Test Time Analysis for IEEE P1687

Farrokh Ghani Zadegan[1], Urban Ingelsson[1], Gunnar Carlsson[2] and Erik Larsson[1]
[1] Linköping University, Linköping, Sweden
[2] Ericsson AB, Stockholm, Sweden
ghanizadegan@ieee.org, urbin@ida.liu.se, erila@ida.liu.se gunnar.carlsson@ericsson.com

***Abstract*—The IEEE P1687 (IJTAG) standard proposal aims at providing a standardized interface between on-chip embedded logic (instruments), such as scan-chains and temperature sensors, and the IEEE 1149.1 standard which provides test data transport and test protocol for board test. A key feature in P1687 is to include Select Instrument Bits (SIBs) in the scan path to allow flexibility in test architecture design and test scheduling. This paper presents algorithms to compute the test time in a P1687 context. The algorithms are based on analysis for flat and hierarchical test architectures, considering two test schedule types - concurrent and sequential test scheduling. Furthermore, two types of overhead are identified, i.e. control data overhead and JTAG protocol overhead. The algorithms are implemented and employed in experiments on realistic industrial designs.**

***Keywords*-Test Time Calculation, IEEE P1687 IJTAG, Test Architectures, Test Schedules**

I. Introduction

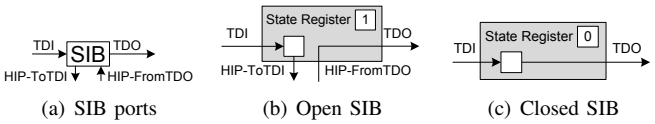
The complexity and reduced feature sizes in recent IC designs, necessitates the provision of on-chip infrastructures for test and debug. The IEEE 1149.1 standard (a.k.a. JTAG), originally intended for board test, has proved useful in this context for ad-hoc access to such on-chip infrastructure, as discussed in [1]. This means that JTAG is required to connect to a wide range of different electronic circuits, such as embedded test logic. Therefore, there is a need to standardize how JTAG circuitry should connect to the embedded logic. The IEEE P1687 standard proposal [2], [3], which describes a flexible test data transport infrastructure to interface JTAG to the embedded test logic, aims to address this need of standardization. When ratified, P1687 will specify methods for access and control of embedded instruments [2]. Here, instrument refers to any device (DFT-specific or not) with a scanable register that could be included in the JTAG scan path. Examples of instruments include embedded sensors and scan-chains. This paper considers using P1687 for testing the internal cores of an IC and therefore, the instruments mentioned in the remainder of this paper are scan-chains.

Before the P1687 proposal, similar efforts for interfacing instruments with JTAG have been presented in [4]–[6], but it should be noted that compared with the approaches in [4]–[6], P1687 introduces a component called Select Instrument Bit (SIB). The SIBs provide flexibility in setting up the scan path by making it possible to include and exclude instruments. An example of this flexibility is the possibility to build hierarchies of SIBs and instruments, as will be discussed in this paper. To setup the scan path, P1687 proposes to transport control data together with test data on a single wire (the JTAG scan path), and this will affect the Test Application Time (TAT) compared to previous approaches [4]–[6]. Since IEEE P1687 has recently been proposed, only a few studies have considered it [4], [7], [8]. In particular, no study has investigated the impact of IEEE P1687 on TAT. Therefore, this paper will analyze (Sections III and IV) the implications of IEEE P1687 on TAT, and it is shown that the TAT depends on several parameters including the placement of instruments and SIBs. To make it possible to calculate the TAT for large designs with complicated structures of instruments and SIBs, a first TAT calculation method IJTAGcalc is presented (Section V) and used to calculate the TAT for a number of ITC'02 benchmark designs (Sections VI and VII).

