

# On-chip Instrument Access Through System Hierarchy

---

SHASHI KIRAN GANGARAJU

MASTER'S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY



# On-chip Instrument Access Through System Hierarchy

Shashi Kiran Gangaraju  
sh6071ga-s@student.lu.se

Department of Electrical and Information Technology  
Lund University

Supervisor: Dr. Erik Larsson  
erik.larsson@eit.lth.se

Examiner: Dr. Pietro Andreani  
pietro.andreani@eit.lth.se

October 29, 2020



---

# Abstract

---

There are many advantages with the development towards Integrated Circuits (ICs) with smaller, faster, and more transistors. However, tighter margins lead to a need of on-chip instruments to test, tune, and configure. These on-chip instruments, which can be in the range of thousands per IC, must be accessed through the life-time. However, access is challenged by complex system hierarchies. When ICs are mounted on Printed Circuit Boards (PCB) there might not be a direct access path and a single access protocol. In this thesis we have developed a generic module to handle communication between two ICs. We propose two alternatives for the communication. A hardware-based where communication is handled in an interrupt-driven manner and a software-based where the communication is handled through polling. We have made experiments using a Xilinx Field-Programmable Gate Array (FPGA) where we compare the solutions in terms of data overhead and area cost. We implemented two ICs on the FPGA where we for one IC used on-chip instruments connected using IEEE Std. 1687 and for the second IC we implemented our module connected using IEEE Std. 1687. The communication between the ICs was performed with Serial Peripheral Interface (SPI) and the communication with the outside world with Universal Asynchronous Receiver Transmitter (UART). The experiments show that the hardware-based solution provides little data and limited area overhead.



---

# Popular Science Summary

---

In the modern times, we use and depend more and more on systems with a part of electronics. The electronic part of a system consists of Integrated Circuits (ICs) mounted on Printed Circuit Boards (PCBs). The development towards Integrated Circuit (IC) with smaller, faster, and more transistors, has many performance advantages. However, smaller and faster transistors lead to tighter margins. These tight margins make it a challenge to perform testing, tuning, and configuration. On-chip instruments (also called as embedded instruments) refer to the integration of test and measurement instrumentation into semiconductor chips (or integrated circuit devices). As these instruments are placed inside the ICs close to the logic, which is important due to tight margins, there is a need to access and analyse these instruments throughout the lifetime of the IC. Recent electronic systems are increasingly complex with more ICs and more advanced ICs, the access through the system hierarchies can be difficult as there might not be a direct access path and a single access protocol.

In this thesis we have developed a generic module to handle communication including synchronization between two ICs. We propose two alternatives for the communication. A hardware-based where communication is handled in an interrupt-driven manner and a software-based where the communication is handled through polling. We have made experiments using an Xilinx Field-Programmable Gate Array (FPGA) where we compare the solutions in terms of data overhead and area cost. We implemented two ICs on the FPGA where we for one IC used on-chip instruments connected using Institute of Electrical and Electronics Engineers (IEEE) Std. 1687 and for the second IC we implemented our module connected using IEEE Std. 1687. The communication between the ICs was performed with Serial Peripheral Interface (SPI) and the communication with the outside world with Universal Asynchronous Receiver Transmitter (UART). The experiments show that the hardware-based solution provides minimum data overhead and limited area overhead, whereas the software based solution provides more data overhead and less area overhead.



---

## Acknowledgment

---

- First and foremost I would like to thank Swedish University Admissions department for providing me the opportunity to study at one of the best Universities in Sweden and in the world and Migrationsverket for understanding the student problems and providing me the proper support to continue my studies.
- I would like to express my sincere gratitude to my supervisor Prof. Dr.Erik Larsson, who supervised this project and the examiner Prof. Dr.Pietro Andreani. I would like to express the deepest appreciation to my International Master's Coordinator - Helene von Wachenfelt,for her patient understanding of my personal problems, encouragement and administrative support.
- My special gratitude goes to Mr. R.Vasanth Kumar and Mr. R.V. Rohit Kumar from Kumar Electrical for funding me through out my educational journey. Without their support I would not be have been able to study this far.
- I would also like to thank my peer Mr. Prathamesh Murali. for his unstinted guidance and support during the thesis implementation.
- Last but not least, I would like to express my gratitude to my parents, families and friends Ravi, Manu, Jayanth, Pramod, Vinay, Nagesh, Vinod, Ajay, Praveen, Srinivasan, Ghanshyam, Gustav, Robel and Ennio. Their motivation and support helped me to mentally prepare and face challenges during my masters studies at Lund University.





---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation	3
1.2	Thesis objectives	3
1.3	Thesis organization	4
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	System overview	5
2.2	Institute of Electrical and Electronics Engineers (IEEE) Std. 1149.1	6
2.3	IEEE Std. 1687 (Internal Joint Test Action Group (IJTAG))	8
2.3.1	SIB	8
2.3.2	Description language	9
2.4	Related work	11
2.4.1	Data overhead classification	12
2.5	Communication technique	13
<b>3</b>	<b>Proposed solutions</b>	<b>15</b>
3.1	Synchronization module	15
3.1.1	Sending data	15
3.1.2	Receiving data	16
3.2	Hardware-based solution	16
3.2.1	Universal Asynchronous Receiver Transmitter (UART) master controller	17
3.2.2	Serial Peripheral Interface (SPI) system	18
3.2.3	SPI master controller	18
3.2.4	Hardware description of the solution	18
3.2.5	Retargeting flow for the hardware-based solution	21
3.3	Software-based solution	24
3.3.1	IC-2	24
3.3.2	Polling function	25
3.3.3	Check-up function	26
3.3.4	Hardware description of the solution	26
3.3.5	Retargeting flow for the software-based solution	27
<b>4</b>	<b>Experiments and Results</b>	<b>31</b>

4.1	Results of experiment-1 . . . . .	33
4.2	Results of experiment 2 . . . . .	35
<b>5</b>	<b>Conclusion and Future Work</b> _____	<b>39</b>
	<b>Bibliography</b> _____	<b>41</b>
<b>A</b>	<b>Appendix 1</b> _____	<b>43</b>
A.1	UART standard . . . . .	43
A.2	SPI bus interface . . . . .	44
A.3	Retargeting tool algorithms . . . . .	46

---

## List of Figures

---

2.1	System overview . . . . .	6
2.2	A conceptual view of IEEE Std. 1149.1 circuitry . . . . .	7
2.3	IEEE Std. 1149.1 TAP Controller state diagram. Courtesy: [3] . . . . .	7
2.4	Flat IEEE Std. 1687 network consists three instruments . . . . .	8
2.5	Implementation of SIB Schematic . . . . .	9
2.6	Conceptual schematic of Instrument Connectivity Language (ICL) for IC-4 . . . . .	10
2.7	Procedural Description Language (PDL) example . . . . .	10
2.8	Full-featured case . . . . .	11
2.9	Developed protocol . . . . .	12
3.1	Synchronization module overview . . . . .	15
3.2	Architecture of hardware-based solution . . . . .	16
3.3	ASMD of DTFSM . . . . .	19
3.4	ASMD of RSFSM . . . . .	20
3.5	Bits generation by retargeting tool of iWrite PDL . . . . .	22
3.6	Bits generation by retargeting tool of iRead PDL . . . . .	23
3.7	Architecture of software-based solution . . . . .	25
3.8	Re-Targeting tool functions . . . . .	25
3.9	ASMD of the Software Solution . . . . .	27
3.10	Bits generation by retargeting tool of iRead PDL . . . . .	28
3.11	Bits generation by retargeting tool of iRead- polling and check-up process . . . . .	28
4.1	Digilent Nexys 4 FPGA. Courtesy[10] . . . . .	33
4.2	Data Overhead comparison Single 1687 Network case and Two 1687 Network case . . . . .	35
4.3	Comparison of data overhead between the hardware-based versus and the software-based solution . . . . .	37
4.4	Comparison of Area between the Hardware Solution versus the Software Solution . . . . .	38
A.1	Break down of UART transmission. Courtesy: Soliton Technologies[12] . . . . .	44
A.2	SPI Single Master - Multiple slaves architecture . . . . .	44

A.3 SPI bus timing. Courtesy:[14] . . . . . 45

---

## List of Tables

---

4.1	Experiment 1: Data overhead test results . . . . .	34
4.2	Experiment 2: Data overhead test results Hardware and software solutions . . . . .	36
4.3	Field-Programmable Gate Array (FPGA) resource utilization of the software-based solution . . . . .	37
4.4	FPGA resource utilization of the hardware-based solution . . . . .	38
A.1	SPI protocol modes definition . . . . .	45



---

## Listings

---

A.1	Algorithm for the iWrite PDL hardware-based . . . . .	47
A.2	Algorithm for the iRead PDL hardware-based . . . . .	48
A.3	Algorithm for the iRead PDL software-based . . . . .	49
A.4	Algorithm for the iApply command . . . . .	51





---

## List of Acronyms

---

<b>CLB</b>	Configurable Logic Block
<b>CPHA</b>	Clock Phase
<b>CPOL</b>	Clock Polarity
<b>CSU</b>	Capture-Shift-Update
<b>DTFSM</b>	Data Transfer FSM
<b>FF</b>	flipflop
<b>FPGA</b>	Field-Programmable Gate Array
<b>FSM</b>	Finite State Machine
<b>IC</b>	Integrated Circuit
<b>ICL</b>	Instrument Connectivity Language
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IJTAG</b>	Internal Joint Test Action Group
<b>ILM</b>	Instrument Length Memory
<b>LSB</b>	Least Significant Bit
<b>LUT</b>	Look-Up Table
<b>MISO</b>	Master In Slave Out
<b>MOSI</b>	Master Out Slave In
<b>ODU</b>	Output Discard Unit
<b>PDL</b>	Procedural Description Language
<b>RSFSM</b>	Read Status FSM

---

<b>SCLK</b>	Serial Clock
<b>SCR</b>	SIB Control Register
<b>SIB</b>	Segment Insertion Bit
<b>SPI</b>	Serial Peripheral Interface
<b>SS</b>	Slave Select
<b>TDI</b>	Test Data In
<b>TDO</b>	Test Data Out
<b>UART</b>	Universal Asynchronous Receiver Transmitter





# Introduction

---

## 1.1 Motivation

The development towards Integrated Circuits (ICs) with smaller, faster, and more transistors, has advantages as the possibility to implement more functionality at high performance and at smaller footprint. However, smaller and faster transistors lead to tighter margins. These tight margins make it a challenge to perform testing, tuning, and configuration.

Traditionally, external instrumentation has been used for testing, tuning, and configuration. There are two shortcomings with external instrumentation. First, due to tight margins, it is difficult to ensure that the on-chip signal is correctly transported to the external instrument. Second, new fault types, such as wear-outs, require that instrumentation is available also while the IC is in operation. To address these short-comings, on-chip instruments are increasingly used in modern ICs.

There is a need to access the on-chip instruments throughout the lifetime of the IC. As electronic systems are increasingly complex with more ICs and more advanced ICs, the access through the system hierarchies can be difficult as there might not be a direct access path and a single access protocol.

## 1.2 Thesis objectives

The overall objective of the thesis is to develop protocol, software support and hardware support to enable communication from system-level to instruments embedded in ICs through an access path consisting of several and different protocols and several ICs. In particular, the objectives are:

- Develop protocol to specify commands to perform communication.
- Develop software to generate access patterns according to the protocol.
- Develop hardware to handle communication including synchronization between ICs.

- Validate the developed solutions on an FPGA in terms of data overhead and area utilization.

### 1.3 Thesis organization

The thesis is organised as follows. The background material is presented in chapter 2. The proposed synchronization module, the communication protocol, and the hardware-based solution with interrupts and the software-based solution with polling are presented in chapter 3. In chapter 4, the experimental results where the proposed solution is implemented on an FPGA and data overhead and area utilization is evaluated on the BASTION benchmarks [1]. Conclusions and future work are in chapter 5.

This chapter contains background material. First, we describe a typical system with its components and connections. Then we discuss dedicated infrastructures. First IEEE Std. 1149.1 and then IEEE Std. 1687. Finally, we describe on-going and related work.

