IEEE Std. P1687.1: translator and protocol

Larsson, Erik; Murali, Prathamesh; Kumisbek, Gani

Published in:
International Test Conference

DOI:
10.1109/ITC44170.2019.9000135

2019

Document Version:
Publisher's PDF, also known as Version of record

IEEE Std. P1687.1: Translator and Protocol

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

***Abstract*—The IEEE Std. P1687.1 working group is currently exploring alternatives to IEEE Std. 1149.1 test access port (TAP) as the interface between the boundary of integrated circuits (ICs) and IEEE Std. 1687 networks. In this paper, we investigate the use of universal asynchronous receiver-transmitter (UART) to access IEEE Std. 1687 networks. We have developed a protocol to describe the information transported over UART and a hardware component to translate (retarget) information between UART and IEEE Std. 1687. The objective is to minimize the amount of information transported over UART and the area of the hardware component while maintaining the flexibility to access an arbitrary combination of instrument in the IEEE Std. 1687 network. We have developed software for the protocol translation, implemented the hardware component and IEEE Std. 1687 networks of different sizes in an field-programmable gate array (FPGA). For comparison, we developed a number of alternatives, all implemented on FPGA. The experimental results show that proposed scheme gives low overhead in terms of transported information (data) and low area of the hardware component.**

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

I. Introduction

The constant semiconductor technology development enables integrated circuits (ICs) with smaller, faster and more transistors. The development gives advantages, such as the possibility to implement more functionality to ever-increasing performance. There are, however, growing challenges to avoid malfunctioning. Smaller and faster transistors lead to tighter margins, both in device sizes and timing, which in combination with higher transistor count increase the need of testing, tuning, configuration, and so on. 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. The number of instruments in modern ICs increases and can be in order of 1000 [1] [2].

IEEE Std. 1687 [3] was developed to offer flexible and scalable access to embedded instruments. The flexibility to access arbitrary instruments is achieved by 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. Figure 1 shows the schematic equivalent of the network's ICL consisting of three instruments, three scan-chains and three SIBs. PDL describes how to operate on instruments. Figure 1 shows an example of PDL with one iApply group to write data to instrument *i1* and read data from instrument *i3*[1].

Not included in IEEE Std. 1687 is support to translate (retarget) PDL. For the example in Figure 1, the detailed PDL is

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

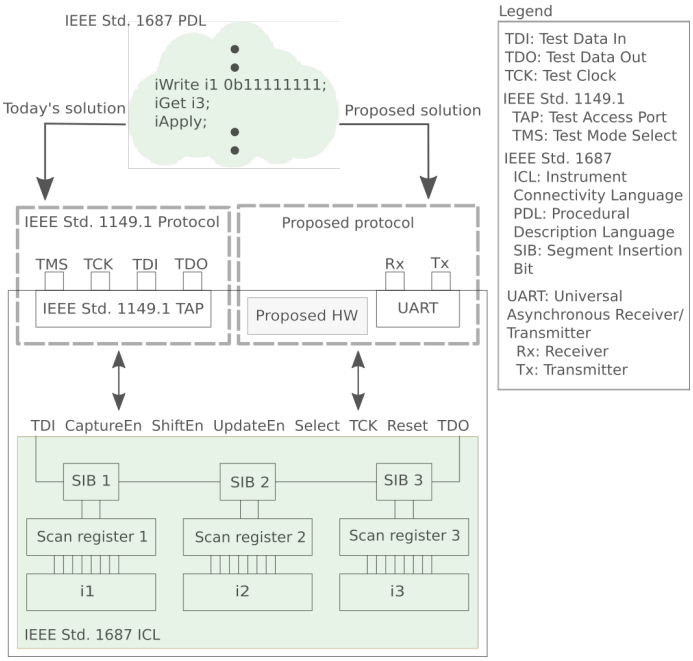


Fig. 1. Illustration of today's and proposed solution to an IEEE Std. 1687 network

first translated by a Electronic Design Automation (EDA) tool or an embedded controller, from a text format to shift data as specified by IEEE Std. 1149.1. Second, the IEEE Std. 1149.1 [4] test access port (TAP) handles the retargeting to and from the IEEE Std. 1687 network and at scan registers data is retargeted between serial and parallell format to access instruments.

The IEEE Std. 1149.1 TAP is the primary interface between IC's boundary (pins) and an IEEE Std. 1687 network, see Figure 1. However, the IEEE Std. P1687.1 [5] working group is currently exploring the use of other interfaces, like serial peripheral interface (SPI), inter-integrated circuit (I2C), universal serial bus (USB), and advanced microcontroller bus architecture (AMBA). We have in this paper explored the use of universal asynchronous receiver-transmitter (UART) to access IEEE Std. 1687 networks. We have developed a protocol specifying data transported over UART and a hardware component for the translation of data between UART and IEEE Std. 1687, see Figure 1. Our hypothesis is that with a suitable protocol specification and a small hardware component it is possible to keep the amount of data transported over UART low, while maintaining the flexibility to access arbitrary instruments in IEEE Std. 1687 networks.

The paper is organized as follows. Related work is briefly overviewed in Section II. An analysis of data overhead is in Section III and based on the analysis, we propose a hardware translator and a protocol, described in Section IV and Section

978-1-7281-4823-6/19/$31.00

V, respectively. The hardware translator and the protocol are for the working example in Figure 1 shown in detail in Section VI. We have implemented proposed scheme, as well as a number of alternatives on field-programmable gate arrays (FPGAs) to compare the overhead in terms of transported information over UART and the needed area. We compared the alternatives using three IEEE Std. 1687 networks of size 50, 100, and 150 instruments, also implemented on FPGA. The results show that proposed scheme gives low overhead in terms of transported information and low area of the hardware component, Section VII. The paper is concluded in Section VIII.

II. Related Work

Early in the process of developing IEEE Std. 1687, Rearick *et al.* described problems, activities and instruments [6]. The listed problems to be addressed by IEEE Std. 1687 included power management, clock control, chip configuration, memory test, scan test, logic built-in self-test (BIST), debug/diagnosis, phase locked-loop (PLL) control, reduced pin count test, and fault insertion. Activities benefiting from IEEE Std. 1687 include any on-chip circuit for test, debug, diagnosis, monitoring, characterization, configuration, or functional use that can be accessed by, configured from, or communicate with a TAP controller. As examples of instruments Rearick *et al.* listed scan chains, BIST engines, cyclic redundancy check (CRC) registers, packet counters, performance monitors, waveform analog analog-to-digital converters (ADCs), remapping registers, trace buffers, PLL controls, and power managers.

