# Reusing IEEE 1687-Compatible Instruments and Sub-Networks over a System Bus

Farrokh Ghani Zadegan[*], Zilin Zhang[*], Kim Petersén[*], and Erik Larsson[†]
[*]Ericsson, Sweden
[†]Lund University, Lund, Sweden
Email: erik.larsson@eit.lth.se

***Abstract*—Accessing embedded test and monitoring circuitry (the so-called embedded instruments) in in-field products can reduce maintenance and diagnostics costs. Performing such access can be facilitated when done over an internal system bus, due to that it might be faster and less cumbersome to reach a system processor (on an in-field product) over a network interface, compared with the effort and speed of gaining access to a test interface on the same product. Enabling such access might require that, at the component level, the embedded instruments in a system-on-chip (SoC) become accessible both from a chip interface and from an on-chip processor over a system bus. Although this reuse of embedded instruments can be achieved by already existing standards, such as IEEE 1687, the system bus might become a scalability bottleneck when the number of instruments that are to be reused increases. In this paper, we propose two solutions that address the scalability in this type of reuse while maintaining compatibility with IEEE 1687 tools. We also discuss the trade-offs associated with each approach and present timing analyses that by considering system parameters such as clock rates determine how the correct operation can be guaranteed. To validate the proposed solutions, we have implemented them on an FPGA using AXI as system bus, and have used standard IEEE 1687 tools to access the instruments. We present some details of the implementation to highlight practical issues such as clock domain crossing, as well as how the presented timing analyses can be used to adjust design parameters.***

***Index Terms*—IEEE 1687, embedded instruments, reuse**

## I. Introduction

Modern complex system-on-chips (SoCs) pose diverse challenges throughout their entire life-cycle, from system design and integration, through prototyping (initial bring-up and debug) and manufacturing tests, to in-field health-management and repair diagnostics on returned products. To address some of these challenges, it has become common to embed additional circuitry in the SoCs to help in each of the above-mentioned life-cycle stages, such as self-tests and debug circuitry. Over the years, the increase in number and complexity of these added circuitry (the so-called on-chip instruments) and the desire to reuse them in different products and life-cycle phases has led to standardized interfaces and description languages such as the specifications in IEEE 1149.1 [1] (JTAG) and its successor 1149.1-2013 [2], IEEE 1500 [3], IEEE 1687 [4], etc.

One aspect in the reuse of embedded instruments is to be able to access them both from an on-chip processor and externally through a chip interface. This "sharing" of instruments can, for example, enable an on-chip processor to perform in-field self-test and fault-isolation by using the same instruments used during manufacturing test or those used for diagnostics. Ideally, this sharing should be done in such a way that the benefits from existing standards can still be reaped. Additionally, it would be beneficial to reuse an existing system bus for accessing the shared instruments, to avoid adding another port to the processor and possibly another bus to the system. The reuse of system buses (functional interconnects) for non-mission mode purposes have been discussed before [5]–[8].

In this work, we address the following sharing problem. For standardized access to shared instruments, we consider IEEE 1687 networks, mainly due to that IEEE 1687 is purposed for access and control of embedded instruments and due to that it is being increasingly adopted by the industry. More precisely, we consider instrument sharing in an IEEE 1687 network with an on-chip processor such that from the perspective of external tools the network is still seen compliant with IEEE 1687.

This paper is organized as follows. In Section II, we present some basics of IEEE 1687. Section III outlines our proposed hardware solutions for accessing shared instruments, and presents advantages and disadvantages of each solution. In Section IV, we present timing analyses for the proposed hardware solutions that can be used to determine how the correct operation can be guaranteed by considering system parameters such as clock rates. In Section V, we present some details from our validation experiments. Finally, Section VI presents concluding remarks and future directions for the presented work.

## II. Background and prior work

The basic idea in IEEE 1687 can be summarized as follows:

- let instrument vendors/providers describe procedures for operating each instrument at its terminals by using standard read, write, wait, etc. commands,
- let the system integrators (1) use a standard description language to describe the network that connects instruments to a chip-level interface, as well as (2) describe at the chip-level which instrument procedures (and at which order) should be run, and
- have an algorithm generate the chip-level commands (or bit-vectors) for configuring the network and carrying out the described instrument operations.

This work is partly funded by Vinnova.

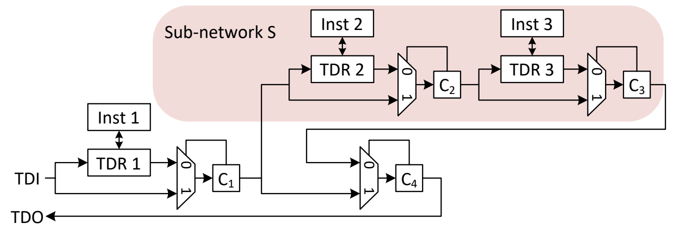


Fig. 1. An example IEEE 1687 network

The above points are discussed in more details in the following paragraphs.

To connect instruments to a chip interface, IEEE 1687 defines a reconfigurable and hierarchical scan-path, which is described in Instrument Connectivity Language (ICL). Fig. 1 shows an example of an IEEE 1687 network. The figure shows three instruments, each interfaced to the parallel input/output data ports of a dedicated test data register (TDR). The TDRs are placed on a flexible scan path, which can be reconfigured via the multiplexer control bits $C_1$–$C_4$, such that individual instruments or network segments (such as sub-network S) can be switched on and off the scan path. In the figure, only the data path is shown, and signals controlling the shift, capture, and update, as well as clock and reset are omitted. In reality, each TDR reacts to the control signals only when it is *selected* (that is, when it is switched on the scan path). For example, the *select* signal for TDR 2 is asserted when both $C_4$ and $C_2$ are set to '0'. In the rest of this paper, we refer to such reconfigurable scan networks that are compatible with IEEE 1687 as "1687 networks".

In IEEE 1687, the language for describing how to operate the instruments is Procedural Description Language (PDL). In PDL, besides the basic read, write, and wait operations at instrument ports, it is possible to describe complex operations by the use of procedures. These procedures can then be called across the hierarchy defined via ICL, up to the chip interface.

The algorithm that receives the PDL and ICL descriptions and generates the chip-level commands to carry out the procedure calls from the chip interface is called a *retargeter*.

For IEEE 1687, the chip interface mentioned above is the IEEE 1149.1 Test Access Port (TAP), which consists of four mandatory signals, namely, test data in (TDI), test data out (TDO), test mode select (TMS), and test clock (TCK), as well as an optional reset signal (TRST). The IEEE P1687.1 [9] working group is extending the interface to other standard interfaces such as I2C and SPI, while maintaining compatibility with IEEE 1687.

