Optimized Integration of Test Compression and Sharing for SOC Testing

Anders Larsson, Erik Larsson, Petru Eles, and Zebo Peng

Embedded Systems Laboratory
Linköpings Universitet
SE-582 83 Linköping, Sweden

Abstract[1]

The increasing test data volume needed to test core-based System-on-Chip contributes to long test application times (TAT) and huge automatic test equipment (ATE) memory requirements. TAT and ATE memory requirement can be reduced by test architecture design, test scheduling, sharing the same tests among several cores, and test data compression. We propose, in contrast to previous work that addresses one or few of the problems, an integrated framework with heuristics for sharing and compression and a Constraint Logic Programming technique for architecture design and test scheduling that minimizes the TAT without violating a given ATE memory constraint. The significance of our approach is demonstrated by experiments with ITC'02 benchmark designs.

1. Introduction

The technology development has made it possible to build highly complex integrated circuits, so called system chips or System-on-Chips (SOCs). A common way to design SOC is to employ a modular core-based approach where each module can be tested as a separate unit. The testing of these chips requires high test data volume, which leads to long test application times (TAT) and a need for large automatic test equipment (ATE) memory. Different techniques have been proposed to lower the TAT and ATE memory requirement such as; test architecture design, test scheduling, test data compression, and test sharing.

Test architecture design and test scheduling techniques have been proposed while minimizing TAT [13, 14] which also has been extended for an abort-on-fail environment [15]. These techniques efficiently reduces the TAT and the utilization of the ATE memory. However, the test data volumes are increasing and techniques such as test data compression and test sharing have been proposed.

For test data compression several schemes have been investigated, for instance Huffman [3], Colomb [4], and Frequency-Directed Run-Length (FDR) [2] codes. The general scheme is that the high number of unspecified bits [1, 2], so called don't-care bits, in the test stimuli (TS) are filled such that high test data compression ratio is achieved. Compressed TS for each core are stored in the ATE, and at test application the code words are sent to the system under test, decompressed and applied to the cores. The produced responses (PR) are usually compressed using either a multiple-input signature-register (MISR) or by some other compaction logic such as combinational compactors. In the case when MISRs are used, at the end of the testing the MISR signature is shifted out and compared with the expected signature. The draw-backs with MISRs and combinational compactors are the sensitivity to unknown values. Additional logic must therefore be added for tolerating them [10]; however only a limited number of unknown bits can be handled. Further, by using a MISR the testing cannot be terminated immediately when a fault is present (abort-on-fail testing). Instead, the testing must continue until the final signature is produced. To address these problems, an architecture that does not make use of MISRs has been proposed [11].

For test sharing, the aim is to find overlapping tests where an overlapping test set is used to test several cores [5, 6, 9]. For an overlapping tests, the same TS are sent to all cores that share the test, and the PR are transported on dedicated test access mechanism (TAM) wires separately, since cores that share a test can output different PR these cannot share TAM to the ATE for evaluation. The advantage of sharing is that it reduces the test time, the ATE memory requirement and the TAM wire usage (as TS is broadcasted to several cores).

In this paper we address the test architecture design and test scheduling problems as well as the test compression and test sharing problems. We assume that given is a core-based modular SOC with a dedicated TAM and tests for each core. We make use of an architecture that does not require MISRs [11] but instead of using a processor for decompression we use the nine code (9C) compression technique [16]. The following trade-offs have been identified and addressed: between compression and sharing in terms of test data volume (number of bits), and between sharing and test architecture design in terms of TAT.

The trade-off between compression and sharing in terms of the test data volume is explained by considering two tests. By sharing, i.e., finding overlapping sequences in the two tests, which is used to create a new test, the amount of don't-care bits will decrease. Since the shared test will have less don't-care bits, it is likely that it will suffer from a lower compression ratio compared to when the tests are compressed individually. This means that the size of the compressed shared test could be larger than the sum of the two separately compressed tests. Hence, it is not obvious to determine which tests that should be shared and which tests that should be compressed.

The trade-off between sharing and test architecture design in terms of TAT is explained as follows: as described above, only the TS are shared between cores, the PR are transported on separate

1. The research is partially supported by the Strategic Integrated Electronic Systems Research (STRINGENT) program.

TAM wires, hence the TAM wire architecture will be different when using sharing compared when sharing is not used, consequently affecting the TAT.

The major contribution of this paper is twofold. First, we demonstrate that the integration of test sharing and test compression for core-based SOCs will lead to decreased test data volume. Second, we address the test scheduling and test architecture problem, exploring the trade-off between sharing and compression, while minimizing the TAT under ATE memory constraints. The problem has been implemented and solved using Constraint Logic Programming (CLP) [8]. The efficiency of the proposed technique has been demonstrated by experiments using ITC'02 benchmark designs.

The rest of this paper is organized as follows. In Section 2 the used SOC test architecture is described. The problem is formulated in Section 3 and the techniques for test sharing and compression are described in Section 4. In Section 5 the test application time minimization is described and the experimental results are presented in Section 6. Conclusions are in Section 7.

2. SOC test architecture

In this section the test architecture used for the test data transportation and decompression is described.

Let us first describe the common practice test architecture. The cores are scan tested and the scanned elements at each core are formed to wrapper chains that are connected to TAM wires. The TAM wires are connected to the ATE. At test application, TS are sent to the SOC and the PR are sent to the ATE. The ATE compares the PR with the expected test responses (ER) to determine if the chip is faulty.