A key feature of IEEE Std. 1687 is dynamic configuration of the active scan-path. However, dynamically re-configurable scan-paths leads to a number of challenges. Zadegan *et al.* analyzed the challenge of computing the access time [7] and the need of design methods [8]. Dynamic configuration may require a number of configuration steps to change from accessing one set of instruments to accessing another set of instruments. Baranowski *et al.* developed a method to limit the search space in terms of number of configuration steps such that the limit is high enough to include optimal solution but as low as possible to speed-up the search [9]. Cantoro *et al.* proposed techniques to test the IEEE Std. 1687 network [10]. Jutman *et al.* proposed the first work with IEEE Std. 1687 for fault management [11]. To speed-up fault detection, the network was complemented with an additional infrastructure. A similar infrastructure as proposed by Jutman *et al.* [11] was used by Petersen *et al.* [12]. Zadegan *et al.* proposed self-configuring SIBs to speed-up fault localization [13].

Crouch *et al.* describe the need of making it possible to access IEEE Std. 1687 through alternative ports, as SPI, I2C, USB, AMBA, and others [14]. von Staudt and Spyronasios describe an industrial case where I2C is used to interface IEEE Std. 1687 [15].

III. Analysis of data overhead

In this section we analyze and classify the data (information) transported to and from an IEEE Std. 1687 network. An IEEE Std. 1687 network is operated using three operations, capture (C), shift (S) and update (U), known as CSU-cycles. These operations perform as follows:

- C: instrument data is *captured* in scan registers
- S: data is serially *shifted* through the active scan-path
- U: instruments are *updated* with data from scan registers

The C and U operations move data between scan-chains and instruments. Each of these operations takes one clock cycle. The S operation moves data in and out of the IEEE Std. 1687 network. The number of clock cycles needed for an S operation depends on the length of the active scan-path.

We analyze the data moved in and out of an IEEE Std. 1687 network during an S operation. At each clock cycle, one bit of data is shifted-in and one bit of data is shifted-out. For the analysis we make use of the *iApply* group in Figure 1 where the data 0b11111111 should be written to instrument *i1* and data from instrument *i3* should be read:

```
iWrite i1 0b1111111;
iGet i3;
iApply;
```

The shortest active scan-path for the IEEE Std. 1687 network in Figure 1 includes instruments *i1* and *i3* while instrument *i2* is excluded. The shortest active scan-path is achieved by activating SIB1 and SIB3 while SIB2 is not activated. The length of this scan-path becomes 19, given by the length of instruments *i1* and *i3*, each 8 bits long, and by the fact that there are 3 SIBs on the scan-path, each of length 1. Hence, the shift sequence contains 19 (8+8+3) bits.

A shift sequence consists of the shift-in sequence and the shift-out sequence. For the shift-in sequence, the 19 bits are as follows; 8 bits are instrument data, 0b1111111, for instrument *i1*, 8 bits of data must be shifted in because of the read operation of instrument *i3* (the instrument data must be pushed out), and 3 bits are needed for the SIBs. The shift-out sequence also contains 19 bits, where 8 bits come from instrument *i1* when its current value is shifted out (pushed out), 8 bits come from reading the content of instrument *i3* and 3 bits come from the SIBs. Hence, the total number of bits for the shift sequence is 38 (19+19). Out of these 38 bits, we observe that the 8 bits written to instrument *i1* and the 8 bits from reading instrument *i3* are *useful* data. All other bits are overhead bits. The overhead bits are the 6 bits for the SIBs, which we call *SIB overhead*. The rest of the overhead bits are due to the need of shifting out data from instrument *i1* to make it possible to shift in useful data (write) and the need of shifting in data to instrument *i3* to get out the useful data (read). We denote this as *shift overhead*.

To generalise SIB and shift overhead, assume the IEEE Std. 1687 network in Figure 1 has N SIBs where $SIB(i)$ controls if instrument i is included in the active scan-path or not, such that $SIB(i) = 1$ if instrument i is included in the active scan-path, and 0 otherwise. The length of an instrument i is given by $l(i)$. As all SIBs are always on the active scan path, we find that:

$$SIB\,overhead \;=\; N \times 2 \tag{1}$$

The shift overhead depends on the number of instruments included in the active scan path and the type of operation. All instruments on the active scan path cause overhead, either due to read or due to write. The shift overhead becomes:

$$shift\,overhead \;=\; \sum_{i=1}^{N} SIB(i) \times l(i) \tag{2}$$

Given the analysis of the overhead (Equations 1 and 2), we propose a hardware translator and a data protocol aiming at minimizing the SIB and shift overhead.

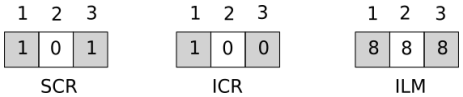


Fig. 2. SCR, ICR, and ILM content for the PDL and ICL in Figure 1 to include instrument *i1* and *i3* in the active scan-path (SCR), perform a write to instrument *i1* and read to instrument *i3* (ICR) and shift 8 bits (the length of instruments *i1* and *i3* (ILM))

IV. Hardware Translator

The proposed hardware component translates data between UART and IEEE Std. 1687. Prior to detailing the proposed hardware component, we briefly review a solution with IEEE Std. 1149.1 TAP. The IEEE Std. 1149.1 TAP is based on a finite state machine (FSM) to translate data between the four IEEE Std. 1149.1 pins, that is test data in (TDI), test data out (TDO), test mode select (TMS), and test clock (TCK), and the eight IEEE Std. 1687 signals, that is TDI for scan-in (SI), TDO for scan-out (SO), CaptureEn, Select, Reset, ShiftEn, UpdateEn, and TCK, illustrated in Figure 1.

We assume that the UART circuitry ensures that data arrive to the IC byte-per-byte, which is stored in an input buffer, and that data to be sent from the IC over UART is stored in an output buffer of size one byte. These buffers are implemented as shift-registers controlled by our FSM. We assume that the sender of UART data, which is the tester, embedded controller, etc., sends data so that our FSM consumes data before new data arrive. If the FSM has consumed all data and need more data, the FSM stops until new UART data arrive. If the FSM has filled the output buffer and needs to send more data, the FSM stops until the output buffer is empty.

The FSM in the proposed hardware component is complemented with a SIB control register (SCR), an instrument control register (ICR), and an instrument length memory (ILM). These components are detailed below.

A. SIB Control Register

The SIB control register (SCR) controls which instruments to include in the active scan path. There is one bit per SIB to determine if a particular SIB should be active (open) or not (closed). In general, there are N SIBs, hence there are N bits in the SCR, where a 1 at position i indicates that SIB i should be included in the active scan-chain. For the example in Figure 1 when instruments *i1* and *i3* are in the active scan path, Figure 2 shows the content of the SCR. The SCR for each instrument (SIB i) is programmed with the control command, detailed in Section V. The SCR is cleared after the completion of an iApply group.

