Analysis and Optimization for Testing Using IEEE P1687

By

Farrokh Ghani Zadegan

LIU-IDALITH-EX-A-10/040-SE

2010-10-13
Final Thesis

Analysis and Optimization for Testing Using IEEE P1687

by

Farrokh Ghani Zadegan

LIU-IDA/LITH-EX-A--10/040--SE

Supervisor : Erik Larsson
Dept. of Computer and Information Science
at Linköpings Universitet

 Examiner : Urban Ingelsson
Dept. of Computer and Information Science
at Linköpings Universitet
Abstract

The IEEE P1687 (IJTAG) standard proposal aims at providing a standardized interface between on-chip embedded test, debug and monitoring logic (instruments), such as scan-chains and temperature sensors, and the Test Access Port of IEEE Standard 1149.1 mainly used for board test. A key feature in P1687 is to include Segment Insertion Bits (SIBs) in the scan path. SIBs make it possible to construct a multitude of different P1687 networks for the same set of instruments, and provide flexibility in test scheduling. The work presented in this thesis consists of two parts. In the first part, analysis regarding test application time is given for P1687 networks while making use of two test schedule types, namely concurrent and sequential test scheduling. Furthermore, formulas and novel algorithms are presented to compute the test time for a given P1687 network and a given schedule type. The algorithms are implemented and employed in extensive experiments on realistic industrial designs. In the second part, design of IEEE P1687 networks is studied. Designing the P1687 network that results in the least test application time for a given set of instruments, is a time-consuming task in the absence of automatic design tools. In this thesis work, novel algorithms are presented for automated design of P1687 networks which are optimized with respect to test application time and the required number of SIBs. The algorithms are implemented and demonstrated in experiments on industrial SOCs.

Keywords: IEEE P1687, IJTAG, Test Architectures, Test Time Calculation, Design Automation, Test Time Optimization
Acknowledgements

This thesis work marks the end of my Master studies at Linköping University, Sweden. I would like to express my heartfelt gratitude to those who contributed to my success during this period and specifically in carrying out my Master thesis.

In the first place, I would like to thank Sweden for providing me with high quality education for free.

I would very much like to thank Erik Larsson for suggesting such an interesting topic for my thesis, and for encouraging and providing me with a scholarship to make two papers based on this thesis work.

I gratefully acknowledge Urban Ingelsson for his supervision and his constructive comments on this work. I would not have been able to carry out all the assigned tasks successfully if it had not been for his kind help and support.

It was a great opportunity for me to meet Gunnar Carlsson and have my many technical questions regarding this thesis kindly answered by him.

My special thanks go to my family: To my mother to whom words fail me to express my gratitude, to my brother who supported me both emotionally and financially, and to my sister who has always encouraged me to progress.

I was extremely fortunate in having many great friends beside me during my Master studies in Sweden. It is my pleasure to mention: Ali, Amin, Amir, Asieh, Edris, Iman, Kaveh, Mahdad, Marjan, Mehdi, Ning, Noora, Omid, Payman and Rana for helping me whenever I needed and for inviting me to their friendly parties and gatherings. Many thanks also go to Andréas Karlsson, Kristian Chevalier and specially Patrik Björn whose friendship made my stay in Sweden so pleasant and memorable.
# Contents

1 Introduction and Background  
  1.1 Introduction ........................................ 1  
  1.2 How This Report Is Organized ......................... 4  
  1.3 Introduction to IEEE Standard 1149.1 ............... 4  
  1.3.1 Overview ......................................... 4  
  1.3.2 Hardware ......................................... 5  
  1.4 Introduction to IEEE Standard 1500 ................. 8  
  1.5 IEEE P1687 (IJTAG) Architecture and Terminology .... 10  
  1.5.1 Overview ......................................... 10  
  1.5.2 Interfacing P1687 to JTAG TAP .................... 14  
  1.5.3 Segment Insertion Bit (SIB): The Internal Circuitry 14  
  1.5.4 Interfacing P1687 to IEEE Standard 1500 Wrapped  
    Cores ................................................. 16  
  1.5.5 Possible Architectures ........................... 17  
  1.6 Summary ............................................. 19  

2 Test Time Calculation ........................................ 20  
  2.1 Test Application Time for the Flat Architecture .... 21  
    2.1.1 Concurrent Test Schedule ......................... 22  
    2.1.2 Sequential Test Schedule ....................... 26  
  2.2 Test Application Time for the Hierarchical Architecture 28  
    2.2.1 Concurrent Test Schedule ......................... 29  
    2.2.2 Sequential Test Schedule ....................... 31  
  2.3 Test Time Calculation Method: IJTAGcalc ............... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1</td>
<td>IJTAGcalc for the Concurrent Test Schedule</td>
<td>35</td>
</tr>
<tr>
<td>2.3.2</td>
<td>IJTAGcalc for the Sequential Test Schedule</td>
<td>37</td>
</tr>
<tr>
<td>2.4</td>
<td>IEEE Standard 1500 Wrapped Cores As Instruments</td>
<td>39</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Flat Architecture, Concurrent Test Schedule</td>
<td>40</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Flat Architecture, Sequential Test Schedule</td>
<td>41</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Hierarchical Architecture, Concurrent Test Schedule</td>
<td>44</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Hierarchical Architecture, Sequential Test Schedule</td>
<td>46</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Comparison with scan-chains as instruments</td>
<td>48</td>
</tr>
<tr>
<td>2.4.6</td>
<td>IJTAGcalc for the Concurrent Test Schedule</td>
<td>49</td>
</tr>
<tr>
<td>2.4.7</td>
<td>IJTAGcalc for the Sequential Test Schedule</td>
<td>52</td>
</tr>
<tr>
<td>2.5</td>
<td>Experiments with the ITC’02 Benchmarks</td>
<td>54</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Experimental Setup</td>
<td>54</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Experimental Results for Unwrapped Cores</td>
<td>57</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Experimental Results for Wrapped Cores</td>
<td>61</td>
</tr>
<tr>
<td>2.6</td>
<td>Summary</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Design of P1687 Networks</td>
<td>67</td>
</tr>
<tr>
<td>3.1</td>
<td>SIB Overhead Reduction for the Sequential Schedule</td>
<td>68</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Analysis</td>
<td>68</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Algorithms for Sequential Schedules</td>
<td>73</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Handling a Special Case</td>
<td>74</td>
</tr>
<tr>
<td>3.1.4</td>
<td>The Proposed Design Method</td>
<td>75</td>
</tr>
<tr>
<td>3.2</td>
<td>SIB Overhead Reduction for the Concurrent Schedule</td>
<td>77</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Analysis</td>
<td>77</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Algorithm for Concurrent Schedules</td>
<td>79</td>
</tr>
<tr>
<td>3.3</td>
<td>Experimental Setup</td>
<td>80</td>
</tr>
<tr>
<td>3.4</td>
<td>Experimental Results for Wrapped Cores</td>
<td>81</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary</td>
<td>86</td>
</tr>
<tr>
<td>4</td>
<td>Conclusion and Future Work</td>
<td>87</td>
</tr>
<tr>
<td>4.1</td>
<td>Conclusion</td>
<td>87</td>
</tr>
<tr>
<td>4.2</td>
<td>Future Work</td>
<td>88</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction and Background

1.1 Introduction

The increasing complexity of Integrated Circuits (ICs) and the need for shorter time to market have made design reuse an attractive and common practice. For example, an ASIC from Ericsson contains 64 processors where each processor has its dedicated data memory and instruction memory, and a number of SERDESes and hardware accelerators [1]. Therefore, more than 200 blocks of logic are reused to design this large ASIC. The manufacturing process of ICs and the process of assembling them onto a board are not perfect. Therefore, tests will be necessary at different stages of a chip’s life cycle, i.e. prototyping, wafer test, board test, system test and in-field test [2]. Unfortunately, tests designed for a block of logic do not lend themselves well to the design reuse paradigm. The reason is that tests cannot be parameterized (such as selecting algorithms or the seed patterns for BIST) and they are developed having a specific test application equipment in mind [3]. Therefore, the system integrator faces the task of reusing the tests for the logic blocks at chip level, board level and system level. Furthermore, in a large IC such as the ASIC mentioned above, each
block of logic may contain embedded test, debug and monitoring logic (referred to as instruments) such as Memory BIST, Logic BIST, scan-chains, and temperature sensors, which are used to ensure testability and reliability of the manufactured IC. However, there is no standard way (and thus no EDA support) for accessing the on-chip instruments. Therefore, a DFT standard is needed to facilitate the test reuse and standardize the access to the instruments. The IEEE P1687 standard proposal is such an effort which aims at providing a protocol and a uniform connectivity method for accessing the instruments.

P1687 proposes to use the IEEE 1194.1 (a.k.a. JTAG) Test Access Port (TAP) for accessing the instruments from outside the chip. Therefore, P1687 has received the informal name of IJTAG (Internal JTAG). JTAG will be briefly introduced in Section 1.3. The choice of using JTAG TAP to access the on-chip instruments is made due to the widespread adoption of JTAG TAP in ad hoc access to the internal test and debug features [3]. Although JTAG was originally meant for board test and not for accessing on-chip instruments, it has found such application in recent years. Among examples of such application are Altera’s SignalTap Logic Analyzer, Xilinx’s ChipScope Pro real-time debug and verification tool and the circuits explained in the works of [4], [5] and [6]. Furthermore, in [7] a technique is presented for accessing the internal scan chains of the chip through the JTAG TAP.

Since using JTAG to access the on-chip embedded logic is beyond the scope of the JTAG standard, there are some drawbacks associated with such usage [8]:

- JTAG circuitry will not scale well with increasing the number of instruments, since the instruction decoder becomes more and more complex with the addition of each new instrument.

- Boundary Scan Definition Language (BSDL) which is part of the IEEE Standard 1149.1 [9] and is used to describe the boundary scan devices, is neither efficient nor sufficient to describe all types of instruments (both current and future instruments) [8].

- Since different solutions utilize JTAG port to connect to on-chip embedded logic (instruments) in different ways, no EDA tool vendor is
able to provide designers with a tool to automate the insertion of the instrument access network.

The IEEE P1687 standard proposal addresses the above problems and standardizes the way that on-chip instruments are accessed through JTAG. A key feature of P1687 is a component called Segment Insertion Bit (SIB). By using SIBs, it is possible to construct an instrument access network that has a flexible scan-path. The term P1687 network in this report will be applied to any arrangement of SIBs and instruments that form the P1687 scan-path. SIBs provide flexibility in setting up the scan-path by adding segments of P1687 network to the scan-path. A segment of P1687 network can be a single instrument or a smaller network of SIBs and instruments.

Although at the time of writing of this report the final draft of IEEE P1687 is not yet released, a few studies have already considered it. In [2] a possible prototype of P1687 is used to access the embedded measurement circuitry. Also in [10] and [4] the authors have considered future integration of their proposed techniques with P1687. However, no study has yet considered the impact of placing SIBs on the scan-path in terms of time required to access the instruments. Also automated design of optimized (with respect to test time and number of SIBs) P1687 networks is not addressed yet. Without automated design, design of instrument access networks manually when there are many instruments, such as the mentioned ASIC from Ericsson, might be extremely time consuming. In this thesis work these issues will be addressed while keeping the focus mainly on using P1687 to access Design-for-Test (DFT) instruments, i.e. scan chains and IEEE standard 1500 wrapped cores. The main contributions of this thesis work can be listed as follows:

- A test time calculation method is presented called IJTAGCalc, which is able to handle a wide range of test architectures that are implemented using P1687, and two types of schedules, namely concurrent test scheduling and sequential scheduling.

- Design automation algorithms are presented for design of P1687 networks which are optimized with respect to test time and gate overhead. The algorithms take as input a set of instruments and a test
1.2 How This Report Is Organized

Since following the material presented in this report requires familiarity with IEEE standards 1149.1 and 1500, the required features of these standards will be introduced in Sections 1.3 and 1.4. In Section 1.5, P1687 circuitry will be introduced and it will be explained how P1687 can be used to connect the IEEE standard 1500 wrapped cores to JTAG circuitry. What is presented in this section is based on the author’s understanding of the material made public through the IEEE P1687 work group’s website [11]. Chapter 2 will describe the work done for test time calculation when P1687 is in use to transfer the test stimuli and gather the test responses. Some observations in this chapter will be used to guide the work described in chapter 3 to design P1687 networks which are optimized with respect to the trade-off between test application time (TAT) and the required hardware components. Chapter 4 presents the conclusion of this thesis work and possible future directions.

1.3 Introduction to IEEE Standard 1149.1

1.3.1 Overview

IEEE standard 1149.1, also known as JTAG, describes logic to be incorporated into integrated circuits (ICs) to standardize the following [9]:

- Printed circuit board (PCB) test: JTAG can be used to test the printed circuit boards to detect the faults in the interconnections between ICs (tracks), placement of ICs and soldering.
Accessing the internal design-for-test circuitry of an IC, e.g. built-in-self-test (BIST) engines.

In the next section (Section 1.3.2) the JTAG circuitry will be introduced as much as required for understanding the rest of this report. The information provided in Section 1.3.2 is mainly from the standard draft [9].

### 1.3.2 Hardware

Figure 1.1 shows a conceptual view of JTAG circuitry in a chip [12] [9]. The access to the on-chip JTAG circuitry is provided through test access port (TAP). The TAP includes four mandatory connections, namely test data input (TDI), test data output (TDO), test mode select (TMS) and test clock (TCK), and one optional reset input which is not shown in Figure 1.1. As can be seen only one data input (TDI) and one data output (TDO) are available for transporting both instructions and test data. The transported data is shifted in and out serially one bit with every clock pulse on TCK signal. The TCK signal is the clock for the test logic accessed through TAP, which is used to (independently from component specific clocks) synchronize the operation of the test circuitry inside one chip and among different chips in a board or system. The TMS signal is decoded by a standard state diagram, shown in Figure 1.2, to generate the control signals required by the test circuitry. This state diagram is implemented in the TAP Controller state machine described later in this section. The IEEE Standard 1149.1
1.3. Introduction to IEEE Standard 1149.1

requires the following hardware components in addition to the TAP and TAP Controller:

- Two test data registers (TDRs), (1) a boundary scan register (BSR) and (2) a bypass register (BR). Optionally, design specific TDRs can be included. In Figure 1.1 Gateway is such a custom TDR.

- An instruction register (IR) which is a shift and update register used to store the instruction shifted in through TDI. The shifted instruction should be latched, so that subsequent shifts do not have any impact on the current instruction. Any further change in the latched instruction will happen only by passing the Update-IR or Test-Logic-Reset states (see Figure 1.2).

- An instruction decoder to decode the instruction latched at the parallel outputs of the IR. This instruction may either select a TDR or a test, such as BYPASS, SAMPLE, PRELOAD, EXTEST and etc., to be performed.
The TAP Controller is responsible for shifting the instruction data into IR, shifting the test data into any of the TDRs, and generating the control signals to perform test actions such as capture, shift and update [13]. Capture is defined as loading a value into the IR or any of the TDRs, and update is defined as transferring logic values from the shift-register stage of the IR or any of the TDRs to their latched parallel output [9]. The state diagram, implemented in the TAP Controller, has two similar branches, one for performing operations on test instruction data, which is the IR branch, and one for performing operations on test data, which is the DR branch.

The IR branch of the state diagram is used to load the instructions to IR. To shift the test instruction data into IR, the state machine should be taken into the Select-IR-Scan state. This requires three TCKs (assuming the initial state of Test-Logic-Reset) for the first of which TMS should be set to logic zero and for the rest it should be set to logic one (see Figure 1.2). Once the state machine is in the Select-IR-Scan state, by selecting appropriate values for TMS, i.e. logic zero, the state machine will be further taken into the Shift-IR state where it is possible to shift in the instruction bits serially through TDI pin of the TAP. By keeping TMS at logic zero it is possible to shift in as many instruction bits as necessary and once done, go through Exit1-IR and Update-IR states (by setting TMS to logic one) to latch the shifted instruction.

The procedure for shifting the data into the current TDR is almost the same except that the DR branch of the state diagram is used. Current TDR is determined by the output of the instruction decoder (marked as IR Decoder in Figure 1.1). In this report, the progression of five of the states in the DR branch, i.e. Exit1-DR, Update-DR, Select-DR-Scan, Capture-DR and Shift-DR, is of special interest and is called a CUC (short for capture/update cycle). The CUCs are required for applying test patterns. The procedure for test application is that after shifting in the test stimuli (in Shift-DR state), the traversal of these five states is required for the application of the test stimuli (in Update-DR state) and capturing of the test responses (in Capture-DR state). Once back in the Shift-DR state, the captured test responses can be shifted out while the next test stimuli are being shifted in. Since no actual test data is transported during a CUC, the time taken to perform a CUC is considered as overhead in the rest of this report.
1.4 Introduction to IEEE Standard 1500

IEEE standard 1500 for embedded core test (SECT) aims at providing a test reuse methodology for the intellectual property (IP) cores used in system-on-chip (SOC) designs [14]. This is achieved by defining circuitry, known as wrapper, to be added around a core. Wrappers also facilitate the interfacing of the core to the test access mechanisms (TAMs) which are responsible for the transportation of test data [13]. As shown in Figure 1.3(a), a wrapper is required to have a wrapper serial port (WSP), wrapper instruction register (WIR), wrapper bypass register (WBY) and wrapper boundary register (WBR), and may optionally have a user defined wrapper parallel port (WPP) as well. Besides the required WBY and WBR, optionally other wrapper data registers (WDRs) can be used. It is also possible to connect core data registers (CDR) to the wrapper circuitry. The WIR, WBY and WBR are accessed through the WSP. Since JTAG (which uses a serial protocol) is used to transport the test instructions and data in the context of this thesis work, only the WSP (and not the WPP) is of interest in this report.