II. IEEE P1687 IJTAG Overview

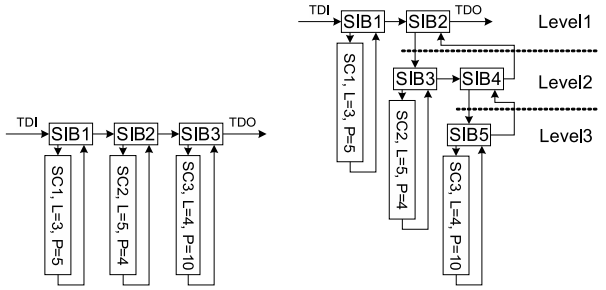
The IEEE P1687 standard proposal is at the time of this paper's writing in the final stage of review, and is sufficiently mature, so that no changes are expected that would affect the study presented in this paper. P1687 proposes an interface for connecting instruments with JTAG for test data transport. The interface is implemented by adding a Test Data Register called Gateway, composed of one or more SIBs, to the JTAG circuitry. The Gateway is selected by loading an instruction called Gateway Enable (GWEN) through the IR-scan procedure [2]. Loading GWEN makes the Gateway accessible from the JTAG Test Access Port (TAP) terminals, Test Data Input (TDI) and Test Data Output (TDO). Once the GWEN instruction is set, any further access, configuration and control of instruments through P1687 will be done through DR-scans [2]. The IR-scan and the DR-scan procedures are controlled by a state machine which is defined by JTAG and implemented in the JTAG TAP controller.

Fig. 1(a) shows a simplified view of a SIB. In addition to the TDI and TDO ports, the SIB has a hierarchical interface port (HIP) that is used to connect to a P1687 segment which can be either an instrument or other SIBs. A SIB acts as a doorway since it has two states, it is either open (Fig. 1(b)) and includes the segment on the HIP in the scan path, or it is closed (Fig. 1(c)) and transfers the data from its TDI port to its TDO port, excluding the segment on the HIP. Whether the SIB is open or closed, it corresponds to a 1-bit data register on the scan path. The state of the SIB is set by scanning in a control bit into its register, and the next time the JTAG state machine [9] is in its Update-DR state, the SIB will transfer the control bit to its state register (shown in Fig. 1(b) and Fig. 1(c)). The Update-DR state is part of the progression of five states in



(a) SIB ports (b) Open SIB (c) Closed SIB

Fig. 1. Simplified view of SIB component



(a) Flat architecture (b) Hierarchical architecture

Fig. 2. Example test data transport infrastructures

the JTAG TAP controller, controlling test stimulus application and capturing of the corresponding test response (Exit1-DR, Update-DR, Select-DR, Capture-DR and Shift-DR). In this paper, the progression of the five states will be referred to as a capture-and-update cycle (CUC). The control bits for the SIBs are embedded in each test stimuli vector.

Fig. 2(a) shows a test data transport infrastructure and three instruments (SC1, SC2 and SC3) including three SIBs, one for each instrument. In Fig. 2(a), L stands for the length of the scan-chain for each instrument and P stands for the number of test patterns in an instrument-specific test. The typical test application process is to scan in test stimuli from TDI, through the SIBs, into the scan-chains of the instruments, apply the test and scan out the captured test responses, 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.

The type of test architecture that is implemented by the SIBs in Fig. 2(a), is called a flat architecture in the remainder of this paper. In the flat architecture no SIB is connected to the HIP of another SIB. Fig. 2(b) shows another structure of SIBs that connect the same three instruments (SC1, SC2 and SC3) as in Fig. 2(a). Here there are five SIBs and three of these SIBs are connected from the HIP of SIB2. This type of SIB structure is called hierarchical architecture in the remainder of this paper. Each SIB that has another SIB on its HIP represents the doorway to another level of hierarchy, such as SIB2 and SIB4 in Fig. 2(b). In this context, SIB placement impacts TAT.

III. Analysis: Flat architecture

In this section, the flat architecture is analyzed regarding the TAT. Two test schedules are considered. In Section III-A, all the tests are started at the same time (the concurrent schedule). In Section III-B, the tests are performed sequentially, one test at a time (the sequential schedule).