B. Instrument Control Register

The basic and fundamental operations on instruments are read and write. The instrument control register (ICR) determines the type of operation to perform on a given instrument. In general, there are N instruments, hence there are N bits in the ICR. For a given instrument i where the operation is *read* the bit is 0 while if the operation is *write*, the bit is 1. For the example in Figure 1 where instrument *i1* is performing a *write* operation and instrument *i3* performs a *read*, Figure 2 shows in gray the content of the ICR. The ICR for each instrument is programmed with the control command, detailed in Section V. The ICR is cleared after the completion of an iApply group.

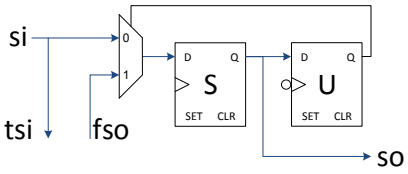


Fig. 3. Illustration of a SIB

C. Instrument Length Memory

The instrument length memory (ILM) keeps track of the length in bits of each instrument. In general, there are N instruments, which means there are N entries, one for each instrument, indicating the length in bits of a given instrument. For the example in Figure 1 the length of instrument *i1* and *i3* are accessed, Figure 2 shows in gray the length of these instruments. The content of the ILM is fixed at design time.

D. Finite State Machine

The FSM operates on one *iApply* group at a time, for example the one in Figure 1. Assume that SCR and ICR are programmed using control commands, detailed in Section V, the FSM first defines the active scan-path by traversing the SCR and shifting content to the SIBs of the IEEE Std. 1687 network. The data in the shift-out sequence is of no interest, hence discarded.

Once the active scan-path is set, the FSM creates the shift-in sequence and handles the shift out sequence. We will for illustration purposes describe them separately. First we explain the shift-in sequence, Algorithm 1 and then the shift-out sequence, Algorithm 2. One important aspect is the order of information to SIBs and instruments. In this paper, we assume that a SIB is implemented as in Figure 3. Figure 3 shows that the content of the SIB is shifted out (SO) before the instrument data is shifted out. This means that for the shift-in sequence, the SIB data for SIB i comes before eventual data to instrument i. As SIBs are implemented prior to the hardware translator, the hardware translator can be adjusting according to used SIB implementation.

Common for Algorithms 1 and 2 are the following functions:

- $SCR(i)$ returns the SCR content for instrument i. If $SCR(i) = 1$, SIB i should be made active such that instrument i is included in the active scan path, while if $SCR(i) = 0$, SIB i is not active, meaning that instrument i should not be included in the active scan path.
- $ICR(i)$ returns the ICR content for instrument i. If $ICR(i) = 0$, instrument i is in *read* mode and if $ICR(i) = 1$, instrument i is in *write* mode.
- $ILM(i)$ gives the length in bits of instrument i.

Below we detail Algorithms 1 and 2.

1) Creating shift-in sequence: The input to Algorithm 1 is UART data and the output is the shift-in sequence to the IEEE Std. 1687 network, including control signals. For illustration purpose, the sequence is captured in $SHIFTIN$, which is a bit string of K bits, corresponding to the length of the active scan part.

Algorithm 1 begins by line 1 where variable k, which keeps track on current bit in the active scan path, is initiated to 0. The FOR loop at line 2 iterates over the N SIBs. At line 3, k, current bit, is incremented, and at line 4, the bit to be shifted in is the value of SIB i, which is received from SCR by $SCR(i)$. Hence, the content of $SCR(i)$ is added to the shift in $SCANIN$ sequence. The following bits to shift in depends on the value of $SCR(i)$, line 5. If $SCR(i) = 1$, it means SIB i is active and hence instrument i should be included in the active scan path. The data that is shifted in to an instrument depends on the type of operation: *read* or *write*. At line 6, $ICR(i)$ determines if the data for instrument i is *read* or *write*. If $ICR(i) = 0$, the operation for instrument i is a *read* operation, which means that dummy data is created by the FSM as the interesting information is the output from instrument i. On the other hand, if $ICR(i) = 1$, the operation for instrument i is a *write* operation. In this case data from the UART register is shifted in. The FOR loops at lines 7 and 11 are used to ensure the correct amount of shifting, which is determined by the length of instrument i. The length of instrument i is given by $ILM(i)$.

Algorithm 1: Creating $shift - in$ sequence

Input: Data sent over UART; $SCR(i)$=1 if SIB i is active, 0 otherwise; $ICR(i)$=0 if instrument at SIB i is in read mode and 1 if instrument at SIB i is in write mode; $ILM(i)$ gives the length in bits of instrument i;

Output: The shift-in sequence to a IEEE Std. 1687 network, for illustration captured in $SCANIN$

```
1  k = 0
2  for i=N downto 1 do
       /* Iterate over the N SIBs                    */
3      k = k + 1
4      SCANIN(k) = SCR(i)
       /* SIB content given by SCR(i)                */
5      if SCR(i) = 1 then
           /* SIB i is active, which means
              instrument i should be
              included in active scan-path
              */
6          if ICR(i) = 0 then
               /* Instr. i in read mode              */
7              for j = 1 to ILM(i) do
                   /* ILM(i) dummy bits created
                      by FSM and shifted in          */
8                  k = k + 1
9                  SCANIN(k) = 1
10         else
               /* Instr. i in write mode             */
11             for j = 1 to ILM(i) do
                   /* ILM(i) bits from UART
                      shifted in                     */
12                 k = k + 1
13                 SCANIN(k) = one bit from UART
```

2) Handling shift-out sequence: Algorithm 2 takes as input the shift-out sequence, which appears bit-by-bit, from the active scan path of the IEEE Std. 1687 network. The shift-out sequence is for illustration purposes captured in $SHIFTOUT$, which is a bit string with a length of K bits, corresponding to the length of the active scan path. The output of Algorithm 2 is the data that is to be transported from the IC over UART.

Algorithm 2 begins by line 1 where variable k, which keeps track on current bit in the active scan path, is initiated to 0. The FOR loop at line 2 iterates over the N SIBs. At line 3, k, current bit, is incremented and the bit that is shifted out is the value of SIB i. If the shifted out bit ($SCANOUT(k)$) is equal to 1, the SIB is active and hence instrument i is included in the scan path, line 4. By checking the content of $ICR(i)$ the FSM knows if the command is *read* or *write*. If $ICR(i) = 0$, instrument i is in read mode. Hence, the content of instrument i should be sent over UART. If $ICR(i) = 1$, instrument i is in write mode. Hence, the content of instrument i can be discarded as the output information is of no value. At lines 6 and 10, the FOR loops are used to ensure correct amount of shifting, which is determined by the length of instrument i. The length of instrument i is given by $ILM(i)$.