## 2.1 System overview

A typical system is shown in Figure 2.1. The system contains four ICs mounted on a PCB. The ICs contain functional logic and additional logic for testing, tuning, and configuration. This additional logic includes instruments, such as Memory Built-In Self-Test (MBIST) and temperature sensors, as well as infrastructures, like IEEE Std. 1687, to connect the instrument to enable access from the outside of the IC. The ICs are connected with different functional ports and protocols. For example, IC-3 is connected to IC-4 using Serial Peripheral Interface (SPI). A system may also have dedicated test infrastructure like IEEE Std. 1149.1 connecting ICs to enable testing, tuning, and configuration. For example, IC-1 is connected to the outside using IEEE Std. 1149.1. To operate on instruments, there is a need to transport data over several ICs and several protocols. For example, to write data to instrument 3 and read data from instrument 1 in IC-4, there is a need to transport data from the outside of the system via UART to access IC-3 and via SPI to IC-4 and then to the individual instruments in IC-4. There are several steps in this communication. Firstly, the user's commands to operate the instruments are retargeted to fit UART in IC-3 and then the user needs a dedicated path to transport the commands to operate the instruments through SPI in IC-4. Secondly, synchronization is needed between UART and SPI protocols in terms of frequency and baud rate to avoid data loss during information transaction. Reconfigurable Scan Networks (RSNs), like IEEE Std. 1687 networks, offers infrastructure to dynamically configure the targeted instruments in ICs. The flexibility to access arbitrary instruments is achieved by means of Segment Insertion Bits (SIBs), so that only the desired instruments are included active scan-path. IEEE Std. 1687 uses communication protocols vide description languages namely Instrument Connectivity Language(ICL) and Procedural



Description Language(PDL). ICL describes the scan paths and the scan registers that are associated with each scan path, whereas PDL uses the user's commands to perform read/write operations on the instruments. Finally, a software support is essential for interpretation of PDL commands into input vectors format, where interpretation is achieved using retargeting tool.

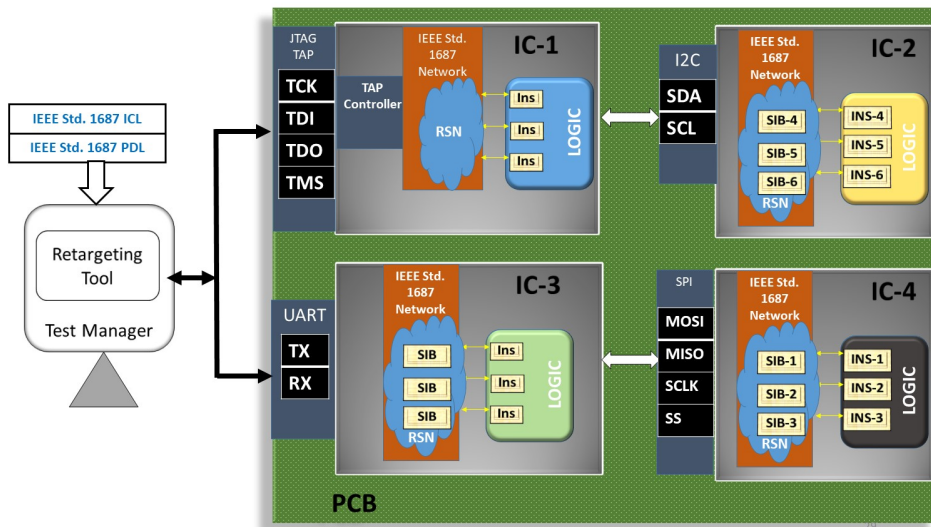


Figure 2.1: System overview

## 2.2 IEEE Std. 1149.1

IEEE Std. 1149.1, also known as Boundary Scan or Joint Test Access Group (JTAG), is a commonly used standard mainly developed to test interconnects between ICs. Figure 2.2 shows the conceptual view of IEEE Std. 1149.1 circuitry [2]. The main parts are the test access port (TAP), the TAP controller, the instruction register (IR), the IR decoder, and a number of test data registers (TDRs). The TAP includes four mandatory signals, namely test data input (TDI), test data output (TDO), test mode select (TMS), and test clock (TCK) which are the means through which the system is able to communicate to the on-chip circuitry. The signals TMS and TCK are used to control the TAP controller, which consists of two main part: the IR scan part to control which TDR to operate on and the DR scan part to control the operation on a selected TDR, see Figure 2.3.

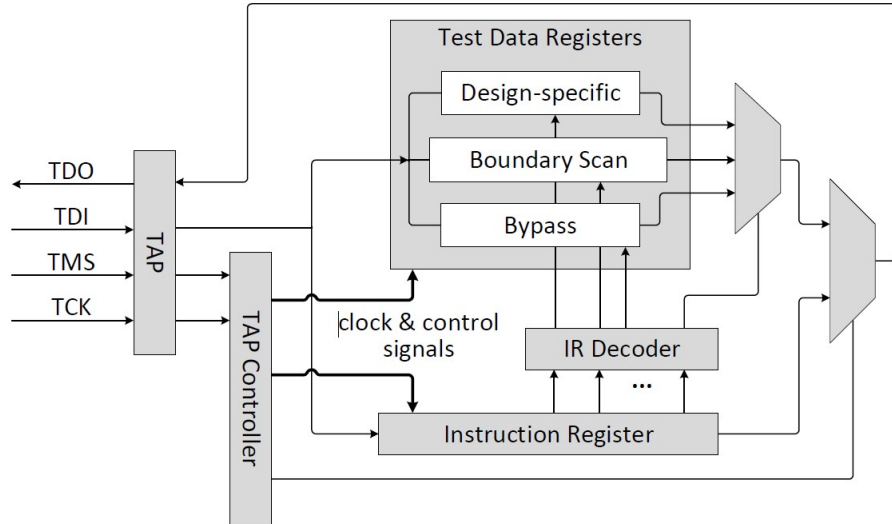


Figure 2.2: A conceptual view of IEEE Std. 1149.1 circuitry

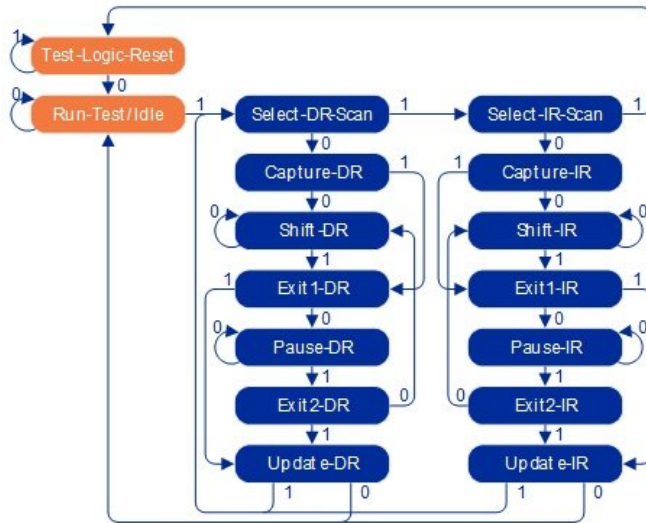
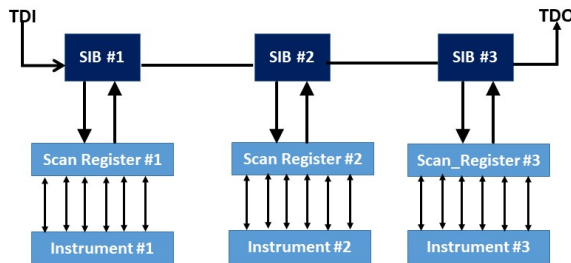


Figure 2.3: IEEE Std. 1149.1 TAP Controller state diagram. Courtesy: [3]

## 2.3 IEEE Std. 1687 (IJTAG)

IEEE Std. 1687, also known as internal JTAG (IJTAG), was developed to enable flexible and scalable access to on-chip instruments [4]. The key to enable flexible and scalable access to on-chip instruments is to allow for dynamic reconfiguration, for example, by the use of Segment Insertion Bits (SIBs). Figure 2.4 shows an example with three instruments using IEEE Std. 1687. The SIBs allow to dynamically include or exclude instruments; hence the length of the scan-chain changes depending on which instruments are included in the active scan-path. The detailed information about SIB is explained below.



**Figure 2.4:** Flat IEEE Std. 1687 network consists three instruments

### 2.3.1 SIB

A SIB is the key element of IJTAG standard. The combination consists of a two-input ScanMux and control bit referred to as one segment [5]. Selecting a particular SIB can activate the scan path and associated instruments in that portion of the scan path. Contrarily, de-selecting a SIB will deactivate the scan path and thus instruments on that segment are inaccessible. The SIB implementation is shown in Figure 2.5 which has selection Muxes, and two flip-flops namely shift and update denoted by 'S' and 'U' respectively.

The SIB is enabled/disabled by inserting the bit '1'/'0' into the 'S' flip-flop and latching the adjacent 'U' flip-flop. The 'S' and 'U' flip-flop operates at the rising edge of the CLK. When the SIBs are disabled the scan path is from Test Data In (TDI) to Test Data Out (TDO) and instruments are excluded from that segment by the scan H MUX. During the enable mode the scan path includes the instruments and shift register of the segment. The control sequence is defined by the state machine phases (shift phase, capture phase, update phase) and consequently, the

state machine generates the control signals (capture\_en, shift\_en, update\_en).

- During shift phase, when the control signal shift\_en='0' it remains in the current state through feedback from K1 MUX. When the control signal shift\_en='1', new value gets updated into 'S' flip-flop via H MUX and K1 MUX.
- During the update phase of the state machine, when the control signal update\_en='0', it retains the current state through feedback from K2 MUX. When the control signal update\_en='1', the value stored in the 'S' flip-flop is loaded into 'U' flip-flop via K2 MUX. In this case H MUX select pin being '1', input from FSO terminal will be multiplexed. The segment between TSI and FSO is also included in the scan path by the ToSel terminal. According to the SIB implementation[5], the control signals are active in some states and disabled in other states due to different phases of the state machine. The purpose of the capture\_en and shift\_en are used to enable parallel loading data between instruments on-chip and shift register and vice-versa.

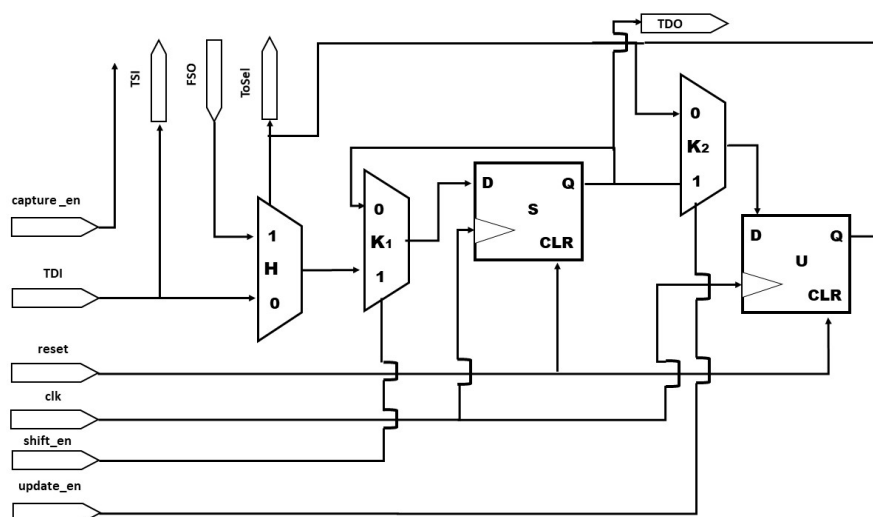


Figure 2.5: Implementation of SIB Schematic

### 2.3.2 Description language

There are two description languages introduced by the IEEE Std. 1687: Instrument Connectivity Language(ICL) and Procedural Description Language(PDL). ICL is used to describe the hardware architecture implemented on a chip, i.e., the way the instruments are positioned with different data lengths within the network. ICL is also used to describe the different scan paths by activating or deactivating the SIBs. For example, a possible ICL structure for IC-4 in Figure 2.1 is shown in Figure 2.6. The possible structure contains three SIBs to activate the scan

path and at scan registers, data is retargeted between serial and parallel format to access instruments. PDL defines the operations on the instruments within the network. There are two levels of PDL commands: Level-0 and Level-1. Level-1 PDL is used for more advanced designs, and level-0 PDL commands are commonly used to execute read and write operations on the instruments implemented on-chip. Figure 2.7 shows an example of PDL to operate on the instruments in the IEEE Std. 1687 network in IC-4 in Figure 2.1