Fig. 2 shows the TAP controller finite state machine (FSM) that controls the operation of IEEE 1149.1 circuitry. The traversal of the FSM is controlled by the TMS signal (shown as labels on the transition arcs) and TCK. There are two main branches in the FSM: the DR-Scan branch for performing operations on the TDRs and the IR-Scan branch for loading the instruction register (IR). There are control signals generated at each of the capture, shift, and update operations. For the DR-Scan branch, these signals are named Capture-DR, Shift-DR, and Update-DR, accordingly. Although IEEE 1687 has foreseen the use of other chip interfaces, the stipulated sequence of read (capture), write (update), and data transfer (shift) operations—called a CSU cycle—closely follows that of the shown FSM. The other states in the FSM that have no associated control signals are referred to as Nop (short for no operation) in IEEE 1687, where each Nop takes one clock cycle to go through.

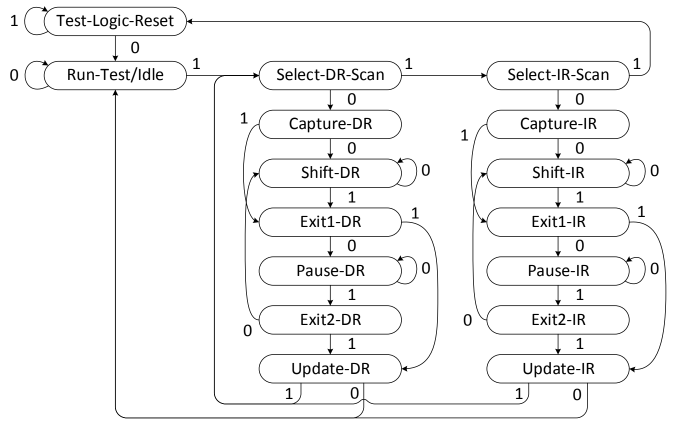


Fig. 2. IEEE 1149.1 TAP Controller FSM

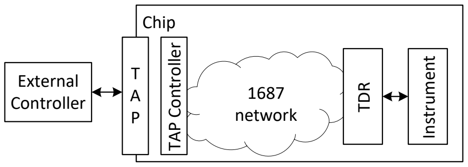
In PDL, each CSU cycle is described as a group of read and write commands to instrument ports. The language format is such that read and write operations are specified with *iRead* and *iWrite* instructions (referred to as *setup* commands), which are applied as a CSU cycle upon encountering an iApply command (or any other *action* command). In addition to the read and write operations, the *iRunLoop* command can be used to generate a specified minimum number of clock cycles that should pass before the next operation can be carried out.

## III. Proposed Hardware Solutions

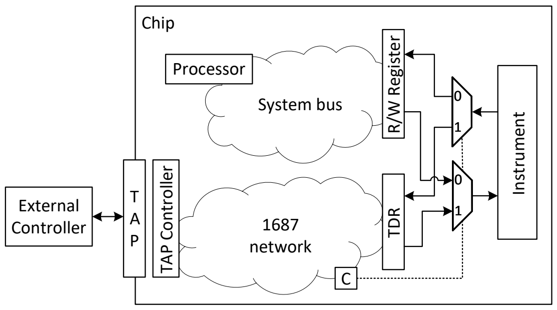
In this section, we present two hardware solutions for accessing shared instruments and discuss pros and cons of each solution. It should be noted that the proposed solutions address the physical shared access to an instrument, and higher-level protocols are needed to ensure that, for example, the CPU's operation of the shared instruments is not interrupted by the external access from the JTAG port.

Fig. 3(a) shows an instrument connected to the JTAG TAP via a 1687 network. Assume that we would like to access the same instrument from an on-chip processor via a system bus. Fig. 3(b) shows one way to achieve this sharing, in which mux control bit C selects which entity can access the instrument. In this example, it is the processor that by default (after startup) gets the access path to the instrument, and External Controller takes over by first writing to the C control bit.

The advantage of the solution shown in Fig. 3(b) is its simplicity and standard compliance. Besides, this solution allows for accessing the instrument even if the system bus is not operational, which can prove useful when an early



(a) Standard access



(b) Standard shared access

Fig. 3. Standard instrument access and sharing

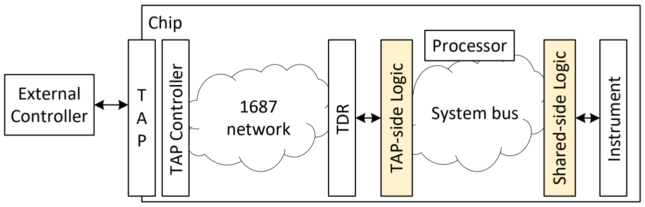


Fig. 4. Instrument sharing over a system bus

prototype has defects or the system clocks are not running. The disadvantage with this solution is that it might not scale well with number of instruments. For example, dedicating a port on the system bus (which could be a network-on-chip, AMBA bus, etc.) to each instrument would not be practical for hundreds of instruments. In this case, using a register bank in which a register is dedicated to each instrument would require only one bus port and thus alleviate the scaling issue, but would then involve routing many long parallel wires from each register to its corresponding instrument.

To address this scalability issue, we present and discuss two solutions in which a segment of a 1687 network is shared with an on-chip processor over a system bus. This sharing is done such that in the ICL description of the network, the shared segment is described as if it were directly connected the main 1687 network without the system bus and related components in between. This way, any standard retargeter can be used to perform retargeting for the proposed solutions. The first solution is based on parallel transfer of data (Section III-A), and the second solution is based on serial data transfer (Section III-B).

### A. Parallel Transfer

We start by a simplified scenario in which only one instrument is shared. Fig. 4 illustrates outline of this solution for such a scenario. This solution in itself does not address the scalability issue discussed for the solution in Fig. 3(b), and will only serve as a stepping stone towards more scalable solutions. In the solution in Fig. 4, instead of directly interfacing an instrument to its TDR via parallel data ports, the data ports are sampled on both sides and the samples are transferred across the system bus. TAP-side Logic and Shared-side Logic contain circuitry for implementing the bus protocol, as well as for sampling and copying the changed data in both directions. There are several ways that the copying process can be initiated, each way resulting in different implementation overhead and timing behavior. As discussed later (Section IV), for this work, we have chosen an implementation in which TAP-side Logic initiates the data transfer in both directions.