Algorithm 2: Handling $shift - out$ sequence

Input: Shift out data from a IEEE Std. 1687 network, for illustration captured in $SCANOUT$; $ICR(i)$ is 0 if instrument at SIB i is in read mode and 1 if instrument at SIB i is in write mode; $ILM(i)$ is the length of instrument i connected to SIB i;

Output: The data to be transported over UART, which for illustration is captured in $UART$

```
1  k = 0
2  for i=N downto 1 do
       /* Iterate over the N SIBs                    */
3      k = k + 1
4      if SCANOUT(k) = 1 then
           /* SIB=1 means an active
              instrument                             */
5          if ICR(i) = 0 then
               /* Instr. i in read mode              */
6              for j = 1 to ILM(i) do
                   /* ILM(i) bits sent on UART
                      */
7                  k = k + 1
8                  UART = SCANOUT(k)
9          else
               /* Instr. i in write mode             */
10             for j = 1 to ILM(i) do
                   /* Discard ILM(i) bits            */
11                 k = k + 1
```

V. Data Protocol

The protocol specifies the format of data (information) transported over UART. The protocol consists of two parts; data to the IC and data from the IC. For data to the IC, each iApply group is translated into one or more control commands followed by one or more data commands. The data sent from the IC is in a raw format, byte by byte, including only useful information as the FSM removes from the shift-out sequence the SIB information and all overhead data. The details in the

raw format is extracted by software using information about the ICL and currently used iApply group.

Below we detail the commands for control and data. The control command defines the active scan-path and the type of operation for each instrument, *read* or *write*. The active scan-path is set by loading *SCR* and the type of operation is set by loading *ICR*. A control command consists of two mandatory bytes where the 16 bits are defined as follows:

- Bit 15 (bit 7 in the first byte) is used to distinguish if the command is a control command or a data command, 0 for control and 1 for data.
- Bit 14 (bit 6 in the first byte) determines if the instrument associated with the SIB will perform a read or a write operation, 0 for read and 1 for write.
- The remaining 14 bits are used to specify the address (number) to the SIB to be set active. The control command enables an IEEE Std. 1687 network with 2^{14} SIBs.

Below we show the control command for: `iWrite i1 0b11111111;`, in Figure 1:

$b_{15} = 0$ - the command is a control command

$b_{14} = 1$ - the instrument should perform a write

$b_{13} - b_1$ are 0 and $b_0 = 1$ - address to SIB 1

A data command consists of two mandatory bytes and a number of bytes with data. The two mandatory bytes are specified as follows. Bit 15 (bit 7 in the first byte) distinguish if the command is a control command or a data command. The next 15 bits determines the number of bytes with data that follows. The 15 bits enables that up to 2^{15} bytes of data can be transported with one data command. Below we show the data command for `iWrite i1 0b11111111;`, in Figure 1:

$b_{15} = 1$ - the command is a data command

$b_{14} - b_1$ are 0

$b_0 = 1$ - there is one byte of data

The byte with data (”11111111”) is:

1	1	1	1	1	1	1	1

VI. Example

In this section we illustrate the translator (described in Section IV) and the protocol (described in Section V) using the PDL and ICL of the IEEE Std. 1687 network in Figure 1.

The iApply group from the PDL description in Figure 4 is retargeted into two control commands and one data command, in total 7 bytes of information. The two control commands, byte 1 to 4 in Figure 4, set required values in SCR and ICR. The first control command, byte 1 and 2 in Figure 4, makes SIB 1 active and sets instrument *i1* in write mode. 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 $b_6 = 1$ in the first byte indicates that a *write* operation should be performed. The following 14 bits, which holds the value 1, indicates that SIB 1 should be active so that instrument *i1* is included in the active scan-path. The next two bytes, byte 3 and 4 in Figure 4, are also forming a control command, indicated by bit $b_7 = 0$ in

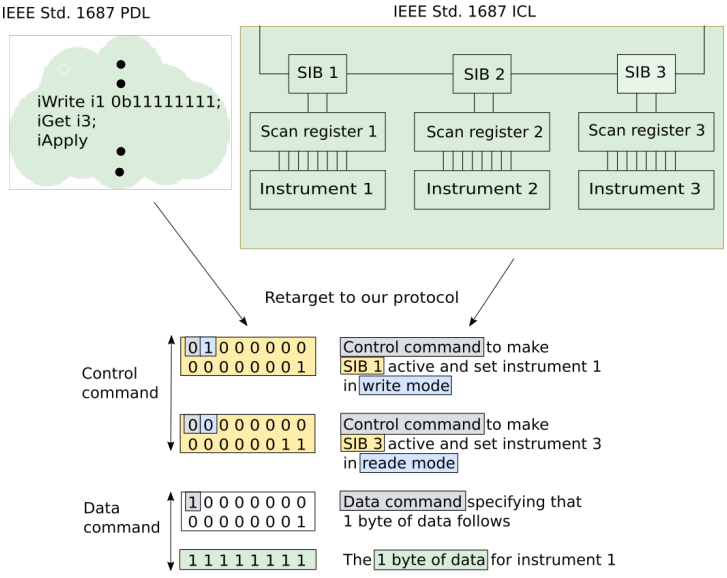
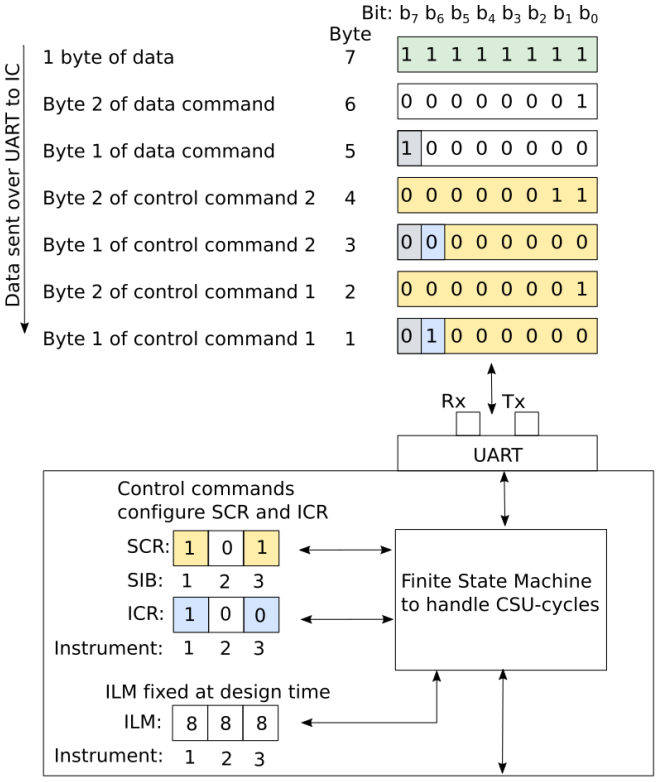


Fig. 4. Retargeting the iApply group to proposed format for UART

byte 3. This control command has $b_6 = 0$, which informs that a *read* operation should be performed. The following 14 bits, which holds the value 3 (0b11) informs that SIB 3 should be active so that instrument *i3* is included in the active scan-path. The following 3 bytes, byte 5, 6, and 7 in Figure 4, forms a data command as $b_7 = 1$ in byte 5. The remaining 15 bits in byte 5 and 6 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. The data in byte 7 is the data that should be written to instrument *i1*.