Figure 1.3(b) shows the eight required terminals of WSP [14]. Wrapper serial input (WSI) and wrapper serial output (WSO) are used to transport test control and data in and out of the wrapper. SelectWIR, if set to logic one, unconditionally selects the WIR regardless of the current decoded instruction or selected WDRs/CDRs. WRCK and WRSTN provide the clock and reset signals for the wrapper, respectively. CaptureWR, ShiftWR and UpdateWR terminals are used to select capture, shift and update operations, respectively. These operations have the same definition as those introduced in Section 1.3.2. Figure 1.4 shows how these signals can be provided by the JTAG TAP [14]. Later in Section 1.5 it will be described how these signals are interfaced to the P1687 circuitry.

Figure 1.3(a) also shows how required components of the IEEE standard 1500 are interfaced to one another. Comparing Figure 1.1 and Figure 1.3(a), shows that WSP circuitry is very similar to the JTAG circuitry, with the difference that JTAG has a state machine, i.e. TAP Controller, to generate the required shift, update and capture signals whereas WSP requires these to applied from outside the wrapper.

In the rest of this report, the simplified wrapped core shown in Fig-
Introduction and Background

(a) How WSP signals, WIR and data registers are interfaced

(b) Wrapper serial port (WSP)

Figure 1.3: Wrapper serial port of an IEEE Standard 1500 wrapped core. Figure 1.5(a) will be used. Only one wrapper data register (WDR) is shown for the core in Figure 1.5(a) which will be enough to explain the test application time in case of wrapped cores as instruments. In the figure, L denotes the length of the scan-chain inside WDR/WIR and P denotes the number of test patterns that should be applied to the WDR. To apply tests to the WDR, it should be first selected by scanning in the appropriate instruction to the WIR. To do so, SelectWIR should be asserted high, so that WIR is selected and connected in the WSI to WSO scan-path as shown in Figure 1.5(b), and the instruction data be shifted in. After the UpdateWR
Figure 1.4: Interfacing the JTAG TAP Controller to the WSP

signal is asserted, which is done when the JTAG TAP controller state machine is in the Update-DR state (see Figure 1.2), the instruction is stored, decoded and the WDR is selected. Then, before scanning in the test data for the WDR, SelectWIR should be asserted low, so that the selected WDR is connected to the WSI-WSO path (as shown in Figure 1.5(c)).

1.5 IEEE P1687 (IJTAG) Architecture and Terminology

1.5.1 Overview

IEEE P1687 introduces a configurable (programmable) component called Segment Insertion Bit (SIB) or Select Instrument Bit (as was presented in earlier documents of P1687 working group) [11]. The SIBs are placed on the scan-path and through their hierarchical interface port (HIP) can change the scan-path by concatenating another P1687 network segment to it, or by excluding a segment which is not currently being used. This capability of SIBs can be compared to a daisy-chain architecture shown in Figure 1.6 [15] where by controlling the multiplexers, it is possible to change the scan-path by excluding a scan chain whose tests are finished, or including a scan chain according to the test schedule. In the daisy chain architecture, the multiplexer control signals (or the single control signal when a shift
Figure 1.5: Simplified IEEE 1500 wrapped core for which one wrapper data register (WDR) is shown.

Figure 1.6: Daisy chain architecture which is used to include the scan chains in (or exclude them from) the scan-path.

Figure 1.7(a) shows a component model of a SIB. Serial data in (SDI) and serial data out (SDO) are used to transport configuration and test data.
1.5. IEEE P1687 (IJTAG) Architecture and Terminology

(a) SIB with all terms (b) Simplified model of SIB

Figure 1.7: SIB as a component

to and through the SIB. SCK and Reset provide the test clock (see Section 1.3.2) and reset signals to the P1687 circuitry, respectively. ShiftEn, CaptureEn and UpdateEn signals are generated when the JTAG TAP Controller is in Shift-DR, Capture-DR and Update-DR states, respectively. These signals are shared among all components in the P1687 network and therefore are gated internally by using the Select signal, so that they only become effective when a component is selected. In Figure 1.7(a) the hierarchical interface port (HIP) signals are shown with the prefix HIP. The hierarchical interface port is used to pass the Select signal and the scan-path connections on to another segment of the P1687 network (which might be only an instrument). Other signals such as SelectWIR may optionally be included in the HIP [11], as will be explained in Section 1.5.4.

In this report whenever a SIB is used to connect the scan-path to an instrument, it is referred to as an Instrument SIB and when it is used to connect to a larger segment of the P1687 network containing SIBs and instruments, it is called a Doorway SIB. The concept of hierarchy will be elaborated on more in Section 1.5.5.

Since SCK, Reset, ShiftEn, CaptureEn and UpdateEn signals are shared among all P1687 components, they will not be shown in later chapters of this report and the symbol shown in Figure 1.7(b) will be used instead to represent the SIB. For further simplification of the drawings the Select and HIP_ToSel signals are also removed from the symbol and it will be assumed that any P1687 component (i.e. SIB or instrument) connected to the HIP of a SIB has its select signal connected to the HIP_ToSel of that SIB.
The basic idea is that the SIB is programmed by shifting the required logic value into its Shift-Capture (S/C) flip-flop and storing it in its Update (U) flip-flop. Depending on the value stored in the U flop, the SIB will be either closed or open. Figure 1.8(b) shows a closed SIB where the dashed line shows the scan-path. When the SIB is open, (Figure 1.8(c)) the P1687 logic connected to the hierarchical interface port (HIP) of the SIB, will be concatenated to the scan-path. After programming the SIB, test stimuli are shifted in the scan-path and responses are shifted out. Since the Scan/Capture (S/C) flip-flop is always on the scan-path, the SIB represents a one-bit delay during the scan operation.
1.5. Interfacing P1687 to JTAG TAP

Interfacing P1687 to JTAG TAP is done by adding a user defined test data register (TDR) to the JTAG circuitry, as permitted by the IEEE standard 1149.1. This TDR is called Gateway in the P1687 terminology and may be a single SIB or composed of a group of SIBs in series, i.e. connected SDI-to-SDO. Figure 1.9 shows a Gateway composed of two SIBs. To start transporting configuration and test data to the P1687 logic, the Gateway should be selected first. This is done by loading a custom command called Gateway Enable (GWEN) into the JTAG instruction register (IR). As mentioned in Section 1.3.2, this is done by going through the IR branch of the TAP Controller state diagram (Figure 1.2).

1.5.3 Segment Insertion Bit (SIB): The Internal Circuitry

Figure 1.10 shows a more detailed RTL view of an example SIB. This example is suggested based on the documents and presentations available on the IEEE P1687 group’s website [11].

As mentioned earlier the ShiftEn, CaptureEn, and UpdateEn signals should be gated using the Select signal. This is done using the three AND gates in Figure 1.10. Since SCK is connected internally to all flip-flops, the keeper multiplexers (marked by K) are used to make the flip-flops retain...
their value when the SIB is not selected. The multiplexer labeled D can be used for diagnostic purposes as it captures the value of the Update (U) flip-flop in the S/C flop to be scanned out and evaluated. To avoid race conditions, a pipelining flip-flop, i.e. PL1, is added such that a SIB and the segment connected to its Hierarchical Interface Port are not selected at the same cycle.
1.5.4 Interfacing P1687 to IEEE Standard 1500 Wrapped Cores

In this section the connection of IEEE standard 1500 wrapped cores (called wrapped cores hereafter) to P1687 network will be explained. To interface wrapped cores to P1687 networks, the localized-control concept introduced in [11] will be used. As mentioned in Section 1.4, only the connection of the wrapper serial port (WSP) to P1687 is of interest in this thesis work. To establish the interface, the connection of eight required signals of WSP shown in Figure 1.3(b) should be handled. The provision of SelectWIR signal through the HIP of a SIB requires another pair of shift-update flops inside the SIB as shown in Figure 1.11(a). The implication of an extra pair of shift-update flops is an extra delay in the scan operation. The SIB provides the SelectWIR and also the Select signal, which is used to gate the ShiftEn, CaptureEn and UpdateEn global signals, to provide the ShiftWR, CaptureWR and UpdateWR signals locally. WCK should be connected to TCK and WRSTN should be connected to the Reset signal which is assumed to be provided directly from the JTAG TAP to all P1687 logic. Finally, WSI and WSO should be connected to HIP_ToSdi and HIP_FromSdo terminals of the SIB, respectively.

Before accessing any of the wrapper data registers (WDR) in a wrapped core, the Select signal should be asserted high to enable the control signals such as ShiftWR, CaptureWR and UpdateWR for the wrapper. It
is also required that the WDR to be tested, is selected by the appropriate instruction, as explained in Section 1.4. The assertion of Select and SelectWIR signals can be done simultaneously, but when the instruction data are shifted and decoded, the Select signal should remain active (high) and the SelectWIR should become inactive (low), so that the test data be transported to the WDR selected by the decoded instruction. In Section 2.4 the effect of the WDR selection and the SIB programming on the total test application time (in the context of P1687) will be calculated. Figure 1.11(b) shows a simplified model of the SIB for the wrapped cores which will be used in the rest of this report. For the sake of simplicity, this SIB will be referred to as the wrapper SIB hereafter.

1.5.5 Possible Architectures

In this section, different possible P1687 network infrastructures (connectivity types) will be discussed and categorized. These will be referred to in the later chapters of this report as test architectures. In general two test architectures may be considered: (1) Flat and (2) Hierarchical. In the flat case, every SIB receives its Select signal directly from the JTAG IR Decoder and all the SIBs are connected SDI-to-SDO with the first SIB having its SDI connected to JTAG TDI and the last SIB having its SDO connected to JTAG TDO (via multiplexers). This means that all the SIBs in the design are inside the P1687 Gateway and instruments are connected to the HIPs of these SIBs. Furthermore, a flat architecture implies that there are no doorway SIBs in the P1687 network. An example of this test architecture is shown in Figure 1.12(a). Assuming that all SIBs are closed initially, the instruments, i.e. scan chains, are not on the scan-path and depending on what values are programmed into the SIBs, different test schedules are possible. For example to test scan chain 3 singly, logic 0s should be programmed into SIB\(_1\) and SIB\(_2\), and a logic 1 should be programmed into SIB\(_3\). This way, only scan chain 3 will be included in the scan-path. It should be noted that all test data should however pass through the S/C flops of all the SIBs.

If however, any SIB in the P1687 logic receives its Select signal and scan-path connections from the HIP of another SIB, this P1687 network infrastructure will be called hierarchical. Figure 1.12(b) shows an example
of hierarchical test connectivity type where only SIB\textsubscript{1} and SIB\textsubscript{2} receive their Select signal directly from the JTAG IR Decoder. In this network SIB\textsubscript{1}, SIB\textsubscript{3} and SIB\textsubscript{5} are Instrument SIBs and SIB\textsubscript{2} and SIB\textsubscript{4} are Doorway SIBs, as were defined in Section 1.5.1. This is in contrast to the flat type where all SIBs are Instrument SIBs. In this case if only scan chain 1 is to be tested, SIB\textsubscript{2} will stay closed and therefore the test data only passes through the S/C flops of SIB\textsubscript{1} and SIB\textsubscript{2}, but for testing SIB\textsubscript{3} singly, test data should pass the S/C flops of all of the SIBs.

Figure 1.12(c) shows an alternative hierarchical network where an instrument is connected in series with SIB\textsubscript{3}. The reason that this type is also considered hierarchical in this report, is that SIB\textsubscript{3} is connected to the HIP of SIB\textsubscript{2}. In this example, to test SIB\textsubscript{3} singly all test data has to pass scan chain 2 as well, and this reduces the efficiency of the test schedule. Therefore, this type of test architecture will not be considered in this thesis work.
and the focus will be only on the flat (Figure 1.12(a)) and hierarchical (Figure 1.12(b)) architectures. However, the algorithms presented in Chapter 2 are capable of handling this alternative test architecture (Figure 1.12(c)).

1.6 Summary

In this chapter, the scope and contributions of this thesis work as well as the organization of this report were presented. Furthermore, the IEEE Standard 1149.1, IEEE Standard 1500 and IEEE P1687 standard proposal were introduced as much as required for following the material presented in the rest of this report.
Chapter 2

Test Time Calculation

In Chapter 1 the fundamentals of P1687 were introduced. In this chapter those concepts will be used to study P1687 networks from test application time (TAT) point of view, and the overhead caused by the P1687 protocol and hardware components. As of writing of this report, no other study has considered TAT calculation for P1687 networks. The observations made in this chapter will be used as guidelines in Chapter 3, to design optimized P1687 networks with respect to TAT and the number of SIBs.

To calculate TAT, the flat and hierarchical test architectures described in Section 1.5.5 will be considered. Each of these architectures will be studied using two test schedules, (1) sequential schedule and (2) concurrent schedule. In sequential test schedule, each instrument is tested separately and completely before testing the next instrument. But in the concurrent schedule, the tests of all instruments start at the same time. Also for each architecture, the calculation will be done for two cases, (1) when the instrument is a scan chain and (2) when the instrument is an IEEE standard 1500 wrapped core. For the flat architecture, we propose a formula for the test time calculation (for each of the schedules and instrument types), whereas algorithms will be proposed for the hierarchical architecture. The reason that a formula is not presented for the hierarchical architecture will be explained in Section 2.2. However, since a flat architecture can be considered a one-level hierarchical architecture, the proposed algorithms
will be applicable to the flat architecture as well. These algorithms will then be employed to calculate TAT for the selected SOCs of the ITC’02 benchmarks [16] and the results will be discussed.

The rest of this chapter is organized as follows: In Section 2.1, a small sample network (with scan-chains as instruments) having flat architecture will be considered and its test application steps will be thoroughly explained, both for sequential and concurrent test schedules. Based on the knowledge gained from this sample network, formulas will be presented to calculate TAT of any P1687 network having flat architecture, for each of the test schedules. In Section 2.2, a small sample network (with scan-chains as instruments) having hierarchical architecture will be considered, its test application will be explained for both concurrent and sequential schedules, and algorithms will be presented for both concurrent and sequential schedules to calculate TAT of any network having hierarchical architecture. In Section 2.4 necessary changes will be made to the presented formulas and algorithms to support the IEEE standard 1500 wrapped cores as instruments. In Section 2.5.1, experimental setup to apply the algorithms to the ITC’02 benchmarks will be explained, followed by the presentation and discussion of the experimental results in Section 2.5.2 and Section 2.5.3. Finally, comes the conclusion to the chapter.

2.1 Test Application Time for the Flat Architecture

Figure 2.1 shows a small P1687 network with a flat architecture. In this network, the scan-chains SC$_1$, SC$_2$, and SC$_3$ are the instruments. In Figure 2.1, $L$ stands for the length of the scan-chain and $P$ stands for the number of test patterns that exist in a test for the scan-chain.

The typical test application process is to scan in test stimuli from TDI, through the SIBs, into the scan-chains, where the test stimuli is applied and test responses are captured in the flip-flops of the scan-chain. Subsequently, the test responses are scanned out, through the SIBs to TDO. It should be noted that while a test response is scanned out, it is possible to scan in the next test stimuli. In the following, test application steps in a concurrent
Figure 2.1: A sample network with a flat architecture having scan-chains as instruments

2.1. Test Application Time for the Flat Architecture

2.1.1 Concurrent Test Schedule

Regarding the concurrent schedule, the following will describe how to calculate the test application time for the flat test architecture, with the help of Table 2.1 and Figure 2.2. Before applying the first test pattern, the SIBs must be opened, since the scan-path initially only consists of the SIBs, as shown in Figure 2.2(a). To open the SIBs, three bits are scanned in (one bit for each SIB) and subsequently a CUC is performed. The three bits each correspond to the 1 bit S/C flop of a closed SIB (Figure 1.8(b)), and the three bits are accounted for on the row marked Setup-sequence in Table 2.1 in column “SIBs”. After the CUC, all instruments are included in the scan-path, as shown in Figure 2.2(b). At this point, test patterns can be applied to all three instruments, with a total scan-path length of \(1_O + 3SC_1 + 1O + 5SC_2 + 1O + 4SC_3=15\) bits, where \(1_O\) corresponds to the 1 bit SIB register that is between each SIBs TDI port and its instrument (Figure 1.8(c)). The total number of such \(1_O\) bits is accounted for on the row marked Scan-sequence 1 in Table 2.1, in the column “SIBs”. Similarly, the number of bits (called \(3SC_1, 5SC_2, 4SC_3\) above) for the three instruments are counted in the columns SC\(_1\), SC\(_2\) and SC\(_3\). After four test patterns have been applied, the test for instrument SC\(_2\) is complete and its scan chain should be excluded from the scan-path by setting the control bit so that SIB\(_2\) is closed while keeping SIB\(_1\) and SIB\(_3\) open. Closing SIB\(_2\) cannot occur until the test response for the last test pattern of SC\(_2\) has been scanned out. Therefore, a fifth scan-sequence is required during which the
last test response of SC₂ is scanned out and the SIB control bits to exclude SC₂ from the scan-path are scanned in. The sixth scan-sequence, has a total scan-path length of $1_C + 3_{SC1} + 1_C + 1_O + 4_{SC3} = 10$ bits (Table 2.1, the row marked Scan-sequence 6). Here, $1_C$ corresponds to the 1 bit register between the TDI and TDO ports of a closed SIB (Figure 1.8(b)). The scan-path is as shown in Figure 2.2(c). After the sixth scan-sequence, the test for instrument SC₁ is complete and SIB₁ is closed. The scan-path becomes as shown in Figure 2.2(d). For Pattern 7 to Pattern 11, four test patterns remain for instrument SC₃ and one scan-sequence is used to scan out the last of the test responses for instrument SC₃, while closing SIB₃. For these last five test patterns the total scan-path length is $1_C + 1_C + 1_O + 4_{SC3} = 7$ bits.