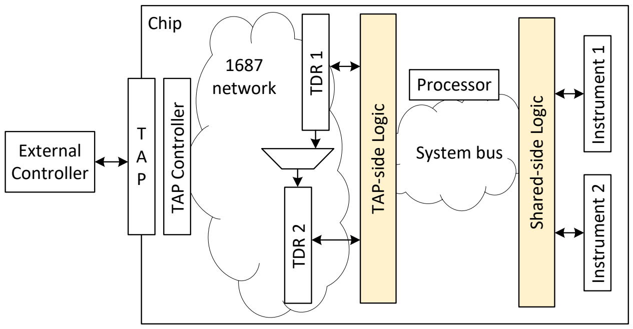


Fig. 5. Network segment sharing over a system bus

To maintain compliance with 1687 network specifications, the above-mentioned copying should be concealed from External Controller. To keep the copying hidden, a first requirement is to have bounds on data transport delays on the system bus. Such bounds exist, for example, if system bus supports arbitration priorities. In the rest of this paper we assume that the above-mentioned data transport bounds exist. In Section IV, we present timing analyses that based on the bounds on data transport delays and clock frequencies determine if and how data integrity can be guaranteed.

The solution shown in Fig. 4 can be extended (from sharing one instrument) to sharing a 1687 network segment. One way to do so is to interface the parallel data ports of all TDRs whose instruments are to be shared to TAP-side Logic as if those TDRs are parts of the same TDR, as illustrated in Fig. 5 for two instruments. In this approach no control signal (and in general nothing related to the structure) of the shared segment is copied to the instrument side. That is, if a TDR is not selected (and therefore does not react to the control signals) its contents will not change and as a result, the same old value will be transferred to its corresponding instrument during the copying process. The advantage of this approach is its simplicity, and that the shared instruments do not necessarily need to be in the same segment of the 1687 network. The disadvantage is that size of the network segment being shared becomes limited to the maximum bit-width allowed by the system bus. For example, if the system bus allows a maximum of 512 bits of parallel data transfer, then the total length of TDRs in the shared segment cannot be more than 512 bits.

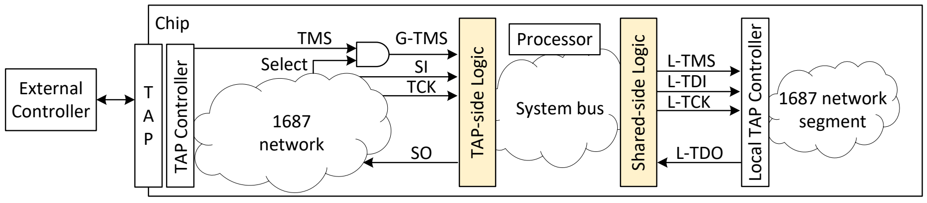


Fig. 6. Using Serial Transfer to share a network segment over a system bus

This limitation, however, can to some degree be overcome if the system bus supports bursts.

## B. *Serial Transfer*

In contrast to the Parallel Transfer approach in which only the shared instruments themselves were separated from their TDRs and placed across the system bus, in the Serial Transfer approach the shared instruments along with their TDRs are separated from the network and placed across the system bus. This way, even a whole segment of the network, complete with its muxes and control bits, can be shared. To do so, that segment is provided *locally* with its own TAP Controller circuitry so that only the serial in (SI), serial out (SO), TCK, and TMS signals are transferred over the system bus, and the control signals for the shared segment are generated locally by the local TAP Controller FSM circuitry. This local FSM is then kept in sync with the main TAP Controller FSM by intercepting (gating) of the chip TAP's TMS signal with a local select signal. This way, whenever the gated TMS (G-TMS) is de-asserted (which always happens when both FSMs are at the Update-DR state), the local TAP Controller FSM is parked in the Run-Test/Idle state, and as soon as G-TMS is asserted (which always happens when the main FSM is in the Update-DR state), both FSMs move to Select-DR-Scan together). The idea of keeping another FSM in sync with the main TAP Controller FSM exists already as ”tapped instrument” (a.k.a. embedded TAP or eTAP) in IEEE Standard 1687. The difference between our proposed Serial Transfer solution and eTAP is that (1) in the proposed solution the syncing is performed over the system bus, and (2) since there is no IR (and IR-related circuitry) used in the proposed solution, the eTAP support from the retargeter tool is not needed. As a matter of fact, the local TAP Controller FSM will even be invisible to the retargeter.

Fig. 6 illustrates this approach for sharing a segment of 1687 network. In this approach, three signals, namely, SI, G-TMS, and TCK, are sampled by TAP-side Logic and transported over the system bus to Shared-side Logic where these signals are reproduced and delivered to the local TAP controller. In the opposite direction, the local TDO (L-TDO) is sampled by Shared-side Logic and is sent over the system bus to TAP-side Logic where it is fed back into the 1687 network as SO. One advantage of this approach to sharing is that this solution does not pose any limitation on the number of instruments in the shared segment. Another advantage is that the network structure in the shared segment remains the same (as contrasted with the solution illustrated in Fig. 5). Therefore, the benefits of serial connectivity in a 1687 network are maintained, hence, less routing clutter. On the other hand, this approach requires tougher requirement on data transfer bounds in the system bus, since this approach requires that data is transferred across the system more frequently. Additionally, keeping the two FSMs in sync requires careful design of the timing for the G-TMS and L-TMS signals, so that a change in G-TMS (which happens at the falling edge of TCK) is transferred to Local TAP Controller FSM before the next rising edge of TCK (when the state changes happen). This timing requirement can be relaxed a bit if the main TAP Controller FSM is always taken to Run-Test/Idle after Update-DR.

Finally, we note that sharing a network segment requires the local processor to perform some retargeting. Such retargeting can be either done in software or as a hardware accelerator, similar to the on-chip retargeting engine proposed in [10].

## IV. System Model and Timing Analysis

As mentioned earlier, the aim with the proposed solutions in Section III is to share instruments such that standard retargeters still remain applicable, which requires making the additional hardware blocks transparent to the retargeter and External Controller. An implication of such transparency is the need to guarantee that the round-trip time over the system bus is less than the delay expected by External Controller. To provide such guarantees, in this section, we describe system models for the proposed hardware solutions, and perform timing analysis for different possible instrument access scenarios w.r.t. order and proximity of read and write instructions targeting that instrument. The presented analyses can be used to adjust the system parameters globally, and/or locally add extra delays in PDL only where extra wait time is needed.

## A. *Parallel Transfer*

In the Parallel Transfer solution, any update at TDR should be transferred to the instrument, and any change at instruments outputs should be transferred back, to be captured by TDR. We will refer to the former as a TAP-to-instrument transfer and to the latter as an instrument-to-TAP transfer. These two transfers can be done in a variety of ways, each having its own timing characteristics. In this work, we consider and analyze the following way: the TAP-side Logic block initiates the TAP-to-instrument transfer upon detecting a change at TDR's parallel data outputs. The instrument-to-TAP transfer is also initiated by TAP-side Logic through polling.