Figure 5 illustrates the translation of UART data into shift-in data. When a control command arrives, the hardware translator automatically resets SCR and ICR. Control commands set SCR and ICR as needed for the iApply group. In this example, the SCRs corresponding to SIB 1 and SIB 3 are set to 1, indicating that SIB 1 and SIB 3 should be active. The first control command makes the hardware translator set the bit in ICR corresponding to instrument *i1* to 1 to indicate that a *write* operation should be performed. The second control command sets the bit in ICR corresponding to instrument *i3* to 0, to indicate that a *read* operation should be performed.

When the data command arrives, the hardware translator begins operating the IEEE Std. 1687 network. First, the active scan path is set by traversing SCR at the highest value, in this example 3 ($SCR(3)$), and includes that bit in the shift-in sequence. Next, $SCR(2)$ is shifted in and finally $SCR(1)$. The bits shifted out are ignored (discarded). Second, the hardware translator creates the shift-in sequence and handles the shift-out sequence concurrently for the active scan path. We first describe the shift-in sequence. The hardware translator begins by checking the SCR at the highest value, in this example 3 ($SCR(3)$), and includes that bit in the shift-in sequence. If $SCR(3) = 1$ the hardware translator checks the value of $ICR(3)$ to learn what type of operation to perform. In this example 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 shift-in data is created by the hardware translator. The hardware translator checks $ILM(3)$ to know how many cycles to shift. After required number of shifts for instrument *i3*, the next bit to be shifted in is the value of



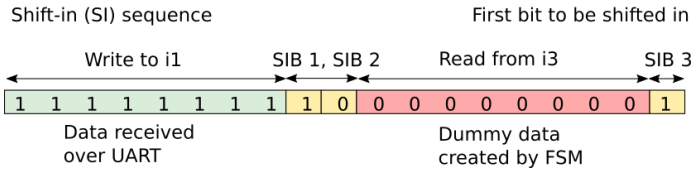


Fig. 5. The hardware translator forming the shift-in sequence from the retargeted data in Figure 4

$SCR(2)$ to SIB 2. As $SCR(2) = 0$, indicating that instrument *i2* is not in the active scan-path, the hardware translator shift-in a 0 and directly start to focus on next bit in SCR, which is $SCR(1)$. $SCR(1) = 1$, which means instrument *i1* should be included in the active scan-path. The hardware translator checks $ICR(1)$ to learn what type of operation to perform. As $ICR(1) = 1$ a *write* operation should be performed. The hardware translator gets the length of instrument *i1* from $ILM(1)$ and takes data from the UART buffer and adds it to the shift-in sequence. Figure 6 shows the created shift-in sequence and how its information will set the SIBs and the instruments.

The first bit that is shifted out corresponds to the content of SIB 3, see Figure 7. This information bit is of no interest so the hardware translator discards this bit. As the content of SIB 3 = 1 the hardware translator understands that instrument *i3* is included in the active scan path. The hardware translator knows the length of instrument *i3* by checking $ILM(3)$. The hardware translator checks $ICR(3)$ and finds that a *read* operation has been performed on instrument *i3*. Hence, this is useful information that should be sent back over UART. When the translator completed the shift-out bits corresponding to the length of instrument *i3*, the next shifted-out bit is the value of SIB 2, this value is 0, which is discarded and informs the translator that next shifted out value corresponds to the value of SIB 1. SIB 1 is 1, so instrument *i1* is active and by checking

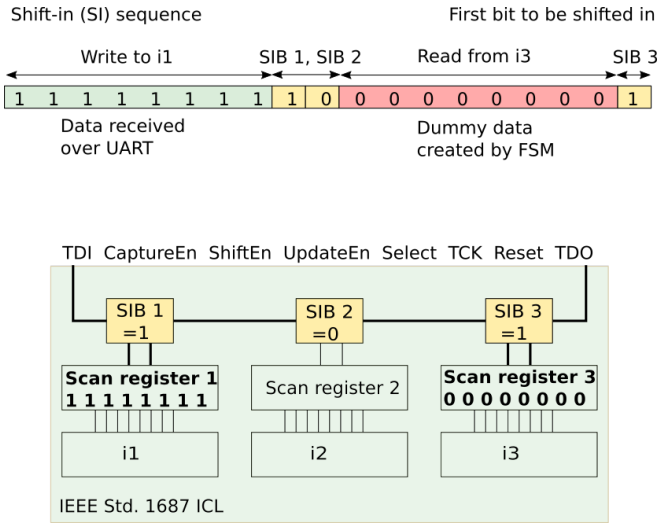


Fig. 6. The shift-in sequence and the created active scan-path in the IEEE Std. 1687 network

$ICR(1)$ the translator knows that instrument *i1* is in write mode, which means that $ILM(1)$ of shifted out bits can be discarded. The data that the translator returns over UART is the data that is read from instrument *i1*, the only useful information. At the receiver side, the one that initiated the iApply group, knows that the received data corresponds to reading from instrument *i1*.

VII. Experimental Results

The objective of the experiment is to evaluate the overhead of proposed scheme, that is the hardware translator (described in Section IV) and the protocol (described in Section V). The overhead is computed as described in Section VII-A and the proposed scheme is compared against bit banging, proposed scheme without on-chip handling of dummy data, proposed scheme without ILM, and a naive approach, all outlined in Section VII-B. The benchmarks in the experiments are detailed in Section VII-C, the results are presented in Section VII-D and the results are discussed in Section VII-E.

A. Overhead

Operating an IEEE Std. 1687 network results in overhead, see Section III. We illustrate the overhead using the iApply group that writes data to instrument *i1* and reads data from instrument *i3*, see Figure 4. The shift-in and the shift-out sequences (Figures 4 and 5) result in:

- SIB overhead are bits to set SIBs
- Useful bits in the shift-in sequence are bits for writing data to instrument *i1*
- Useful bits in the shift-out sequence are bits from reading data from instrument *i3*
- Dummy bits in the shift-in sequence are bits needed to push out data from instrument *i3*
- Dummy bits in the shift-out sequence are bits needed to push out data from instrument *i1*

The proposed data and control commands aim at reducing the overhead to operate IEEE Std. 1687 networks, but do themselves generate overhead. We illustrate this overhead using the iApply group in Figure 4. The two control commands, byte