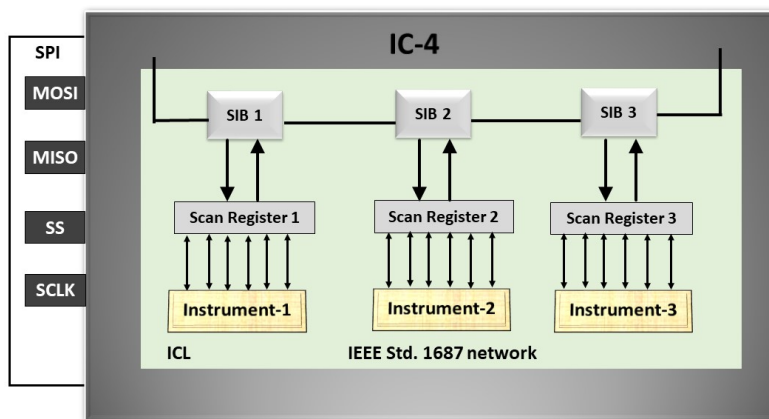


Figure 2.6: Conceptual schematic of ICL for IC-4

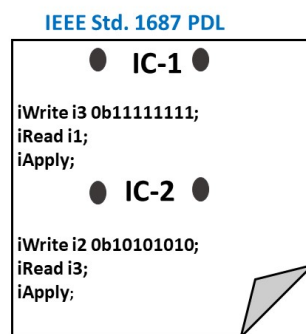
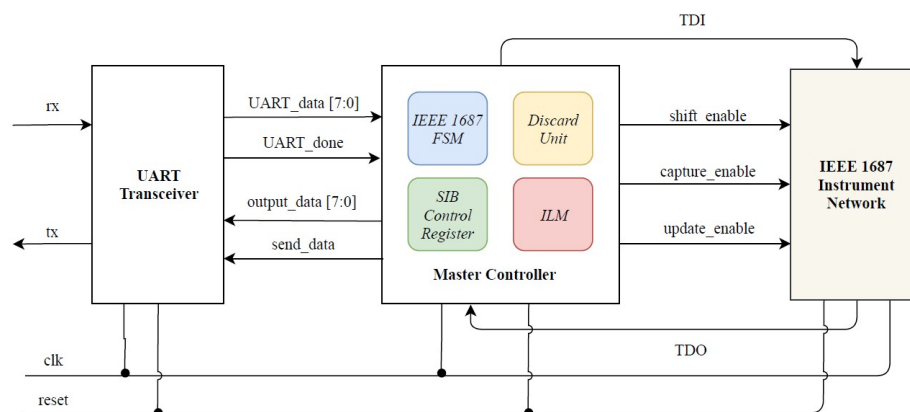


Figure 2.7: PDL example

## 2.4 Related work

A hardware component and protocol to operate on IEEE Std. 1687 over a functional protocol was developed by Larsson et al. [6]. The work is in-line with the on-going standardization initiative IEEE Std. P1687.1, which aims to make use of a functional port instead of the IEEE Std. 1149.1 port to access IEEE Std. 1687.



**Figure 2.8:** Full-featured case

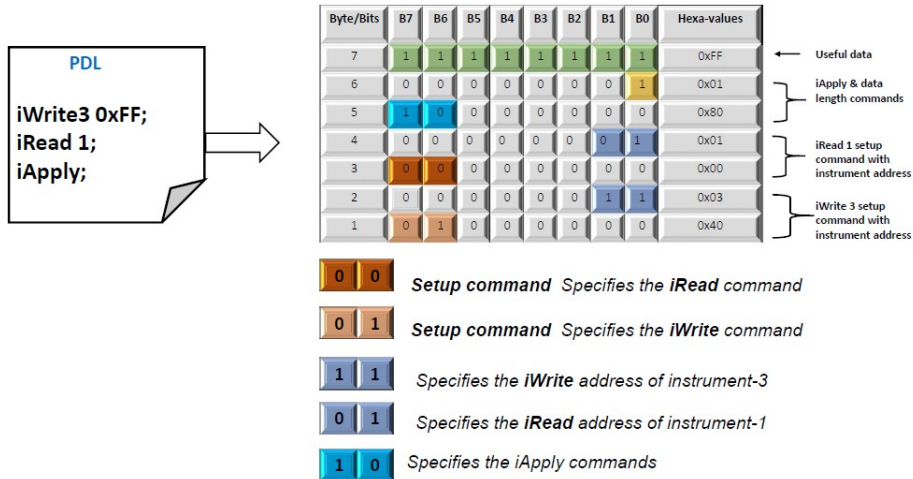
The developed component, shown in Figure 2.8, consists of the following blocks: SIB Control Register (SCR), Instrument Length Memory (ILM), Output Discard Unit (ODU) and Finite State Machine (FSM). The FSM serves as the interpreter of commands received from re-targeting tool and controls the shift, update and capture functions of the 1687 network. The FSM performs the duties of the TAP controller in IEEE Std. 1149.1. The SCR stores information about which instruments are part of the active scan path and what command is to be executed on them. The ILM holds information about the data lengths of the instruments in the network and the position of each instrument. The ODU removes bits outputted from the network which can be considered as garbage bits.

The protocol developed consists of two commands; setup and apply. The setup command is used to store values in SCR. An example is shown in Figure 2.9 where the PDL to write to instrument 3 and read from instrument 1 are translated into the proposed protocol and the result is 7 bytes of data. Each command consists of two bytes. The PDL in Figure 2.9 is translated into the following commands:

- The first command, byte 1 and 2, is a setup command to set that instrument 3 should be in the active scan-path.
- The second command, byte 3 and 4, is a setup command to set that instrument 1 should be in the active scan-path.
- The third command, byte 5 and 6, is an apply command that tells how

many bytes of data that follows. In this example, the command includes the value 1, which means that one byte of data follows.

- Byte 7 is the data that should be applied to instrument 3



**Figure 2.9:** Developed protocol

Larsson et al. have implemented proposed solution on an FPGA and use Universal Asynchronous Receiver-Transmitter (UART) as the functional protocol to communicate with the IEEE Std. 1687 network.

#### 2.4.1 Data overhead classification

The UART controller components explained in the above section such as SCR, ILM, and ODU assist the controller with interpreting the protocol and generate input vectors to be shifted into the IEEE Std. 1687 network [7]. One of their project objectives was to calculate the impact of data overhead of these components. Based on that the overall data overheads are divided as:

- SIB overhead required to configure the SIBs. The setup commands with instrument addresses fall under this category. In Figure 2.9 the SIB overhead is 32 bits. The first two bytes (byte 1 & 2) are for write operation with instrument address and the next two bytes (byte 3 & 4) are for read operation.
- PDL overhead, which specifies the overhead of action commands that are transmitted via UART channel to trigger the controller FSM. In Figure 2.9 the PDL overhead is 16 bits in which 1 byte is used to indicate data command and another byte for specifying the total number of data bytes into instruments.

- Output overhead - which specifies overhead generated by the network during various Capture-Shift-Update (CSU) cycles. For instance, output bits produced while activating instruments. These overhead are discarded by the controller components called ODU.
- Useful bits - Actual data bits that are shifted-in and shifted-out of the instruments. In Figure 2.9 the useful data is indicated in byte 7.

## 2.5 Communication technique

In order to handle communication between ICs in the system shown in Figure 2.1, synchronization is necessary because of possible loss of data due to the difference in data transfer rate or frequency. In hardware-based communication, synchronization is handled based on the interrupt-driven manner and the software-based with polling [8].

- In hardware-based interrupt driven, an interrupt signal sent from an external device or hardware to communicate with the processor indicating that interrupt signal requires immediate attention.
- Software polling is the process where the computer or controlling device waits for an external device to check for its readiness or state.





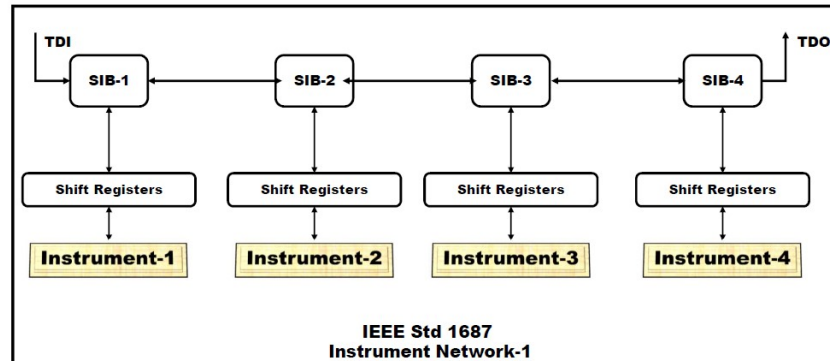
---

## Proposed solutions

---

In this chapter we present the synchronization module enabling communication between two IEEE Std. 1687 networks connected with a functional protocol and a software-based and a hardware-based solution to perform the communication.

### 3.1 Synchronization module



**Figure 3.1:** Synchronization module overview

The synchronization module consists of two parts; one for sending data and one for receiving data. These parts are formed as IEEE Std. 1687 compliant instruments. Figure 3.1 shows the instruments and the internal IEEE Std. 1687 architecture.

#### 3.1.1 Sending data

The part for sending consists of an instrument of length 9 bits where 8 bits are for the data that is to be sent and one control bit to inform the SPI Master when

ending can be performed. To send data, the path is first set up using a setup command of two bytes to include the instrument in the active scan-path.

### 3.1.2 Receiving data

The part for receiving data consists of three instruments as follows:

- Instrument 2 is 8-bit register holding arrived data.
- Instrument 3 is a single flag bit register activated during read operations. The flag is set when data has arrived completely in instrument 2.
- Instrument 4 is a single flag bit to acknowledge the SPI controller when data in instrument 2 has been read (consumed).

## 3.2 Hardware-based solution

In this section we present hardware-based solution which is built upon an hardware interrupt-driven approach to handle communication between two ICs. The communication between the ICs is performed with SPI and the communication with the outside world with UART. As mentioned in chapter 2 Larsson et al. [6] have implemented solution using UART as the functional protocol to communicate with the IEEE Std. 1687 network. In this thesis we inherited the hardware components shown in Figure 2.8 to design IC-1 and IC-2 shown in Figure 3.2. As a result we developed hardware-based solution module which includes synchronization module shown in Figure 3.1 to handle communication between two ICs. Also to enable access path to transport information from the test manager to IEEE Std. 1687 network-2 new hardware components are introduced namely Data Transfer FSM (DTFSM) and Read Status FSM (RSFSM).

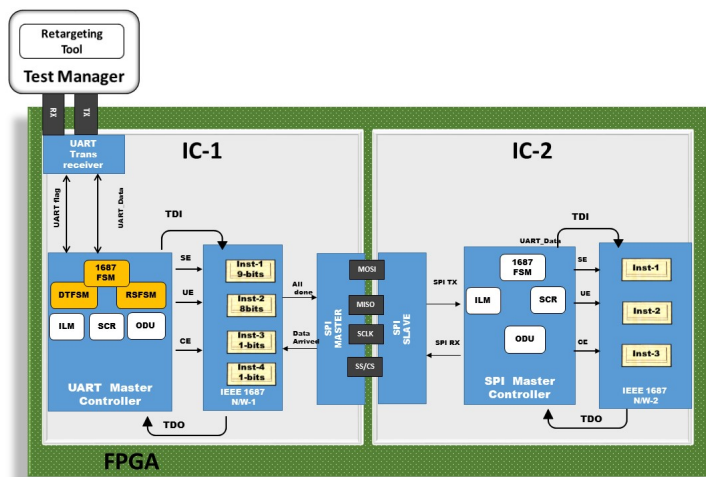


Figure 3.2: Architecture of hardware-based solution

### 3.2.1 UART master controller

In the IC-1 the main hardware block is the UART master controller which includes sub-hardware components - 1687 FSM is extended with more control execution flow using DTFSM and RSFSM. The brief discussion of this two components is explained in below subsections, whereas the other hardware components like SCR, ILM, and ODU plays important roles during synchronization process. The SCR stores information about the instrument 1 during write operation, whereas information about instrument 2, and instrument 3 are stored during read operation. The ILM holds the information about data lengths and positions. The data length of each instruments within the IEEE Std. 1687 network 1 is indicated in Figure 3.2. ODU discards activation bits (1's and 0's) used to activate SIBs of network 1 and also bytes in instrument 1 during new arrival of bytes during data transfer process to next stages. The IEEE Std.1687 network 1 includes the hardware synchronization module. Here instrument are implemented as simple shift register and their access and control methods are described by the IEEE Std.1687 description languages.

#### 3.2.1.1 DTFSM

DTFSM is active during the data transfer phase; receives the commands from the re-targeting tool and configures the SIB-1 then transfers data/commands to the subsequent stages. This is achieved by activating the instrument 1 in the network-1. As observed in the Figure 3.2, the instrument 1 is a 9-bit shift register which transports 8 bits of data and an extra bit, i.e., the 9th bit acts as a flag signal for the SPI master to receive the commands as data from re-targeting tool to convey to the network-2.