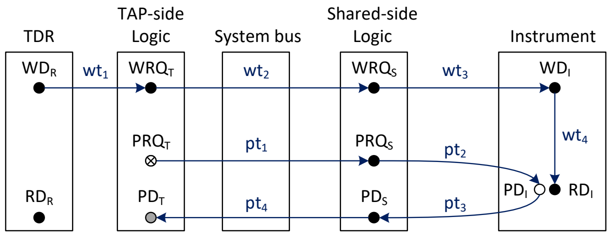


Fig. 7. System model for the Parallel Transfer solution

Fig. 7 shows the system model that we use for timing analysis. The model is based on the hardware architecture described in Section III-A. As was discussed at the end of Section III-A and illustrated in Fig. 5, parallel data signals from multiple TDRs can be bundled and transmitted together as long as the total length of TDRs included in the bundle is less than or equal to the bit width of the system bus. If this condition is met, the presented analysis holds for more than one TDR and instrument pair.

In Fig. 7 each dot and the associated notation represent a point in time at which a certain event happens in the respective component. For example, $\mathrm{WD_R}$ represents the time at which data is written to the TDR. The indices R, T, S, and I signify register (TDR), TAP-side Logic, Shared-side Logic, and Instrument, respectively. Some of the dots are marked with a different fill color or pattern to be spotted easier later in the timing diagrams (Fig. 8).

The following is a complete description of the notations used in the model:

- $\mathrm{WD_R}$: data is written to the update stage of the TDR (at the falling edge of TCK when the TAP Controller FSM is at the Update-DR state)
- $\mathrm{WRQ_T}$: a write request is generated in TAP-side Logic for transferring data to the instrument
- $\mathrm{WRQ_S}$: $\mathrm{WRQ_T}$ is received at Shared-side Logic, which subsequently makes the received data available to the instrument ports
- $\mathrm{WD_I}$: data presented by Shared-side Logic is read by the instrument
- $\mathrm{RD_I}$: the (immediate) response by the instrument (such as a busy flag being set) is produced
- $\mathrm{PRQ_T}$: a read request is generated by the periodic polling process to sample data at the instrument's outputs
- $\mathrm{PRQ_S}$: $\mathrm{PRQ_T}$ is received at Shared-side Logic, which subsequently initiates sampling of the instrument's outputs
- $\mathrm{PD_I}$: output data from instrument is sampled
- $\mathrm{PD_S}$: the sampled data from instrument is sent to TAP-side logic
- $\mathrm{PD_T}$: sampled data transmitted at $\mathrm{PD_S}$ is received by TAP-side Logic and is placed on the parallel inputs of the TDR
- $\mathrm{RD_R}$: data present on the parallel inputs of the TDR is captured into the TDR (when the TAP Controller FSM is at the Capture-DR state)
- $\mathrm{wt_1}$–$\mathrm{wt_4}$: delays (in system clock cycles) between the time points initiated by Update-DR
- $\mathrm{pt_1}$–$\mathrm{pt_4}$: delays (in system clock cycles) between the time points of the events initiated by the polling process

For $\mathrm{RD_I}$, it should be noted that the "immediate response" here refers to acknowledging the reception of a command by an instrument, rather than the final output prepared in response to the command itself. For example, in case of a built-in self-test instrument, the immediate response to starting a test could be asserting the busy flag, whereas the final output could be the pass/fail status. For the former, the PDL developer expects that the instrument's reaction is captured in the next Capture-DR phase without any additional delay, whereas for the latter, typically iRunLoop commands are used to indicate the minimum number of clock cycles the instrument needs to prepare the results. Needless to say, if an instrument can prepare the final results without added iRunLoop delays, the following discussion will be also applicable to such an instrument.

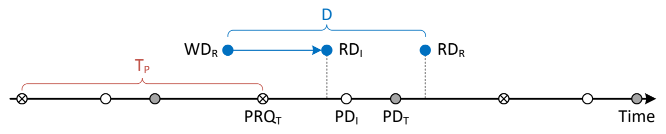
Moreover, $\mathrm{wt_2}$ represents the longest time it takes to transfer the data over system bus. This way, $\mathrm{wt_2}$ should be replaced by the data transport bound for a write operation for the given system bus. Such a write transfer could, for example, involve a four-way handshake in which first a write address is sent and acknowledged followed by the data sent and acknowledged. Such handshaking might also apply to the interface between Shared-side Logic and the instrument. Similar to $\mathrm{wt_2}$, $\mathrm{pt_1}$ and $\mathrm{pt_4}$ represent the longest time it takes to read data over system bus, where $\mathrm{pt_1}$ represents the address transfer time and $\mathrm{pt_4}$ represents data transfer time.

In this analysis, we assume that the system bus does not support overlapping read/write requests. Therefore, we assume that a request of any type (i.e., read or write) should complete before the next request of that type can start. We, however, assume that requests of different types can overlap in time. Based on these assumptions, we can write the following for the polling period $\mathrm{T_P}$:

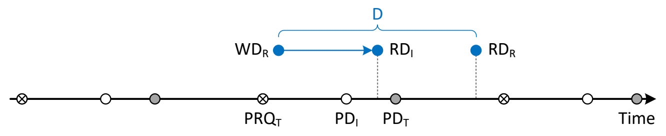
$$T_P \geq \sum_{i=1}^{4} pt_i \tag{1}$$

For the timing analysis, we consider two scenarios for the read and write commands targeting the same TDR: (1) when the commands are carried out under the same CSU cycle (that is, the corresponding iRead and iWrite instructions are under the same iApply group), as well as (2) when commands are executed under two consecutive CSU cycles. If we design the system such that these two scenarios are correctly handled, the other scenarios where there is an arbitrary number of CSUs between read and write commands will also be handled correctly (irrespective of whether those commands target the same register or different registers).

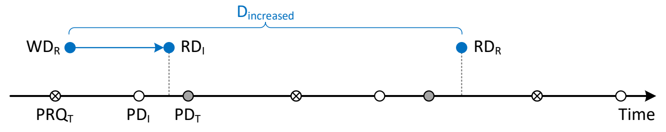
When there are read and write commands targeting the same TDR in the same CSU cycle, the read is carried out when the TAP Controller FSM is in the Capture-DR state, and the write is carried out later when the TAP Controller FSM is in the



(a) Correct read



(b) Incorrect read



(c) Correct read via increasing D

Fig. 8. Timing diagrams for the write followed by read case

Update-DR state. Therefore, the written data cannot affect the read data as the latter is already captured and shifted out before the new data is written to the TDR in the Update-DR state.

When read and write happen in consecutive CSU cycles, the following four cases should be considered:

*1) Write Followed by a Read:* For this case, we can consider the following relation between the $\mathrm{RD_R}$ and $\mathrm{WD_R}$ time points:

$$RD_R = WD_R + (2 + T_L)K \tag{2}$$

where K is the ratio of the frequency of the system clock to that of TCK, 2 denotes the smallest number of TCKs needed by the TAP Controller FSM to transition from the Update-DR state to the Capture-DR state, and $\mathrm{T_L}$ is the extra delay that might have to be added between the consecutive iApply groups by using the iRunLoop command.