A. Concurrent schedule

Regarding the concurrent schedule, the following will describe how to calculate the TAT for the flat architecture, with the help of Table I and Fig. 3. In Fig. 3, the gray

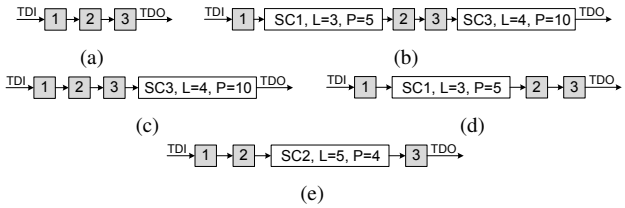


Fig. 3. Scan path configurations of the flat architecture example

TABLE I
Flat architecture, concurrent schedule

Sequence type	Scanned bits SIBs	SC1	SC2	SC3	$\sum$	Scanned bits +CUC
Setup-sequence	3	0	0	0	3	$3+5$
Scan-sequence 1-5	3	3	5	4	15	$(15+5)\cdot 5$
Scan-sequence 6	3	3	0	4	10	$10+5$
Scan-sequence 7-11	3	0	0	4	7	$(7+5)\cdot 5$
TAT						$\sum$=183

boxes represent the data register inside the correspondingly numbered SIBs. Before applying the first test pattern, the SIBs must be opened, since the scan path initially only consists of the SIBs, as shown in Fig. 3(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 corresponds to the 1 bit register of a closed SIB (Fig. 1(c)), and they are accounted for on the row marked Setup-sequence in Table I, column "SIBs". After the CUC, all instruments are included in the scan path, as shown in Fig. 2(a). At this point, test patterns can be applied to all three instruments, with a total scan-chain length of $1_O + 3_{SC1} + 1_O + 5_{SC2} + 1_O + 4_{SC3}$=15 bits, where 1_O corresponds to the 1 bit SIB register that is between each open SIBs' TDI port and its instrument (Fig. 1(b)). The total number of such 1_O bits is accounted for on the row marked Scan-sequence 1 in Table I, in the column "SIBs". Similarly, the number of bits (called 3_{SC1}, 5_{SC2}, 4_{SC3} above) for the three instruments are counted in the columns SC1, SC2 and SC3. After four test patterns have been applied, the test for instrument SC2 is complete and its scan-chain should be excluded from the scan path by setting the control bit so that SIB2 is closed while keeping SIB1 and SIB3 open. This operation, to close SIB2, cannot occur until the test response for the last test pattern of SC2 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-chain length of 1_O+3_{SC1}+1_C+1_O+4_{SC3}=10 bits. Here, 1_C corresponds to the 1 bit register between the TDI and TDO ports of a closed SIB (Fig. 1(c)). The scan path is as shown in Fig. 3(b). After the sixth scan-sequence, the test for instrument SC1 is complete and SIB1 is closed. The scan path becomes as shown in Fig. 3(c). 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 scan-sequences the total scan-chain length is 1_C+1_C+1_O+4_{SC3}=7 bits.

Table I shows the number of bits of different types (columns) that are scanned in for each sequence (rows). The

TABLE II
FLAT ARCHITECTURE, SEQUENTIAL SCHEDULE

Sequence type	Scanned bits SIBs	SC1	SC2	SC3	$\sum$	Scanned bits +CUC
Setup-sequence	3	0	0	0	3	8
Scan-sequence 1-6	3	3	0	0	6	$11 \cdot 6$
Scan-sequence 7-11	3	0	5	0	8	$13 \cdot 5$
Scan-sequence 12-22	3	0	0	4	7	$12 \cdot 11$
TAT						$\sum$=271

column marked $\sum$ sums the bits to get the total scan path length. These are the number of bits that are scanned for each sequence. The last column shows the number of bits to scan in for each sequence plus the five clock cycles that are required to perform a capture-and-update cycle (CUC) for JTAG [9] (see Section II), times the number of sequences. The TAT is the sum of the values in the last column of Table I, as shown on the last row. In this example, the TAT is 183 clock cycles.

It should be noted that the SIB control bits contribute to TAT by 36 clock cycles. Furthermore, the number of clock cycles spent performing CUC is 60. These 36+60=96 clock cycles spent scanning SIB control bits and performing CUC is considered overhead, because no actual test data is transported during this time. Thus the overhead ratio is $96/183 \approx 0.52$ in this example. In general, many test patterns and short scan-chains lead to a high overhead ratio. Of course, the length of the instrument scan-chains does not impact the overhead (the number of scanned SIB control bits or the number of CUCs), but long scan-chains effectively limit the overhead ratio.