Table 2.1 shows the number of bits of different types (columns) that are scanned in for each test pattern (rows). The column marked $\sum$ sums the bits to get the total scan-path length. These are the number of bits that
2.1. Test Application Time for the Flat Architecture

Table 2.1: Flat test architecture, concurrent schedule

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>SIBs</th>
<th>SC1</th>
<th>SC2</th>
<th>SC3</th>
<th>∑</th>
<th>+CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup-sequence</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Scan-sequence 1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Scan-sequence 2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Scan-sequence 3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Scan-sequence 4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Scan-sequence 5</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Scan-sequence 6</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Scan-sequence 7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Scan-sequence 8</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Scan-sequence 9</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Scan-sequence 10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Scan-sequence 11</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td><strong>TAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∑=183</td>
<td></td>
</tr>
</tbody>
</table>

are scanned for each scan-sequence. The last column shows the number of bits to scan in for each scan-sequence plus the five clock cycles that are required to perform a capture-and-update cycle (CUC) for JTAG [9] (see Section 1.3.2). Calculating the test application time is to sum up the values in the last column of Table 2.1, as shown on the last row. In this example, the test application time is 183 clock cycles.

Equation 2.1 gives the test application time (TAT) for the flat test architecture and the concurrent schedule, when provided with the values for the following variables:

- $N$ is the number of instruments.
- $S$ is the number of SIBs (in practice the same as $N$ for the flat test architecture).
- $P_i$ is the number of test patterns for instrument $i$. The instruments are sorted in ascending order on the number of test patterns and enumerated accordingly. (In the example of Figure 2.1, this leads to the order SC2:SC1:SC3.)
Test Time Calculation

- $L_i$ is the scan-chain length of instrument $i$.
- $C$ is the number of clock cycles required for a capture-and-update cycle. The value for $C$ is five for the analysis in this paper and this value is given by the JTAG TAP state machine (see Section 1.3.2).

\[ TAT = C + S + \sum_{i=1}^{N} (P_i - P_{i-1}) \cdot \left( C + S + \sum_{j=i}^{N} L_j \right) \] (2.1)

Equation 2.1 quite literally represents the tabulation of the shifted bits as in Table 2.1. Firstly, $C + S$ corresponds to the Setup-sequence. Secondly, $S + \sum_{j=i}^{N} L_j$ corresponds to the number of shifted bits for each scan-sequence, and $C$ gives the number of clock cycles for the CUC of each scan-sequence. Finally, $P_i - P_{i-1}$ describes the number of lines that have the same active instruments, and therefore the same number of scanned bits in the scan-sequences, as indicated by the horizontal lines in Table 2.1. It should be noted, that for $i = 1$, $P_{i-1} = P_0 = -1$ and there is no instrument 0. This notation is used to mark the additional scan-sequence for shifting out the test responses for each instrument. Therefore, there are five lines for $i = 1$, where all instruments are included in the scan-path. The five lines correspond to the four test patterns of SC$_2$ and one additional scan-sequence. Then comes a single line because for $i = 2$, $P_i - P_{i-1} = P_{SC1} - P_{SC2} = 5 - 4 = 1$. The same reasoning leads to five lines for the last group of scan-sequences.

From Equation 2.1 it is interesting to note how an increase in the number of patterns affects the ratio between amount of shifted test bits $\sum_{i=1}^{N} (P_i + 1) \cdot L_i$ and overhead in terms of scanned SIB control bits $S$ and CUC $C$. An increase in $P_i$ for the instrument with the most test patterns causes additional scan-sequences, which means that the total overhead increases in each scan-sequence with the SIB-overhead for the scan-sequence and a CUC. This increase will be most noticeable for scan sequences for which the number of shifted test data bits is low, for example when testing an instrument with a short scan chain. A similar observation should be made about an increase in $L$, the length of an instrument scan-chain. An increase in $L$, increases the test stimuli volume, but in contrast to the observation about $P$, an increase in $L$ will not increase the overhead in SIBs.
2.1. Test Application Time for the Flat Architecture

Figure 2.3: Steps to apply tests to a flat architecture using the sequential schedule or CUCs. Taken together, many test patterns and short scan chains will lead to a relatively large amount of overhead, whereas long scan-chains will effectively limit the relative amount of overhead.

2.1.2 Sequential Test Schedule

In this section, TAT will be calculated for the flat test architecture considering the sequential test schedule. Figure 2.3 and Table 2.2 will be used to explain the steps taken to calculate TAT. Before the test process starts, the scan-path is as shown in Figure 2.3(a), and three bits are used in the first sequence (the row marked Setup-sequence in Table 2.2) to open SIB$_1$ so that for the six following scan-sequences, the scan-path is as shown in Figure 2.3(b). The row marked Scan-sequence 1-6 in Table 2.2 shows that the three bits of SC$_1$ are included in scan-path. After Scan-sequence 6, the five test patterns of the test for SC$_1$ have been applied and the test responses
Table 2.2: Flat test architecture, sequential schedule

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>SIBs</th>
<th>Scanned bits</th>
<th>Scanned bits +CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SIBs</td>
<td>SC1</td>
<td>SC2</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scan-sequence 1-6</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Scan-sequence 7-11</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Scan-sequence 12-22</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

have been scanned out while closing SIB1 and opening SIB2 so that the scan-path becomes as shown in Figure 2.3(c). For this configuration of the scan-path, four test patterns are applied to complete the test for instrument SC2 followed by a scan-sequence to scan out the last test responses (Scan-sequence 7-11 in Table 2.2). Figure 2.3(d) shows the scan-path as it is after Scan-sequence 11. Finally, Scan-sequence 12-22 (Table 2.2) are applied to complete the test for SC3 and in the final scan-sequence the last test responses are scanned out and SIB3 is closed.

As can be seen from Table 2.2, TAT for the sequential schedule is 271 clock cycles, which should be compared to 183 clock cycles for the concurrent schedule discussed in Table 2.1. In the two cases, the number of test patterns and the length of the scan-chains were the same. The difference in TAT can be explained by a larger number of scan-sequences performed in the sequential schedule, which leads to more SIB and CUC overheads.

TAT for the sequential schedule and the flat test architecture is given by Equation 2.2. The equation is the sum of the test times for the individual instruments. For each instrument, the test time is calculated by multiplying the number of clock cycles spent on each test pattern by the number of patterns (including shifting out the responses of the last test stimuli). The number of clock cycles spent on each test pattern is the sum of the length of the instrument \( L_i \), all the number of SIBs on the scan-path \( S \) and the number of clock cycles spent in the capture-and-update cycle of each scan-sequence \( C \). Similar to in Equation 2.1, the first \( S \) and \( C \) terms correspond to the first test row of Table 2.2 marked Setup-sequence.
2.2 Test Application Time for the Hierarchical Architecture

Figure 2.4: A sample network with a hierarchical architecture having scan-chains as instruments

$$TAT = C + S + \sum_{i=1}^{N} ((C + S + L_i) \cdot (P_i + 1))$$ \hspace{1cm} (2.2)

2.2 Test Application Time for the Hierarchical Architecture

Figure 2.4 shows a small P1687 network with a hierarchical architecture. In this network, the scan-chains SC1, SC2, and SC3 are the same as those in the Figure 2.1, i.e. the same set of instruments are used here as well. This will allow for the comparison between the TAT of flat and hierarchical architectures for the same set of instruments.

Although in Section 2.1 formulas have been presented for test application time (TAT) calculation for flat architecture, it should be noted that it is not trivial to present formulas for TAT calculation for a hierarchical architecture. The reason is that in a P1687 network having hierarchical architecture, the length of the scan-path during the test of each instrument varies based on the placement of that instrument in the network. The situation becomes more complex in case of concurrent test schedule where
Table 2.3: Hierarchical test architecture, concurrent schedule

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Scanned bits</th>
<th>Scanned bits +CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SIBs</td>
<td>SC1</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Scan-sequence 1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Scan-sequence 2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Scan-sequence 3-6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Scan-sequence 7-13</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the scan-path for each instrument will change as soon as another layer of hierarchy is opened or testing another instrument is finished. So, not only the number of patterns and the length of scan-chains in instruments are required for TAT calculation, but the network itself should be taken into account which makes formulation of TAT complicated.

Here again, as was the case in Section 2.1, first the concurrent schedule will be studied for the design shown in Figure 2.4.

2.2.1 Concurrent Test Schedule

Regarding the concurrent schedule, Table 2.3 shows the steps to calculate TAT. Figure 2.5(a) shows the scan-path when all SIBs are closed, before the test process starts. The Setup-sequence row in Table 2.3 represents the control bits that open SIB1 and SIB2, leading to the scan-path in Figure 2.5(b). In Scan-sequence 1, the first pattern is applied for instrument SC1, while SIB3 and SIB4 are opened, leading to Figure 2.5(c). Similarly, Scan-sequence 2 represents the second test pattern for SC1 and the first test pattern for SC2, while SIB5 is opened so that in the following four scan-sequences (Scan-sequence 3-6 in Table 2.3) all five SIBs are open and the scan-path is as shown in Figure 2.5(d). After Scan-sequence 6, both SC1 and SC2 have been tested completely and for SC3 remains six test patterns plus the scan-sequence to scan out the last test response (Scan-sequence 7-13 in Table 2.3). For these seven scan-sequences, the scan-path is as shown in Figure 2.5(e).

As can be seen from Table 2.3, TAT for the hierarchical test archi-
Figure 2.5: Steps to apply tests to a hierarchical architecture using the concurrent schedule
Test Time Calculation

Architecture and the concurrent schedule is 223 clock cycles, which should be compared to 183 clock cycles for the corresponding test schedule and the flat test architecture. In this example, the hierarchical test architecture leads to a longer TAT because of two factors. Firstly, the overhead from the additional SIBs affects TAT, in particular for the instruments that are on a level of hierarchy far away from the primary TDI and TDO ports, such as instrument SC₃ in the example. Secondly, the overhead in terms of capture-and-update cycles (CUC) is higher, because of the scan-sequences required to open up all the SIBs on all the levels of hierarchy.

From Table 2.3, it should be noted that the SIB overhead varied according to the number of hierarchy levels that are included in the scan-path. For the Setup-sequence, only the first level of hierarchy is open and the SIB overhead is two. For Scan-sequence 1, the SIB overhead is four, corresponding to two open levels of hierarchy with two SIBs each, and for Scan-sequence 2-13, the SIB overhead is five clock cycles per scan-sequence.

2.2.2 Sequential Test Schedule

In the following the sequential schedule will be considered and TAT will be calculated for the hierarchical test architecture, with the help of Figure 2.6 and Table 2.4. Similar to the Setup-sequence of the previously discussed schedule, two control bits are shifted in to change the scan-path from Figure 2.5(a), but with the sequential schedule, only SIB₁ is opened, leading to the scan-path in Figure 2.6(b). With this scan-path, six scan-sequences are applied to complete the test for instrument SC₁, as described by Scan-sequence 1-6 in Table 2.4. At this point, an additional Setup-sequence is required to configure the scan-path in two steps, via Figure 2.6(c) to Figure 2.6(d). Subsequently, Scan-sequence 7-11 complete the test for SC₂. Yet another Setup-sequence produces the scan-path in Figure 2.6(e) and then the scan-path in Figure 2.6(f). The last 11 scan-sequences complete the test for SC₃ and TAT is 310 clock cycles. This result should be compared with 271 clock cycles for the sequential schedule and the flat test architecture. The reason for the higher TAT with the hierarchical test architecture is more SIB-overhead and more CUCs. Especially, two extra Setup-sequences add to the TAT.

In Section 2.1.1 it was discussed how TAT and the overhead ratio scale
Figure 2.6: Steps to apply tests to a hierarchical architecture using the sequential schedule
Table 2.4: Hierarchical test architecture, sequential schedule

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Scanned bits</th>
<th></th>
<th></th>
<th>∑</th>
<th>Scanned bits +CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SIBs</td>
<td>SC₁</td>
<td>SC₂</td>
<td>SC₃</td>
<td></td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Scan-sequence 1-6</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Scan-sequence 7-11</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Scan-sequence 12-22</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Table 2.4: Hierarchical test architecture, sequential schedule

The observations made there apply also to the hierarchical test architecture with a few modifications. The number of scan-sequences, and therefore the CUC overhead, depends on the level of hierarchy for an instrument and on the number of scan-sequences spent only on configuring SIBs (such as in Table 2.4).

It should be noted that in the example discussed in Section 2.1 and this section, the flat test architecture and the concurrent schedule led to the lowest TAT. This is not a general conclusion, since other examples may show lower TAT on other test architectures and test schedules.

### 2.3 Test Time Calculation Method: IJTAG-calc

This section will describe a method called IJTAGcalc for calculation of test application time (TAT) for a given P1687 network and a given test schedule which can be either concurrent or sequential. The method consists of two sets of algorithms corresponding to the concurrent and the sequential test schedules.

The terminology used in the algorithms, when defining variable names, is from a tree structure. The JTAG TAP is the root of the tree and the SIBs define the nodes. For each SIB $s$, there is a subtree of SIBs that are accessed through the HIP of $s$. This subtree is empty in case the HIP only
34

2.3. Test Time Calculation Method: IJTAGcalc

Table 2.5: Variables Associated with a SIB

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILength*</td>
<td>Length of an instrument (0 if no instrument)</td>
</tr>
<tr>
<td>IPatterns*</td>
<td>Number of patterns of an instrument (Constant)</td>
</tr>
<tr>
<td>IRemaining*</td>
<td>Remaining patterns of an instrument (-1 if test complete)</td>
</tr>
<tr>
<td>SRemaining</td>
<td>Remaining patterns in a subtree (-1 if test complete)</td>
</tr>
<tr>
<td>Children</td>
<td>Set of children (SIBs)</td>
</tr>
<tr>
<td>IsOpen</td>
<td>State of the SIB (Boolean)</td>
</tr>
</tbody>
</table>

connects to an instrument. SIBs that are in the subtree of s and on the next hierarchy level are referred to as children of s. An example is shown in Figure 1.12(b), where SIB2 has the subtree consisting of SIB3, SIB4 and SIB5. SIB2, which is on Level 1, is the parent of SIB3 and SIB4, since they are on Level 2.

So far, the structure of SIBs in P1687 can be described by the tree mentioned above. To describe the placement of instruments, it is considered that each SIB can have at most one instrument and it will be connected in series with its children SIBs. To illustrate this, consider SIB2 in Figure 1.12(a), which has an instrument SC2 but no children SIBs. In Figure 1.12(b), SIB2 has no instrument but two children SIBs (SIB3 and SIB4). Furthermore, in Figure 1.12(c), SIB2 has an instrument SC2 and one child SIB (SIB3).

Table 2.5 describes the variables used in the algorithms. Each SIB has the variables shown in Table 2.5. The variables ILength, IPatterns and IRemaining (marked with *) represent any instrument connected to the SIB. If an instrument is connected to a SIB s, then s.ILength, s.IPatterns and s.IRemaining define the properties of the instrument. s.IPatterns and s.IRemaining are initially set to the number of patterns of the instrument. For each test stimuli that is applied, s.IRemaining is decremented. When s.IRemaining has reached 0, the final test response for the instrument is to be shifted out. Therefore, a negative number in s.IRemaining represents that the instrument has been completely tested. If SIB s has no instruments, s.ILength and s.IPatterns are set to 0, and s.IRemaining is set to -1, so that it can be handled the same way as an instrument that is already completely tested. The variable s.SRemaining for a SIB s will
Algorithm 1 IJTAGcalcConcurrent

1: for each SIB $s$ do
2:     $SRemaining := \max\{IPatterns \text{ found in subtree of } s\}$
3: end for
4: while $TAP.SRemaining > -1$ do
5:     $SSLength:=0$ // Scan sequence length
6:     $TAP.SRemaining:=\text{Traverse}(TAP)$
7:     $TAT := TAT + SSLength + CUC$
8: end while

at all times hold the maximum of the value of the $IRemaining$-variables over all the SIBs in the subtree of $s$. In practice, this means that when $s.SRemaining$ reaches a negative value, SIB $s$ can be closed, since there are no more patterns to apply for the SIBs in the subtree, and no more test responses to shift out.

2.3.1 IJTAGcalc for the Concurrent Test Schedule

This section describes the IJTAGcalc method for the concurrent test schedule as shown in Algorithm 1 IJTAGcalcConcurrent and Algorithm 2 Traverse. On the first three lines of Algorithm 1, the $SRemaining$ variable is initialized for all the SIBs. The remaining lines in Algorithm 1 describe a loop where each iteration contains a call to Traverse (Algorithm 2). Each iteration corresponds to a scan-sequence (see Table 2.1) and by summing the number of bits in each scan-sequence ($SSLength$) with the CUC for each scan-sequence, the test application time $TAT$ is added up (penultimate line). The iterations finish, when there are no more test patterns to apply, as given by the $SRemaining$ variable. At this point, the test application time $TAT$ will have been found.

As can be seen in IJTAGcalcConcurrent, Algorithm 1, Traverse is an important function, which returns the value for $SRemaining$. It also updates the $SSLength$ variable which keeps track of the number of bits that have been scanned in during each scan sequence. It should be noted that $TAT$ is the test application time and that $CUC$ is the capture-and-update cycle time, typically five clock cycles. Traverse is shown in Algorithm 2. The basic operation of the Traverse function is to inspect the children nodes
of the node that was used to call Traverse, and for these children nodes, the number of remaining test patterns is calculated as the return value of Traverse. Since each child is a SIB, the \textit{SSLength} variable is incremented by one to represent the time it takes to scan in a control bit for the SIB (line 3). If the SIB is closed but there are still test patterns to be applied to any instrument in its subtree (as indicated by the \textit{SRemaining} and \textit{IRemaining} variables, line 4), the SIB is opened (line 6). In the opposite situation, when there are no more test patterns to be applied for the subtree of a SIB, that SIB is closed (line 13). For an open SIB with remaining test patterns, a recursive call to Traverse (Algorithm 2) is performed (line 8). The \textit{SSLength} variable is incremented by \textit{ILength} which signifies the shifting of the bits of one test stimuli while reducing the number of remaining test patterns by one (lines 9 and 10 respectively). The number of remaining test patterns for the subtree for which Traverse was called is calculated by taking the maximum number of test patterns remaining for

\begin{algorithm}
\caption{Traverse(node)}
\begin{algorithmic}[1]
\State \textit{subtreeSPatternList} := \{-1\}
\For{each child \in node.children}
\State \textit{SSLength} := \textit{SSLength} + 1
\If{\textit{child.SRemaining} > -1 \textbf{or} \textit{child.IRemaining} > -1}
\If{\textit{child.IsOpen} = \text{False}}
\State \textit{child.IsOpen} := \text{True}
\Else
\State \textit{child.SRemaining} := \text{Traverse}(\textit{child})
\State \textit{SSLength} := \textit{SSLength} + \textit{child.ILength}
\State \textit{child.IRemaining} := \textit{child.IRemaining} − 1
\EndIf
\Else
\State \textit{child.IsOpen} := \text{False} // might already be closed
\EndIf
\State \text{append} \ \max\{\textit{child.SRemaining}, \ \textit{child.IRemaining}\} \ \text{to} \ \textit{subtreeSPatternList}
\EndFor
\State \textbf{return} \ \max\{\textit{subtreeSPatternList}\}
\end{algorithmic}
\end{algorithm}
any of the child nodes (line 15 and line 17).

