described in Section V-C. The results are presented in Section V-D and the results are discussed in Section V-E.

A. Overhead

The number of useful bits that needs to be transported can be extracted directly from PDL and ICL. For example, assume a write operation on a 32-bit long instrument. This operation requires 32 bits of useful data, the information that is written to the instrument. A read operation on the same instrument results in that 32 bits are taken from the instrument. All additional bits needed to enable operations are considered as overhead. For proposed scheme, the communication over a functional port to an IEEE Std. 1687 network is formed into control command and data command. One control command requires 16 bits of data, which we call control overhead. One data command requires also 16 bits of data, which we call data overhead.

B. Bit-banging

Bit-banging is a straight forward scheme that uses a minimum of hardware to control and operate an IEEE Std. 1687 network. We assume that the data that should be applied is stored in a memory and the content of the memory is directly applied to the IEEE Std. 1687 interface. An IEEE Std. 1687 interface consists of eight signals, that is seven inputs, *ScanIn, CaptureEn, ShiftEn, UpdateEn, Select, Reset,* and one output, *ScanOut*. As clock (TCK) is one input, there is a need to store and handle information when TCK is low and when TCK is high. Hence, each clock cycle requires 16 bits for the eight signals.

C. Benchmarks

We have used three IEEE Std. 1687 benchmarks, named Flat SIB-based, General Flat and SIB-based Perfect Tree. For each benchmark we have made three versions with different number of instruments and for each version we applied variations of PDL.

The Flat SIB-based benchmark, presented in [15], has one SIB per instrument and the SIBs are connected in a flat manner. For a setup with 30 instruments, there are 30 SIBs connected in a chain and each SIB controls one instrument. The length of instruments in bits are as follows. First instrument is of length 8, second 16, third 32, fourth 8, and so on. We have used three sizes of this benchmark with 30, 60 and 90 instruments.

We created the General Flat benchmark and the SIB-based Perfect Tree benchmark. The General Flat benchmark is based on a fixed module that is repeated n number of times. Figure 14(a) shows the fixed module, which contains three instruments, one SIB and one ScanMux control. The instruments are of length 8, 16, and 32 bits. To scale the benchmark, we concatenate n of these modules, as shown in Figure 14(b). We have used three sizes of this benchmark, with 30, 60 and 90 instruments. The SIB-based Perfect Tree benchmark is based on a perfect binary tree structure. A perfect binary tree of height h has $2^{(h+1)} - 1$ elements. We have fixed the height of the benchmark to 5, which means we have $(2^{(h+1)} - 1 =)$ 63 SIBs and 32 instruments. To scale the benchmarks such that there are more instruments, we keep the height fixed and include additional instruments on intermediate levels, that means we do not include additional instruments at top-level and bottom-level. On top-level we always keep one SIB and on bottom-level there are always 32 instruments. If we want to have a benchmark with 48 instruments, that is 16 additional instruments, we distribute them evenly on the four levels as follows. In this case, four instruments per level, see Figure 15. We note that the four instruments on level 1 are distributed such that two instruments are after SIB1.1 and two instruments are after SIB 1.0. For the four additional instruments on level 2, there is one instrument

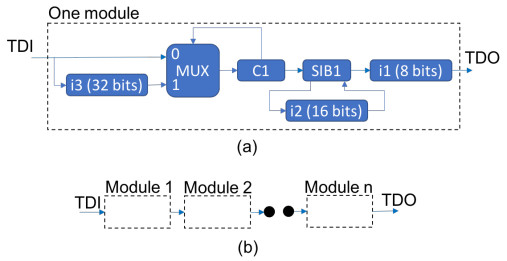


Fig. 14. General Flat benchmark: (a) A single module (b) Concatenation of n modules

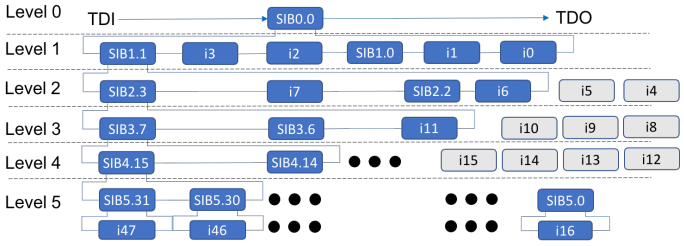


Fig. 15. Illustration of SIB-based Perfect Tree with 16 additional instruments

after each SIB. The length of instruments in bits are as follows. First instrument 8, second 16, third 32, fourth 8, and so on. We have used three sizes of this benchmark, with 48, 96 and 144 instruments.