B. Sequential schedule

In this section, the TAT will be calculated for the flat architecture considering the sequential schedule. Fig. 3 and Table II will be used to explain the steps of the test. Before the test process starts, the scan path is as shown in Fig. 2(a), and three bits are used in the setup-sequence to open SIB1 so that for the six following scan-sequences, the scan path is as shown in Fig. 3(d). The row marked Scan-sequence 1-6 in Table II shows that the three bits of SC1 are included in scan path. After Scan-sequence 6, the five test patterns of the test for SC1 have been applied and the test responses have been scanned out while closing SIB1 and opening SIB2 so that the scan path becomes as shown in Fig. 3(e). 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 II). Fig. 3(c) shows the scan path as it is after Scan-sequence 11. Finally, the Scan-sequences 12-22 (Table II) are applied to complete the test for SC3 and scan out the last test responses, while closing SIB3. As can be seen from Table II, the TAT for the sequential schedule is 271 clock cycles, which should be compared to 183 clock cycles for the concurrent schedule discussed in Table I. 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 overhead.

IV. ANALYSIS: HIERARCHICAL ARCHITECTURE

Similar to the test time analysis for the flat architecture (Section III), this section will discuss the TAT for the hierarchical architecture shown in Fig. 2(b). Table III and Table IV show the steps to calculate the TAT regarding the concurrent and sequential schedules, respectively. In Tables III and IV, the column "Scan path" refers to the scan path that relates to each scan sequence.

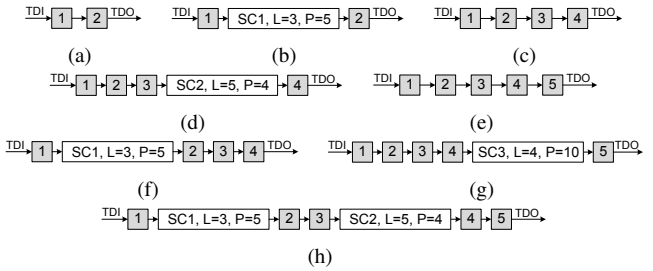


Fig. 4. Scan path configurations of the hierarchical architecture example

TABLE III
HIERARCHICAL ARCHITECTURE, CONCURRENT SCHEDULE

Sequence type	Scanned bits SIBs	SC1	SC2	SC3	$\sum$	Scanned bits +CUC	Scan path
Set-up	2	0	0	0	2	7	Fig. 4(a)
Scan 1	4	3	0	0	7	12	Fig. 4(f)
Scan 2	5	3	5	0	13	18	Fig. 4(h)
Scan 3-6	5	3	5	4	17	$22 \cdot 4$	Fig. 2(b)
Scan 7-13	5	0	0	4	9	$14 \cdot 7$	Fig. 4(g)
TAT						$\sum$=223	

As can be seen from Table III, the TAT for the hierarchical architecture and the concurrent schedule is 223 clock cycles, which should be compared to 183 clock cycles for the corresponding schedule and the flat architecture. In this example, the hierarchical architecture leads to a longer TAT because of two factors. Firstly, the overhead from the additional SIBs affects the TAT. Secondly, the overhead in terms of capture-and-update cycles (CUC) is higher. From Table III, it should be noted that the SIB overhead varied depending on the scan-sequence and the hierarchy level for the tested instrument.

As for the sequential schedule (Table IV), the total TAT is 310 clock cycles, which should be compared with 271 clock cycles for the sequential schedule and the flat architecture. The reason for the higher TAT with the hierarchical architecture is more SIB-overhead and more CUCs. In Section III-A it was discussed how the TAT and the overhead ratio scale with different numbers of test patterns P and the scan-chain length L for the instruments. The observations made there apply also to the hierarchical 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.

It should be noted that in the example discussed in Section III and this section, the flat 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 architectures and schedules.