### 2.3.2 IJTAGcalc for the Sequential Test Schedule

This section describes the IJTAGcalc method for the sequential test schedule. The algorithm is called IJTAGcalcSequential and is shown in Algorithm 3. IJTAGcalcSequential considers the same type of tree representation as was discussed in Section 2.3. The key idea which makes the test application time calculation possible is that there are \( P_i + 1 \) scan sequences for each instrument \( i \), for which the number of shifted bits per scan-sequence is constant. This can be seen in Table 2.4. The number of shifted bits during the tests depends on the length of the scan chain of the tested instrument and the hierarchy level.

To calculate \( TAT \), IJTAGcalcSequential should be called with \( TAP \) as parameter (the root node of the tree). Before the call to IJTAGcalcSequential, the variables \( SIBs \), \( TotILength \) and \( TAT \) should be set to 0. Here, \( SIBs \) is a variable that counts the number of SIBs on the scan-path, which will vary depending on what instrument is being tested. \( TotILength \) should be explained and understood in relation to the alternative architecture shown in Figure 1.12(c). It can be seen in the figure, that for testing Scan Chain 3, the test data should pass through Scan Chain 2. In the context of the sequential test schedule, dummy bits should be shifted in for Scan Chain 2 during the test of Scan Chain 3. \( TotILength \) counts the number of the dummy bits that are shifted in each scan-sequence. Finally, \( TAT \) is the variable that will contain the test application time when IJTAGcalcSequential terminates.

As mentioned above, the number of SIBs on the scan-path will vary according to the location of the instrument that is being tested within the P1687 network. Therefore, IJTAGcalcSequential (Algorithm 3), keeps track of the SIBs that must be traversed to reach the level of hierarchy on which the tested instrument is located. Each level of hierarchy is marked by a recursive call (line 6). When the IJTAGcalcSequential function is called, it enters a previously not visited level of hierarchy and therefore \( SIBs \) is incremented with the number of SIBs on this level (line 1). Similarly, when the call is complete (line 10), the function leaves that same level of hierarchy, and \( SIBs \) is reduced to the previous value, corresponding to
the previous level of hierarchy. In each call to the function, the parameter node will be a SIB. If the SIB has an instrument on its HIP, then TAT will be incremented with the test time required for applying all the instrument’s patterns and the scan-out of the last test response (line 2). This is similar to the grouping of scan-sequences in Table 2.4. For example scan-sequences 1-6 correspond to the testing of instrument SC1. If the SIB passed as the node parameter has no instrument on its HIP, then this implies that this SIB needs to be opened to reach another level of hierarchy, such as SIB2 in Figure 1.12(b). Here, node.ILength and node.IPatterns are both 0 and TAT is increased by the sum of SIBs, TotILength and CUC. These correspond to the number of SIB control bits and the dummy bits that must be shifted in to reach the considered SIB. If the SIB passed as the node parameter has children SIBs, the IJTAGcalcSequential function will be called recursively for each of these children. To take possibility of an instrument connected in series with the children SIBs into account, TotILength is adjusted accordingly.

**Algorithm 3** IJTAGcalcSequential(node)

1. $SIBs := SIBs + \text{size} \left( node.\text{Children} \right)$
2. $TAT := TAT + \left( node.ILength + SIBs + \text{TotILength} + \text{CUC} \right) \cdot \left( node.\text{IPatterns} + 1 \right)$
3. if $\text{size} \left( node.\text{Children} \right) > 0$ then
4. $\text{TotILength} := \text{TotILength} + node.ILength$
5. for each child $\in \text{Children} \left( node \right)$ do
6. IJTAGcalcSequential(child)
7. end for
8. $\text{TotILength} := \text{TotILength} – node.ILength$
9. end if
10. $SIBs := SIBs – \text{size} \left( node.\text{Children} \right)$
2.4 IEEE Standard 1500 Wrapped Cores As Instruments

In previous sections of this chapter, test application time (TAT) for P1687 networks was studied, formulas were presented for the flat architecture and algorithms were presented for all three architectures in Figure 1.12. All the studied networks in the previous sections had scan-chains as their instruments. In this section, wrapped cores as instruments will be studied. Also, the presented formulas and algorithms will be accordingly modified. It will be assumed that each wrapped core is interfaced to the P1687 circuitry as explained in Section 1.5.4. Therefore, to test a desired WDR, the following steps should be taken: The SIB connected to the core wrapper is programmed to have both Select and SelectWIR signals high. This requires shifting appropriate control data followed by a CUC. Next, the instruction data for the selection of the WDR should be shifted into the wrapper instruction register (WIR) followed by another CUC. This will be referred to as instruction loading in the following sections. In this report, the length of the WIR for all of the wrapped cores will be assumed to be 10. While the instruction data are shifted into the WIR, the wrapper SIB should be programmed to have its SelectWIR signal deactivated. After the CUC, the instruction is decoded and the desired WDR is selected and connected to the WSI-WSO path of the wrapper (and also SDI-SDO path of the wrapper SIB). The rest of the test application procedure is the same as the procedure mentioned in the previous sections of this chapter for the scan-chains. The key difference is that the wrapper SIB will represent two delays in TAT calculations.

It should be noted that the procedure mentioned above will be elaborated on more in the following sections, where test application for flat and hierarchical architectures having wrapped cores as instruments will be presented in the context of concurrent and sequential schedules. To be able to make comparisons (in Section 2.4.5) between an architecture having scan-chains as instruments and its counterpart having wrapped cores as instruments, the same scan-chains will be considered to be the WDRs inside each wrapped core, i.e. the same length and the same number of patterns. In Section 2.4.6 and Section 2.4.7 algorithms will be presented.
2.4. IEEE Standard 1500 Wrapped Cores As Instruments

![Diagram of a sample network with a flat architecture having wrapped cores as instruments.](image)

Figure 2.7: A sample network with a flat architecture having wrapped cores as instruments for the test time calculation when the instruments in a P1687 network are IEEE 1500 standard wrapped cores, and for concurrent and sequential schedules, respectively.

2.4.1 Flat Architecture, Concurrent Test Schedule

Figure 2.7 shows a P1687 network in a flat architecture having wrapped cores as instruments, and Figure 2.8 shows the steps that should be taken to apply tests to it. In Figure 2.7, SIBs are wrapper SIBs described in Section 1.5.4 and gray boxes are wrapped cores explained in Section 1.4. Table 2.6 will be used to describe TAT calculations. The structure of the table is quite similar to those used in Sections 2.1.1, 2.1.2, 2.2.1 and 2.2.2. The only difference is that a “WIRs” column is added which shows the number of shifts, in the scan sequence, spent on loading instructions into the WIR.

Initially, the circuit is as shown in Figure 2.8(a). These wrapper SIBs should be programmed to become open and have their SelectWIR signal active, which is done by scanning in six bits followed by a CUC. This is represented by the Setup-sequence row in Table 2.6. After the CUC, all wrapped cores are selected and have their WIRs connected to the TDI-TDO path, as shown in Figure 2.8(b). To load the instruction into WIRs of the cores, 30 bits of instruction data should be shifted in, as well as...
six bits to program the SIBs to remain open and have their SelectWIR disabled. After another CUC, the loaded instructions are decoded and the WDRs (here scan-chains) are selected and connected to the TDI-TDO path. This is marked by the row Instruction loading in Table 2.6. Once the scan-chains are selected and connected to the TDI-TDO path, the rest of the test application procedure will be the same as what was explained in Section 2.1.1 which is marked by rows Scan-sequence 1-11 in Table 2.6. It should be noted that in contrast to Table 2.1 where SIBs represented three shifted bits, here SIBs represent six bits.

Equation 2.3 is the modified version of the Equation 2.1 to reflect the effect of the wrapper SIBs and instruction loading. The variable definitions are the same as those mentioned for Equation 2.1 and the new $W_i$ variable is the length of the WIR of wrapped core $i$. One of the $C + 2 \cdot S$ terms represents the Setup-sequence in Table 2.6. This term differs from its counterpart in Equation 2.1 ($C + S$) in that here each SIB requires two bits to be configured. The other $C + 2 \cdot S$ term and $\sum_{i=1}^{N} W_i$ together represent the time taken to load the instructions into the WIRs of the wrapped cores, as marked by the Instruction loading row of Table 2.6. These two terms were not present in Equation 2.1 and are new here. The last summation term, was present in Equation 2.1 but is again modified to reflect that each wrapper SIB requires two bits to be configured. This summation term is represented by rows Scan-sequence 1 to Scan-sequence 11 in Table 2.6.

The effect of scaling the number of patterns as was presented for Equation 2.1 is also valid here. However, the effect of increasing the number of patterns on overhead is more significant in case of the wrapped cores, due to the coefficient of 2 for the $S$ parameter in Equation 2.3.

$$TAT = 2 \cdot (C + 2 \cdot S) + \sum_{i=1}^{N} W_i + \sum_{i=1}^{N} (P_i - P_{i-1}) \cdot \left( C + 2 \cdot S + \sum_{j=i}^{N} L_j \right)$$  \hspace{1cm} (2.3)$$

2.4.2 Flat Architecture, Sequential Test Schedule

As was shown in previous section, the procedure of applying tests to individual scan-chains was quite similar to the same procedure for scan-chains
2.4. IEEE Standard 1500 Wrapped Cores As Instruments

Figure 2.8: Steps to apply tests to a flat architecture using the concurrent schedule inside wrapped cores. The key difference between the two procedures was that with scan-chains being inside wrapped cores, an extra step was re-
As for the sequential schedule, TAT of P1687 network in Figure 2.7 shows an increase compared with that of the network in Figure 2.1. The reasons for this increase are (1) WIR overhead, (2) the doubled SIB overhead and (3) more CUCs for instruction loading.
Table 2.7: Flat test architecture, sequential schedule, wrapped cores

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Scanned bits</th>
<th>SIBs</th>
<th>WIRs</th>
<th>SC1</th>
<th>SC2</th>
<th>SC3</th>
<th>∑</th>
<th>+CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup-sequence</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Instruction loading</td>
<td></td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Scan-sequence 1-6</td>
<td></td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>14.6</td>
</tr>
<tr>
<td>Instruction loading</td>
<td></td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Scan-sequence 7-11</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>16.5</td>
</tr>
<tr>
<td>Instruction loading</td>
<td></td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Scan-sequence 12-22</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>15.11</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>403</td>
</tr>
</tbody>
</table>

Equation 2.4 gives the test application time for the flat test architecture, wrapped cores and the sequential schedule. The term \( C + 2 \cdot S \), which is similar to the term \( C + S \) in Equation 2.2, corresponds to the initial Set-up of the SIBs, as marked by the Setup-sequence row of Table 2.7. The term \( (C+2\cdot S + W_i) \) represents the time required to load the scan-chain selection instruction into the WIR of the corresponding core, as shown by the rows marked Instruction load in Table 2.7. The \( (C+2\cdot S + L_i) \cdot (P_i + 1) \) term is similar to the term \( (C + S + L_i) \cdot (P_i + 1) \) in Equation 2.2, and is modified so that each wrapper SIB represents two shifts in the test application time calculation.

\[
TAT = C + 2 \cdot S + \sum_{i=1}^{N} \left( (C + 2 \cdot S + L_i) \cdot (P_i + 1) + (C + 2 \cdot S + W_i) \right)
\]

(2.4)

### 2.4.3 Hierarchical Architecture, Concurrent Test Schedule

Figure 2.9 shows a sample network with a hierarchical test architecture. In this network, two SIBs (i.e. SIB\(_2\) and SIB\(_4\)) are used to connect to another level of hierarchy and each requires one bit to be configured. Also, to connect the wrapped cores to the P1687 logic, three wrapper SIBs are used each requiring two bits for their configuration. The procedure of applying
Test Time Calculation

Figure 2.9: A sample network with a hierarchical architecture having wrapped cores as instruments

tests will be discussed by using Table 2.8 and Figure 2.10. Initially the circuit is as shown in Figure 2.10(a) and the SIBs should be opened before applying tests. Therefore, three bits (two bits for SIB\(_1\) and one bit for SIB\(_2\)) should be scanned in followed by a CUC. The Setup-sequence row in Table 2.8 represents this and Figure 2.10(b) shows the result. In the next step, SC\(_1\) should be selected and SIB\(_3\) and SIB\(_4\) should be opened. So, six bits (four bits for SIB\(_1\) and SIB\(_3\), and two bits for SIB\(_2\) and SIB\(_4\)) should be scanned in to configure the SIBs and 10 bits of instruction data should be scanned in the WIR. After applying a CUC the circuit will be as shown in Figure 2.10(c). The Instruction loading row reflects the step just mentioned. Now SC\(_1\) is selected and the first test stimuli can be scanned in for it. This can be done along with the instruction loading for selection of SC\(_2\) and programming the SIBs. This time, in addition to the previously opened SIBs, SIB\(_5\) should also be opened, which makes the total SIB programming bits become eight. This is represented by the row marked Scan-sequence 1 in Table 2.8 and the circuit is as shown in Figure 2.10(d). At this stage, the second test stimuli for SC\(_1\), and the first
Table 2.8: Hierarchical test architecture, concurrent schedule, wrapped cores

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>SIBs</th>
<th>WIRs</th>
<th>SC₁</th>
<th>SC₂</th>
<th>SC₃</th>
<th>∑</th>
<th>+CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup-sequence</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Instruction loading</td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Scan-sequence 1</td>
<td>8</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>Scan-sequence 2</td>
<td>8</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>26</td>
<td>31</td>
</tr>
<tr>
<td>Scan-sequence 3-6</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>25·4</td>
</tr>
<tr>
<td>Scan-sequence 7-13</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>17·7</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∑=305</td>
</tr>
</tbody>
</table>

Test stimuli for SC₂ can be shifted in while the response to the first stimuli of SC₁ is scanned out. Again, this can be done along with loading the instruction for selection of SC₃ into its corresponding WIR, as shown in row Scan-sequence 2 in Table 2.8. After applying a CUC, the circuit will be as shown in Figure 2.10(e). Now, both SC₁ and SC₂ have three test stimuli left to be applied and one last response to be scanned out. This is done along with applying the first four stimuli of SC₃ which is now selected. The row Scan-sequence 3-6 in Table 2.8 represents this step. At the same time that the response to the last stimuli of SC₁ and SC₂ is scanned out, SIB₁ and SIB₃ can be closed. After closing SIB₁ and SIB₃ the circuit will be as shown in Figure 2.10(f). After this, the remaining patterns of SC₃ are left to be applied which is represented by row Scan-sequence 7-13 in the Table 2.8.

2.4.4 Hierarchical Architecture, Sequential Test Schedule

The test application procedure for the sequential schedule and the network shown in Figure 2.9 consists of three similar steps, one for each scan-chain. Each of these steps for each instrument consists of three sequences: (1) Configuring the scan-path, (2) loading the instruction into WIR and (3) applying the tests to the scan-chain. This similarity could also be seen in Table 2.4. There, in Section 2.2.2, each step consisted of only a scan-path configuration sequence (Setup-sequence) and test application sequence...
Figure 2.10: Steps to apply tests to a hierarchical architecture using the concurrent schedule (Scan-sequence). Table 2.9 shows the test application procedure for the network shown in Figure 2.9 and in the context of a sequential schedule. Due to the similarity between the test application procedure for each of
2.4. IEEE Standard 1500 Wrapped Cores As Instruments

Table 2.9: Hierarchical test architecture, sequential schedule, wrapped cores

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>SIBs</th>
<th>WIRs</th>
<th>SC1</th>
<th>SC2</th>
<th>SC3</th>
<th>∑</th>
<th>Scanned bits +CUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup-sequence</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Instruction loading</td>
<td>3</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>Scan-sequence 1-6</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>11·6</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Instruction loading</td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Scan-sequence 7-11</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>16·5</td>
</tr>
<tr>
<td>Setup-sequence</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>Instruction loading</td>
<td>8</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>Scan-sequence 12-22</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>17·11</td>
</tr>
<tr>
<td>TAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∑=427</td>
</tr>
</tbody>
</table>

scan-chains, not all the test application steps are shown in Figure 2.11. Initially the circuit is as shown in Figure 2.11(a). Three bits should be scanned in to configure SIB$_1$ to be opened and SIB$_2$ to remain closed. The Setup-sequence row in Table 2.9 shows the three bits as well as the CUC that follows. After the CUC, the circuit will be as shown in Figure 2.11(b). In the next step, SC$_1$ should be selected by loading the selection instruction in the WIR of the wrapper containing SC$_1$. This is represented by row Instruction loading in Table 2.9. After the CUC, SC$_1$ will be selected and ready to be tested, as shown in Figure 2.11(c). The Scan-sequence 1-6 row in Table 2.9 represents the bits that should be scanned in and out to complete the tests for SC$_1$. As the last response of SC$_1$ is being scanned out, configuration bits should be scanned in to close SIB$_1$ and open SIB$_2$. After the CUC, the circuit will be as shown in Figure 2.11(d). The test application for SC$_2$ and SC$_3$ will be similar to that for SC$_1$ and as can be seen from Table 2.9.