In the following discussion, let:

$$D = (2 + T_L)K \tag{3}$$

Therefore, we can simplify (2) by (3):

$$RD_R = WD_R + D \tag{4}$$

To correctly carry out a write followed by a read operation, the following must hold:

$$PD_I > RD_I \tag{5}$$
$$PD_T < RD_R \tag{6}$$

In plain language, (5) states that we must not sample instrument data before the immediate response of the instrument is ready, and (6) states that the sampled immediate response must arrive at TDR before Capture-DR is asserted. One such correct read operation is illustrated in Fig. 8(a). However, as TAP is operated by an external controller, no assumption on time of $\mathrm{WD_R}$ can be made, which can result in an incorrect read. As an example, Fig. 8(b) illustrates how a small shift in timing of $\mathrm{WD_R}$ results in data being sampled too early at the instrument, thus violating (5). The read issue illustrated in Fig. 8(b) cannot be addressed by faster polling.

Since we have made no assumptions on any synchronization between the operation of TAP and the polling cycles, the only option for guaranteeing the correct reading of instrument's immediate response is to increase D beyond one polling cycle such that if data is sampled too early by the current polling cycle the next cycle corrects the sampling. This increase of D beyond one polling cycle is conceptually illustrated in Fig. 8(c), where a violation of (5) in the first polling cycle is compensated for at the next cycle. To select D such that correct reads are guaranteed we need to delay the read from External Controller so much that (1) the write command reaches the instrument and the immediate response becomes ready to be captured (i.e., $\sum_{i=1}^{4} wt_i$), (2) a complete polling cycle passes to guarantee the response is captured by the polling process (namely, $\mathrm{T_P}$), and (3) the captured response arrives at TDR (namely, $pt_3 + pt_4$). The following inequality captures this:

$$D > \sum_{i=1}^{4} wt_i + T_P + pt_3 + pt_4 \tag{7}$$

Since higher D implies slower application of PDL instructions, it is desirable to decrease the right-hand side of (7) so that D can be reduced. Lowering the right-hand side can be achieved via choosing smaller $\mathrm{T_P}$ (which translates into faster polling, thus higher traffic on the system bus), and lowering the internal delay of the system. On the other hand, D can be increased to meet this lower bound via two configuration knobs (see (3)): (1) increasing the ratio of system clock frequency to TCK frequency, and (2) adding wait cycles in the PDL script. Increasing K affects the whole system, whereas adding the wait cycles only affects the case where a write to a TDR is immediately followed in the next CSU cycle by a read targeting the same TDR.

*2) Write Followed by a Write:* Following the above-mentioned assumption of no outstanding writes, a write operation should finish before another one can start. Assuming the shortest possible TDR length of one bit, the shortest sequence of operations for two consecutive writes is as Update-DR, Select-DR-Scan, Capture-DR, Shift-DR, Exit1-DR, Update-Dr. Denoting the time points for the first and second writes as $WD^1_R$ and $WD^2_R$, respectively, we have:

$$WD^2_R = WD^1_R + (5 + T_L)K \tag{8}$$

where K and $\mathrm{T_L}$ have the same definition as before—see under (2)—and 5 represents the sequence mentioned above. For $\mathrm{T_L}$ we note that the delay added for this "write followed by a write" is not necessarily the same as the delay added for the "write followed by a read" case discussed above. To ensure that one write operation completes before the next one starts we should have:

$$(5 + T_L)K > \sum_{i=1}^{4} wt_i \tag{9}$$

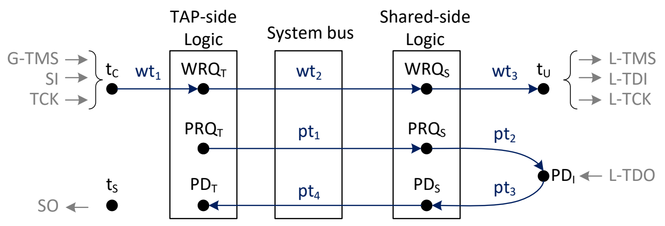


Fig. 9. System model for the Serial Transfer solution

*3) Read Followed by a Read:* Before discussing the timing analysis for this case, we note that two consecutive reads from an instrument without any writes in between could be, for example, for reading status flags, temperature, etc. Again, assuming no multiple outstanding reads at any point, a read operation should finish before any other read operation can start. Assuming the shortest possible TDR length of one bit, the shortest sequence of operations for two consecutive reads is as Capture-DR, Shift-DR, Exit1-DR, Update-DR, Select-DR-Scan, Capture-DR. Denoting the time points for the first and second writes as $RD_R^1$ and $RD_R^2$, respectively, we have:

$$RD_R^2 = RD_R^1 + (5 + T_L)K \tag{10}$$

where K and $\mathrm{T_L}$ have the same definitions as before, and 5 represents the sequence mentioned above. To ensure that a read operation finishes before the next one starts, we need:

$$(5 + T_L)K > T_P + pt_3 + pt_4 \tag{11}$$

This is to ensure that at least one polling round completes and data is ready to be captured, before the next read operation.

*4) Read Followed by a Write:* For this case, we note that the read and write belong to different CSU cycles. That is, assuming again the shortest possible shift-path, the sequence of operations to carry out a read followed by a write is as follows: Capture-DR, Shift-DR, Exit1-DR, Update-DR, Select-DR-Scan, Capture-DR, Shift-DR, Exit1-DR, Update-DR. For this case, External Controller uses the data sampled and shifted out at the first Capture-DR and Shift-DR as the read data. This way, this read operation cannot be affected by the write that follows in the next CSU cycle.

### *B. Serial Transfer*

The system model for the Serial Transfer solution (see Fig. 9) is similar to (but somewhat simpler than) the model for Parallel Transfer. In Serial Transfer, a change detected on any of the three TAP-side wires shown in Fig. 6, namely, G-TMS, SI, and TCK, is sent by TAP-side Logic to Local TAP Controller over the system bus. TAP-side Logic also polls for samples of L-TDO signal by sending read requests to Shared-side Logic.