V. TEST TIME CALCULATION METHOD: IJTAGCALC

This section will describe a method called IJTAGcalc for calculating the TAT for general instances of the flat and hierarchical architecture types in Fig. 2. The method consists

TABLE IV
HIERARCHICAL ARCHITECTURE, SEQUENTIAL SCHEDULE

Sequence type	Scanned bits SIBs	SC1	SC2	SC3	$\sum$	Scanned bits +CUC	Scan path
Set-up	2	0	0	0	2	7	Fig. 4(a)
Scan 1-6	2	3	0	0	5	$10 \cdot 6$	Fig. 4(b)
Set-up	4	0	0	0	4	9	Fig. 4(c)
Scan 7-11	4	0	5	0	9	$14 \cdot 5$	Fig. 4(d)
Setup	5	0	0	0	5	10	Fig. 4(e)
Scan 12-22	5	0	0	4	9	$14 \cdot 11$	Fig. 4(g)
TAT						$\sum$=310	

of two sets of algorithms corresponding to the concurrent and the sequential 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 and the SIB is a leaf node in case the HIP only 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 Fig. 2(b), where SIB2 has the subtree consisting of SIB3, SIB4 and SIB5. SIB2, which is on Level1, is the parent of SIB3 and SIB4, since they are on Level2. To describe the placement of instruments, it is considered that each SIB can have at most one instrument connected to its HIP. The properties of this instrument, i.e. L and P of its scan-chain, are associated with the connected SIB. If an instrument is connected to a SIB s, then $s.ILength$, $s.IPatterns$ and $s.IRemaining$ define the properties of the instrument. Initially, $s.IPatterns$ and $s.IRemaining$ are set to the number of patterns (P) of the instrument, and $s.ILength$ is set to the length of the instrument scan-chain (L). 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 a subtree on its HIP, $s.ILength$ 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 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.

A. IJTAGcalc for the Concurrent Schedule

This section describes the IJTAGcalc method for the concurrent 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 I) and by summing the number of bits in each scan-sequence ($SSLength$) with the CUC for each scan-sequence, the TAT (TAT) is added up (line 7). The iterations finish, when there are no more test patterns to apply, as given by the

Algorithm 1 IJTAGcalcConcurrent

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

Algorithm 2 Traverse($node$)

```
1: subtreeSPatternList := {−1}
2: for each child ∈ node.children do
3:     SSLength := SSLength + 1
4:     if child.SRemaining > −1 or child.IRemaining > −1 then
5:         if child.IsOpen=False then
6:             child.IsOpen := True
7:         else
8:             child.SRemaining := Traverse(child)
9:             SSLength := SSLength + child.ILength
10:            child.IRemaining := child.IRemaining − 1
11:        end if
12:    else
13:        child.IsOpen := False // might already be closed
14:    end if
15:    append max{child.SRemaining,child.IRemaining} to
       subtreeSPatternList
16: end for
17: return max{subtreeSPatternList}
```

$SRemaining$ variable. At this point, the TAT (TAT) will have been found.

As can be seen in IJTAGcalcConcurrent (Algorithm 1), Traverse (Algorithm 2) is an important function, which returns the value for $SRemaining$. It also updates the global variable $SSLength$ which keeps track of the number of bits that have been scanned in during each scan sequence. 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 $SSLength$ 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 $SRemaining$ and $IRemaining$ 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 $SSLength$ variable is incremented by $ILength$ 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 any of the child nodes (line 15 and line 17).

B. IJTAGcalc for the Sequential Schedule

This section describes the IJTAGcalc method for the sequential schedule. The algorithm is called IJTAGcalcSequential (Algorithm 3). IJTAGcalcSequential considers the same type

Algorithm 3 IJTAGcalcSequential($node$)