For PDL, we have used the following iApply groups: Write to instrument 1, Read from instrument 1, Write to all instruments, and Read from all instruments. In addition, we have used the PDL scheme from the BASTION benchmarks [17], which is to first perform one iApply group with a 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 a benchmark with 30 instruments the scheme results in 62 iApply groups.

D. Results

The results are produced using an Artix-7 (XC7A100T-1CSG324C) field-programmable gate array (FPGA) where we used the universal asynchronous receiver-transmitter (UART) as the functional interface.

The results on transported data are in Tables I, II and III. In Table I, proposed scheme is compared against [15] and Bit-banging. The first column states the number of instruments, the second column indicates the PDL that has been used, and the third column reports the number of useful bits, which is given directly from the PDL and the ICL. Columns four to seven list data overhead, control overhead, total overhead and the ratio of useful bits versus total number of bits transported for proposed scheme. Columns eight to eleven list the same for the scheme in [15]. Column twelve reports the total overhead for the Bit-banging scheme and column thirteen reports the ratio of useful bits versus total number of bits transported.

Tables II and III report results where proposed scheme is compared against Bit-banging. Note that the scheme in [15] is not applicable on these benchmarks. Tables II and III are organized in the same way. The first column states the number of instruments, the second column indicates the PDL that has been used, and the third column reports the number of useful bits, which is given directly from the PDL. Column four to seven list data overhead, control overhead, total overhead and the ratio of useful bits versus total number of bits transported for proposed scheme. Column eight reports the total overhead and column nine the ratio of useful data over total number of bits transported for Bit-banging.

The results on area overhead are reported in Tables IV, V, VI and VII, which are organized in the same way. Column one lists the number of instruments. There are four columns for each scheme listing the number of flip-flops (FF) and the number of look-up tables (LUT) for the UART transceiver, proposed controller, the IEEE Std. 1687 network, and total area. In addition, we list the relative number (%) of FFs and LUTs in relation to the IEEE Std. 1687 network.

E. Discussion

We collected the transported data overhead in Figure 16 and the area overhead in Figure 17 from the experiments using BASTION PDL. The first observation we make is that Bit-banging gives huge data overhead, which leads us to conclude that some smarter approach is needed. Our second observation is that data overhead for proposed scheme is worse than the approach presented at ITC'19. It should be noted that the scheme presented at ITC'19 is limited to a particular style of IEEE Std. 1687 networks, known as flat SIB-based. Given results on three benchmarks we observe that the design of IEEE Std. 1687 networks, the way instruments are connected, has a quite large impact on the data overhead. When developing solutions to access IEEE Std. 1687 networks without some dedicated test interface, like the work of the IEEE Std. P1687.1 working group, one should be careful not violating the basic concepts of IEEE Std. 1687, which is to provide flexibility and scalability. In respect to area overhead, a Bit-banging scheme provides no area. However, in order to use bit banging via a functional port, that port's clock must become the source for the TCK of the 1687 network. While proposed scheme gives more area than the scheme at ITC'19, it is interesting to note that the relative area of the proposed controller in respect to the IEEE Std. 1687 network decreases as the number of instruments increases, which indicates that area is kept under control as design sizes increases.