The following is a complete description of the notations used in the model:

- $\mathrm{t_C}$: a change happens at any of the shown signals
- $\mathrm{WRQ_T}$: the change is detected and a write request is generated in TAP-side Logic for transferring data to Shared-side Logic
- $\mathrm{WRQ_S}$: $\mathrm{WRQ_O}$ is received at Shared-side Logic, which subsequently writes the received data to Local TAP Controller
- $\mathrm{T_U}$: data is updated at the input ports of Local TAP Controller
- $\mathrm{PRQ_T}$: a read request is generated by the periodic polling process to sample data at L-TDO
- $\mathrm{PRQ_S}$: $\mathrm{PRQ_T}$ is received at Shared-side Logic, which initiates sampling of L-TDO
- $\mathrm{PD_I}$: L-TDO is sampled
- $\mathrm{PD_S}$: L-TDO sample is sent to TAP-side Logic
- $\mathrm{PD_T}$: sampled data transmitted at $\mathrm{PD_S}$ arrives at TAP-side Logic
- $\mathrm{t_S}$: data at the output of TAP-side Logic is sampled on the rising edge of TCK
- $\mathrm{wt_1}$–$\mathrm{wt_3}$: delays (in system clock cycles) between the time points initiated by Update-DR
- $\mathrm{pt_1}$–$\mathrm{pt_4}$: delays (in system clock cycles) between the time points of the events initiated by the polling process

Since we are using a gated copy of the TMS signal, we note that according to the JTAG standard [1], it is expected that External Controller changes the TMS and TDI values on the falling edge of TCK. Additionally, by choosing the lock-up latch type implementation of the scan path in 1687 networks [4], we ensure that SI and SO also change at the falling edge. This way, at each cycle of TCK, we have only two samples to transfer: one at the falling edge of the TCK and one at the rising edge. Therefore, we consider that if we (1) allow enough time for the serial data to be transferred from TAP-side Logic to Shared-side Logic ($\sum_{i=1}^{3} wt_i$), (2) allow for a whole polling period ($\mathrm{T_P}$) to pass so that we ensure the L-TDO is sampled, and (3) allow for sampled L-TDO to be transferred back from Shared-side Logic to TAP-side Logic (namely, $pt_3 + pt_4$), then we can ensure that the local TAP is kept in sync with the main TAP and the L-TDO is transferred back in time. The above points are captured in the following inequality:

$$K > 2 \times (\sum_{i=1}^{3} wt_i + T_P + pt_3 + pt_4) \tag{12}$$

Coefficient 2 in (12) represents the above-mentioned fact that two samples must be transferred per each TCK period.

## V. Implementation and Experimental Results

We have implemented the proposed solutions on an FPGA to validate them against standard IEEE 1687 tools. For the implementation, we chose the 1687 network shown in Fig. 1, where each TDR is 16 flip-flops long and each instrument is just an array of inverters (so that any data written to the instrument is immediately inverted, and the next read operation gets the inverted data back). For the Parallel Transfer solution, we shared Inst 2 and Inst 3 over the system bus, and for the Serial Transfer approach we shared the segment marked as sub-network S over the system bus. As for the system bus itself, we chose AXI [11] Interconnect.

In this section, we present our general validation scheme. Moreover, we present some implementation aspects (such as clock domain crossing considerations) that we had to address, as well as the delay numbers associated with our implementation. Furthermore, we will present how the presented timing analysis can in practice be used to adjust the system and PDL-related parameters.

### A. Validation Approach

A major feature of the presented solutions is that, from the perspective of retargeter tools, the shared instruments appear as if they were directly connected to the 1687 network that is interfaced to the JTAG TAP. That is, the tools can be kept completely unaware of the fact that the implementation deviates significantly from what is described in ICL.

To validate this feature, we had a commercial retargeter generate test patterns in Serial Vector Format (SVF) for the example network shown in Fig. 1, and applied the generated patterns via OpenOCD (Open On-Chip Debugger) [12] to our FPGA implementation. OpenOCD applies the test stimuli in the SVF file to the implemented hardware and compares the test responses with the expected values (also available in the generated SVF patterns).

For the ICL description, as intended by IEEE 1687, only the parallel interfaces of the instruments (i.e., the inverters in our case) are described. Therefore, only the structural connectivity of the TDRs, multiplexers and their associated controller bits, as well as the length of the TDRs are described in ICL.

As for the PDL code, we created a long sequence of ”writes immediately followed by reads” targeting Inst 2 and Inst 3 with alternating patterns, similar to what is shown in Fig. 10. In the shown PDL snippet, Inst refers to the ICL definition of the pair of a TDR and its associated inverter instrument. In the test procedure that is defined for such a pair, the iRead instruction in Line 3 expects to receive FFFFh since after reset the contents of TDRs are 0000h. The iRead instruction in Line 4 expects to receive 5555h since iWrite in Line 3 has written AAAAh to the TDR, and so on. The last iWrite in test writes zeros, so that the procedure can be called many times (as the first iRead expects FFFFh). The Run_test procedure calls test on Inst 2 and Inst 3 hundreds of times in *iMerge* blocks (which give the retargeter the option to schedule the operations concurrently). Close examination of the generated SVF patterns showed that the writes and reads to the two instruments are indeed scheduled at the same.

For both hardware implementations described below (sections V-B and V-C), the generated SVF patterns were successfully applied by OpenOCD to the hardware on the FPGA.

### B. Implementation Based on Parallel Transfer

In the following, we describe our hardware implementation of the Parallel Transfer solution, and present and discuss the timing aspects.

*1) Implementation Details:* One implementation issue to address for Parallel Transfer is transferring data between the

```
1   iProcsForModule Inst
2   iProc  test {}   {
3       iWrite  Ins.DI 0xAAAA;    iRead  Ins.DO 0xFFFF;    iApply;
4       iWrite  Ins.DI 0x5757;    iRead  Ins.DO 0x5555;    iApply;
5       iWrite  Ins.DI 0x1234;    iRead  Ins.DO 0xA8A8;    iApply;
6       iWrite  Ins.DI 0x0000;    iRead  Ins.DO 0xEDCB;    iApply;
7   }
8
9   iProcsForModule Chip
10  iProc Run_test {} {
.   ...
.       iMerge -begin
.           iCall Inst_2.test;
            iCall Inst_3.test;
        iMerge -end
    ...
    }
```

Fig. 10. Excerpt from the PDL used for validation of the 16-bit ”inverter” instruments

JTAG and system clock domains while maintaining data coherency of instrument data. This clock domain crossing (CDC) should be done considering that TCK is not free running. In our implementation, we have used the CDC approach shown in Fig. 11, which relies on the free-running and faster Clk to transfer Update-DR to the system domain. To reduce clutter, the clock signals are not explicitly shown, and it should be assumed that flip-flops to the right side of the dashed vertical line are clocked by Clk and the flip-flops to the left side of the dashed line are clocked by TCK.