1: $SIBs := SIBs$+size($node.Children$)
2: $TAT := TAT + (node.ILength + SIBs + CUC) \cdot (node.IPatterns + 1)$
3: **if** size($node.Children$)>0 **then**
4: **for** each $child \in Children(node)$ **do**
5: IJTAGcalcSequential($child$)
6: **end for**
7: **end if**
8: $SIBs := SIBs-$size($node.Children$)

of tree representation as was discussed in Section V. The key idea which makes the TAT 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 the example of Table IV. 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 $SIBs$ and TAT should be set to 0. Here, $SIBs$ is a variable that counts the number of SIBs on the scan path, and TAT is the variable that will contain the TAT when IJTAGcalcSequential terminates. The number of SIBs on the scan path will vary according to the location of the instrument that is being tested within the P1687 structure. 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 5). 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 8), 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 IV. 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 Fig. 2(b). Here, $node.ILength$ and $node.IPatterns$ are both 0. The TAT is increased by the sum of $SIBs$ and CUC. If the SIB passed as the $node$ parameter has children SIBs, the IJTAGcalcSequential function will be called recursively for each of these children.

VI. Experimental Setup

The IJTAGcalc method has been used in experiments to analyze six ITC'02 SOC Test Benchmarks [10] with regard to the TAT. Since in the context of P1687 all test patterns are transported through a single wire, the scan-chains and boundary cells corresponding to the core inputs and outputs 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 bi-directional terminal is counted as three cells [9], [11]. To consider the testing requirements for hierarchical cores, the boundary cells of each embedded child core is appropriately included in the set of boundary cells of its direct parent core [11]. 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 V 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 $ShiftedTestData = \sum_{i=1}^{N} L_i \cdot (P_i + 1)$, where 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. It should be noted that Module 0, mentioned in the ITC'02 benchmarks, as well as modules containing BIST-engines are not included in the experiments and are also not accounted for in Table V. All six SOCs were included in an experiment with the flat architecture (denoted with "F") and three of the designs (P22810, P34392 and A586710 which have hierarchical cores) were also used in an experiment with the hierarchical architecture (denoted with "H"). In the hierarchical architectures the children cores are assigned to the next hierarchy level in relation to their parent core. This is to make a one-to-one correspondence to the hierarchy in the SOC itself, for the sake of experiment. 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 TAT, we require one such implementation and we have chosen to take inspiration from the notion of the hierarchical cores.

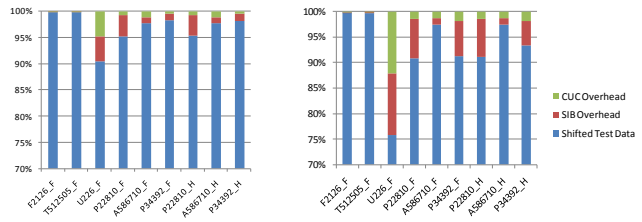
TABLE V
Considered benchmark designs

SOC	# Cores	Patterns		Length		Shifted
		Min	Max	Min	Max	Test Data
F2126	4	103	422	502	8875	5330439
T512505	31	3	3370	7	43922	165400967
A586710	5	2945	1914433	69	21324	843806808
P34392	19	128	12336	11	9669	16728056
P22810	28	1	12324	52	11878	8172047
U226	5	15	2666	20	1201	252929

VII. Experimental Results

Fig. 5(a) shows experimental results using the concurrent schedule. 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 TAT. Since the overhead constitutes a small portion of TAT, the vertical axis is scaled from 70% to allow details to be visible. On the horizontal axis are nine designs and the six left-most designs have flat architecture as indicated by the labels.

Fig. 5(a) shows that F2126_F and T512505_F both have relatively low overhead. 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_F. For T512505_F, 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



(a) Concurrent schedule (b) Sequential schedule

Fig. 5. Experimental Results

the overall shifted test data. Therefore, this core made such impact on TAT that the overhead from the other cores became negligible. U226_F contains some cores that have short core-chains and a large number of test patterns. Therefore, the overhead ratio for the TAT of U226_F, which is about 10%, is larger than that of the other designs. 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 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 V) 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 architecture typically applies also to the corresponding designs (from the same SOC) with hierarchical architecture, as can be seen in Fig. 5(a). It should be noted that even though there was a noticeable difference between the TAT of the flat architecture and that of the hierarchical architecture for the small example of Fig. 2, see Section IV, the experiments with SOC benchmarks show little difference in overhead ratio. P22810_H and P22810_F show similar results because there are few hierarchical levels relative to the number of cores. Therefore, P22810_H has characteristics similar to those of its flat counterpart.