VI. Conclusions

Reconfigurable scan networks, like IEEE Std. 1687, offer flexible and scalable access to the increasing number embedded (on-chip) instruments that are needed in modern integrated circuits (ICs). As not all ICs, have a dedicated test port, like IEEE Std. 1149.1, the IEEE Std. P1687.1 is working towards a standard to enable the use of functional ports to access IEEE Std. 1687 networks. We have proposed a hardware component and protocol to enable access to general IEEE Std. 1687 network for ICs without a dedicated test port, like IEEE Std. 1149.1. We have implemented our scheme on an FPGA and made experiments on several benchmarks with various number of instruments. A key feature of our scheme is that our hardware component ensures that instruments maintain their original content after shift operations through a scheme that directly feedback shift-out data to the shift-in, in cases when instruments should not be updated.

TABLE I. TRANSPORTED DATA FOR THE FLAT SIB-BASED BENCHMARK

Instruments	PDL	Useful bits	Proposed				ITC'19				Bit-banging	
			Data overhead	Control overhead	Total overhead	Useful data (%)	Data overhead	Control overhead	Total overhead	Useful data (%)	Total overhead	Useful data (%)
30	iWrite 1	16	32	48	80	16.7	16	16	32	33.3	1104	1.4
	iRead 1	16	32	48	80	16.7	16	16	32	33.3	1104	1.4
	Allwrite	560	32	1440	1472	27.6	16	480	496	53.0	34560	1.6
	Allread	560	32	1440	1472	27.6	16	480	496	53.0	34560	1.6
	BASTION	2240	1968	6608	8576	20.7	992	1920	2912	43.5	168844	1.3
60	iWrite 1	16	32	48	80	16.7	16	16	32	33.3	1884	0.8
	iRead 1	16	32	48	80	16.7	16	16	32	33.3	1884	0.8
	Allwrite	1120	32	2880	2912	27.8	16	960	976	53.4	115920	1.0
	Allread	1120	32	2880	2912	27.8	16	960	976	53.4	115920	1.0
	BASTION	4480	3888	13328	17216	20.6	1952	3840	5792	43.6	548814	0.8
90	iWrite 1	16	32	48	80	16.7	16	16	32	33.3	2664	0.6
	iRead 1	16	32	48	80	16.7	16	16	32	33.3	2664	0.6
	Allwrite	1680	32	4320	4352	27.9	16	1440	1456	53.6	244080	0.7
	Allread	1680	32	4320	4352	27.9	16	1440	1456	53.6	244080	0.7
	BASTION	6720	5808	18624	24432	21.6	2912	5760	8672	43.7	1139384	0.6

TABLE II. TRANSPORTED DATA FOR THE GENERAL FLAT BENCHMARK

Instruments	PDL	Useful bits	Proposed				Bit-banging	
			Data overhead	Control overhead	Total overhead	Useful data (%)	Total overhead	Useful data (%)
30	Write 1	16	32	48	80	16.7	3564	0.4
	Read 1	16	32	48	80	16.7	3564	0.4
	Allwrite	560	32	1120	1152	32.7	90620	0.6
	Allread	560	32	1120	1152	32.7	90620	0.6
	BASTION	2240	1776	5360	7130	23.8	428494	0.5
60	Write 1	16	32	48	80	16.7	7424	0.2
	Read 1	16	32	48	80	16.7	7424	0.2
	Allwrite	1120	32	2240	2272	33.0	432800	0.3
	Allread	1120	32	2240	2272	33.0	432800	0.3
	BASTION	4480	3536	11120	14565	23.4	2126080	0.3
90	Write 1	16	32	48	80	16.7	11104	0.2
	Read 1	16	32	48	80	16.7	11104	0.2
	Allwrite	1680	32	3360	3392	33.1	970800	0.2
	Allread	1680	32	3360	3392	33.1	970800	0.2
	BASTION	6720	5296	16720	22016	23.3	4669920	0.2