#### 3.2.1.2 RSFSM

RSFSM enables the synchronization during the read data operation between the the two ICs and it is only active during the read operations. The RSFSM controls read operation by monitoring the status of instrument 2, 3 and 4 of the network-1. It transfers the read data from the network-2 by polling instrument 3 in network-1. The monitoring process of instruments during RSFSM process are summarized as follows:

- Instrument 2 is an 8-bit shift register. It is activated during read, stores the incoming read data from instruments in network-2. The read data will be shifted from this segment to the re-targeting tool.
- Instrument 3 is a single bit register, activated during the read operation. This instrument is read at specific intervals during the read operation, i.e., when it gets updated from 0 to 1. This implies that the read data arrives at the SPI Master. In the same clock cycle, the data is sent to the instrument 2 and the FSM sends signal back to the re-targeting tool so that it sends the setup and action commands required in order to extract the data from the instrument 2.

- Instrument 4 is a single bit register, activated during the read operation. It is updated by the re-targeting tool by writing 1 to it, when the read data is received at the re-targeting tool from instrument 2.

### 3.2.2 SPI system

To integrate IC-1 and IC-2 and also to transfer the data from test manager to network-2 we implemented SPI communication protocol as hardware component. We used mode '0' (see Table A.1 in Appendix 1 for the details) to sample data between two ICs. In mode '0' data is sampled at the rising edge of the clock. In order to match baud rate of UART protocol a clock divider logic is implemented in this system. The SPI protocol with four terminals Master Out Slave In (MOSI), Master In Slave Out (MISO), Serial Clock (SCLK), and Slave Select (SS) are shown in Figure 3.2. Additional details of SPI interface can be found in Appendix 1.

### 3.2.3 SPI master controller

To design the SPI master controller we used a hardware structural model shown in Figure 2.8. The hardware components are 1687 FSM, SCR, ILM, and ODU. The basic behaviour of each components are same and explained in the background chapter in related work [6] section. However input and output ports are changed w.r.t SPI system. Here SPI slave receives the input vectors from the SPI master and send them to the SPI controller using the input and output ports SPI\_TX and SPI\_RX. An example of input vectors passing through input and output ports is represented Figure 3.5. In particular the bytes/bits from 5<sup>th</sup> to 9<sup>th</sup>. The IEEE Std. 1687 network-2 consists of instruments which are simple inverters with variable data lengths. The data lengths of the inverters vary between 8 bits, 16 bits or 32 bits. The network architecture designed based on flat IEEE Std. 1687 network architecture (see Figure 2.4).

### 3.2.4 Hardware description of the solution

Previously, we introduced the newly implemented hardware components which enable communication between two ICs in sub-sections 3.2.1.1 and 3.2.1.2. Let us clarify their functionality in detail in this section.

The DTFSM is implemented to configure instrument 1 within network-1 and it shifts in commands-as-data to the next stages. This process is essential according to the new protocol which has been explained in the below section. The states in this FSM as shown in the Figure 3.3, it starts with the configuration phase (left portion) i.e., activating the SIB-1 and connecting the instrument 1 into the active scan path. Then it is followed by the data transfer phase( right portion ), where the data is sent from the retargeting tool to the IC-2 instrument network-2.

During the configuration phase the FSM transitions through IDLE, SHIFT\_CONTROL, UPDATE\_CONTROL and CAPTURE. The first state transition from IDLE to SHIFT\_CONTROL occurs when the control\_ready signal value becomes '1',

along with the arrival of iApply-"0x80" action command (see Figure 3.5). To indicate the FSM that the setup commands are available at the SCR. The bits stored in the SCR will be shifted into the network-1 during the SHIFT\_CONTROL state. The shift-in sequence contains four bits configured as "0x40"(1000) which activates the SIB-1 in network-1 while deactivating the SIB-2, SIB-3, and SIB-4. During UPDATE\_CONTROL state the SIB-1 is activated and pulls the corresponding shift registers into the active scan path. Here the signal control\_en is used to check the number of data bytes required for the network-2. Finally, in the CAPTURE state, the expected read data and number of write bytes are captured and the state machine returns to the IDLE state, where it waits for the data to be transmitted by the retargeting tool.

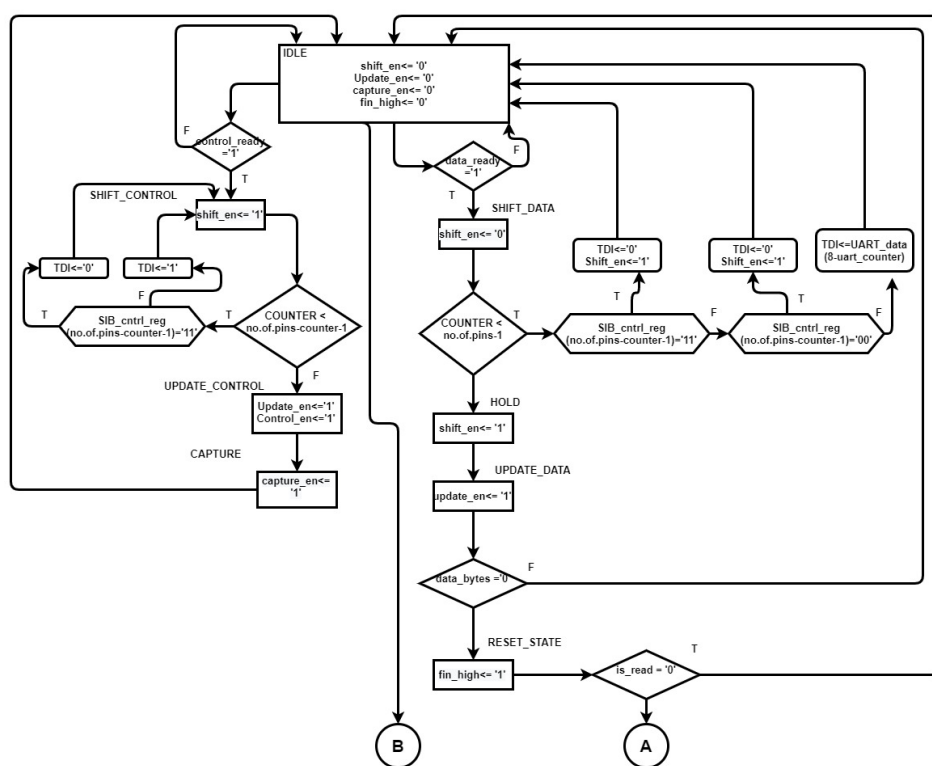


Figure 3.3: ASMD of DTFSM

During the data transfer phase the FSM transitions through the IDLE, SHIFT\_DATA, and UPDATE\_DATA states. The IDLE to SHIFT\_DATA transitions occur when the data\_ready control signal value becomes '1'. In the SHIFT\_DATA state, prior to the shift-in operation, the logic ensures which of the SIBs are active and which operations does the setup commands imply. In the case of iWrite, the useful data gets shifted into the network, while for the iRead, dummy bits (zeros) are shifted-in to extract the useful read data. The FSM then transitions to the UPDATE\_DATA state where the state machine loads the data from the shift register to the active

instruments which are available in the active scan path. The FSM finally reaches the reset state where the data\_bytes signal value becomes '0' indicating that data is transferred successfully. After that, the counters and registers are reset and the hardware is ready to begin a new operation.

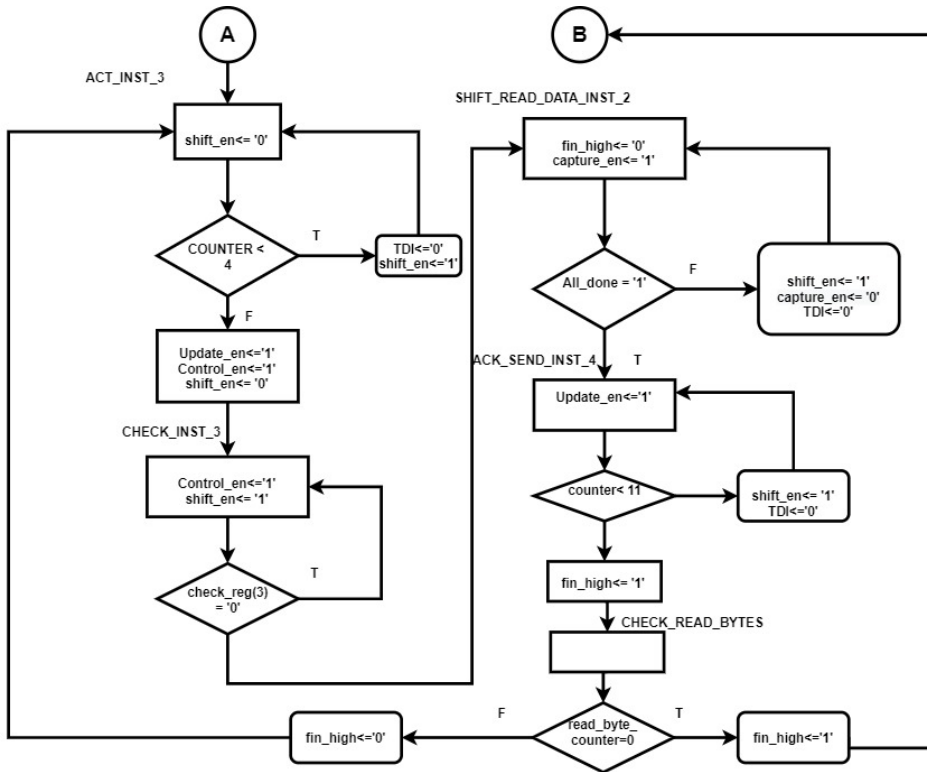


Figure 3.4: ASMD of RSFSM

The purpose of RSFSM, shown in Figure 3.4 is to receive the read data from IC-2 and send it to the system manager during read operations. The read operation is performed when the control signal is\_read value becomes '1'. It begins from the ACT\_INST\_3 state where it configures the instrument 3 within the instrument network-1 to begin the polling process. In CHECK\_INST\_3 state, it continuously reads the status of the instrument 3 (polling), and if the read status becomes '1', the state transitions to SHIFT\_READ\_DATA\_INST\_2 state to extract the read data from the instrument 2; whereas in case the read status is '0', the state doesn't transition. After a byte of read data is extracted, the FSM transitions to ACK\_SEND\_INST\_4 state where an acknowledge signal is generated and sent to the SPI controller to send the next byte of read data. Finally, in the CHECK\_READ\_BYTES state checks if there is more read data yet to arrive compared to the expected (employs read\_byte\_counter), the FSM repeats the process from the ACT\_INST\_3 state; otherwise, it will return to node 'B' i.e., the IDLE state.

### 3.2.5 Retargeting flow for the hardware-based solution

In this thesis the retargeting tool algorithms are built upon the functions defined in previous work [7]. Additionally, a new function 'activate\_network' is added to establish the communication channel between the test manager and IC\_2. This function configures instrument 1 in network-1 using iWrite1 setup and iApply. Depending on the total number of read/write operations being performed on the instruments within network-2, the total bytes are sent to the instrument 1 as write data, which is alternatively known as commands-as-data. In Figure 3.5 4<sup>th</sup> byte is indicating the total number of bytes sent as commands-as-data.

For the write setup commands, the PDL translated into a stream of bits as shown in the Figure 3.5. The bytes 1<sup>st</sup> to 4<sup>th</sup> constitutes the first set of PDL commands which activate instrument 1 within network-1. Out of the above, its initial 2-bytes specify the write operation where the first 2 bits ("01") indicate write setup command and the remaining 14 bits specify the instrument address, for instance 0x"0001" indicates instrument 1 being addressed. The next 2-bytes hold the iApply command along with the total number of setup commands being sent to the network-2. In this example it is indicated in the 4<sup>th</sup> byte as "0000 0101" meaning 5 bytes in total is required to transfer.

Based on the above example, the second set of PDL commands will constitute 5 bytes representing the write operation onto instrument 3 within network-2. It follows the same write command principle as described above but changes according to the target instrument address and write data. Bytes 5<sup>th</sup> and 6<sup>th</sup> are correspondingly used to set the write operation and the instrument address. Further, the bytes 7<sup>th</sup> and 8<sup>th</sup> correspondingly represent the action command and the total write data size. Finally, the 9<sup>th</sup> byte holds the actual instrument data ("0x55") which gets shifted in to instrument 3. This is because, we have considered the instrument length memory to be 1-byte.