Fig. 5(b) shows results for the experiments that use sequential test scheduling. From the two figures it can be seen that the overhead ratio is larger when using a sequential schedule compared to using a concurrent 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 schedules. Design P34392 shows a noticeable difference in overhead ratio between the flat architecture, marked by P34392_F, and the hierarchical architecture, marked by P34392_H, in the case of the sequential schedule. For the flat architecture the overall overhead is about 9% and for the hierarchical architecture the overhead is about 7% with the difference mainly due to a lesser SIB overhead. In P34392_F, every scan-sequence, independent of the core, includes 19 SIBs (the same number as the number of cores). However, in P34392_H, the scan-sequences contain on average 13 SIBs. Since this is significantly less than 19 SIBs, this explains the noticeable difference in SIB overhead.

To summarize, the IJTAGcalc method has been applied to calculate the TAT for six SOCs and the impact of the SIB structure (flat or hierarchical) has been observed.

VIII. Conclusion

In expectation of ratification of IEEE P1687 IJTAG, this paper has presented an analysis of how test time is to be calculated. P1687 aims to standardize access to embedded logic through IEEE 1149.1 JTAG. In this context the accessed logic is called an instrument. The access is through a single wire interface for data transport. This interface can be dynamically configured through a component called Select Instrument Bit (SIB) to create a multitude of alternative scan paths, by opening up levels of hierarchy. A number of SIBs are placed on the scan path and they are configured using the same single wire interface as the one used for the data transport.

The analysis explored the configuration possibilities provided by SIBs and showed that overhead in terms of clock cycles spent shifting other bits than test data can be put into two categories, namely time spent shifting test control data (SIB overhead) and time spent performing test stimuli application and test response capture sequences in the JTAG controller (CUC overhead). In the analysis it has been observed how long scan-chains and a low number of test patterns lead to a low overhead ratio, compared to the total TAT.

This paper presents a test time calculation method 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 scheduling and sequential scheduling. The IJTAGcalc method was employed to perform experiments on ITC'02 benchmark designs. The results show that industrial and academic SOCs from the set of benchmarks can have an overhead ratio of up to 9% and 24%, respectively. The results can be well explained by the observations made in the analysis. For a particular benchmark, it was seen that the test architecture has a noticeable impact on the overhead.

References

[1] J. Rearick, B. Eklow, K. Posse, A. Crouch, and B. Bennetts, "IJTAG (Internal JTAG): A Step Toward a DFT Standard," in *Proc. ITC*, 2005.
[2] IJTAG, "IJTAG - IEEE P1687," 2010. [Online]. Available: http://grouper.ieee.org/groups/1687
[3] A. L. Crouch, "IJTAG: The Path to Organized Instrument Connectivity," in *Proc. ITC*, 2007, pp. 1–10.
[4] L.-T. Wang *et al.*, "Turbo1500: Toward Core-Based Design for Test and Diagnosis Using the IEEE 1500 Standard," in *Proc. ITC*, 2008, pp. 1–9.
[5] E. J. Marinissen and T. Waayers, "Infrastructure for modular SOC testing," in *Proc. CICC*, 2004, pp. 671–678.
[6] Y. Zorian and A. Yessayan, "IEEE 1500 utilization in SOC design and test," in *Proc. ITC*, 2005, pp. 1–10.
[7] M. Higgins, C. MacNamee, and B. Mullane, "SoCECT: System on Chip Embedded Core Test," in *Proc. DDECS*, 2008, pp. 326–331.
[8] J. Rearick and A. Volz, "A Case Study of Using IEEE P1687 (IJTAG) for High-Speed Serial I/O Characterization and Testing," in *Proc. ITC*, 2006, pp. 1–8.
[9] IEEE association, "IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture," 2001.
[10] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *Proc. ITC*, 2002, pp. 519–528.
[11] S. K. Goel, "Test-access planning and test scheduling for embedded core-based system chips," Ph.D. dissertation, University of Twente, 2005. [Online]. Available: http://doc.utwente.nl/48260/