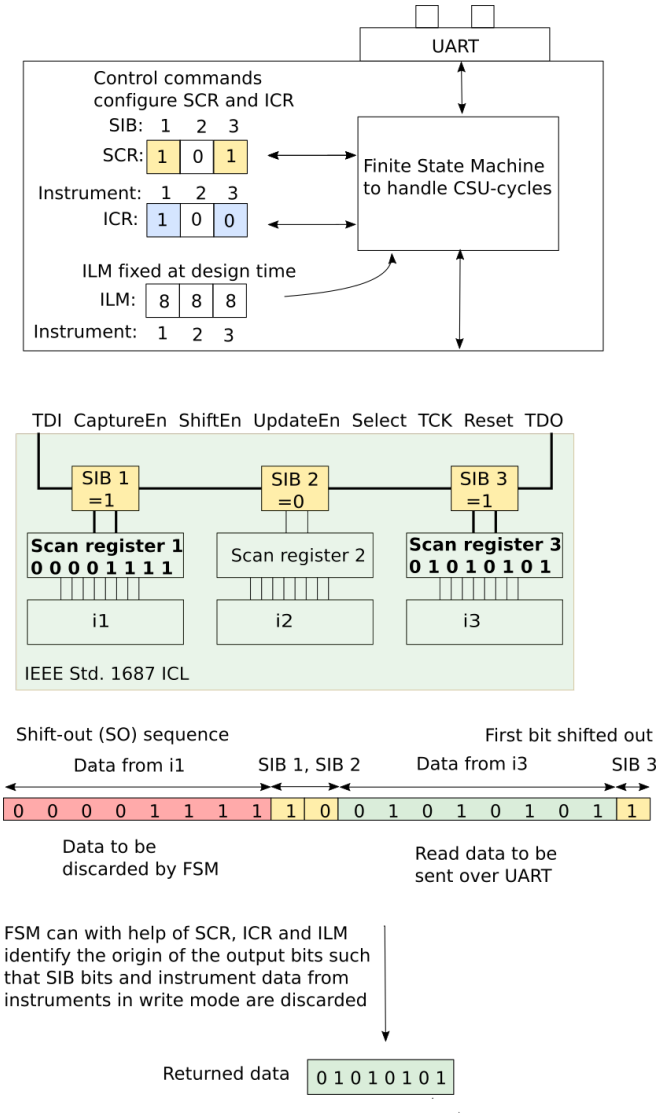


Fig. 7. The extraction of useful data from a shift-out sequence from an IEEE Std. 1687 network

1, 2, 3 and 4, generates *control overhead*. The data command generates three bytes where the two mandatory bytes, byte 5 and 6, are *data overhead*. The third byte in the data command, byte 7, contains *useful data*, to be written to instrument *i1*. Note, these commands assume a hardware translator capable of handling, creating and discarding, *dummy overhead*.

To estimate area overhead, we report the number of Xilinx configurable logic blocks (CLBs), which constitutes the basic FPGA cell.

B. Alternatives to proposed scheme

The alternatives we compare proposed scheme against are:

1) Bit banging: For bit banging, the received byte of data is directly applied to the seven inputs at the IEEE Std. 1687 interface, TDI(SI), CaptureEn, ShiftEn, UpdateEn, Select, TCK and Reset. The output from the IEEE Std. 1687 network, TDO(SO), is placed in one byte and sent over UART. The number of clock cycles for a CSU-cycle is one for C, one for U, and the length of the active scan-path for S. We report one byte of data is received and one byte of data is sent over UART per

TABLE I. AREA OVERHEAD IN CONFIGURABLE LOGIC BLOCKS (CBLS)

Approaches	Instruments		
	50	100	150
IEEE Std. 1687 network	369	742	1118
Naive	45	45	45
No ILM	96	118	140
No Dummy	103	129	147
Proposed	100	123	144

clock cycle. In general, as TCK is sent over UART to control the IEEE Std. 1687 network, there is a need to send two bytes of data per clock cycle.

2) Without dummy data handling: In the proposed scheme, the FSM creates shift-in (push out) data for instruments performing a read operation and removes shift-out data for instruments performing write operations as well as SIB content. We have made an implementation where we excluded this feature.

3) Without ILM: In the proposed scheme, the ILM keeps track on the length of each instrument. We have made an implementation where we excluded this feature. To perform experiments where the ILM is not present, the protocol and the FSM are modified. The control command is extended from two bytes to three bytes where the additional byte is used to include the length of the instrument. The FSM is modified such that instead of checking the ILM to get the length of an instrument, the FSM receives the input on instruments length from a command via UART.

4) Naive: In the naive case, all features in the proposed scheme, that is dummy data handling, ILM, SCR, and ICR, are excluded. We have written software for the retargeting so that the shift-in sequence is created and formed into packets of size one byte, which is sent over UART. Instead of having the hardware translator creating the shift-in sequence as in Figure 5, the software creates this sequence, see Figure 8. First information of size is sent (number of packets with scan-data) and then packets with scan-data. For illustration Figure 8 shows the formation of the scan data sequence into three packets (one byte each) of data, with additional padding bits in the last byte to match the size of packets.

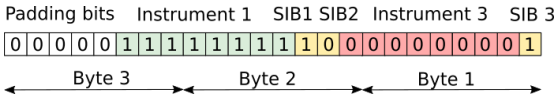


Fig. 8. Forming shift-in sequence into three packets of bytes

C. Benchmarks

We have created ICL and PDL to perform experiments. We created ICL for three IEEE Std. 1687 designs with 50, 100, and 150 instruments, respectively. The instruments are connected in a flat manner with one SIB per instrument, as illustrated in Figure 1. The instruments are of length 8, 16, and 32 bits. The length is set as follows: Instrument one has length 8, instrument two has length of 16, instrument three has length of 32, and instrument four has length of 8, and so on. The instruments perform bit-wise inversion, the value written to the instrument is inverted. The motivation for this simple instrument is that it eases validation of the implementation.

We have made use of four trivial PDL descriptions with one iApply group each, `iGet`[2] of instrument 1, `iWrite` of

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

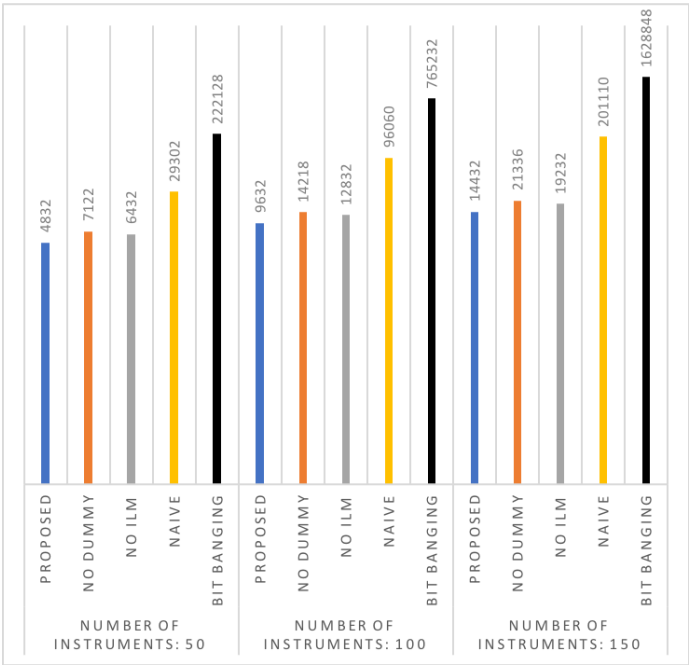


Fig. 9. The total overhead (logarithmic scale) for the approaches in Table II applying the PDL according to the BASTION scheme