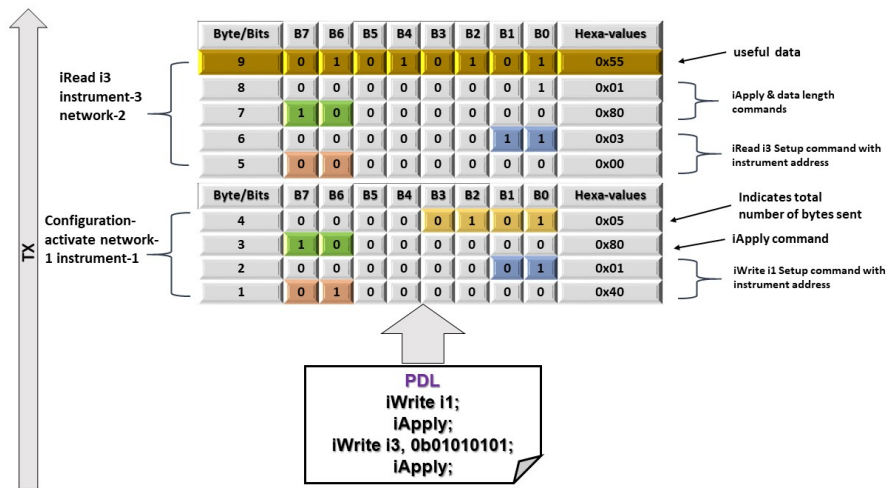
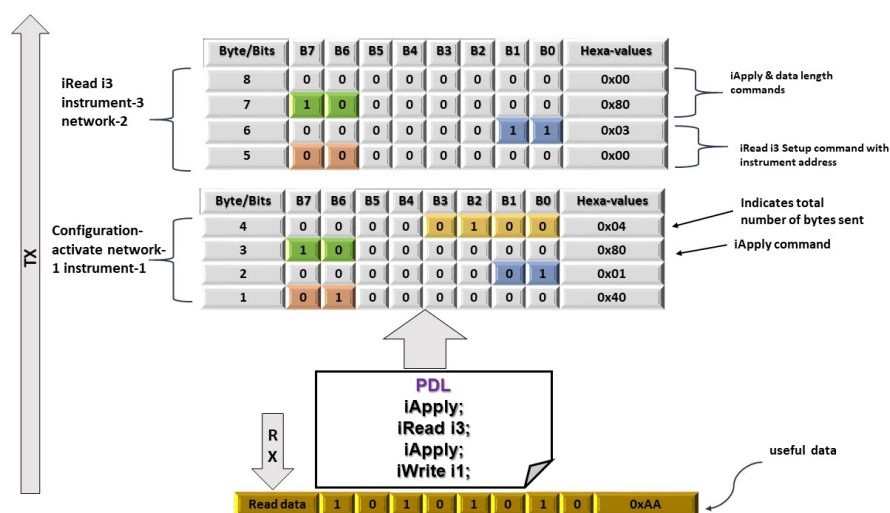


Figure 3.5: Bits generation by retargeting tool of iWrite PDL

The Figure 3.6 shows an example of iRead PDL translated into bytes/bits input vector stream. As per our new protocol initial configuration principle from bytes 1<sup>st</sup> to 4<sup>th</sup> explained in the iWrite algorithm example applies here also; with an exception that, instead of shifting-in the instrument write data, the FSM shifts out the read data from the instruments. Look closely, that the initial 3 bytes are exactly the same as the previous example, but the 4<sup>th</sup> byte varies depending upon total number of operations performed on instruments within network-2. Here, only one read operation is performed to instrument 3, so the total number of bytes sent to network-2 is set as 4 bytes. During the read operation, data is expected from the instruments within the instrument network-2, bytes 5 and 6 correspondingly represent the iRead setup command; their starting 2 bits are set to "00" which indicate the read operation of instrument 3. Additionally bytes 7<sup>th</sup> and 8<sup>th</sup> represent the iApply commands. It is interesting to note that the Least Significant Bits (LSBs) of the 8<sup>th</sup> byte are set to 0x"0" to tell the FSM that no data is being shifted in to the instrument 3.



**Figure 3.6:** Bits generation by retargeting tool of iRead PDL

Finally, as a result, useful data (0x"AA") is obtained successfully in the read operation.

### 3.3 Software-based solution

In this solution, our goal was to analyse the data consumption and area utilization if the communication is enabled through the software approaches. The hardware mechanism through interrupts are replaced by the software polling approaches. To achieve this facility we removed the hardware components in UART master controller, in particular DTFSM and RSFSM. We developed polling-based communication technique using the software functions called polling and check-up functions. These two functions are operated externally inside the test manager. In Figure 3.7 this difference is clearly observed in IC-1 UART master controller main block and test manager. The 1687 FSM in IC-1 enables the instrument 1 to transport data from the re-targeting tool to the subsequent stages. The hardware synchronization module remains unchanged, however operations to control the synchronization module is implemented in a polling-based manner where the re-targeting tool in test manager monitors if data is ready to be read in instrument 2 by polling instrument 3.

- The PDL to perform polling of instrument 3 is:

```
iRead instrument3:
iApply;
```

- The PDL to perform read data from instrument 2 is:

```
iRead instrument2:
iApply;
```

For example, in Figure 3.11 the bytes from 1<sup>st</sup> to 8<sup>th</sup> translates to one setup command of 2 bytes and one action command of 2 bytes. When data in instrument 2 has been read, the UART controller sets the flag into instrument 4 to inform the SPI controller that data has been read.

#### 3.3.1 IC-2

IC-2 hardware structural model is the same shown in Figure 3.2. The hardware components are 1687 FSM, SCR, ILM, and ODU. Basic behaviour of each of the components are same as explained in the hardware-based solution. The SPI slave receives the input vectors from the SPI master and sends them to the SPI controller using the input and output ports SPI\_TX and SPI\_RX. An example of input vectors passing through input and output ports is represented in Figures 3.10 and 3.11. Here also the IEEE Std. 1687 network-2 consists of instruments which are simple inverters with variable data lengths. The data lengths of the inverters vary between 8 bits, 16 bits or 32 bits. The network architecture is designed based on flat IEEE Std. 1687 network architecture (see Figure 2.4).

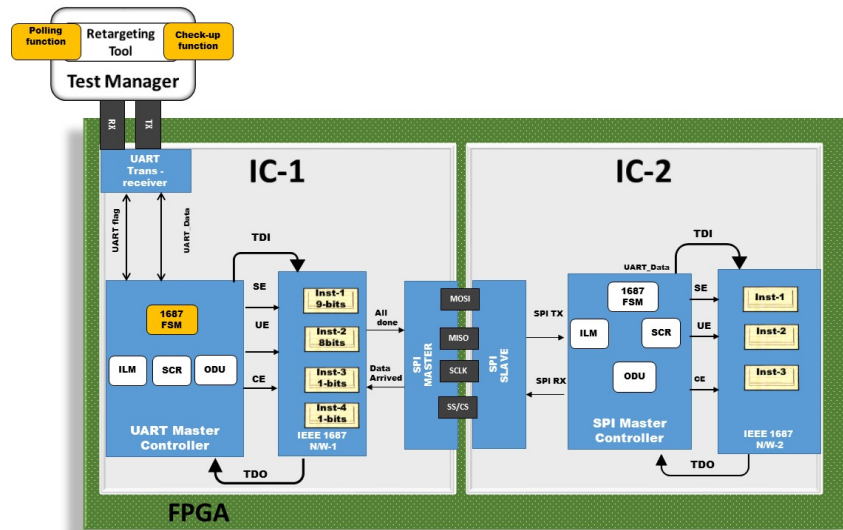


Figure 3.7: Architecture of software-based solution

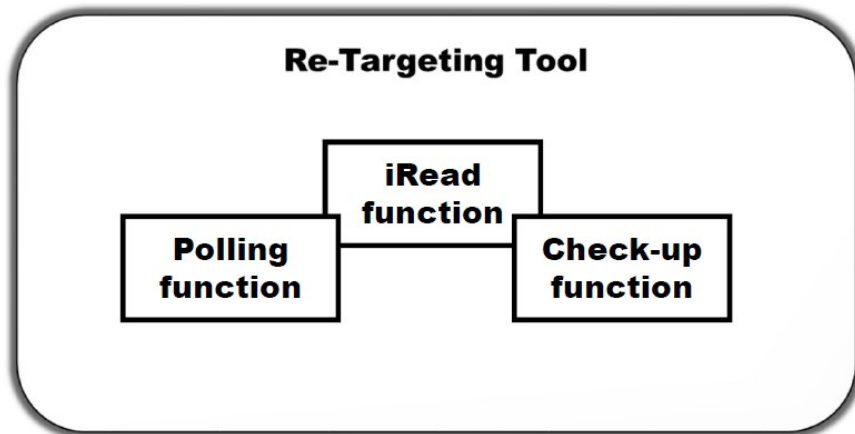


Figure 3.8: Re-Targeting tool functions

### 3.3.2 Polling function

This function is called inside the iRead function. This function starts polling continuously, by sending the iRead commands to the instrument 3 (1 bit) of IEEE Std 1687 network-1 at every 0.1ms delay to capture its status. If the instrument 3 captures a '0', it indicates that no read data has arrived at the SPI master whereas when it captures a '1', it indicates that the read data is available at SPI master.

Simultaneously, the SPI master will shift the read data to the instrument 2 within the network-1, after which, the UART master controller will perform the padding along with the captured 1bit from instrument 3 with seven '0's (zeros). It is then sent back to the re-targeting tool as '0x80' since the UART can only accept 8 bits of data as default. The '0x80' acts as a flag signal to the iRead function inside the re-targeting tool, which later sends the read command to extract the actual data from the instrument 2.

### 3.3.3 Check-up function

The check-up function holds the expected read bytes, based on the apply group, which were sent by the re-targeting tool, to perform the read operation to the instruments in IEEE Std 1687 network-2. The counter used in the re-targeting tool helps to keep track of the read data while receiving, i.e, whenever a byte of read data arrived counter will be decremented. If the counter value is greater than '0', the polling function will be called again, which will generate the necessary iRead setup and action procedure using the iRead function. If the counter value is equal to '0', it would signify that all the expected read data has fully arrived at the re-targeting tool.

To complete the read process the 1687 FSM will be actively involved by writing '1' to the instrument 4 within the instrument network-1. This '1' is send as the acknowledgment flag signal to the SPI controller in order to convey that, all the read data has been received at the network-1 and transferred back to the re-targeting tool; following the completion of which, it is ready to accept a new set of read data. This scenario emerges during multiple reads.

### 3.3.4 Hardware description of the solution

The 1687 FSM performs operation of the DTFSM which has explained in the hardware solution. The RSFSM operations are removed and developed as software algorithms to check the feasibility of the polling process. However sending the acknowledgement bit to the SPI controller through instrument 4 in network-1 is done by this state machine the ASMD diagram is as shown in Figure 3.9.