TABLE III. TRANSPORTED DATA FOR THE SIB-BASED PERFECT TREE BENCHMARK

Instruments	PDL	Useful bits	Proposed				Bit-banging	
			Data overhead	Control overhead	Total overhead	Useful data (%)	Total overhead	Useful data (%)
48	iWrite 1	16	32	144	176	8.3	1157	1.4
	iRead 1	16	32	144	176	8.3	1157	1.4
	Allwrite	896	128	3552	3680	19.6	16884	5.3
	Allread	896	128	3552	3680	19.6	16884	5.3
	BASTION	3584	4240	18784	23024	13.5	293620	1.2
96	iWrite 1	16	32	336	368	4.2	4965	0.3
	iRead 1	16	32	336	368	4.2	4965	0.3
	Allwrite	1792	128	5088	5216	25.6	33012	5.4
	Allread	1792	128	5088	5216	25.6	33012	5.4
	BASTION	7168	5776	28768	34544	17.2	1760260	0.4
144	iWrite 1	16	32	528	560	2.8	8773	0.2
	iRead 1	16	32	528	560	2.8	8773	0.2
	Allwrite	2688	128	6624	6752	28.5	49140	5.5
	Allread	2688	128	6624	6752	28.5	49140	5.5
	BASTION	10752	7328	40480	47808	18.4	4372900	0.2

TABLE IV. AREA OVERHEAD FOR THE ITC'19 SCHEME ON THE FLAT SIB-BASED BENCHMARK

Instruments	UART		Controller		Network		Total		Controller compared to network	
	FF	LUT	FF	LUT	FF	LUT	FF	LUT	FF	LUT
30	109	179	261	663	1184	1223	1554	2065	22%	54%
60	109	179	321	759	2364	2437	2794	3375	14%	31%
90	109	179	381	845	3544	3652	4034	4676	11%	23%

TABLE V. AREA OVERHEAD FOR THE PROPOSED SCHEME ON THE FLAT SIB-BASED BENCHMARK

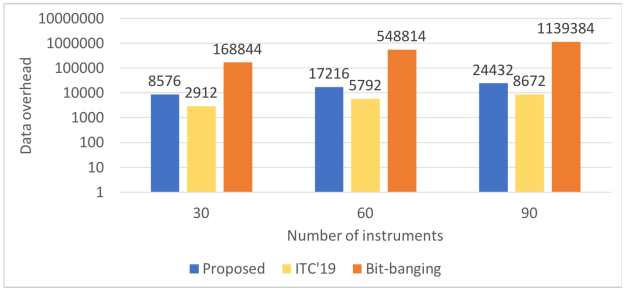
Instruments	UART		Controller		Network		Total		Controller compared to network	
	FF	LUT	FF	LUT	FF	LUT	FF	LUT	FF	LUT
30	110	178	380	821	1184	1222	1673	2221	32%	54%
60	109	178	564	1068	2364	2436	3037	3682	23%	31%
90	109	178	749	1299	3544	3651	4401	5128	21%	36%

TABLE VI. AREA OVERHEAD FOR THE GENERAL FLAT BENCHMARK

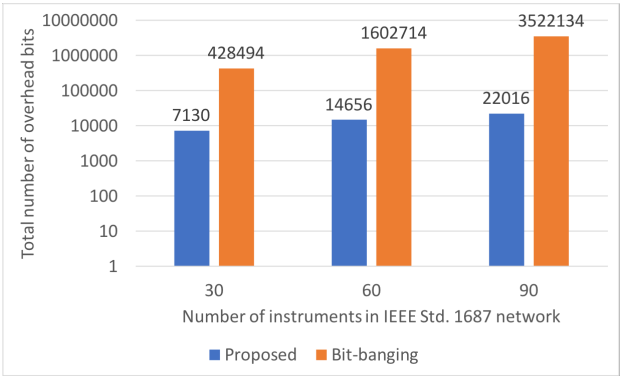
Instruments	UART		Controller		Network		Total		Controller compared to network	
	FF	LUT	FF	LUT	FF	LUT	FF	LUT	FF (%)	LUT (%)
30	109	178	350	828	1167	1185	1626	2191	30%	70%
60	110	178	503	1200	2333	2362	2945	3751	21%	51%
90	109	178	656	1504	3496	3637	4261	5320	19%	41%