As specified by JTAG and IEEE 1687, data is written into the update stage of a TDR at the falling edge of TCK when Update-DR is asserted. Therefore, by the time that Update-DR itself is transferred to the Clk domain, the data in the update stage has become stable. The transferred Update-DR is then used as clock-enable (CE) of the flip-flops marked with an ’S’, resulting in instrument data being coherently registered in the S flip-flops. TAP-side Logic, samples the outputs of ’S’ flip-flops at every cycle of Clk and upon detecting a change with the previous value, starts a new transfer. It should be noted that, depending on the clock ratio K, Update-DR might stay active for more than one cycle of Clk, resulting in the same data being sampled multiple times, which is no problem as TAP-side Logic reacts only to changes in the sampled data. In the reverse direction, the instrument response that TAP-side Logic presents to the capture ports of the TDR should in principle be stable before the capture happens, as ensured by the presented analyses for the ”write followed by a read” case (Section IV-A1).

Fig. 12 shows our implementation of the Parallel Transfer approach. As can be seen, the two instruments originally belonging to sub-network S, namely, Inst 2 and Inst 3, are connected to Shared-side Logic. TDR 2 and TDR 3 together form the TDR (virtually), and their parallel data outputs are connected to TAP-side Logic via the scheme shown in Fig. 11. Controller in TAP-side Logic detects changes in the contents of TDR, and instructs the AXI bus manager (AXI Manager 1) to send the data over to AXI Subordinate. The received data is then delivered to Controller in Shared-side Logic to be applied to Inst 2 and Inst 3. Additionally, Controller in TAP-side Logic

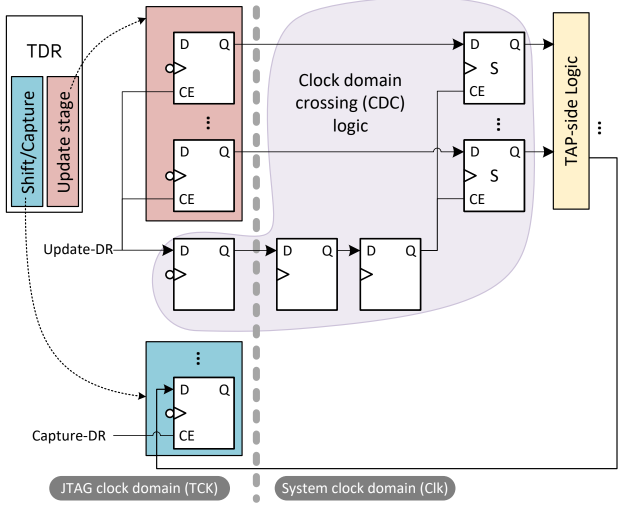


Fig. 11. Parallel Transfer: Circuitry for clock domain crossing from JTAG clock (TCK) domain to system clock (Clk) domain

performs periodic polling by instructing AXI Manager 1 to read data from AXI Subordinate. Upon detecting the read request, AXI Subordinate samples the instrument ports and sends the data back, which is then presented to the capture ports of TDR. The data is then captured into TDR 2 and TDR 3 during the next Capture-DR event. AXI Manager 2 in this implementation is a traffic generator and consumer IP, which has a lower priority than Manager 1 and is added for a more realistic evaluation of the delays. The AXI Interconnect IP is configured in "Shared Address Multiple Data" (SAMD) mode, in which a delay mainly happens when TAP-side Logic starts a transaction on the AXI address channel when a transaction of another type is already using that channel.

*2) Delays and Discussion on Timing Analysis:* Table I presents the calculated delays according to the model presented in Fig. 7. The delays reported for the system bus (namely, $\mathrm{wt}_2$, $\mathrm{pt}_1$, and $\mathrm{pt}_4$) are calculated for the worst case according the data sheet for the AXI Interconnect IP and validated in the simulations. Due to the SAMD configuration of the Interconnect IP, the worst case happens for a transaction when another transaction has already taken the address channel of AXI Interconnect. It should be noted that an AXI write transaction involves the bus manager writing the address and having it acknowledged by the bus subordinate, followed by the bus manager writing the data and having it acknowledged by the bus subordinate, followed by the subordinate sending a write response and having it acknowledged by the bus manager. In the reported delay for $\mathrm{wt}_2$, we have only included the number of clock cycles until the data is delivered to Shared-side logic, since the rest of the steps in the handshake process overlap in time with $\mathrm{wt}_3$ and $\mathrm{wt}_4$.

These delay numbers allow for polling as fast as every 17 system clock cycles according to (1). Plugging $\mathrm{T_P}$=17 into (7), (9), and (11) gives us the following constraints for the three

TABLE I
MEASURED DELAYS (IN SYSTEM CLOCK CYCLES), SEE FIG. 7

| $\mathrm{wt}_1$ | $\mathrm{wt}_2$ | $\mathrm{wt}_3$ | $\mathrm{wt}_4$ | $\mathrm{pt}_1$ | $\mathrm{pt}_2$ | $\mathrm{pt}_3$ | $\mathrm{pt}_4$ |
|---|---|---|---|---|---|---|---|
| 7 | 7 | 2 | 0 | 9 | 1 | 1 | 6 |

relevant cases discussed in Section IV-A:

$$(2 + T_{L,WR})K > 40 \tag{13}$$
$$(5 + T_{L,WW})K > 16 \tag{14}$$
$$(5 + T_{L,RR})K > 24 \tag{15}$$

The iRunLoop delays, namely $\mathrm{T_L}$ values, are indexed differently (i.e., WR for write followed by a read, WW for write followed by a write, and RR for read followed by a read) as they do not need to be the same for different cases. To simplify the following discussion, we assume that no additional iRunLoop delays are to be added. That is, $\mathrm{T_L}$ is zero in (13), (14), and (15). It follows that among the above three constraints, it is (13) that puts the most demanding requirement on the clock ratio, namely, $K > 20$. In plain language, in order for the solution to work for any given PDL, the system clock should be at least 20 times faster than the JTAG clock. We note that with longer polling periods (i.e., increased $\mathrm{T_P}$) even larger clock ratio $K$ will be needed. It is, on the other hand, possible to achieve a lower $K$ by adding $\mathrm{T_L}$ delays in PDL where needed (i.e., between any two iApply groups in which the same instrument is being accessed).

### C. Implementation Based on Serial Transfer

The clock domain crossing in this approach is much simpler than the Parallel Transfer solution and consists of a few synchronization flip-flops on each JTAG signal. Considering the relatively much higher rate of the system clock, and the fact that TMS and TDI are expected to change value at the falling edge of the JTAG clock, there is plenty of time for TDI and TMS to settle to the right value before the rising edge of TCK arrives.

In the following, we describe our hardware implementation of the Serial Transfer solution, and present and discuss the timing aspects.

*1) Implementation Details:* Fig. 13 outlines our hardware implementation. The Tap-side Logic and Shared-side Logic blocks are the same as the ones in Fig. 12 and are, therefore, not detailed. In contrast to Parallel Transfer, here the whole sub-network S is shared. In our implementation, the logic on Shared-side is reset using both the system reset and a reset generated by the Local TAP when its FSM is in the Test-Logic-Reset state. The TCK, gated TMS (G-TMS), and SI signals are sampled by TAP-side Logic and, upon detecting a change, the sample is sent to Shared-side Logic to be applied to L-TCK, L-TMS, and L-TDI, respectively. TAP-side Logic, also received samples of L-TDO through polling, and drives the SO signal. The FSM in Local TAP Controller generates the control signals for the operation of sub-network S locally.