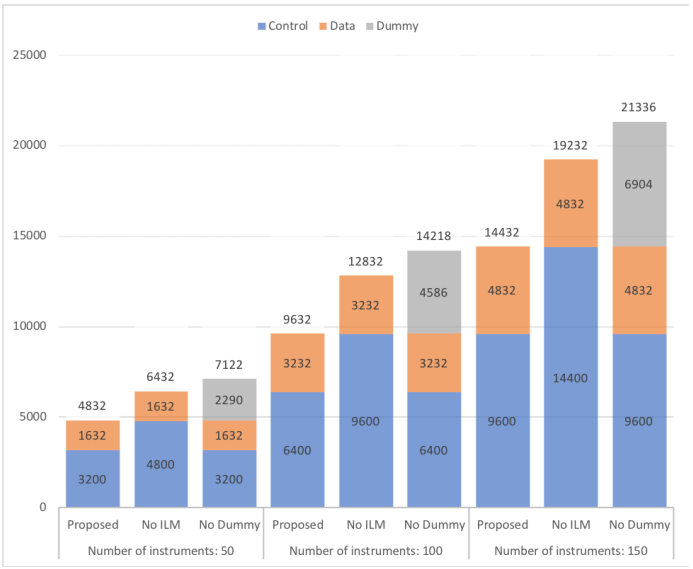


Fig. 10. Comparing amount of control, data and dummy overhead of the methods; Proposed, No ILM, and No Discard

instrument 1, `iGet` of all instruments, `iWrite` of all instruments. We have also made use of the PDL scheme used in the BASTION benchmarks [16]. The PDL scheme in BASTION 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 the benchmark with 50 instruments the BASTION PDL scheme results in 102 iApply groups.

D. Results

We have on an Nexys 4 DDR with an Artix-7 (XC7A100T-1CSG324C) FPGA implemented the proposed scheme (described in Section IV), the proposed scheme without on-chip

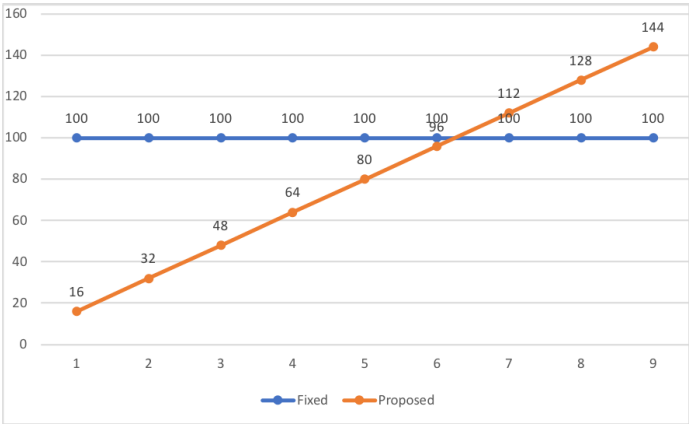


Fig. 11. Theoretical comparison of proposed control command and a shift-in scheme

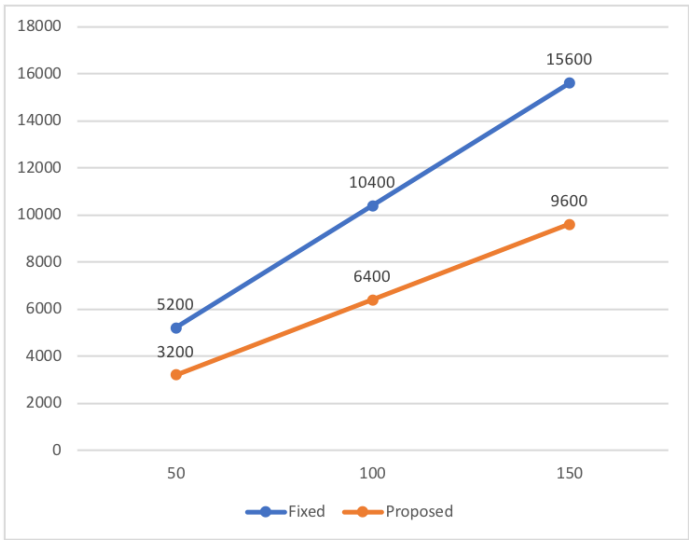


Fig. 12. Theoretical comparison of proposed control command and a shift-in scheme on BASTION PDL

handling of dummy data (Section VII-B2), proposed scheme without ILM (Section VII-B3), the naive approach (Section VII-B4), and the IEEE Std. 1687 networks. For the bit banging (Section VII-B1) we computed the overhead without making an implementation on an FPGA. We have also implemented software for the creation of data transported over UART for all cases except for bit banging.

The experimental results on UART overhead are collected Table II. Table II is organized as follows. The benchmarks of size 50, 100, and 150 instruments are in column one, the applied PDL is listed in column two and the amount of useful data bits is in column three. In column four, the types of overhead is listed, including the total overhead and the amount of useful data in relation to total overhead. The approaches bit banging, naive, no ILM, no dummy handling and proposed are listed in columns five to nine.

The experimental results on area overhead are collected in Table I, which is organized as follows. The first column lists the approaches, the IEEE Std. 1687 network itself and the four approaches, naive, no ILM, no dummy, and proposed. Column two to four list the number of CBLs for each of the IEEE Std. 1687 networks.

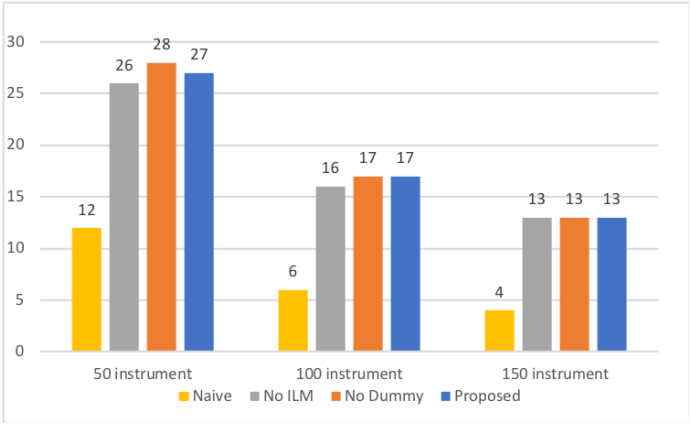


Fig. 13. Estimated area of implemented approaches in relation (%) to the IEEE Std. 1687 networks

E. Discussion

Based on the results, we make the following observations:

- Bit banging and naive schemes generate highest overhead and the data overhead increases dramatically as design complexity increase (number of instruments), see Figure 9 that shows the total overhead, note logarithmic scale, for the approaches using PDL from the BASTION scheme.
- Proposed, no ILM, and no dummy schemes scale for control, data, and dummy overhead nearly linear with the number of instruments, see Figure 10.
- ILM has a significant impact on reducing overhead, compare no ILM and proposed schemes in Figure 10.
- Proposed scheme where FSM handles and creates dummy data has a significant impact on reducing overhead, compare no discard and proposed schemes in Figure 10.
- Control overhead is relatively high in all cases, see Figure 10. An alternative to proposed control command is to send control data, for example to SCR for SIB content, as shift data that is shifted-in to SCR for each iApply group. We compare for a design with 100 SIBs the amount of data required using a shift-in sequence of 100 bits per iApply group (fixed scheme) against proposed scheme when accessing 1 to 9 instruments, see Figure 11. The results indicate that when 6 or more instruments are accessed, the shift-in scheme is more suitable. On the other hand, we made a theoretical computation of the amount of SCR data needed for the three designs using PDL as in the BASTION scheme. The result indicates that proposed control command generates less control overhead compared to a shift-in scheme (fixed scheme), see Figure 12.
- The estimated area for the approaches relative to the IEEE Std. 1687 networks decreases as the number of instruments increase (%), Figure 13.
- We note that the naive scheme gives the lowest area and that all other approaches performs basically the same. The fact that the proposed, no ILM and no dummy schemes perform the same when their functionality differs can be that the proposed functions (ILM and dummy handling) do not cost much in area or that these designs are based on proposed scheme where the functions have been removed, hence no optimization. We also note that the relative impact of the hardware for the approaches decreases as the IEEE Std. 1687 network increases in number of instruments.

VIII. Conclusions

We developed hardware and protocol where we, instead of using IEEE Std. 1149.1, made use of UART to access IEEE Std. 1687 networks. The overall aim is to minimize the overhead in terms of data transported over UART and the area of the hardware component. We implemented on FPGAs proposed scheme and a number of alternatives as well as supporting software for retargeting. Experiments with benchmarks with 50, 100, and 150 instruments using the BASTION PDL scheme show the benefit of SIB control register (SCR), instrument control register (ICR) to create and handle dummy data, and instrument length memory (ILM) to know the length of instrument during shifting.

Future work may include the handling of general IEEE Std. 1687 networks, analysis of the need of other operations on instruments than read and write, built-in synchronisation of communication, handling of misalignment in data transfer, and other protocols like SPI, I2C, and so on.

References

[1] "Embedded Instrumentation: Its Importance and Adoption in the Test and Measurement Marketplace, Frost and Sullivan, Whitepaper, 2010, 20 p."

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

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

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

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[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] H. M. von Staudt and A. Spyronasios, "Using ijtag digital islands in analogue circuits to perform trim and test functions," in *IEEE International Mixed-Signals Testing Workshop (IMSTW)*, June 2015, pp. 1–5.

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

TABLE II. DATA TRANSPORTED OVER UART

Benchmark	PDL	Useful data	Type of overhead	Bit banging	Naive	No ILM	No dummy handling	Proposed
50	iGet 1	8	Control overhead		120	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		111	0	111	0
			Total overhead	1776	255	40	143	32
			Useful data (%)	0.4	3.0	16.7	5.3	20
	iWrite 1	8	Control overhead		112	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		118	0	118	0
			Total overhead	1776	254	40	150	32
			Useful data (%)	0.4	3.1	16.7	5.1	20
	iGet all	920	Control overhead		1032	1200	800	800
			Data overhead		24	16	16	16
			Dummy overhead		226	0	170	0
			Total overhead	16368	1282	1216	986	816
			Useful data (%)	5.3	41.8	43.1	48.3	53.0
	iWrite All	920	Control overhead		112	1200	16	800
			Data overhead		24	16	800	16
			Dummy overhead		1031	0	974	0
			Total overhead	16368	1167	1216	1790	816
			Useful data (%)	5.3	44.1	43.1	33.9	53.0
	BASTION	3680	Control overhead		13152	4800	3200	3200
			Data overhead		2448	1632	1632	1632
			Dummy overhead		13702	0	2290	0
			Total overhead	222128	29302	6432	7122	4832
			Useful data (%)	1.6	11.2	36.4	34.1	43.2
100	iGet 1	8	Control overhead		216	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		205	0	205	0
			Total overhead	3376	445	40	237	32
			Useful data (%)	0.2	1.8	16.7	3.3	20
	iWrite 1	8	Control overhead		208	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		212	0	212	0
			Total overhead	3376	444	40	244	32
			Useful data (%)	0.2	1.8	16.7	3.2	20
	iGet all	1856	Control overhead		2064	2400	1600	1600
			Data overhead		24	16	16	16
			Dummy overhead		439	0	335	0
			Total overhead	32944	2527	2416	1951	1616
			Useful data (%)	5.3	42.3	43.4	48.8	53.5
	iWrite All	1856	Control overhead		208	2400	1600	16
			Data overhead		24	16	16	1600
			Dummy overhead		2063	0	1958	0
			Total overhead	32944	2295	2416	3574	1616
			Useful data (%)	5.3	44.7	43.4	34.2	53.5
	BASTION	7424	Control overhead		45520	9600	6400	6400
			Data overhead		4848	3232	3232	3232
			Dummy overhead		45692	0	4586	0
			Total overhead	765232	96060	12832	14218	9632
			Useful data (%)	1.0	7.2	36.7	34.3	43.5
150	iGet 1	8	Control overhead		312	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		349	0	299	0
			Total overhead	4976	685	40	331	32
			Useful data (%)	0.2	1.2	16.7	2.4	20
	iWrite 1	8	Control overhead		304	24	16	16
			Data overhead		24	16	16	16
			Dummy overhead		356	0	306	0
			Total overhead	4976	684	40	338	32
			Useful data (%)	0.2	1.2	16.7	2.3	20
	iGet all	2800	Control overhead		3104	3600	2400	2400
			Data overhead		24	16	16	16
			Dummy overhead		1245	0	501	0
			Total overhead	49648	4373	3616	2917	2416
			Useful data (%)	5.3	39.0	43.6	49.0	53.7
	iWrite All	2800	Control overhead		304	3600	2400	2400
			Data overhead		24	16	16	16
			Dummy overhead		3103	0	2950	0
			Total overhead	49648	3431	3616	5366	2416
			Useful data (%)	5.3	44.9	43.6	34.3	53.7
	BASTION	11200	Control overhead		97104	14400	9600	9600
			Data overhead		7248	4832	4832	4832
			Dummy overhead		96758	0	6904	0
			Total overhead	1628848	201110	19232	21336	14432
			Useful data (%)	0.7	5.3	36.8	34.4	43.7