2.4.5 Comparison with scan-chains as instruments

Test application time calculations in Sections 2.4.1, 2.4.2, 2.4.3, and 2.4.4 show that for all four studied cases where instruments were wrapped cores, TAT has increased compared with respective counterpart of each case hav-
Figure 2.11: Steps to apply tests to SC1 in the hierarchical architecture shown in Figure 2.9

ing scan-chains as instruments. The increase in TAT is caused by the following factors: 1) WIR overhead which did not exist in case of instruments being scan-chains, (2) the doubled SIB overhead due to the use of wrapper SIBs and (3) more CUC overhead caused by instruction loading for the wrappers.

2.4.6 IJTAGcalc for the Concurrent Test Schedule

In Section 2.3.1, the IJTAGcalcConcurrent algorithm (see Algorithm 1) was proposed for the test time calculation of a P1687 network in the con-
Algorithm 4 Traverse(node)

1: subtreeSPatternList := \{-1\}
2: for each child ∈ node.children do
3:   if child.IsWrapperSIB = True then
4:     SSLength := SSLength + 2
5:   else
6:     SSLength := SSLength + 1
7:   end if
8:   if child.SRemaining > −1 or child.IRemaining > −1 then
9:     if child.IsOpen = False then
10:        if child.IsWrapperSIB = True then
11:           if child.IsWIRSelected = False then
12:              child.IsWIRSelected := True
13:           else
14:              SSLength := SSLength + WIRLength
15:              child.IsOpen := True
16:          end if
17:        else
18:            child.IsOpen := True
19:        end if
20:     else
21:        child.SRemaining := Traverse(child)
22:        SSLength := SSLength + child.ILength
23:        child.IRemaining := child.IRemaining − 1
24:     end if
25:   else
26:     child.IsOpen := False // might already be closed
27: end if
28: append max{child.SRemaining, child.IRemaining} to subtreeSPatternList
29: end for
30: return max{subtreeSPatternList}
Table 2.10: Variables Associated with a SIB

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILength*</td>
<td>Length of an instrument (0 if no instrument)</td>
</tr>
<tr>
<td>IPatterns*</td>
<td>Number of patterns of an instrument (Constant)</td>
</tr>
<tr>
<td>IRemaining*</td>
<td>Remaining patterns of an instrument (-1 if test complete)</td>
</tr>
<tr>
<td>SRemaining</td>
<td>Remaining patterns in a subtree (-1 if test complete)</td>
</tr>
<tr>
<td>Children</td>
<td>Set of children (SIBs)</td>
</tr>
<tr>
<td>IsOpen</td>
<td>State of the SIB (Boolean)</td>
</tr>
<tr>
<td>IsWrapperSIB</td>
<td>Indicates if the SIB is a wrapper SIB (Boolean)</td>
</tr>
<tr>
<td>IsWIRSelected</td>
<td>State of the SelectWIR signal of the wrapper SIB (Boolean)</td>
</tr>
<tr>
<td>WIRLength</td>
<td>Length of the WIR of the wrapped core connected to the wrapper SIB</td>
</tr>
</tbody>
</table>

The first required update is considering that a wrapper SIB requires two bits to be configured. This is in contrast to the other types of SIB (i.e. doorway SIBs and instrument SIBs that connect to scan-chains) that required only one bit. To implement this update, Line 3 in Algorithm 2 is replaced by Line 3 to Line 7 in Algorithm 4. SSLength is a variable of concurrent schedule. Algorithm 1 contained a call to a function called Traverse (see Algorithm 2), which calculated the number of bits that have been scanned in for each scan sequence. Traverse also returned the remaining patterns in the subtree of the SIB passed as its node parameter. In this section changes will be proposed that make these algorithms capable of handling P1687 networks containing wrapper SIBs. Algorithm 1 does the variable initialization, sums up the total number of scanned bits and required CUCs, and also checks the termination condition. Since the actual network is traversed by the Traverse function, the SIBs and instruments in the network are hidden from Algorithm 1 and therefore no changes will be required in it. On the other hand, Algorithm 2 should be modified to recognize wrapper SIBs and take the instruction loading sequence into account in test time calculations. Algorithm 4 is the updated version of Algorithm 2 which uses three new variables (in addition to those defined in Table 2.5). Table 2.10 lists and describes the variables used in the updated algorithm. Three variables, i.e. IsWrapperSIB, IsWIRSelected and WIRLength, are new compared to Table 2.5. The IsWrapperSIB variable helps the algorithm identify a wrapper SIB, the IsWIRSelected variable is used to identify the sequence required to load the scan-chain selection instruction into WIR, and WIRLength holds the length of WIR.
that keeps the number of bits in each scan sequence. Line 3 in Algorithm 4 checks the type of the SIB and in case of a wrapper SIB, adds two bits to the scan sequence length (stored in $SSLength$), otherwise one bit will be added.

The other necessary change is taking the time required to perform the instruction loading into account. To implement this change, Line 6 in Algorithm 2 is replaced by Line 10 to Line 19 in Algorithm 4. The basic idea is to check the type of the SIB before opening it, and if the SIB is a wrapper SIB, postpone the opening of the SIB to the next scan sequence and in the current sequence add the length of the WIR, i.e. $WIRLength$ to the $SSLength$ variable. The procedure in this basic idea is not the same as the procedure explained in Section 2.4.1 and Section 2.4.3 for applying tests. There, it was explained that the wrapper SIB is opened at the same time as the SelectWIR signal is activated. However, it should be noted that the algorithm proposed here is for the test time calculation and is not meant to fully comply with the actual test application sequence. Here, $IsOpen$ is merely used to mark the start of applying tests to the scan-chain inside the wrapper. This is in contrast to the actual test application procedure where the SIB should be opened before accessing the WIR.

In the modified algorithm, i.e. Algorithm 4, Line 10 checks to see if the current child of node is a wrapped SIB. If so, the instruction load time should be taken into account. Variable $IsWIRSelected$, which is assumed to be initially False, is used in Line 11 and Line 12 to make sure that one scan sequence is dedicated to the activation of the SelectWIR signal. This is done by setting $IsWIRSelected$ to True in Line 12 and checking it in the next scan sequence in Line 11. In the next scan sequence, the instruction should be loaded into the WIR. The algorithm considers this by adding the length of the WIR, i.e. $WIRLength$, to $SSLength$ in Line 14. In the same sequence, the algorithm sets the $IsOpen$ variable to True and the rest of the test time calculation procedure, starting from Line 21 is the same as what was explained for Algorithm 2.

### 2.4.7 IJTAGcalc for the Sequential Test Schedule

The algorithm suggested in Section 2.3.2 for TAT calculation in the context of sequential schedule, i.e. Algorithm 3, cannot handle wrapper SIBs.
Algorithm 5 IJTAGcalcSequential(node)
1: \( SIBs := SIBs + \text{BitCount}(node) \)
2: \( TAT := TAT + (node.WIRLength + SIBs + TotILength + CUC) \)
3: \( TAT := TAT + (node.ILength + SIBs + TotILength + CUC) \cdot (node.IPATTERNS + 1) \)
4: \( \text{if} \ \text{size}(node.Children) > 0 \ \text{then} \)
5: \( \text{TotILength} := \text{TotILength} + node.ILength \)
6: \( \text{for each} \ child \in Children(node) \ \text{do} \)
7: \( \text{IJTAGcalcSequential(child)} \)
8: \( \text{end for} \)
9: \( \text{TotILength} := \text{TotILength} - node.ILength \)
10: \( \text{end if} \)
11: \( SIBs := SIBs - \text{BitCount}(node) \)

Algorithm 6 BitCount(node)
1: \( Bits := 0 \)
2: \( \text{for each} \ child \in Children(node) \ \text{do} \)
3: \( \text{if} \ child.IsWrapperSIB=\text{True} \ \text{then} \)
4: \( Bits := Bits + 2 \)
5: \( \text{else} \)
6: \( Bits := Bits + 1 \)
7: \( \text{end if} \)
8: \( \text{end for} \)
9: \( \text{return} \ Bits \)

This section suggests modifications to Algorithm 3 for handling wrapper SIBs. Algorithm 5 shows the necessary modifications. The first modification concerns \( SIBs \) which is the variable that counts the number of SIBs on the scan-path. In Line 1 of Algorithm 3, the \( SIBs \) variable is incremented by the number of the SIBs in the newly opened level of hierarchy, and since wrapper SIBs require two configuration bits, this line of algorithm is replaced by Line 1 in Algorithm 5. BitCount is a function, described by Algorithm 6, that iterates on the \textit{Children} of the SIB passed to it as its \textit{node} parameter and counts and returns the total number of bits required for the configuration of the scan-path. The second modification is adding
a new line, i.e. Line 2, to Algorithm 5 to take into account the time it takes to perform the instruction loading, as represented by the Instruction loading rows in Table 2.7 and Table 2.9.

2.5 Experiments with the ITC’02 Benchmarks

The test application time (TAT) calculation algorithms proposed in this chapter are implemented and used on Six ITC’02 SOC Test Benchmarks [16] to obtain experimental results on realistic industrial SOCs, see Table 2.11. The algorithms are slightly modified to report the SIB overhead and CUC overhead in addition to the test application time. Every SOC contains cores for which the number of test patterns, number of I/O terminals and the number and length of internal scan-chains are given. In the following sections, it will be described how these SOCs are used to perform experiments (Section 2.5.1), and the experimental results will be presented and discussed for the case where the cores inside each SOC are considered unwrapped (Section 2.5.2) and the case where the cores are considered wrapped (Section 2.5.3).

2.5.1 Experimental Setup

Since in the context of P1687 all test patterns are transported through a single wire, the scan-chains and boundary cells of each core are concatenated to form a core-chain. To calculate the number of the boundary scan cells, each input/output terminal is counted as one cell and every bidirectional terminal is counted as three cells [9] [17]. To consider the testing requirements for hierarchical cores, the boundary cells of each embedded child core will be appropriately included in the set of boundary cells of its direct parent core [17]. To do the concatenation, the number of the boundary scan cells is added to the sum of the lengths of the internal scan-chains for each core. Table 2.11 shows for each of the six SOCs the number of cores, the minimum and maximum of pattern counts and core-chain lengths found among the cores, as well as the amount of test data that needs to be shifted. The amount of test data to be shifted is calculated as shown in 2.5.
ShiftedTestData = \sum_{i=1}^{N} L_i \cdot (P_i + 1) \quad (2.5)

In 2.5, \(N\) is the number of cores, \(P_i\) and \(L_i\) are the number of patterns and the length of the core-chain for core \(i\), respectively. ShiftedTestData calculated this way, can be used to verify that the algorithms are performing the TAT calculation correctly. That is, since algorithms report TAT, SIB overhead and CUC overhead, ShiftedTestData can be calculated as ShiftedTestData = TAT − SIB overhead − CUC overhead. If ShiftedTestData calculated based on the output of the algorithms is the same as ShiftedTestData calculated using 2.5, the algorithms perform the TAT calculation correctly.

It should be noted that Module 0, which is mentioned in the ITC’02 benchmarks and is the top-level SOC design, is not included in the experiments. The reason is that terminals for this module correspond to SOC pins (which are connected to the JTAG BSR) and their test patterns are not applied using P1687 network. Furthermore, modules containing BIST-engines are excluded from the experiments since no test patterns are transported over P1687 network for these modules.

Two experiments were conducted for six of the ITC’02 benchmark designs assuming no IEEE standard 1500 wrappers for the cores. All six designs were included in an experiment with the flat test architecture and three of the designs (P22810, P34392 and A586710) were also used in an experiment with the hierarchical test architecture. The two experiments
2.5. Experiments with the ITC’02 Benchmarks

(a) Structure used to map hierarchical cores to a hierarchical test architecture

(b) Overview of design P34392

(c) Hierarchical test architecture in P34392_HIER

Figure 2.12: Example for creation of hierarchical test architectures

The results of these experiments will be presented and discussed in Section 2.5.2. The same set of experiments were repeated for the case that the cores have IEEE standard 1500 wrappers, and the results will be presented in Section 2.5.3. To distinguish the two sets of experiments, the terms unwrapped cores and wrapped cores will be used.

Figure 2.12: Example for creation of hierarchical test architectures

correspond to the two test schedule types discussed previously in this report, namely the concurrent test schedule and the sequential test schedule.

To create hierarchical test architectures, a structure as the one shown in Figure 2.12(a) has been applied to P22810, P34392 and A586710. These three designs have hierarchical cores and the children cores are assigned to a hierarchy level which is accessed through the parent core. An example is shown in Figure 2.12(b) and Figure 2.12(c) for design P34392_FLAT. In Figure 2.12(c), only a part of the architecture is shown. It should be noted that there is no correlation between the concept of hierarchy in hierarchical
cores and the concept of hierarchy in P1687. However, for the sake of experiments on the hierarchical test architecture, the author has chosen to make a one-to-one correspondence between the parent-child relationship in hierarchical cores and hierarchy levels in a P1687 network. It should be noted that hierarchy in terms of P1687 can be implemented in a huge variety of ways, but to calculate and analyze the test application time, one such implementation of hierarchy was required and the author has chosen to take inspiration from the notion of the hierarchical cores.

### 2.5.2 Experimental Results for Unwrapped Cores

Table 2.12 and Figure 2.13 show experimental results using the concurrent test schedule. Table 2.12 presents the number of test clock cycles spent on each of the components of the test application time, i.e. Shifted Test Data, SIB overhead and CUC overhead. In Figure 2.13 the results are presented as a normalized stacked column chart for the ratios between the amount of shifted test data, SIB overhead and CUC overhead, such that their sum (100%) gives the test application time. Since the overhead constitutes a small portion of test application time, the vertical axis is scaled from 60% to allow details to be visible. On the horizontal axis are nine designs and the six left-most designs have flat test architecture as indicated by the labels.

Figure 2.13 shows that F2126_FLAT and T512505_FLAT both have relatively low overhead. As mentioned in Section 2.1.1, many test patterns and short scan chains will lead to a relatively large amount of overhead, whereas long scan-chains will effectively limit the relative amount of over-
2.5. Experiments with the ITC’02 Benchmarks

Figure 2.13: Results with the concurrent schedule

head. Seen from another perspective, long scan chains and a small number of test patterns should result in relatively low overhead. This is the case for all the cores of F2126_FLAT. For T512505_FLAT, there are some cores that have short core-chains but in those cases the number of test patterns is also low. On the other hand, there is one core with a very long core-chain and the shifted test data for this core corresponds to about 90% of the overall shifted test data. Therefore, this core made such impact on test application time that the overhead from the other cores became negligible.

U226_FLAT contains some cores that have short core-chains and a large number of test patterns. The design also contains four cores that are tested by BIST-engines and the test time for these cores is much larger than for the other cores of U226. To show that tests with short scan chains and a large number of test patterns lead to a large amount of overhead, the four cores with BIST-engines have been excluded from the experiment. Indeed, the overhead ratio for the test application time of U226_FLAT is about 10%.

In A586710_F, there is a core with about two million test patterns and a core-chain length of 326 cells. However, this large number of patterns has not resulted in a large SIB overhead ratio. The reason is that the 326-cell core-chain is significantly longer than the number of SIBs in the scan path.
Table 2.13: Results with the sequential schedule

<table>
<thead>
<tr>
<th>Design</th>
<th>Shifted Test Data</th>
<th>SIB Overhead</th>
<th>CUC Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2126_FLAT</td>
<td>5,330,439</td>
<td>3,868</td>
<td>4,835</td>
</tr>
<tr>
<td>T512505_FLAT</td>
<td>165,400,967</td>
<td>325,841</td>
<td>52,555</td>
</tr>
<tr>
<td>U226_FLAT</td>
<td>252,929</td>
<td>40,475</td>
<td>40,475</td>
</tr>
<tr>
<td>P22810_FLAT</td>
<td>8,172,047</td>
<td>701,176</td>
<td>125,210</td>
</tr>
<tr>
<td>A586710_FLAT</td>
<td>843,806,808</td>
<td>10,709,500</td>
<td>10,709,500</td>
</tr>
<tr>
<td>P34392_FLAT</td>
<td>16,728,056</td>
<td>1,260,498</td>
<td>331,710</td>
</tr>
<tr>
<td>P22810_HIER</td>
<td>8,172,047</td>
<td>665,898</td>
<td>125,220</td>
</tr>
<tr>
<td>A586710_HIER</td>
<td>843,806,808</td>
<td>10,890,647</td>
<td>10,709,505</td>
</tr>
<tr>
<td>P34392_HIER</td>
<td>16,728,056</td>
<td>841,306</td>
<td>331,725</td>
</tr>
</tbody>
</table>

in A586710_F, which effectively limits the SIB overhead ratio corresponding to this core.

As for P22810_F and P34392_F, the maximum number of patterns among the cores are similar (see Table 2.11) resulting in similar CUC overheads for both designs. However, the ratios of CUC overhead are not similar in these two designs due to the differences in their amounts of shifted test data.

The observations regarding the designs with flat test architecture typically applies also to the corresponding designs (from the same SOC) with hierarchical test architecture, as can be seen in Figure 2.13. It should be noted that even though there was a noticeable difference between the test application time of the flat architecture and that of the hierarchical test architecture for the small example of Figure 2.1 (see Section 2.2), the experiments with SOC benchmarks show little difference in overhead ratio. In this context it is worth noting that the hierarchical test architectures in this experiment only had two levels of hierarchy. Furthermore, the hierarchical test architecture was artificially designed for the experiment, without any optimization, and therefore might not represent a well-designed hierarchical test architecture.

For designs with a large number of cores, such as P22810_HIER, the ratio between the number of instruments per level to the number of levels is high, just as for a flat test architecture.

Similarly to how Table 2.12 and Figure 2.13 presented the results for the experiments that employed concurrent test scheduling, Table 2.13 and Fig-
Figure 2.14: Results with the sequential schedule

Figure 2.14 show results for the experiments that use sequential test scheduling. From the figures it can be seen that the overhead ratio is higher when using a sequential test schedule compared to using a concurrent test schedule. The main reason for this is that the total number of scan-sequences increase, leading to an increase in both CUC and SIB overhead, while the amount of shifted test data stays the same in both test schedules.