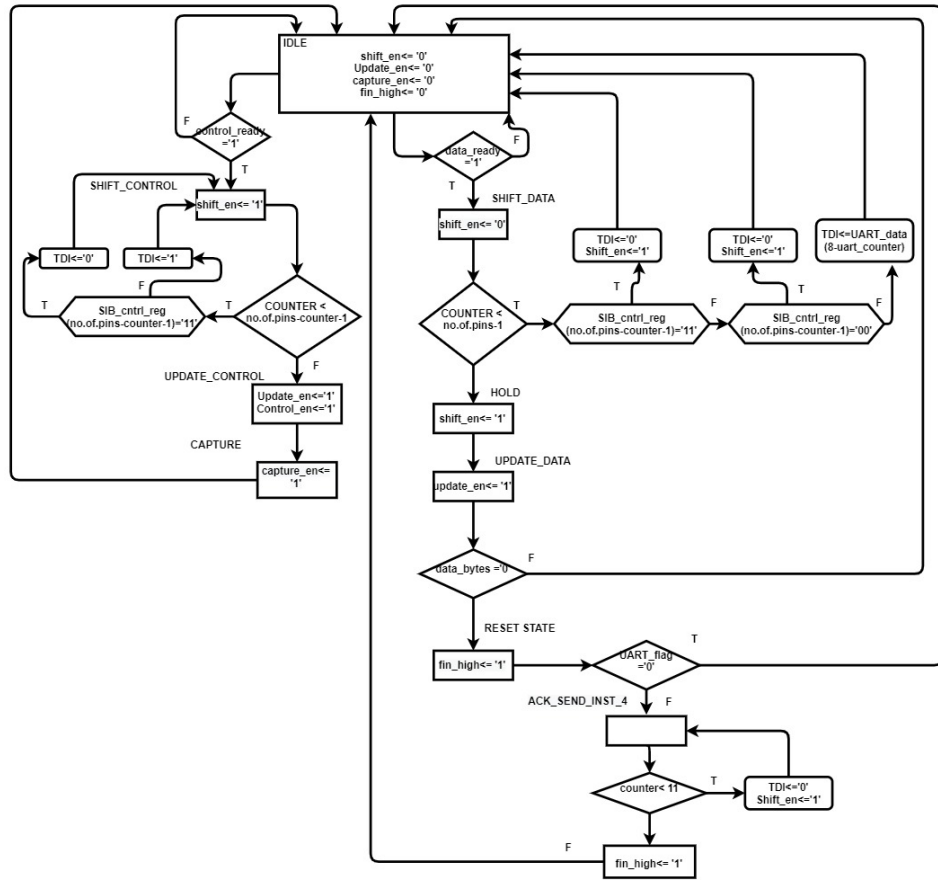
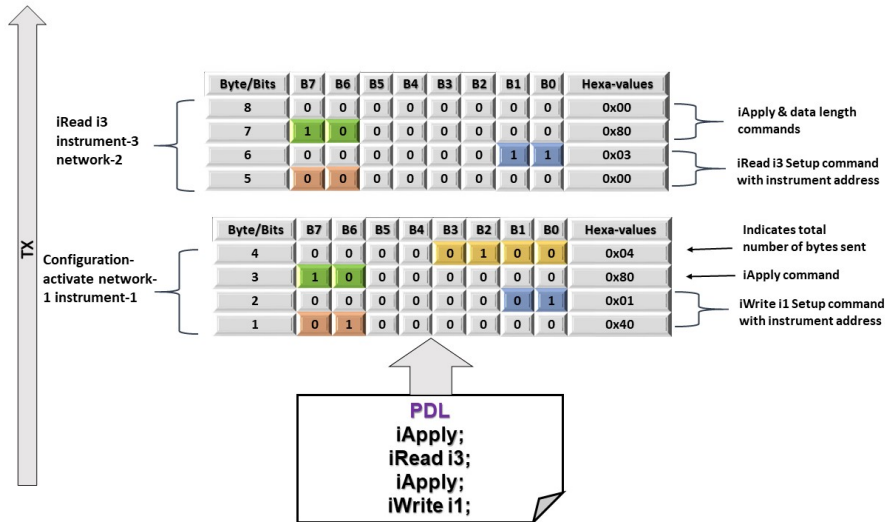


Figure 3.9: ASMD of the Software Solution

### 3.3.5 Retargeting flow for the software-based solution

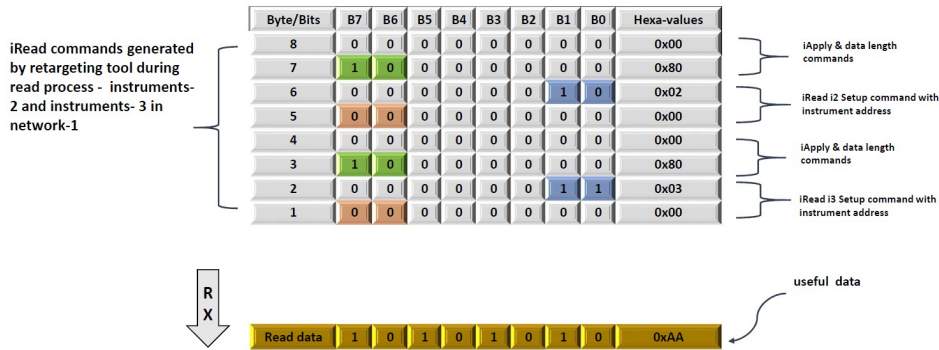
In comparison to the previous retargeting flow of the hardware-based solution, the input vector of iWrite operation is exactly same (see Figure 3.5). Whereas, in the software-based iRead operations adapts new sub-functions inside the iRead main function viz., the polling and the check-up functions to facilitate software-polling on the instrument 3 and to shift the read data from the instrument 2 within in the instrument network-2 when read data is ready. As mentioned previously, the read functions will anticipate the read data and expects it to be the same as received from the instruments.

For the read operation, a stream of bits generated by the retargeting tool is shown in the Figure 3.10. Again, the first set of four bytes from 1<sup>st</sup> to 4<sup>th</sup> set the communication path between IC-1 and IC-2. Also, the total number of read bytes required to access the instruments in network-2 is indicated in 4<sup>th</sup> byte. Here Read operation, targeted on instrument 3 in network-2, PDL iRead i3; iApply; translated into bits stream is indicated from bytes 5<sup>th</sup> to 8<sup>th</sup>.



**Figure 3.10:** Bits generation by retargeting tool of iRead PDL

The bits generated during the iRead function for the single read iteration is as shown in the Figure 3.11. This process is fully automated inside the retargeting tool and user is relieved to ignore the process. The first set of 4 bytes from 1<sup>st</sup> to 4<sup>th</sup> are sent to the instrument 3 to begin the polling process. Once the read data arrives at the SPI master, the next 4 bytes from 5<sup>th</sup> to 8<sup>th</sup> are sent to instrument 2 to extract the read data.



**Figure 3.11:** Bits generation by retargeting tool of iRead- polling and check-up process

---

The iApply command is translated into stream of bits during all types of read and write operations. As an example look closely into the Figures 3.5, and 3.6 it is indicates as iApply commands every 3<sup>rd</sup> byte. Arrival of these commands trigger the FSMs inside the controllers to begin their execution. Its function is to shift-in the commands (either read or write) and the instrument data into the corresponding instruments. In Appendix the retargeting tool algorithms which generates the input vector for the PDL are presented.





---

## Experiments and Results

---

The objective of the experiments is to evaluate the data overhead and the area utilization, first in respect to the length of the access path and second comparing the hardware with interrupt and the software solution with polling. The data overhead can be divided into following categories:

- SIB overhead is the data needed to configure the SIBs.
  - In the case of the work by Larsson et al.[6] where a single protocol is used, the SIB overhead is the bits needed for the setup. In the example in Figure 3.5 where 2 SIBs are activated, there is a need of one setup command for each SIB. As each setup command requires two bytes, the SIB overhead in this example to activate two SIBs is 32 bits (2 commands of 2 byte each).
  - In the case of two protocols, additional data is needed to setup the path. The additional data needed is per PDL group. The setup of the path corresponds to one setup command and one action command. Each command corresponds to 2 bytes, which lead to an additional 32 bits per PDL group.
- PDL overhead is the data needed to operate on an active scan path. In the example in Figure 3.5 two action commands are needed. First action command for activating scan path in network-1 within IC-1 and second action command is for activating scan path in network-2 within IC-2. The PDL overhead in this example to activate two active scan path is 32 bits (two commands of 2 byte each).
- Output overhead is the unwanted data output through the instrument 3, padded with 7 zeros. This serves as the acknowledgement signal to the retargeting tool to send the read setup commands to extract read data from instrument 2. The padding is necessary as UART can only accommodate 8 bits for data transaction.
- Total overhead - Sum of all the above mentioned data overheads produced during instruments access from system manager.

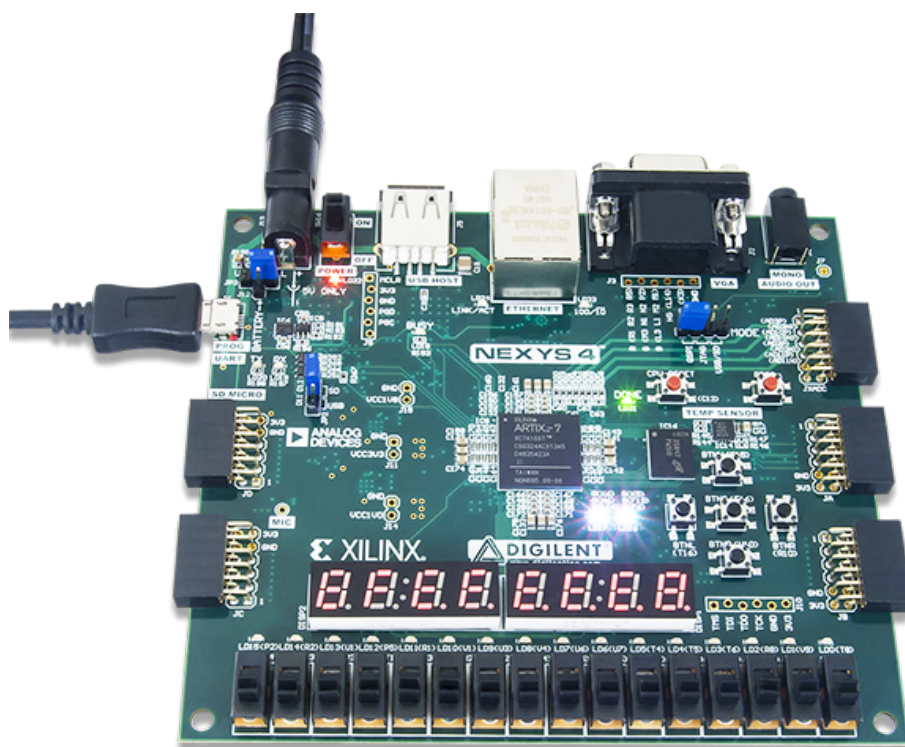
- Useful bits - Actual data bits that are shifted-in and shifted-out of the instruments. The data shifted-in is during a write operation, and data shifted-out is during a read operation.

The area utilization is given by the number of Configurable Logic Blocks (CLBs) which is computed as a function of Look-Up Tables (LUTs) and flipflops (FFs):

$$\text{Number of CLB utilized} = (LUT \div 8) + (FF \div 16) \quad (4.1)$$

We have on an Xilinx Nexys-4 FPGA implemented the proposed solution in a system with two ICs connected where SPI is used as the connection between the ICs and UART is used to connect with the test manager outside, see Figure 3.2. For the experiments, we have made use of the TreeFlat BASTION benchmark [9], which we implemented in IC-2. We use of three versions of the benchmark, 50, 100 and 150 instruments and we have used instrument length of 8, 16, and 32 bits. That is, instrument 1 is of length 8 bits, instrument 2 is of length 16 bits, and instrument 3 is of length 32, and instrument 4 is of length 8, and so on. We have used five versions of PDL for these benchmarks:

- Write to instrument 1.
- Read from instrument 1.
- Write to all instruments.
- Read from all instruments.
- BASTION where we write to instrument 1, read from instrument 1, write to instrument 2, read from instrument 2, and so on and finally we write to all instrument and then we read from all instrument. In the case of 50 instruments, there are 102 iApply groups.



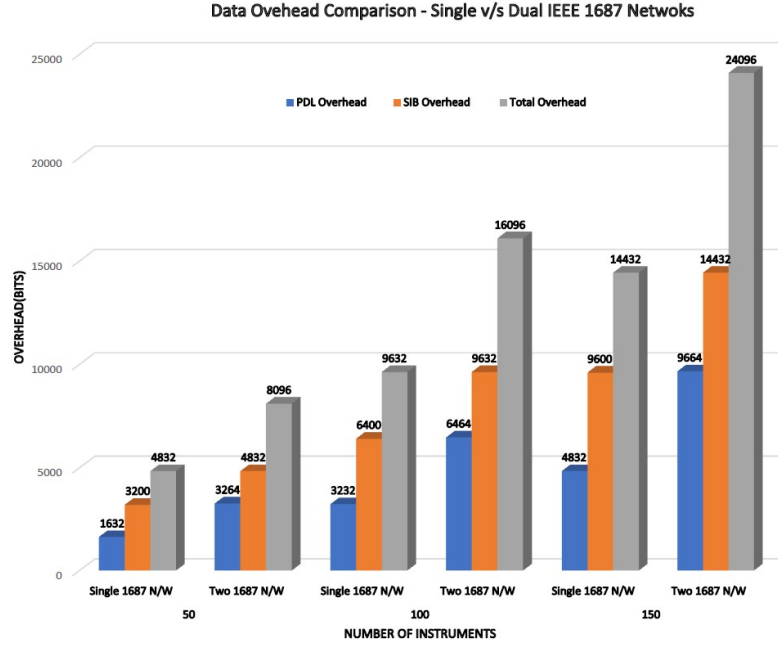
**Figure 4.1:** Digilent Nexys 4 FPGA. Courtesy[10]

#### 4.1 Results of experiment-1

In the first experiment, we investigate the impact of the length of the access path. We compare our solution where we have an access path of two protocols against the access path of one protocol used by Larsson [6]. The results are presented in Table 4.1 and in Figure 4.2. Column 1 shows the number of instruments in the three designs, 50, 100, and 150. Column 2 shows the type of iApply group. The following columns report the overhead, PDL, SIB, Output, and total. The rightmost columns report the useful data and the relation of useful data compared to the total number of bits. For example, the iWrite 1 – write data to instrument 1, which is of length 8 bits, requires an active scan path where SIB 1 is active. To setup an active scan path with one SIB, one setup command of size 2 bytes is needed. Hence, the SIB overhead is 16 bits. To operate – write data to instrument 1 – one action command of size 2 bytes is needed. Hence, the PDL overhead is 16 bits. The number of useful bits is the data written to instrument 1, which is 8 bits. Figure 4.2 shows for the BASTION benchmarks the PDL overhead, SIB overhead and total overhead for the Single protocol alternative and the Two protocol alternative for the three designs.