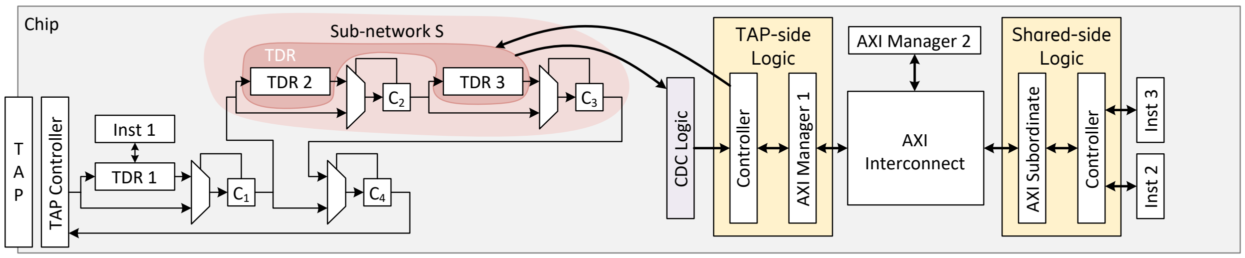


Fig. 12. Example implementation of the Parallel Transfer approach

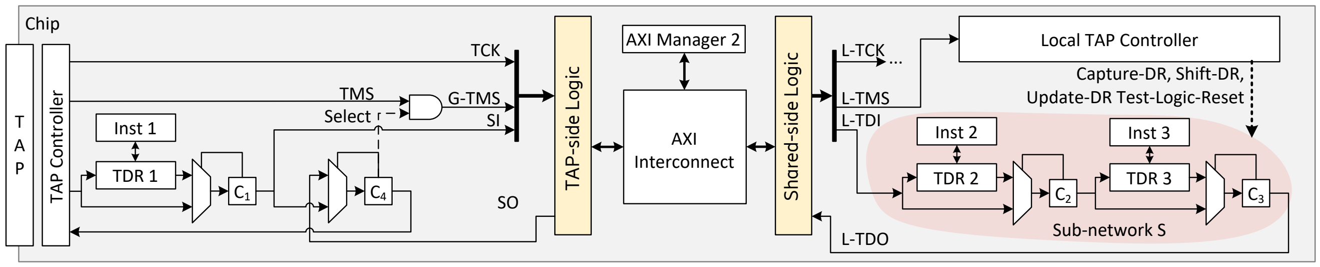


Fig. 13. Example implementation of the Serial Transfer approach

TABLE II
MEASURED DELAYS (IN SYSTEM CLOCK CYCLES), SEE FIG. 9

| $wt_1$ | $wt_2$ | $wt_3$ | $pt_1$ | $pt_2$ | $pt_3$ | $pt_4$ |
|---|---|---|---|---|---|---|
| 6 | 7 | 2 | 9 | 1 | 1 | 6 |

*2) Delays and Discussion on Timing Analysis:* Table II presents the calculated delays according to the model presented in Fig. 9. Since the TAP-side Logic and Shared-side Logic blocks are identical to those used in Parallel Transfer, $wt_2$–$wt_3$ and $pt_1$–$pt_4$ are the same as what was described for Parallel Transfer. By using (12) and choosing the fastest polling ($T_P$=17 clock cycles), we have $K > 78$. That is, the system clock should be at least 78 times faster than TCK to ensure timely transfer of the serial data across the system bus. Choosing longer polling intervals would then require a faster system clock or (more realistically) a slower TCK. In comparison to the Parallel Transfer solution, the Serial Transfer approach imposes a stricter limit on the TCK rate, which might still be justifiable given the advantages of the Serial Transfer approach (Section III-B). For example, for a system with a system clock of 1GHz, our implementation of the Serial Transfer approach still allows for a TCK of about 12.8 MHz, which might still be acceptable.

## VI. Conclusion and Future Work

In this work, we presented two hardware solutions that enable an on-chip processor to access instruments in an IEEE 1687 network over a system bus while maintaining compliance with standard IEEE 1687 tools—a reuse scheme we referred to as ”sharing” in this work. We have validated the solutions through FPGA tests by using standard IEEE 1687 and JTAG tools. In our timing analyses and hardware implementations of the proposed solutions, we have considered a data transfer scheme that in one direction is based on polling. Polling happens irrespective of any change in signal values, which might cause unnecessary traffic on the system bus. Lowering the polling rate on the other hand might limit frequency of the JTAG clock. Therefore, for future work, we consider other data transfer schemes for the proposed solutions.

## References

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

[2] “IEEE standard for test access port and boundary-scan architecture,” *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, 2013.

[3] “IEEE standard testability method for embedded core-based integrated circuits,” *IEEE Std 1500-2005*, 2005.

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

[5] A. Amory, K. Goossens, E. J. Marinissen, M. Lubaszewski, and F. Moraes, “Wrapper design for the reuse of networks-on-chip as test access mechanism,” in *Proc. Eleventh IEEE European Test Symposium (ETS’06)*, 2006, pp. 213–218.

[6] T. Yoneda and H. Fujiwara, “Wrapper and TAM co-optimization for reuse of SoC functional interconnects,” in *Proc. Design, Automation and Test in Europe*, 2008, pp. 1366–1369.

[7] A. van den Berg, P. Ren, E. J. Marinissen, G. Gaydadjiev, and K. Goossens, “Bandwidth analysis for reusing functional interconnect as test access mechanism,” in *Proc. 13th European Test Symposium*, 2008, pp. 21–26.

[8] J. Song, H. Yi, J. Han, and S. Park, “An efficient SoC test technique by reusing on/off-chip bus bridge,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 3, pp. 554–565, 2009.

[9] P1687.1, “Standard for the application of interfaces and controllers to access 1687 IJTAG networks embedded within semiconductor devices,” Mar. 2022. [Online]. Available: https://standards.ieee.org/ieee/1687.1/6900/

[10] A. Ibrahim and H. G. Kerkhoff, “Analysis and design of an on-chip retargeting engine for IEEE 1687 networks,” in *Proc. 21st IEEE European Test Symposium (ETS)*, 2016, pp. 1–6.

[11] Arm, “AMBA® AXI and ACE protocol specification,” Mar. 2022. [Online]. Available: https://developer.arm.com/documentation/ihi0022

[12] OpenOCD, “Open On-chip Debugger,” Mar. 2022. [Online]. Available: https://openocd.org/