Design P34392 shows a noticeable difference in overhead ratio between the flat test architecture, marked by P34392_FLAT, and the hierarchical test architecture, marked by P34392_HIER, in the case of the sequential test schedule. For the flat test architecture the overall overhead is about 9% and for the hierarchical test architecture the overhead is about 7% with the difference mainly due to a lesser SIB overhead. In P34392_FLAT, every scan-sequence, independent of the core, includes 19 SIBs (the same number as the number of cores). However, in P34392_HIER, the scan-sequences contain on average 13 SIBs. For example, Core 5 and Core 19 of P34392_HIER each contains 20% of the total number of test patterns. Figure 2.12(c) shows a part of the test architecture for P34392_HIER. As can be seen, there are eight SIBs in the scan-path that is required to test Core 19 and there are 14 SIBs in the scan-path for testing Core 5. Since the average of 13 SIBs in P34392_HIER is significantly less than 19 SIBs
**Table 2.14: Results with the wrapped cores, concurrent schedule**

<table>
<thead>
<tr>
<th>Design</th>
<th>Shifted Test Data</th>
<th>SIB Overhead</th>
<th>WIR Overhead</th>
<th>CUC Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2126_FLAT</td>
<td>5,330,439</td>
<td>3,400</td>
<td>40</td>
<td>2,125</td>
</tr>
<tr>
<td>T512505_FLAT</td>
<td>165,400,967</td>
<td>200,126</td>
<td>310</td>
<td>16,865</td>
</tr>
<tr>
<td>U226_FLAT</td>
<td>252,929</td>
<td>26,690</td>
<td>50</td>
<td>13,345</td>
</tr>
<tr>
<td>P22810_FLAT</td>
<td>8,172,047</td>
<td>690,312</td>
<td>280</td>
<td>61,635</td>
</tr>
<tr>
<td>A586710_FLAT</td>
<td>843,806,808</td>
<td>19,144,360</td>
<td>50</td>
<td>9,572,180</td>
</tr>
<tr>
<td>P34392_FLAT</td>
<td>16,728,056</td>
<td>468,882</td>
<td>190</td>
<td>61,695</td>
</tr>
<tr>
<td>P22810_HIER</td>
<td>8,172,047</td>
<td>656,860</td>
<td>280</td>
<td>61,640</td>
</tr>
<tr>
<td>A586710_HIER</td>
<td>843,806,808</td>
<td>17,592,210</td>
<td>50</td>
<td>9,572,180</td>
</tr>
<tr>
<td>P34392_HIER</td>
<td>16,728,056</td>
<td>463,196</td>
<td>190</td>
<td>61,700</td>
</tr>
</tbody>
</table>

in P34392_FLAT, this explains the noticeable difference in SIB overhead.

Finally, it is interesting to note that for both concurrent and sequential schedules, comparison between the results for the flat and hierarchical architectures for the same SOC (see Table 2.12 and Table 2.13) shows that the CUC overhead takes very little effect from the network structure while the SIB overhead varies considerably based on the network structure.

### 2.5.3 Experimental Results for Wrapped Cores

To study the effect of IEEE standard 1500 wrappers on the test application time in P1687 networks, comparisons will be made between the test time calculation results for wrapped and unwrapped cores in the ITC'02 benchmarks.

Table 2.14 and Figure 2.15 present the results for the test time calculation of the SOCs listed in Table 2.11, for the concurrent test schedule and when every core has an IEEE standard 1500 wrapper. As mentioned in Section 2.4, in this report the length of the WIR for all of the wrapped cores will be assumed to be 10. In presentation of the results, WIR overhead represents the time spent on loading the scan-chain selection instruction into the WIR of the wrapper that contains the scan-chain. As can be seen, the WIR overhead is negligible compared to the shifted test data. Comparing the results presented in Table 2.14 and Figure 2.15 with those presented by Table 2.12 and Figure 2.13 shows that the Shifted Test Data has not changed and the CUC overhead is only increased by five test clock cycles. The reason that Shifted Test Data remained unchanged is that it only depends on the scan-chain length and the number of patterns for the
2.5. Experiments with the ITC’02 Benchmarks

Figure 2.15: Results with the wrapped cores, concurrent schedule scan-chain. The slight increase in the CUC overhead can be explained by the fact that loading the instruction into WIR requires one CUC which takes five clock cycles.

The comparison also reveals that the SIB overhead is almost doubled in case of the wrapped cores. To study this, first the F2126_FLAT will be considered. In the case of the wrapped cores, each SIB in the design is replaced by a wrapper SIB which requires two bits for its configuration. Therefore, the SIB overhead of 1696 presented in Table 2.12 for the F2126_FLAT design will increase to 1696 × 2 = 3392. However, these configuration bits should also be scanned in during the instruction loading sequence and this slightly adds to the SIB overhead. Since the F2126_FLAT design has four cores (see Table 2.11), the SIB overhead added in the instruction loading sequence (in the concurrent schedule) will be 4 × 2 = 8. Adding 8 to 3392 gives 3400 which is the resulting SIB overhead shown in Table 2.14. The increased overhead (compared to the unwrapped cores) in other flat architectures can be explained by the same argument. For example, in case of the design T512505_FLAT which has 31 cores, the SIB overhead in case of the wrapped cores can be obtained by doubling the SIB overhead in case of the unwrapped cores and adding 31 × 4 to it, i.e. 104, 532 × 2 + 31 × 4 = 209, 126.

In study of the SIB overhead increase in case of the wrapped cores, it
Table 2.15: Results with the wrapped cores, sequential schedule

<table>
<thead>
<tr>
<th>Design</th>
<th>Shifted Test Data</th>
<th>SIB Overhead</th>
<th>WIR Overhead</th>
<th>CUC Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2126_FLAT</td>
<td>5,330,439</td>
<td>7,768</td>
<td>10</td>
<td>4,855</td>
</tr>
<tr>
<td>T512505_FLAT</td>
<td>165,400,967</td>
<td>653,604</td>
<td>310</td>
<td>52,710</td>
</tr>
<tr>
<td>U226_FLAT</td>
<td>252,929</td>
<td>81,000</td>
<td>50</td>
<td>40,500</td>
</tr>
<tr>
<td>P22810_FLAT</td>
<td>8,172,047</td>
<td>1,403,920</td>
<td>280</td>
<td>125,350</td>
</tr>
<tr>
<td>A586710_FLAT</td>
<td>843,806,808</td>
<td>21,419,050</td>
<td>50</td>
<td>10,709,525</td>
</tr>
<tr>
<td>P34392_FLAT</td>
<td>16,728,056</td>
<td>2,521,718</td>
<td>190</td>
<td>331,805</td>
</tr>
<tr>
<td>P22810_HIER</td>
<td>8,172,047</td>
<td>1,283,032</td>
<td>280</td>
<td>125,360</td>
</tr>
<tr>
<td>A586710_HIER</td>
<td>843,806,808</td>
<td>19,639,440</td>
<td>50</td>
<td>10,709,530</td>
</tr>
<tr>
<td>P34392_HIER</td>
<td>16,728,056</td>
<td>1,483,984</td>
<td>190</td>
<td>331,820</td>
</tr>
</tbody>
</table>

should be noted that hierarchical architectures contain SIBs that act as doorways to other levels of hierarchy, and since these SIBs are not replaced by wrapper SIBs, the SIB overhead will be less than doubled in case of the wrapped cores.

Table 2.15 and Figure 2.16 present the results for the test time calculation of the ITC’02 SOCs for the sequential test schedule and when every core has an IEEE standard 1500 wrapper. Comparing the results with those presented by Table 2.13 and Figure 2.14 shows that the Shifted Test Data is unchanged and the CUC overhead has slightly increased. The increase in the CUC overhead can be explained by the fact that in the context of sequential schedule every core is tested individually and this requires a dedicated CUC for each WIR instruction loading. Therefore, the amount of increase in CUC overhead for each SOC equals the number of cores in the SOC times five.

As for the SIB overhead, the comparison shows an almost doubled value in case of the wrapped cores. As explained earlier, a large contribution to this increased value is made by the wrapper SIBs (which require two bits for configuration). Also, scanning in the SIB configuration data during the instruction loading adds slightly to the SIB overhead, which can be easily calculated for the designs having flat architectures. To calculate the SIB overhead because of the WIR instruction loading, it should be considered that in the flat architecture, the shifted data should traverse all the SIBs. Since the number of SIBs in a flat architecture equals the number of cores and each wrapper SIB requires two bits to be configured, the SIB overhead added for each instruction loading will be two times the number of cores. Also by knowing that in the sequential schedule,
every instruction loading is performed individually, the total amount of SIB overhead because of instruction loading in a flat architecture is calculated as $(\text{number of cores})^2 \times 2$. For example, this calculation helps explain the SIB overhead increase for the F2126_FLAT from 3868 in case of the unwrapped cores to 7768 in case of the wrapped cores. Knowing that F2126_FLAT has four cores, the amount of increase in the SIB overhead can be calculated as $3868 \times 2 + 4^2 \times 2 = 7768$.

A noteworthy point about WIR overhead is that similar to the Shifted Test Data that remains unchanged regardless of the schedule, WIR overhead does not depend on the type of the schedule used.

Finally, it should be noted that although the WIR instruction loading overhead (i.e. WIR overhead) is negligible compared to the total test time for all the designs examined, this does not mean that the overhead caused due to the use of the wrappers is negligible. This is because the wrappers require wrapper SIBs which increase the SIB overhead significantly. Another reason is that in the presented discussion, it was assumed that each core has only one scan-chain and therefore only one sequence of instruction loading is required. In practice, this might not be the case since a wrapper may have different wrapper data registers (WDR) and the selection of each of them requires a dedicated instruction loading sequence.
2.6 Summary

In this chapter, the test application procedure for two P1687 architectures, namely flat and hierarchical architectures, was studied. Formulas were presented for calculation of the test application time (TAT) for the flat architecture, and algorithms were presented capable of calculating the test application time for all the sample architectures presented in Figure 1.12. Also, the effect of adding IEEE standard 1500 wrappers was studied and the necessary changes suggested for the formulas and algorithms to take the effect of adding wrappers into account in their test time calculations. The algorithms were employed in experiments on ITC’02 benchmarks, both considering cores as being wrapped and unwrapped in separate studies, and the results were discussed. In the study of the test application time, observations were made regarding the effect of the length and the number of patterns of scan-chains on the test application time. The key observations can be listed as follows:

- The shifted test data is independent of test schedule and network architecture, and solely depends on the length and number of patterns of the instruments.

- Only the number of patterns of a scan-chain (and not the length of it) affects the overhead.

- In the flat architecture and for both concurrent and sequential schedules, the instrument with the largest number of patterns makes the largest contribution to the overhead.

- Hierarchical architectures (as was observed in the experiments) can help reduce the SIB overhead, since they can reduce the average number of SIBs that exist on the scan-path during the test of instruments.

- The CUC overhead takes little effect from the network structure while in contrast, the SIB overhead varies considerably based on the network structure.

- When the instruments are IEEE 1500 standard wrappers and are connected to the P1687 network using localized-control, the SIB overhead
is almost doubled compared with the case that the instruments are scan-chains identical to the WDR inside the wrapped core.

These observations will be used as guidelines in Chapter 3 to construct optimized P1687 networks with respect to trade-off between the test application time and the number of SIBs.
Chapter 3

Design of P1687 Networks

In Chapter 2 calculation of test application time (TAT) for a given P1687 network was studied. It was shown that TAT varies significantly based on the test schedule and the used architecture. In this chapter, the focus will be on designing P1687 networks, for a given set of instruments and a test schedule, which are optimized with respect to TAT and the number of SIBs. In Chapter 2 it was shown that total test time for a P1687 network, having scan-chains as instruments, is composed of three components: (1) shifted test data, (2) CUC overhead, and (3) SIB overhead. To keep the discussion simple, the focus of this chapter will only be on scan-chains as instruments, and not the wrapped cores. However, the same discussion can be extended to the wrapped cores by taking into account that wrapper SIBs require two configuration bits to be programmed.

Shifted test data is independent of test schedule and network structure, and solely depends on properties of instruments, i.e. number of test patterns and length of scan chains, and thus will be considered a constant value in this study. Therefore, to reduce TAT, SIB overhead and CUC overhead should be reduced. In this chapter, it is assumed that each instrument is connected to the P1687 network through a dedicated instrument SIB. Therefore, the number of instrument SIBs will remain constant (given the same set of instruments) for every possible network and the difference in SIB overhead only depends on number and placement of doorway SIBs
in the network. Since doorway SIBs effectively change the length of the scan-path, their impact on the SIB overhead is significant. In contrast, CUC overhead takes a negligible impact from the number and placement of doorway SIBs, as was observed in Section 2.5.2. This narrows the focus of this chapter to the reduction of SIB overhead. Here again two test schedules are considered, (1) sequential schedule where instruments are tested one by one and no testing concurrency exists at all and (2) concurrent schedule in which instruments are tested all at the same time until their tests finish. It is worth mentioning that the length of the scan-chains has no effect on the SIB overhead, as was concluded from the study in Chapter 2. Therefore, no length will be specified for the scan-chains in the rest of this chapter.

3.1 SIB Overhead Reduction for the Sequential Schedule

3.1.1 Analysis

SIB overhead is the number of test clock cycles spent on scanning SIB control bits. This can be shown by Equation 3.1.

\[ O = O_i + O_t \]  

The following describes \( O_i \): As was mentioned in Chapter 2, before transporting test data to any instrument, SIBs should be programmed in a way that the instrument is included in the scan-path. A dedicated scan is required to program the SIBs prior to using the first instrument at each level of hierarchy, but for the rest of instruments at that level this programming can be done while the last response of the previous instrument is being shifted out (it should be reminded that this discussion is in the context of sequential schedule). This initial programming scan, which was referred to as Setup-sequence in Chapter 2, is represented by \( O_i \) in Equation 3.1.

\( C_t \) is the number of control bits that are shifted in during the test of each of the instruments. Equation 3.2 shows how \( O_t \) can be calculated, where \( N \) is the total number of instruments, \( P_i \) is the number of patterns for instrument \( i \) and \( L_{sp,i} \) is the number of SIBs (both instrument SIBs and
doorway SIBs) on the scan path during the test of instrument \( i \), including \( i \)'s own instrument SIB, i.e. the SIB that \( i \) is directly connected to its HIP. Here +1 represents the scan required to shift out the responses to the last stimuli while reprogramming the SIBs.

\[
O_t = \sum_{i=1}^{N} (P_i + 1) \cdot L_{sp,i} \tag{3.2}
\]

Since \( P_i \) is fixed for each instrument, the only alternative for minimizing the SIB overhead (\( O \)) is reducing \( L_{sp,i} \). Equation 3.2 also suggests that to minimize \( O \), instruments with larger values of \( P \) should have fewer SIBs on their scan path (i.e. smaller \( L_{sp,i} \)). Since each instrument is connected to the P1687 logic through a dedicated instrument SIB, one way to remove the instrument SIBs from each other’s scan path (and thus reducing \( L_{sp,i} \)) is using hierarchical architectures.

In this study, for each set of instruments, comparison against a flat architecture (which is the simplest and most straightforward method of creating a P1687 structure) will be made to observe the reduction in the SIB overhead by using hierarchy.

Although it is possible to use hierarchy to reduce the \( O_t \) part of SIB overhead, it should be noted that using hierarchy increases the \( O_i \) part. Therefore, decision cannot be made on what architecture to use, unless the SIB overhead is calculated and compared for both architectures, as clarified by the following example.

**Example 1:** Figure 3.1(a) shows three instruments in a P1687 network having flat architecture and Figure 3.1(b) shows the same instruments in a hierarchical architecture. The SIB overhead for Figure 3.1(a) and Figure 3.1(b) are 75 and 74, respectively. In this case, the hierarchical architecture helped reduce the SIB overhead, but if \( P_2 \) were 8 instead of 3, the SIB overhead for Figure 3.1(a) and Figure 3.1(b) would be 90 and 94, respectively. Here, using the flat architecture resulted in a lower SIB overhead.

The previous example showed that for three instruments and based on their number of patterns, it is possible to select the architecture type that results in less SIB overhead. It should be noted that although the network shown in Figure 3.1(b) is not the only possible hierarchical network for
3.1. SIB Overhead Reduction for the Sequential Schedule

Figure 3.1: The same three instruments tested in different architectures three instruments, it results in the least SIB overhead among them. The reason is that, as argued earlier, the instrument with the largest number of patterns is placed at a lower hierarchical level to have a shorter scan-path during its test.

The following example, studies the same three instruments used in Example 1 among a larger set of instruments. It will be shown that an optimized P1687 network when used as a segment in a larger network, does not necessarily result in the optimum SIB overhead for that larger network.

Example 2: All three P1687 networks shown in Figure 3.2, are constructed out of the same set of four instruments. The SIB overheads corresponding to Figure 3.2(a), Figure 3.2(b) and Figure 3.2(c), are 304, 229, and 230, respectively. It can be seen that both hierarchical networks result in significantly lower SIB overhead compared with the network having flat architecture. Interestingly, the same hierarchical network that resulted in less SIB overhead (compared to the flat network) in Example 1, results in more SIB overhead when included in a larger network (i.e. the dashed segment in Figure 3.2(b) compared with the dashed segment in Figure 3.2(c)). The reason is that when Figure 3.2(c) is used instead of Figure 3.2(b), the increase in the $O_i$ part of the overhead becomes larger than the decrease in the $O_t$ part.

It can be concluded from Example 1 and Example 2 that to design a network for a set $S$ of instruments, it is possible to design optimized network segments for the subsets of $S$ and combine them, by using hierarchical
Figure 3.2: The same four instruments tested in different networks architecture, to construct the network for $S$. This approach helps reduce the $O_t$ part of the SIB overhead but increases the $O_i$ part, which generally leads to a total decrease in the SIB overhead. However, as was shown in Example 2, there might be cases that the amount of increase in $O_i$ becomes larger than amount of decrease in $O_t$ when a segment is combined with the other segments, and this cannot be known until all the whole network is constructed. Therefore, after the construction of the whole network is finished, another step is required to identify the SIBs (such as SIB$_6$ in Figure 3.2(c)) whose removal (i.e. replacing a SIB with the segments on its HIP) leads to less SIB overhead.
3.1. SIB Overhead Reduction for the Sequential Schedule

Figure 3.3: Steps in Huffman Construction

In the following, a construction algorithm is proposed inspired by the Huffman Construction. The Huffman Construction is a method for constructing labeled trees of symbols that model prefix codes, used in variable length coding [18]. The basic idea in Huffman Construction is that symbols with higher frequency of occurrence (weight) are assigned shorter length code words. To construct such a tree, symbols with larger weights are placed closer to the root of the tree.