TABLE VII. AREA OVERHEAD FOR THE SIB-BASED PERFECT TREE BENCHMARK

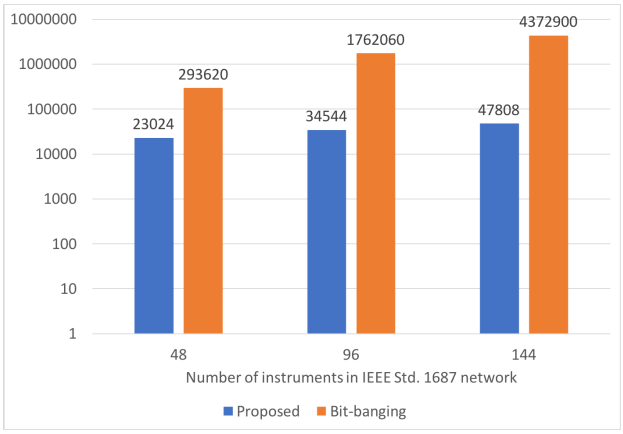
Instruments	UART		Controller		Network		Total		Controller compared to network	
	FF	LUT	FF	LUT	FF	LUT	FF	LUT	FF (%)	LUT (%)
48	109	178	537	1022	1937	1992	2583	3192	28%	51%
96	109	178	683	1249	3732	3789	4524	5216	18%	32%
144	109	178	841	1437	5550	5605	6500	7220	15%	26%



(a) Flat SIB-based



(b) General Flat



(c) Perfect Tree

Fig. 16. Data overhead on benchmarks: Flat SIB-based, General Flat and Perfect Tree

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

[4] J. Rearick *et al.*, "IJTAG (Internal JTAG): A step toward a DFT standard," in *International Test Conference (ITC)*, 2005.

[5] M. Portolan, "A novel test generation and application flow for functional access to IEEE 1687 instruments," in *European Test Symposium (ETS)*, 2016.

[6] F. Zadegan *et al.*, "Access time analysis for ieee p1687," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1459–1472, Oct 2012.

[7] F. Ghani Zadegan *et al.*, "Design automation for IEEE P1687," in *Design, Automation & Test in Europe Conference (DATE)*, 2011.

[8] R. Baranowski, M. Kochte, and H.-J. Wunderlich, "Modeling, verification and pattern generation for reconfigurable scan networks," in *International Test Conference (ITC)*, 2012.

[9] A. Jutman, S. Devadze, and J. Aleksejev, "Invited paper: System-wide fault management based on IEEE P1687 IJTAG," in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, June 2011, pp. 1–4.

[10] K. Petersen *et al.*, "Fault injection and fault handling: an MPSoC demonstrator using IEEE P1687," in *IEEE International On-Line Testing Symposium (IOLTS), 2014*, 2014, pp. 170–175.

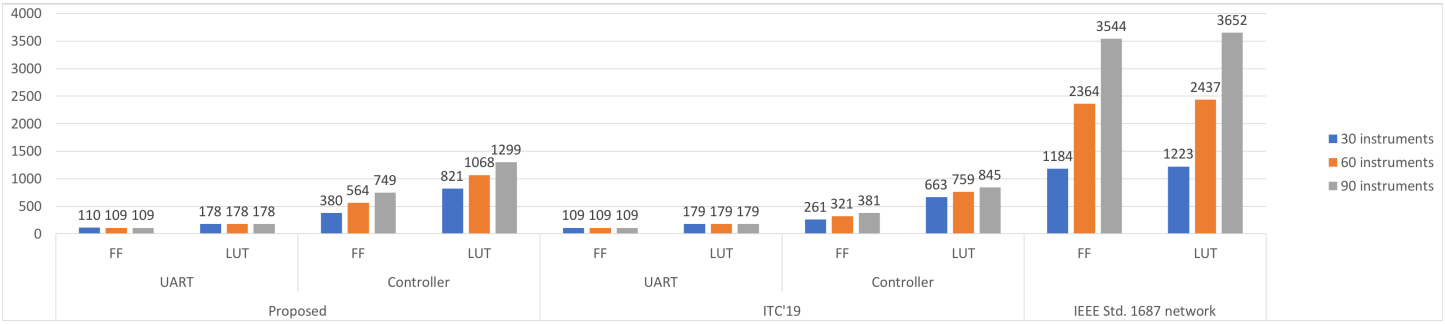
[11] F. G. Zadegan, D. Nikolov, and E. Larsson, "On-chip fault monitoring using self-reconfiguring IEEE 1687 networks," *IEEE Transactions on Computers*, vol. 67, pp. 237–251, 2018.