No. of Instruments	iAppy Group	PDL Overhead		PDL overhead (%)	SIB Overhead		SIB overhead (%)	Output Overhead both network	Total Overhead		Total overhead (%)	No useful bits	Useful data (%)	
		Two network SPI	Single network UART		Two network SPI	Single network UART			Two network SPI	Single network UART				
50	iWrite 1	32	16	100 %	32	16	100 %	0	64	32	33.33%	8	11.10%	20%
	iRead 1	32	16	100 %	32	16	100 %	0	64	32	33.33%	8	11.10%	20%
	Write all	32	16	100 %	816	800	2 %	0	848	816	3.92%	920	52%	52%
	Read all	32	16	100 %	816	800	2 %	0	848	816	3.92%	920	52%	53%
	BASTION	3264	1632	100 %	4832	3200	51%	0	8096	4832	67.54%	3680	31.2%	43%
100	iWrite 1	32	16	100 %	32	16	100 %	0	64	32	100 %	8	11.10%	20%
	iRead 1	32	16	100 %	32	16	100 %	0	64	32	100 %	8	11.10%	20%
	Write all	32	16	100 %	1616	1600	1%	0	1648	1616	1.98%	1856	53%	53%
	Read all	32	16	100 %	1616	1600	1%	0	1648	1616	1.98%	1856	53%	53%
	BASTION	6464	3232	100 %	9632	6400	50.50%	0	16096	9632	67.10%	7424	31.60%	44%
150	iWrite 1	32	16	100 %	32	16	100 %	0	64	32	100 %	8	11.10%	20%
	iRead 1	32	16	100 %	32	16	100 %	0	64	32	100 %	8	11.10%	20%
	Write all	32	16	100 %	2416	2400	0.66%	0	2448	2416	1.32%	2800	53.40%	54%
	Read all	32	16	100 %	2416	2400	0.66%	0	2448	2416	1.32%	2800	53.40%	54%
	BASTION	9664	4832	100 %	14432	9600	50.33%	0	24096	14432	66.96%	11200	31.70%	44%

Table 4.1: Experiment 1: Data overhead test results



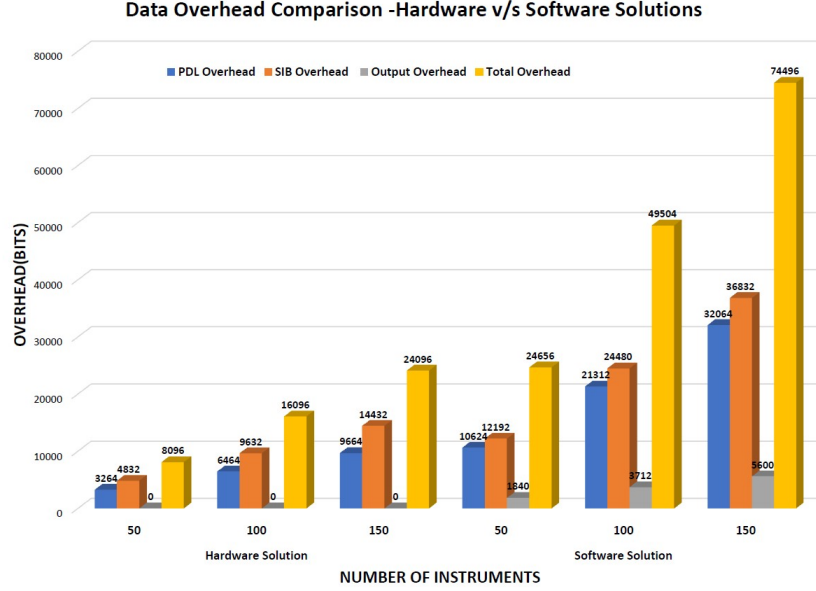
**Figure 4.2:** Data Overhead comparison Single 1687 Network case and Two 1687 Network case

## 4.2 Results of experiment 2

In the second experiment, we compare our hardware solution (interrupt-driven) against our software solution (polling-driven). For the software-based solution we assume optimal polling, that is the result is available at first attempt (only one poll is needed). The results on data overhead are presented in Table 4.2 and in Figure 4.3. Column 1 in Table 4.2 shows the number of instruments in the three designs, 50, 100, and 150. Column 2 shows used iApply group. The following columns report the overhead, PDL, SIB, Output, and total. The rightmost columns report the useful data and the relation of useful data compared to the total number of bits. For iWrite 1 – write of 8 bits to instrument 1 – the hardware solution and the software solution performs the same operations as there is no need to check for any output. For iRead 1 – read of 8 bits of data from instrument 1 – the software solution generates additional overhead due to polling. First, one setup command is needed to setup instrument 3 and one setup command is needed to setup instrument 4 (totally 32 bits). Second, action commands are needed. One to read from instrument 3 and one to write to instrument 4 (total of 32 bits). The read (poll) of instrument 3 generates 8 bits of output overhead. Figure 4.3 shows the data overhead for the BASTION benchmarks.

No. of Instruments	iApply Group	PDL Overhead		PDL overhead (%)	SIB Overhead		SIB overhead (%)	Output overhead		Total Overhead		Total overhead (%)	No useful bits	Useful data (%)	
		Hardware Solution	Software Solution		Hardware Solution	Software Solution		Hardware Solution	Software Solution	Hardware Solution	Software Solution			Hardware Solution	Software Solution
50	iWrite 1	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 1	32	64	100.00%	32	64	100.00%	0	8	64	136	112.50%	8	11.11%	5.56%
	iWrite 2	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 2	32	96	200.00%	32	96	200.00%	0	16	64	208	225.00%	8	11.11%	3.70%
	Allwrite	32	32	0.00%	816	816	0.00%	0	0	848	848	0.00%	920	52.04%	52.04%
	Allread	32	3712	11500.00%	816	4496	450.98%	0	920	848	9128	976.42%	920	52.04%	9.16%
	BASTION	3264	10624	225.49%	4832	12192	152.32%	0	1840	8096	24656	204.55%	3680	31.25%	12.99%
100	iWrite 1	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 1	32	64	100.00%	32	64	100.00%	0	8	64	136	112.50%	8	11.11%	5.56%
	iWrite 2	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 2	32	96	200.00%	32	96	200.00%	0	16	64	208	225.00%	8	11.11%	3.70%
	Allwrite	32	32	0.00%	1616	1616	0.00%	0	0	1648	1648	0.00%	1856	52.97%	52.97%
	Allread	32	7456	23200.00%	1616	9040	459.41%	0	1856	1648	18352	1013.59%	1856	52.97%	9.18%
	BASTION	6464	21312	229.70%	9632	24480	154.15%	0	3712	16096	49504	207.55%	7424	31.56%	13.04%
150	iWrite 1	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 1	32	64	100.00%	32	64	100.00%	0	8	64	136	112.50%	8	11.11%	5.56%
	iWrite 2	32	32	0.00%	32	32	0.00%	0	0	64	64	0.00%	8	11.11%	11.11%
	iRead 2	32	96	200.00%	32	96	200.00%	0	16	64	208	225.00%	8	11.11%	3.70%
	Allwrite	32	32	0.00%	2416	2416	0.00%	0	0	2448	2448	0.00%	2800	53.35%	53.35%
	Allread	32	11232	35000.00%	2416	13616	463.58%	0	2800	2448	27648	1029.41%	2800	53.35%	9.20%
	BASTION	9664	32064	231.79%	14432	36832	155.21%	0	5600	24096	74496	209.16%	11200	31.73%	13.07%

**Table 4.2:** Experiment 2: Data overhead test results Hardware and software solutions



**Figure 4.3:** Comparison of data overhead between the hardware-based versus and the software-based solution

The results on area utilization are in Tables 4.3 and 4.4 and Figure 4.4. Tables 4.3 and 4.4 is organized is as follows. Column1 shows the number of instruments in the three designs, 50, 100, and 150. Column 2 shows the type of IC-1 UART controller. The following column report the IC-2 SPI controller utilization, and IC-2 1687 network-2. We have two solutions. In the first, hardware-based solution where all the hardware components are present. In the second software-based solution where the hardware components like DTFSM and RSFSM are removed in IC-1 UART controller. Hence, the CLBs are more in hardware-based and less in software-based. Figure 4.4 shows the comparison of area utilized between two solutions in terms of controllers and network-2.

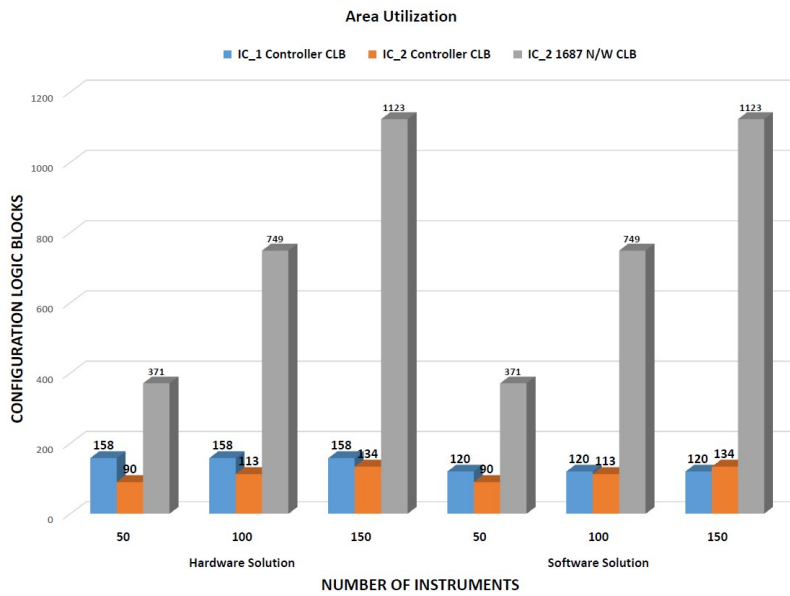
Instruments	IC 1 UART Controller			IC 2 SPI Controller			IC 2 1687 Network-2		
	FF	LUT	CLB	FF	LUT	CLB	FF	LUT	CLB
50	417	753	120	321	797	90	1970	1998	371
100	417	753	120	423	961	113	3983	4020	749
150	417	753	120	521	1102	134	5958	6058	1123

**Table 4.3:** FPGA resource utilization of the software-based solution



Instruments	IC_1 UART Controller			IC_2 SPI Controller			IC_2 1687 Network-2		
	FF	LUT	CLB	FF	LUT	CLB	FF	LUT	CLB
50	482	1021	158	321	797	90	1970	1998	371
100	482	1020	158	423	961	113	3983	4020	749
150	482	1025	158	521	1102	134	5958	6058	1123

**Table 4.4:** FPGA resource utilization of the hardware-based solution



**Figure 4.4:** Comparison of Area between the Hardware Solution versus the Software Solution

## Conclusion and Future Work

---

As on-chip instruments are increasingly needed to test, tune, and configure there is a need accessed them through the life-time of the IC. However, access is challenged due to complex system hierarchies when ICs are mounted on Printed Circuit Boards (PCB) as there might not be a direct access path and a single access protocol. In this thesis we develop a generic module to handle communication between two ICs. We implement two ICs on an FPGA; the first implements the IEEE Std. 1687 as a communication module while the second hosts on-chip instruments connected using the same standard. Communication between the ICs is performed with Serial Peripheral Interface (SPI) and the communication with the outside world with Universal Asynchronous Receiver Transmitter (UART). We propose and compare two alternatives for the communication. They are - a hardware-based solution where communication is handled in an interrupt-driven manner and a software-based solution where the communication is handled through polling. Comparison between the two solution based on data overhead and area utilization shows that hardware-based solution consumes less data overhead and more area utilization, whereas software-based solution consumes more data overhead and less area utilization. The choice of the solution for accessing on-chip instruments through system hierarchy depends upon the user requirements.