In construction of a P1687 network, analogy can be made between weight of a symbol in Huffman Construction, and the number of patterns of an instrument. That is, since instruments with larger number of patterns are accessed more frequently, they should be placed in the P1687 network such that the number of SIBs on their scan-path (which is analogous to the length of the code word for the symbol) becomes relatively low. A key idea in the construction inspired by Huffman Construction is to combine a set $S$ of instruments and treat them as one instrument $x$ where $P_x = \sum_{i \in S} P_i$. To combine the instruments, a doorway SIB (SIB$_d$) is added and the set $S$ of instruments are connected to the HIP of SIB$_d$.

Figure 3.3(f) shows a Huffman tree constructed for binary encoding of seven symbols with these weights: 1, 1, 1, 1, 5, 10, 25. The symbol with the weight of 25 has the largest frequency of occurrence and therefore receives the shortest length binary code word which is “1”, whereas a symbol with the weight of 1 which is the lowest weight among the others, receives the
code word of “01000”. In the following, construction of this tree will be explained with the help of Figure 3.3. Initially, the symbols are sorted in the ascending order based on their weights and the first two symbols are combined, as shown in Figure 3.3(a). The combination can be viewed as a symbol having the weight of 2. Therefore, this combined symbol must be placed in the appropriate position to keep the list sorted, as shown by the dotted arrow in Figure 3.3(a). Again, starting from the beginning of the list, the first two symbols are combined. Since the combined weight is 2, which is less than or equal to the rest of the weights, it will not be repositioned, as shown by Figure 3.3(b). The process of combining the first two symbols in the list and repositioning will continue until only one instrument is left, in which case the construction algorithm terminates. In the example of Figure 3.3(f), the algorithm terminated when the combined symbol with the weight of 42 became the only symbol in the list.

Figure 3.4(a) shows the P1687 network analogous to the Huffman tree shown in Figure 3.3(f). In this network, SIB_8-SIB_12 are doorway SIBs that can be considered as combined instrument SIBs. For example, SIB_8 and SIB_11 can be considered as instruments with 2 and 9 patterns, respectively.

If the instruments used to construct the network in Figure 3.4(a) were arranged in flat architecture, the SIB overhead would be 350 clock cycles, while the SIB overhead for the design in Figure 3.4(a) is 244. As discussed earlier in this section, there might be possibilities for further reduction of the SIB overhead in the network shown in Figure 3.4(a), by identifying which doorway SIBs can be removed. For example, if SIB_8, SIB_9 and SIB_11 in Figure 3.4(a) are removed, as shown in Figure 3.4(b), the overhead will be further reduced to 215. An optimization algorithm is proposed in Section 3.1.2 that analyzes a P1687 network to find the possibilities of TAT reduction by reducing the number of doorway SIBs. Reducing the number of SIBs has the advantage of saving area, power and routing required by each SIB.

### 3.1.2 Algorithms for Sequential Schedules

Algorithm 7 shows the steps to construct a P1687 network out of a given set of instruments, such that the TAT is optimized for the sequential schedule. The algorithm is based on the Huffman Construction method described in
3.1. SIB Overhead Reduction for the Sequential Schedule

Section 3.1.1. Also discussed in Section 3.1.1, was the possibility of further reduction of the SIB overhead, by finding which doorway SIBs to remove. Algorithm 8 outlines the basic idea for such optimization.

3.1.3 Handling a Special Case

If a large number of instruments which have the same (or very similar) number of patterns are to be accessed through P1687 circuitry, the construction algorithm (Algorithm 7) may not generate a very optimized network. The reason is that the basic idea behind that algorithm is prioritizing the instruments with larger number of patterns (over those with smaller number of patterns) by placing them closer to the Gateway. But in case where all the instruments have the same number of patterns, it is not reasonable to prioritize any of the instruments over the others. In fact in this case, all

![Diagram](image)

Figure 3.4: P1687 network analogous to the Huffman tree in Figure 3.3(f)

Algorithm 7 Construction Algorithm for Sequential Schedule

1: $L :=$ list of instruments
2: while Size($L$) > 1 do
3: Find two instruments with the lowest $P$ in $L$
4: Combine the two instruments
5: Remove the two instruments from $L$
6: Add the combined instruments to $L$
7: end while

Section 3.1.1. Also discussed in Section 3.1.1, was the possibility of further reduction of the SIB overhead, by finding which doorway SIBs to remove. Algorithm 8 outlines the basic idea for such optimization.

3.1.3 Handling a Special Case

If a large number of instruments which have the same (or very similar) number of patterns are to be accessed through P1687 circuitry, the construction algorithm (Algorithm 7) may not generate a very optimized network. The reason is that the basic idea behind that algorithm is prioritizing the instruments with larger number of patterns (over those with smaller number of patterns) by placing them closer to the Gateway. But in case where all the instruments have the same number of patterns, it is not reasonable to prioritize any of the instruments over the others. In fact in this case, all
Algorithm 8 Optimization Algorithm for Sequential Schedule

1: for each SIBₐ do
2:  \[ SIBOverhead := \text{SIB overhead of the network} \]
3:  Remove SIBₐ
4:  \[ NewSIBOverhead := \text{SIB overhead of the network} \]
5:  if \[ NewSIBOverhead > SIBOverhead \] then
6:    Restore SIBₐ
7:  end if
8: end for

the instruments in the network should have equal number of SIBs on their scan-path. Since there may be many possible networks for a given set of instruments where all the instruments have equal number of patterns on their scan-path, the following method is proposed to create such network in a way that the SIB overhead is optimized.

1. Starting from the Gateway, the instruments will be divided into two and three groups and the SIB overhead will be calculated for each of the groupings. The construction will continue with the grouping that shows less SIB overhead.

2. Inside each group repeat the previous step until the number of instruments in a group is less than five.

The rationale behind proposing the above method is that using groups of two and three instruments makes it possible to distribute the instruments more uniformly on the HIP of doorway SIBs at all the levels of hierarchy. If however, it happens that, for example, one group of four instruments is more optimal than two groups each having two instruments, the optimization algorithm (Algorithm 8) described in Section 3.1.2 will convert the two groups of two instruments, back to one group of four instruments, by removing the doorway SIBs.

3.1.4 The Proposed Design Method

In this section a design method will be proposed capable of constructing P1687 networks out of a given set of instruments, in a way that the SIB overhead is minimized for the sequential test schedule. To do so, the method employs the construction algorithm (Algorithm 7), the solution suggested
for handling special cases (Section 3.1.3), and the optimization algorithm (Algorithm 8) discussed so far.

Before proceeding to describe the proposed design method, some practical issues regarding the suggested solution for handling the special case where all the instruments have the same number of patterns (Section 3.1.3), should be discussed.

The first issue is that in Section 3.1.3, it was assumed that all the given instruments have exactly the same number of patterns, but in practice it might happen that groups of instruments with the same (or very similar) number of patterns inside each group, appear in the given instruments list. This requires these groups to be identified, and the method suggested in Section 3.1.3 be applied to each group. In this thesis work, cluster analysis (clustering) will be used to identify the groups. To identify the groups (clusters) such that the instruments with similar (or very close) number of patterns appear in the same group, clustering should be done recursively inside each cluster until no more clustering is possible, i.e. the clustering algorithm identifies only one cluster inside each cluster. When clusters are identified, the solution explained in Section 3.1.3 should be applied to the instruments inside each cluster.

Another issue is regarding the construction algorithm proposed in Section 3.1.2. This algorithm operated on a given set of instruments and sorted the instrument SIBs and doorway SIBs based on the number of patterns that existed for the instruments accessed through those SIBs. However, when clustering is to be done before using the construction, the output of the clustering should be usable by the construction method. This can be done by arranging the instruments inside each cluster in a flat architecture (which implies connecting each instrument to the HIP of an instrument SIB as well) and then connecting the resulted structure to the HIP of a new doorway SIB. If a cluster only contains one instrument, it suffices to connect the instrument to the HIP of a doorway SIB. Therefore, the output of clustering will be a list of instrument SIBs and doorway SIBs, which can be used as the input to the construction algorithm (Section 3.1.2).

Based on the discussion above, the following design method is proposed as the solution to the problem of constructing P1687 networks out of a given set of instruments, in a way that the SIB overhead is minimized for the sequential test schedule:
1. Cluster the instruments based on the number of patterns.

2. Inside each cluster use the method explained in Section 3.1.3.

3. Use the construction algorithm proposed in Section 3.1.2.

4. Use the optimization algorithm proposed in Section 3.1.2.

It should be mentioned that each step of the above design method operates on the output generated by the previous step.

3.2 SIB Overhead Reduction for the Concurrent Schedule

3.2.1 Analysis

In this section, reduction of SIB overhead for the concurrent schedule will be studied. Analysis will be presented to motivate the proposed algorithm. Figure 3.5(a) shows $N$ instruments in the flat architecture. $P_i$ is the number of patterns for instrument $i$ such that $P_1 > P_2 > \ldots > P_K > \ldots > P_N$. In the concurrent schedule, all tests start at the same time and by closing the instrument SIBs whose corresponding instruments’ tests are finished, the scan-path will become shorter for the rest of instruments. This however, leaves the closed instrument SIBs themselves on the scan-path, contributing to the SIB overhead for each subsequent test pattern. By using hierarchical architectures, such as the one shown in Figure 3.5(b), it is possible to minimize the SIB overhead caused by the instrument SIBs, by excluding them from the scan-path.

To test the design in Figure 3.5(a) using concurrent schedule, $N$ bits should be shifted in to program the SIBs to be open after a CUC. These $N$ bits are considered overhead since they are not part of the actual test data. Furthermore, each SIB should be programmed for every test pattern, either to remain open or to be closed once all the patterns for its corresponding instrument are applied. Since $P_1$ is the largest number of patterns, the test continues until $P_1$ patterns are applied and the last responses are shifted out. This will lead to $(P_1 + 1) \cdot N$ test clock cycles spent in total on shifting
3.2. SIB Overhead Reduction for the Concurrent Schedule

Figure 3.5: \(N\) instruments in flat and a two-level hierarchical architectures

To test the design in Figure 3.5(b), \(K\) bits should be shifted in to open the SIBs at the first level of hierarchy, i.e. SIB\(_1\) to SIB\(_{K-1}\) and SIB\(_d\), followed by a CUC. As the first set of stimuli for instruments corresponding SIB\(_1\) to SIB\(_{K-1}\) is shifted in, SIB\(_K\) to SIB\(_N\) are programmed to be open. Therefore, \(N + 1\) control bits are shifted in besides the actual test data. Now that SIBs at the second level are open, \(P_K\) more patterns are applied for all the instruments. At this point, all the instruments at the second level are completely tested (and the last responses are shifted out) and SIB\(_d\) should be closed to shorten the scan-path for the rest of test patterns. Closing SIB\(_d\) is done at the same time that responses to the last patterns are shifted out. Applying \(P_K\) patterns and shifting out the last responses require shifting \((P_K + 1) \cdot (N + 1)\) control bits. Once SIB\(_d\) is closed, the rest of the patterns (i.e. those left from \(P_1\)) are to be applied and the last responses be shifted out. This requires \((P_1 - P_K - 1) \cdot K\) more control bits to be shifted in.
Algorithm 9 Construction Algorithm for Concurrent Schedule

1: \( L := 1 \)
2: \( S := \{P_1, P_2, \ldots, P_N\} \)
3: \( N := |S| \)
4: while \( N > 2 \) do
5: \( K \) that satisfies (3.3) for current elements of \( S \)
6: if \((K \text{Found})\) then
7: \( \text{break} \)
8: end if
9: \( I_L := \text{First } K - 1 \text{ elements of } S \)
10: \( S = S - I_L \)
11: \( N := |S| \)
12: \( L := L + 1 \)
13: end while
14: \( I_L := S \)

Therefore, the total SIB overhead for testing the design in Figure 3.5(b) is calculated as \( O = K + (N + 1) + (P_K + 1) \cdot (N + 1) + (P_1 - P_K - 1) \cdot K \).

Based on the above overhead calculations, it can be concluded that if (3.3) is satisfied for the set of \( N \) instruments shown in Figure 3.5, using the hierarchical design in Figure 3.5(b) will result in less SIB overhead.

\[
K + (N + 1) + (P_K + 1) \cdot (N + 1) + (P_1 - P_K - 1) \cdot K < N + (P_1 + 1) \cdot N \tag{3.3}
\]

If there is any \( K \) that satisfies (3.3), the architecture shown in Figure 3.5(b) will be used instead of the flat architecture. It can be seen that if \( P_1 = P_2 = \ldots = P_K = \ldots = P_N = P \), (3.3) is not satisfied for any value of \( P \). That is, when all the instruments have the same number of patterns, the flat architecture will result in the least SIB overhead.

Based on this observation, Section 3.2.2 presents an algorithm for the construction of P1687 networks, optimized for the concurrent schedule.

### 3.2.2 Algorithm for Concurrent Schedules

Based on the analysis presented earlier in this section, Algorithm 9 achieves optimization of test time, given the concurrent schedule and a set of instruments \( S \). The set \( S \) contains \( N \) instruments sorted in the
3.3 Experimental Setup

The ITC’02 Test Benchmarks [16] mentioned in Section 2.5.1 are used to obtain experimental results from the algorithms proposed in this chapter. Three of the SOCs, i.e. A586710, F2126 and U226, are left out due to their small number of cores. Also, two custom SOCs are composed to study how well the algorithms perform in case of large designs and in special cases. Table 3.1 lists the SOCs from the ITC’02 benchmarks suite as well as the two custom SOCs, i.e. Merge12 and S100. Merge12 is a collection of the cores existing in all twelve SOCs from the ITC’02 benchmark suite. It should be noted that the cores that their associated tests have their TAMUSE set to “0” are excluded [16]. The SOC S100 contains 100 scan-chains with the length of 10, each having one pattern to be applied. S100 is introduced here to investigate the case where there are many instruments with the same number of patterns and short scan-chains.

For the set of cores in each of the SOCs listed in Table 3.1, five P1687 networks referred to as designs will be constructed and studied as listed below:

1. A design with the flat architecture which will be marked by F in the
presentation of the results.

2. A design with hierarchical architecture, constructed using the algorithm described in Section 3.1.2 which will be marked by H in the results. It should be noted that this algorithm tries to minimize the SIB overhead for the sequential schedule.

3. An H design (see 2) followed by the optimization algorithm described in Section 3.1.1. This will be marked by HO.

4. An HO design (see 3) preceded by the clustering method described in Section 3.1.3. This will be marked by CHO. To do the clustering, Cluster 3.0 is used which is an open source clustering software [19]. The results that are reported are obtained using Euclidean distance as the similarity metric and Single linkage as the clustering method. It should be mentioned that the results obtained using other similarity metrics or clustering methods supported by the Cluster 3.0 tool, were not significantly different and therefore are not reported.

5. A design marked by P (P for parallel) which is constructed by using the algorithm suggested in Section 3.2. This algorithm constructs the design in a way to have the lowest SIB overhead for the concurrent schedule.

3.4 Experimental Results

Table 3.2 shows the results of the test time calculation for the generated designs, when the sequential schedule is used. Test application time (TAT) is calculated using the algorithms presented in Chapter 2. The column “SOC” lists the SOC which its cores are used to construct networks. The column “Design” marks the type of network that is constructed out of the cores in the SOC mentioned in (the corresponding row) in the column “SOC”. The column “Doorway SIBs” lists the number of doorway SIBs that are used to construct the design. This number is reported here to help compare the efficiency of different algorithms when the reduction in SIB overhead achieved by them is not significantly different. The column “SIB Overhead” presents the calculated SIB overhead for each design. The
Column “SIB Overhead Ratio(%)” presents the ratio of the SIB overhead to the Shifted Test Data (reported in Table 3.1). Since the Shifted Test Data is independent of the network and solely depends on the instruments (here, cores), it remains the same for every SOC and therefore, the SIB overhead ratio gives a good measure to compare the algorithms used to construct each of the designs constructed for the same core. The column “Reduction in TAT(%)” presents (in percents) how much reduction in TAT is achieved compared to the flat architecture (which is the most simple and
straight-forward way of designing a P1687 network). Finally, the column “CUC overhead” presents the CUC overhead associated with each of the designs.

It should be noted that the designs marked by H are intermediate results, i.e. are not meant to be compared to the other designs, and are just reported to show how much reduction in the overhead (both the SIB and the CUC overheads) and the number of doorway SIBs is achieved in the optimization step (see section 3.1.2), which takes the H design as the input and generates the HO design as output. It can be seen that for all of the SOCs, the reduction in the number of doorway SIBs in the optimization step is quite significant, especially in case of SOC S100.

Regarding the results presented in Table 3.2, the following general observations can be made:

- The CHO design, which is the output of the recommended design method for the sequential schedule, achieves the highest reduction in TAT for all the studied SOCs. In cases where the achieved reduction of TAT is equal to the other designs, the number of doorway SIBs for CHO is the least. T512505 seems to be an exception to this rule, but it seems so because the reported numbers for “Reduction in TAT” only show two fractional digits after the decimal point.

- A close look at the values in the column “CUC Overhead”, shows that except for the SOC S100, the CUC overhead does not change significantly among the different designs constructed for the same SOC. On the other hand, in case of the SOC S100, the reduction in SIB overhead is relatively more significant than the increase in the CUC overhead. Therefore, these results confirm the previous statement in Section 3.1 that suggested the optimization focus to be only on SIB overhead reduction.

- For all of the SOCs, the F design (which has flat architecture) shows the largest SIB overhead and the smallest (i.e. zero) number of doorway SIBs, compared to the other designs (which have hierarchical architectures).