[12] R. Cantoro *et al.*, "Test of reconfigurable modules in scan networks," *IEEE Trans. on Computers*, vol. 67, no. 12, pp. 1806–1817, Dec 2018.

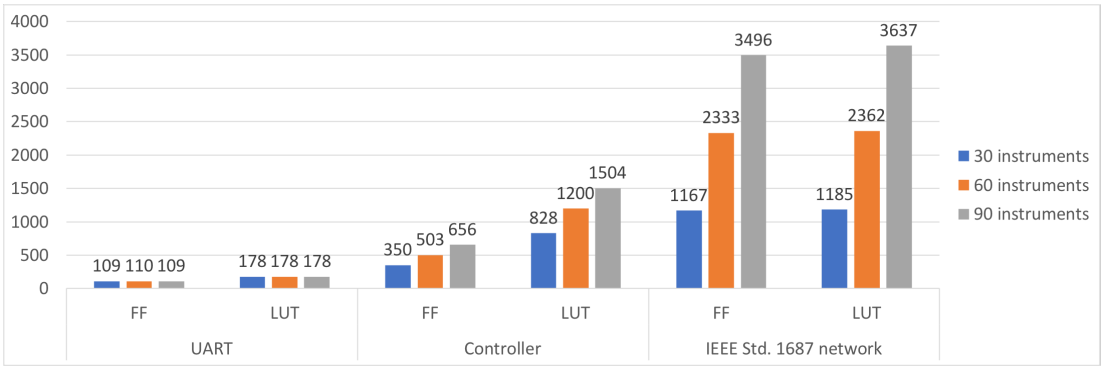
[13] IEEE P1687.1, "Standard for the Application of Interfaces and Controllers to Access 1687 IJTAG Networks Embedded Within Semiconductor Devices," Dec. 2016.

[14] 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.

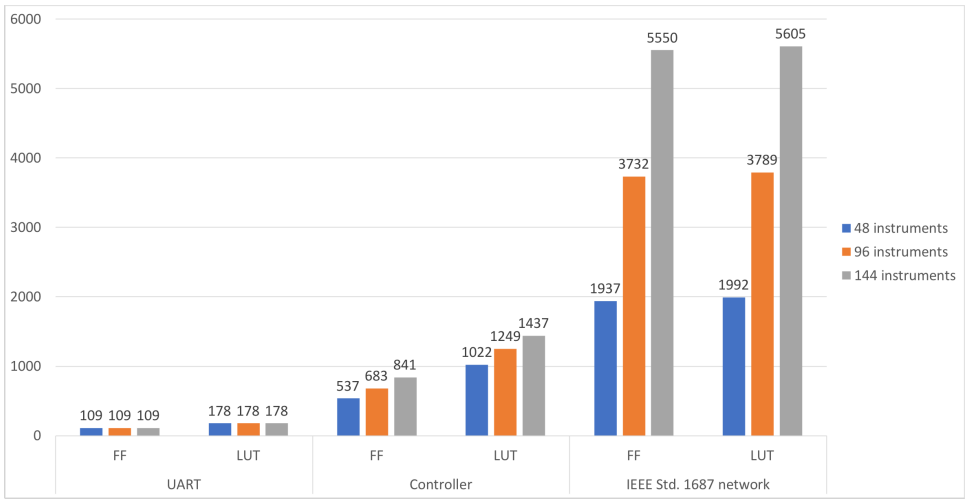
[15] E. Larsson, P. Murali, and G. Kumisbek, "IEEE Std. P1687.1: Translator and Protocol," in *International Test Conference*, 2019, pp. 1–10.



(a) Flat SIB-based



(b) General flat



(c) Perfect tree

Fig. 17. Area overhead on benchmarks: Flat SIB-based, General Flat and Perfect Tree

[16] 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.

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