In the future, this thesis work could become the reference model to access of embedded instruments from system hierarchy. The developed communication module can be used to communicate with more than two ICs with any intermediate interfaces other than SPI bus interface. Further, this work can be used to explore the data and the other performance parameters with different approaches, such as daisy-chained networks and hierarchical networks.



---

## Bibliography

---

- [1] European collaborative research project BASTION, “About BASTION.” <http://fp7-bastion.eu/index.php?page=1>.
- [2] “IEEE Standard Test Access Port and Boundary Scan Architecture,” *IEEE Std 1149.1-2001*, pp. 1–212, 2001.
- [3] Corelis, Technical Guide to JTAG, “TAP State machine,” <https://www.corelis.com/education/tutorials/jtag-tutorial/jtag-technical-primer/>.
- [4] E. Larsson and F. G. Zadegan, “Accessing embedded dft instruments with ieee p1687,” in *2012 IEEE 21st Asian Test Symposium*, pp. 71–76, IEEE, 2012.
- [5] Ghani Zadegan, Farrokh, Reconfigurable On-Chip Instrument Access Networks: Analysis, Design, Operation, and Application. PhD thesis, Lund University, 2017.
- [6] E. Larsson, P. Murali, and G. Kumisbek, “Ieee std. p1687. 1: Translator and protocol,” in *2019 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2019.
- [7] G. Kumisbek and P. Murali, “Reconfigurable instrument access network with a functional port interface,” 2019.
- [8] GeeksforGeeks, “Difference between Interrupt and Polling,” <https://www.geeksforgeeks.org/difference-between-hardware-interrupt-and-software-interrupt/:text=Hardware>.
- [9] European collaborative research project BASTION, “PDL for flat tree network.” <http://fp7-bastion.eu/files/TreeFlat.pdl> .
- [10] Digilent, A National Instruments Company, “Xilinx Nexys 4 Artix-7 FPGA Trainer Board,” <https://reference.digilentinc.com/reference/programmable-logic/nexys-4/start>.
- [11] “IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device,” *IEEE Standard*, pp. 1687–2014, 2014.

- 
- [12] UART PROTOCOL VALIDATION SERVICE, “Soliton Technologies,” 2020.  
<https://www.solitontech.com/uart-protocol-validation-service/>.
  - [13] Nate Eastland, “Digilent Inc FPGA - Configuration Logic Block 2020.”  
<https://blog.digilentinc.com/fpga-configurable-logic-block/>.
  - [14] Balamurugan Natarajan, “SCLK - CPOH and CPOL,”  
<http://blmrgnn.blogspot.com/2018/10/spi-un1.html/>.

## Appendix 1

---

### A.1 UART standard

UART stands for Universal Asynchronous Receiver-Transmitter. It is used to transmit and receive serial data at various transmission speeds. UART consists of two components transmitter and receiver. The transmitter includes shift-register which shift-out the data concurrently at fixed rate. The receiver accepts the incoming data bit-by-bit and stores data in parallel. The start bit is also known as a synchronization bit that is placed before actual data. When start bit '1' indicates idle, start bit '0' indicates data transmission has begun. Followed by 5-8 data bits, an optional parity bit and ends with a stop bit, which is '1'. The optional parity bit is used for error detection. For odd parity, it is set to '0' when the data bits have an odd number of 1s. For even parity, it is set to '0' when data bits have an even number of 1s.

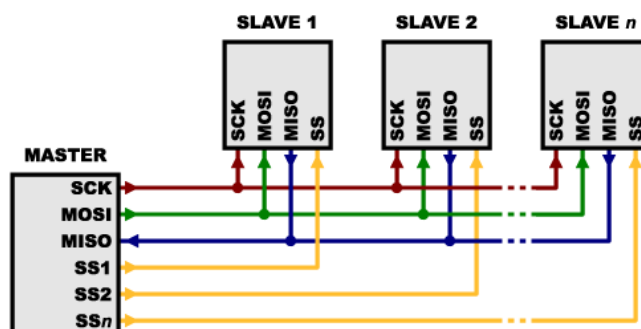
In the UART data transmission there is no clock signal used for synchronizing the output bits. Therefore when transmission starts transmitter and receiver must agree on a data transfer rate in advance, i.e., baud rate which specifies the number of bits transmitted per second (BPS), similarly number of data bits, stop bits and the parity bit. The standard baud rates are 2400, 4800, 9600, 19200, and 115200 BPS. This work uses a UART standard with 8 data bits, no parity bits, 1 stop bit and a baud rate of 115200 BPS.



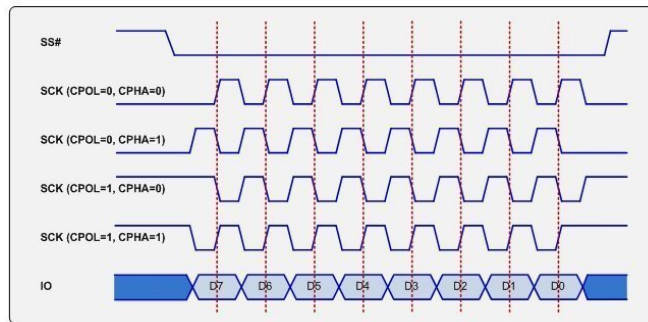
**Figure A.1:** Break down of UART transmission. Courtesy: Soliton Technologies[12]

## A.2 SPI bus interface

The Serial Peripheral interface (SPI) was developed by Motorola to provide full-duplex synchronous serial communication between master and slave devices. The Figure A.2, shows an SPI architecture with multiple slaves connected using single master. The SPI protocol defines a bus with four pins namely SCLK, MOSI, MISO, and SS. The slaves are active only when the signal SS is enabled. The data transferred serially bit-by-bit via two signals MOSI and MISO. The signal SCLK assures the time synchronization between master and slaves, and is always driven by master.



**Figure A.2:** SPI Single Master - Multiple slaves architecture



**Figure A.3:** SPI bus timing. Courtesy:[14]

The SPI standard multiple slave configuration uses four unique modes to provide flexibility in communication between master and slaves as shown in Figure A.3. The Clock Polarity (CPOL) and Clock Phase (CPHA) are the two additional signal settings used during the data sampling. The data sampling descriptions followed by the protocol is as follows:

- Mode 0 when CPOL and CPHA are both '0' data is sampled at the leading rising edge of the clock.
- Mode 1 when CPOL is '1' and CPHA is '0' data is sampled at the leading falling edge of the clock.
- Mode 2 when CPOL is '0' and CPHA is '1' data sampled at on the trailing falling edge.
- Mode 3 when CPOL is '1' and CPHA is '1' data sampled on the trailing rising edge. The table A.1 below depicts the detailed modes.

MODE	CPOL	CPHA
1	0	0
2	0	1
3	1	0
4	1	1

**Table A.1:** SPI protocol modes definition



### A.3 Retargeting tool algorithms

The re-targeting tool developed as software support to interpret the PDL using Python 3 scripting language. This tool benefits a designer by simply input the necessary PDL command and let the tool automatically generate the input vector and deliver it to the shift registers of the instruments. In our project, we used the retargeting tool developed by [7] and extended the algorithms to based on new designs. The 'activate\_network' can be seen in the Listing (A.1), (A.2), and (A.3) which enables the communication path between IC-1 and IC-2. As mention in the chapter 3 the algorithm to translate the iApply command is shown in Listing (A.4). This algorithm is common for both hardware-based and software-based solution.

```

1 iWrite function-hardware solution
2
3 #Network-1#
4 def activateNetwork(): #Main function activates network-1
5     instrument-1
6     act_cmd = [] #holds the setup and action commands
7     Write_cmd = (1<<14).to\_bytes(2, byteorder='big')#"0x40"
8     specifies the iWrite setup command
9     act_cmd.append(write_cmd)
10    Total_bytes= .join(send_cmd) + .join(test_vector) #
11    instrument_addresses and databits
12    CommandsAsData = int(len(.join(Total_bytes)))
13    apply_cmd = ((1<<15) | Total_bytes).to\_bytes(2,
14    byteorder='big')#Action command-iApply
15    act_cmd.append(apply_cmd)
16
17 #Network-2#
18 def iWrite():# instrument_address and databits
19     if instrument_address % 3 == 0: # instrument-1 is
20     selected (1byte)
21         write_cmd = ((1<<14) | instrument_address)#iWrite
22         command with instrument address
23         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
24     else if instrument_address % 3 == 1: # instrument-2 is
25     selected (2byte)
26         write_cmd = ((1<<14) | instrument_address)#iWrite
27         command with instrument address
28         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
29         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
30     else instrument_address % 3 == 2: # instrument-3 is
31     selected (4byte)
32         write_cmd = ((1<<14) | instrument_address)#iWrite
33         command with instrument address
34         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
35         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
36         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
37         testvector.append((0xaa).to\_bytes(1, byteorder='big'))
38     end if
39     return write_cmd

```

**Listing A.1:** Algorithm for the iWrite PDL hardware-based

```

1
2 iRead function-Hardware Solution
3
4 #Network-1#
5 def activateNetwork(): #Main function activates network-1
6     instrument-1
7     act_cmd = [] #list stores the setup and action commands
8     Write_cmd = (1<<14).to\_bytes(2, byteorder='big')#"0x40"
9     specifies the iWrite setup command
10    act_cmd.append(write_cmd)
11    Total_bytes= .join(send_cmd) + .join(test_vector) #
12    instrument_Addresses and databits
13    CommandsAsData = int(len(.join(Total_bytes)))
14    apply_cmd = ((1<<15) | Total_bytes).to\_bytes(2,
15    byteorder='big')#Action command-iApply
16    act_cmd.append(apply_cmd)
17
18 #Network-2#
19 def iRead():# instrument_address and dummybits
20 read_cmd = ((0 << 15) | instrument_address).to\_bytes(2,
21 byteorder='big')
22 if instrument_address % 3 == 0: # then generate dummy
23 bits (1byte)
24     expected_data.append(bin(0x55)[2:])
25 else if instrument_address % 3 == 1: # then generate
26 dummy bits (2byte)
27     expected_data.append(bin(0x55)[2:])
28     expected_data.append(bin(0x55)[2:])
29 else if instrument_address % 3 == 2: # then generate
30 dummy bits (4 byte)
31     expected_data.append(bin(0x55)[2:])
32     expected_data.append(bin(0x55)[2:])
33     expected_data.append(bin(0x55)[2:])
34     expected_data.append(bin(0x55)[2:])
35 end if
36 return read_cmd

```

**Listing A.2:** Algorithm for the iRead PDL hardware-based

```
1 iRead function-Software Solution
2
3 #Network-1#
4 def activateNetwork(): #Main function activates network-1
5     instrument-1
6     act_cmd = [] #list stores the setup and action commands
7     Write_cmd = (1<<14).to\_bytes(2, byteorder='big')#"0x40"
8     specifies the iWrite setup command
9     act_cmd.append(write_cmd)
10    Total_bytes= .join(send_cmd) + .join(test_vector) #
11    instrument_Addresses and databits
12    CommandsAsData = int(len(.join(Total_bytes)))
13    apply_cmd = ((1<<15) | Total_bytes).to\_bytes(2,
14    byteorder='big')#Action command-iApply
15    act_cmd.append(apply_cmd)
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33 def polling(instrument_address=2):# checks for the read data
34     if iRead(instrument_address) == 3: #checks read data is
35     arrived at instrument-3
36     else if
37     iRead(instrument_address) == 1: # read data is arrived at
38     instrument-3
39     read_cmd = (0 << 15) | instrument_address)#activate read
40     command to extract read data from instrument-3
```

```
38   end if  
39   return read_cmd
```

**Listing A.3:** Algorithm for the iRead PDL software-based

```
1 Action command-Hardware/Software Solution
2
3
4 def iApply(command): #Action command
5     if command = iWrite then
6         All_bytes = .join(input_testvector) #actual write data
7         and action command
8         Total_bytes = int(len(All_bytes))# total number of
9         bytes for the network-2
10        Apply_cmd = ((1 << 15) | Total_bytes)# sends commands
11        and data via instr-1 network-1
12    else if
13        command = iRead then
14        Apply_cmd = (1 << 15).to_bytes(2, byteorder='big')#"0
15        x80" iApply action command
16    end if
17    return iApply
```

**Listing A.4:** Algorithm for the iApply command



**LUND**  
UNIVERSITY

Series of Master's theses  
Department of Electrical and Information Technology  
LU/LTH-EIT 2021-801  
<http://www.eit.lth.se>