- For all of the SOCs except for S100, the P design (which is optimized
3.4. Experimental Results

for the concurrent schedule) shows results comparable to the results of HO and CHO designs which are optimized for the sequential schedule.

Additionally, the following SOC specific observations can be made:

- For T512505, the SIB overhead ratio is almost zero for all the designs. The reason is that this SOC contains a core with a relatively very large scan chain \((L = 43922, P = 3370)\) which constitutes about 90% of the Shifted Test Data. As was mentioned in Section 2.5.2, long scan chains reduce the SIB overhead ratio.

- For Merge12, the SIB overhead ratio is very large for the F design and becomes significantly smaller for all other designs which use hierarchical architecture. This can be explained by the fact that Merge12 contains a core with \(L = 326\) and \(P = 1914433\). Placing this core in a flat architecture with the other 166 cores, results in a SIB overhead of \(167 \times 1914433 = 319710311\). This alone, constitutes 84% of the SIB overhead for the F design. The number of SIBs on the scan path to this dominant core (including its own instrument SIB) is 167 for the F design and 2 for all other designs constructed for Merge12. This explains the significant difference between the SIB overhead for the F design and the other designs for Merge12.

- The SOC S100, among the other SOCs, shows the largest values of SIB overhead ratio for all of its corresponding designs. To explain the very high SIB overhead ratio for the F design, it should be noted that in F every instrument has 100 instrument SIBs on its scan path, which is 10 times the length of the scan chain for the instrument. Therefore, for shifting in each test stimuli to the scan chain of any instrument, 100 control bits should be shifted in for programming the SIBs. Knowing that \(L = 10\) for all the instruments in S100, this explains the reason that SIB overhead for the F design is 10 times the Shifted Test Data.

Table 3.3 shows the test time calculation results for the concurrent schedule. The following observations can be made regarding the results presented in Table 3.3:
Table 3.3: The generated designs and their associated SIB and CUC overheads (concurrent schedule)

<table>
<thead>
<tr>
<th>SOC</th>
<th>Design</th>
<th>Doorway</th>
<th>SIB SIBs</th>
<th>Overhead</th>
<th>Ratio(%)</th>
<th>CUC Overhead</th>
<th>Reduction in TAT(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P22810</td>
<td>F</td>
<td>0</td>
<td>345128</td>
<td>4.22</td>
<td>61630</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>26</td>
<td>57526</td>
<td>0.70</td>
<td>61630</td>
<td>3.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HO</td>
<td>15</td>
<td>62805</td>
<td>0.77</td>
<td>61630</td>
<td>3.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHO</td>
<td>14</td>
<td>59160</td>
<td>0.72</td>
<td>61630</td>
<td>3.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>17</td>
<td>46886</td>
<td>0.57</td>
<td>61630</td>
<td>3.48</td>
<td></td>
</tr>
<tr>
<td>P34392</td>
<td>F</td>
<td>0</td>
<td>234422</td>
<td>1.40</td>
<td>61690</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>17</td>
<td>161998</td>
<td>0.97</td>
<td>61700</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HO</td>
<td>11</td>
<td>148337</td>
<td>0.89</td>
<td>61695</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHO</td>
<td>10</td>
<td>125134</td>
<td>0.75</td>
<td>61695</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>10</td>
<td>117842</td>
<td>0.70</td>
<td>61690</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>T512505</td>
<td>F</td>
<td>0</td>
<td>104532</td>
<td>0.06</td>
<td>16860</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>29</td>
<td>31636</td>
<td>0.02</td>
<td>16865</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HO</td>
<td>16</td>
<td>26779</td>
<td>0.02</td>
<td>16860</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHO</td>
<td>18</td>
<td>25639</td>
<td>0.02</td>
<td>16860</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>18</td>
<td>19691</td>
<td>0.01</td>
<td>16860</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Merge12</td>
<td>F</td>
<td>0</td>
<td>319710645</td>
<td>28.91</td>
<td>9572175</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>165</td>
<td>4772942</td>
<td>0.43</td>
<td>9572175</td>
<td>21.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HO</td>
<td>94</td>
<td>4861850</td>
<td>0.44</td>
<td>9572175</td>
<td>21.94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHO</td>
<td>90</td>
<td>4763711</td>
<td>0.43</td>
<td>9572175</td>
<td>21.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>101</td>
<td>4520598</td>
<td>0.41</td>
<td>9572175</td>
<td>21.96</td>
<td></td>
</tr>
<tr>
<td>S100</td>
<td>F</td>
<td>0</td>
<td>300</td>
<td>15.00</td>
<td>15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>98</td>
<td>784</td>
<td>39.20</td>
<td>45</td>
<td>-22.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HO</td>
<td>21</td>
<td>393</td>
<td>19.65</td>
<td>30</td>
<td>-4.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHO</td>
<td>39</td>
<td>471</td>
<td>23.55</td>
<td>30</td>
<td>-8.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>0</td>
<td>300</td>
<td>15.00</td>
<td>15</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- For all SOCs in the experiment, the P design shows the least amount of SIB overhead. Since P design is the output of the construction algorithm for the concurrent schedule (Algorithm 9), it can be seen that this algorithm has been successful in constructing the design that results in the least SIB overhead among the other designs, for the concurrent schedule.

- As was the case for the sequential schedule, the CUC overhead does
not change considerably among different designs for the same SOC.

- As was concluded in Section 3.2, for S100 the P design has a flat architecture and therefore shows the same results as the F design.

Finally, it is interesting to note that for the F design of Merge12, the SIB overhead is similar for the sequential and concurrent schedules. The reason is that as explained above, most of the overhead comes from a dominant core and this dominant core has all 167 doorway SIBs on its scan path for both schedules.

3.5 Summary

In this chapter, automated design of optimized IEEE P1687 networks was studied and time effective algorithms were proposed both for sequential and concurrent test schedules. The algorithms are employed in experiments on industrial and custom-built SOCs and the results are reported and discussed. By using the algorithms, given a set of instruments (embedded test and debug logic) and a test schedule which can be either sequential or concurrent, a P1687 network is designed which is optimized with respect to test time and number of SIBs. The advantage in reducing the number of SIBs is savings in area, power and routing required by each SIB. It was argued (and later observed in the experiments) that CUC overhead (JTAG protocol overhead) is almost independent of the network, and in contrast, SIB overhead (control data overhead) varies considerably based on how SIBs are placed on the scan-path. Therefore, for reducing TAT, this paper only focused on reduction of SIB overhead. To reduce the SIB overhead, providing a shorter scan-path for the instruments which are more frequently accessed, proved effective for both concurrent and sequential test schedules. Providing the shorter path is achieved by using hierarchical structures. Experimental results showed that a hierarchical architecture helps reduce the TAT by up to about 25%, compared to the flat architecture for the same set of instruments.
Chapter 4

Conclusion and Future Work

4.1 Conclusion

IEEE P1687 standard proposal aims at standardizing the access to the on-chip test, debug and monitoring logic (called instruments) through JTAG TAP. To construct the scan-path for accessing the instruments, i.e. P1687 network, P1687 proposes a component called Segment Insertion Bit (SIB). Through its Hierarchical Interface Port (HIP), a SIB is used to connect to instruments and other SIBs. SIBs make it possible to construct numerous P1687 networks for the same set of instruments, and provide flexibility in test scheduling. SIBs are configured through the same scan-path that is used to transport data to/from the instruments.

In this thesis work, instruments having scan-chains that are used for testing are considered. The work itself has two parts. In the first part, test time analysis was presented for P1687 networks, when the tests are applied sequentially or concurrently. In the analysis, it was observed that the test application time consists of three components: Shifted Test Data (i.e. actual test data for the instruments), SIB overhead (control data for configuring the SIBs) and CUC overhead (overhead caused by using the
JTAG protocol). Furthermore, algorithms were presented for calculation of the test application time for a given P1687 network and a given schedule type which can be either concurrent or sequential. The algorithms were used in experiments and the results showed that the Shifted Test Data is independent of the P1687 network structure and the test schedule, while in contrast, the SIB overhead and CUC overhead vary based on the network and the schedule. It is seen from the results that the SIB overhead varies considerably based on the network structure, while in contrast the CUC overhead takes little effect from the network structure. Additionally, it was observed that hierarchical architectures can help reduce the SIB overhead, since they can reduce the average number of SIBs that exist on the scan-path during the test of instruments.

In the second part of this work and based on the analysis presented in the first part, automated design of P1687 networks is studied. In the study, the effect of using hierarchy on SIB overhead and the fact that CUC takes little effect from the network structure, which were observed in the first part of this thesis work, are used as guidelines. Based on these guidelines algorithm are presented that, given a set of instruments and a schedule type which can be either concurrent or sequential, construct a P1687 network which is optimized with respect to test time and the number of SIBs. The algorithms were implemented and demonstrated on industrial benchmark circuits. The experimental results showed that in some circuits, up to 25% reduction in test application time is achieved.

4.2 Future Work

The future work in this area can address the following issues which were not considered in this thesis work:

- Construction and optimization of IEEE P1687 test structures for test schedule types other than sequential and concurrent.

- Considering (in the construction and optimization of the P1687 test structures) the trade-off between the area occupied by doorway SIBs and the amount of the SIB overhead reduction achieved by adding them.
• Considering (in the construction and optimization of the P1687 test structures) the layout (where the instruments actually are placed) and routing issues.
Appendix A

IJTAG Calculator

To implement the suggested algorithms in this report and carry out the experiments, a tool called *IJTAG Calculator* has been developed. This chapter briefly introduces the features of the *IJTAG Calculator* tool. The source code and the user’s manual will be made available through the ESLAB’s website [20].

A.1 Main Features

A.1.1 Construction of P1687 Networks

*IJTAG Calculator* is enabled to receive a list of instruments and create P1687 networks out of them by using the constraints set by the user through the Graphical User Interface (GUI). The user may either select optimized design or custom design. For the optimized design construction the user can either select optimization for the sequential schedule or the concurrent schedule. For the custom design, the user can either select that the instruments are placed into an n-ary tree, or select that the instruments are placed into a randomly constructed tree where the outdegree and the number of instruments per node change at each node randomly.
A.1.2 Test Application Time Calculation

*IJTAG Calculator* calculates the test application time of a given P1687 network for the concurrent and sequential schedule. It is also possible to perform a batch operation where the list of the designs as well as the desired test schedule are specified in a file and *IJTAG Calculator* performs the required test application time calculations and outputs the result in a file.

A.2 Input/Output File Format

Extensible Markup Language (XML) is chosen (instead of plain text) as the input/output file format for *IJTAG Calculator*. The reason is that XML provides a standard way of representing information and also proves useful when it comes to describing hierarchical structures such as P1687 networks. In the following, sample input and output file contents will be shown with regard to the features of *IJTAG Calculator*. The interested reader may refer to the user’s manual for the details.

A.2.1 Describing P1687 test structures

Listing 1 shows the XML description of the design shown in Figure A.1.

A.2.2 Instrument lists

Listing 2 shows a sample instruments list described in XML.
A.2. Input/Output File Format

Listing 2 A sample instruments list

```xml
<?xml version="1.0"?>
<ScanChains>
  <SIB ID="1" SCLength="20" SCPatterns="2666" BIST="0" />
  <SIB ID="2" SCLength="8003" SCPatterns="416" BIST="0" />
</ScanChains>
```

Listing 3 A sample batch operations file

```xml
<?xml version="1.0"?>
<Analyses Name="Sequential" >
  <Analysis ID="1" SRC="P22810_F.xml" Method="0" />
  <Analysis ID="2" SRC="P34392_F.xml" Method="0" />
  <Analysis ID="3" SRC="P93791_F.xml" Method="0" />
</Analyses>
```

A.2.3 Batch Test Application Time Calculations

Listing 3 shows a sample batch operations input file and Listing 4 shows the results of the batch operations shown in Listing 3.
Listing 4 Output of the sample batch operations file in Listing 3

```xml
<?xml version="1.0"?>
<!--
Definition of Acronyms:
ttt: Total Test Time (in TCKs)
ttlbt: Test Time Lower Bound (Theoretical)
ttlbc: Test Time Lower Bound (Calculated)
to: Total Overhead
==================================================================
spo: Overhead due to SIB programming
==================================================================
suf: Shift/Update Flops
pf: Pipeline Flops
swf: SelectWIR Flops
==================================================================
wpo: Overhead due to WIR programming
==================================================================
cuc: Overhead due to Capture/Update cycles
==================================================================
nds: Number of doorway SIBs
-->  
<AnalysesResults SourceName="Sequential">
  <Result ID="1" Design="P22810_F.xml" Method="0" ttt="8998433"
ttlbt="8172047" ttlbc="8172047" spo="701176" suf="701176" pf="0"
swf="0" wpo="0" cuc="125210" nds="0" />
  <Result ID="2" Design="P34392_F.xml" Method="0" ttt="18320264"
ttlbt="16728056" ttlbc="16728056" spo="1260498" suf="1260498" pf="0"
swf="0" wpo="0" cuc="331710" nds="0" />
  <Result ID="3" Design="P93791_F.xml" Method="0" ttt="32248114"
ttlbt="31396374" ttlbc="31396374" spo="736640" suf="736640" pf="0"
swf="0" wpo="0" cuc="115100" nds="0" />
</AnalysesResults>
```
Appendix B

List of Acronyms

Table B.1: List of acronyms used in this report with their description

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>BIST</td>
<td>Built-In Self-Test</td>
</tr>
<tr>
<td>BR</td>
<td>Bypass Register</td>
</tr>
<tr>
<td>BSDL</td>
<td>Boundary Scan Definition Language</td>
</tr>
<tr>
<td>BSR</td>
<td>Boundary Scan Register</td>
</tr>
<tr>
<td>CDR</td>
<td>Core Data Register</td>
</tr>
<tr>
<td>CUC</td>
<td>Capture/Update Cycle</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronic Design Automation</td>
</tr>
<tr>
<td>DFT</td>
<td>Design for Test</td>
</tr>
<tr>
<td>GWEN</td>
<td>Gateway Enable</td>
</tr>
<tr>
<td>HIP</td>
<td>Hierarchical Interface Port</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>IJTAG</td>
<td>Internal JTAG</td>
</tr>
<tr>
<td>IR</td>
<td>Instruction Register</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG</td>
<td>Joint Test Access Group</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SDI</td>
<td>Serial Data Input</td>
</tr>
<tr>
<td>SDO</td>
<td>SErial Data Output</td>
</tr>
<tr>
<td>SECT</td>
<td>Standard for Embedded Core Test</td>
</tr>
<tr>
<td>SIB</td>
<td>Select Instrument Bit / Segment Insertion Bit</td>
</tr>
<tr>
<td>SOC</td>
<td>System-on-Chip</td>
</tr>
<tr>
<td>TAM</td>
<td>Test Access Mechanism</td>
</tr>
<tr>
<td>TAP</td>
<td>Test Access Port</td>
</tr>
<tr>
<td>TAPC</td>
<td>TAP Controller</td>
</tr>
<tr>
<td>TAT</td>
<td>Test Application Time</td>
</tr>
<tr>
<td>TCK</td>
<td>Test Clock</td>
</tr>
<tr>
<td>TDI</td>
<td>Test Data Input</td>
</tr>
<tr>
<td>TDO</td>
<td>Test Data Output</td>
</tr>
<tr>
<td>TDR</td>
<td>Test Data Register</td>
</tr>
<tr>
<td>TMS</td>
<td>Test Mode Select</td>
</tr>
<tr>
<td>TRST</td>
<td>Test Reset</td>
</tr>
<tr>
<td>WBR</td>
<td>Wrapper boundary Register</td>
</tr>
<tr>
<td>WBY</td>
<td>Wrapper Bypass Register</td>
</tr>
<tr>
<td>WDR</td>
<td>Wrapper Data Register</td>
</tr>
<tr>
<td>WIR</td>
<td>Wrapper Instruction Register</td>
</tr>
<tr>
<td>WPP</td>
<td>Wrapper Parallel Port</td>
</tr>
<tr>
<td>WRCK</td>
<td>Wrapper Clock</td>
</tr>
<tr>
<td>WRSTN</td>
<td>Wrapper Reset</td>
</tr>
<tr>
<td>WSI</td>
<td>Wrapper Serial Input</td>
</tr>
<tr>
<td>WSO</td>
<td>Wrapper Serial Output</td>
</tr>
<tr>
<td>WSP</td>
<td>Wrapper Serial Port</td>
</tr>
</tbody>
</table>
Bibliography


The IEEE P1687 (IJTAG) standard proposal aims at providing a standardized interface between on-chip embedded test, debug and monitoring logic (instruments), such as scan-chains and temperature sensors, and the Test Access Port of IEEE Standard 1149.1 mainly used for board test. A key feature in P1687 is to include Segment Insertion Bits (SIBs) in the scan path. SIBs make it possible to construct a multitude of different P1687 networks for the same set of instruments, and provide flexibility in test scheduling. The work presented in this thesis consists of two parts. In the first part, analysis regarding test application time is given for P1687 networks while making use of two test schedule types, namely concurrent and sequential test scheduling. Furthermore, formulas and novel algorithms are presented to compute the test time for a given P1687 network and a given schedule type. The algorithms are implemented and employed in extensive experiments on realistic industrial designs. In the second part, design of IEEE P1687 networks is studied. Designing the P1687 network that results in the least test application time for a given set of instruments, is a time-consuming task in the absence of automatic design tools. In this thesis work, novel algorithms are presented for automated design of P1687 networks which are optimized with respect to test application time and the required number of SIBs. The algorithms are implemented and demonstrated in experiments on industrial SOCs.

<table>
<thead>
<tr>
<th>Nyckelord</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE P1687, IJTAG, Test Architectures, Test Time Calculation, Design Automation, Test Time Optimization</td>
<td></td>
</tr>
</tbody>
</table>
Copyright

Svenska

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådan sammanhang som är krävande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagens hemsida http://www.ep.liu.se/

English

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Farrokh Ghani Zadegan
Linköping, October 13, 2010