In the case when test data compression is used, it is common to make use of MISRs for compaction of the PR. However, as discussed above, there are a number of disadvantages with MISRs and we therefor make use of a MISR free architecture proposed by Larsson and Persson [11]. The general idea is to store compressed TS, compressed ER, and compressed masks (M) in the ATE, as illustrated in Figure 1. The compressed TS, ER, and M are sent to the SOC under test and decompressed on the chip. Test evaluation is also performed on-chip using a comparator. Figure 1 also shows the placement of the decoder and comparator.

In contrast to using a processor core for the decompression [11], we make use of an on-chip decoder [16] to decompress the compressed tests. We assume a decoder where for each TAM wire we have a decoder block. A decoder block can act as an input decoder *d-in* in the case the decoder block is configured to receive TS. If the decoder block is configured as output decoder *d-out* it receives ER and a mask.

3. Problem formulation

Given is a system with n cores $c_1, c_2, \ldots, c_n$ where for each core c_i the following is given:

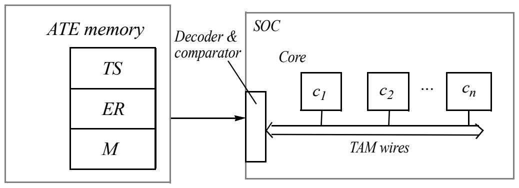


Figure 1. Test architecture and ATE memory organization.

- sc_i - the number of scan chains,
- ff_{ij} - the number of flip-flops in scan chain j,
- wi_i - the number of input wrapper cells,
- wo_i - the number of output wrapper cells,
- $nff_i = \sum_{j=1}^{sc_i} f_{ij}$ - the total number of flip-flops,
- $T_i = \{TS_i, ER_i, M_i\}$ - an initially given dedicated test consisting of TS_i, ER_i, and a test mask M_i,
- $TS_i = (ts_{i1}, \ldots, ts_{il})$ - a sequence of l patterns, where ts_{ik} consists of $nff_i + wi_i$ bits and each bit can be *0, 1*, or *x*,
- $ER_i = (er_{i1}, \ldots, er_{il})$ - a sequence of l patterns, where er_{ik} consists of $nff_i + wo_i$ bits and each bit can be *0, 1*, or *x*,
- $M_i = (m_{i1}, \ldots, m_{il})$ - a sequence of l mask patterns, where m_{ik} consists of $nff_i + wo_i$ bits and each bit can be *0* or *1*. A *1* indicates that the corresponding bit in the PR is a care bit and should be checked with the ER otherwise it is a don't-care bit and should be masked.

Also given for the system is the number of TAM wires, W_{TAM}.

For the ATE the following is given:

- M_{ATE} - the number of bits that can be stored in the ATE memory,
- f_{ATE} - the clock frequency of the ATE.

The *share* and *compress* functions take two tests and one test respectively as input and generate a new test that is added to the list of alternative tests. This process of generating alternative tests is explained using two initially given dedicated tests T_1 and T_2. Table 1 shows how these two tests are used to generate new alternative tests. Column one lists the alternative tests and column two and three list which core(s) is tested by each test (marked as X in the table). For example, core c_1 can be tested using TA_1, TA_3, TA_5, or TA_6 (one test is sufficient in our approach). The fourth column lists the function used to generate the test.

Given the above, our problem is to select one test alternative for each core c_i, determine the architcture (the TAM wire usage), and start time t_i such that the TAT is minimized without exceeding the memory constraint M_{ATE}.

4. Sharing and compression

In this section we describe the sharing and compression techniques. The common objective of sharing and compression is to decrease the size of the test, which is stored in the ATE.

4.1 Sharing

The sharing problem is formulated as follows: for a given number of test sequences (TS and ER), find overlapping sequences that are used to generate a new test such that the size of the new test is minimal. An overlapping between two sequences is found iff for each position in the sequences both has the same value (0, 1, x) or one is a don't-care (x).

Let us use a small example to illustrate the test architecture when sharing is used. The example depicted in Figure 2 consists

Table 1. Test alternatives per core

Alternative test	Core c_1	Core c_2	Note
TA_1	X		*Initially given*
TA_2		X	*Initially given*
TA_3	X		*compress(T_1)*
TA_4		X	*compress(T_2)*
TA_5	X	X	*share(T_1,T_2)*
TA_6	X	X	*compress(share(T_1,T_2))*

of two cores, c_1 and c_2, which are tested by test T_1 and T_2, presented in Figure 3, respectively. The tests consist of TS_1 and TS_2, and ER_1 and ER_2. By using a mask (M_1, M_2 in Figure 3) for each test that marks the positions of each specified bit in the ER, it is possible to determine if the PR from the core are correct or not even in the presence of unspecified values. The latter is important since unspecified bits in the PR are becoming more common with technology scaling. A similar approach with a mechanism for masking unspecified values has been proposed for different purpose [12]. The scan chains, *a* to *d* in c_1 and *e* to *g* in c_2, are partitioned into wrapper chains that are connected to six TAM wires, TAM_1 to TAM_6.

The example illustrated in Figure 2(a) shows how the scan chains have been partitioned to wrapper chains and connected to TAM wires when the cores are tested using the dedicated tests, i.e., no sharing is used. In the example illustrated in Figure 2(b), it is assumed that test T_1 and T_2 have been merged into a new shared test, which will be broadcasted to c_1 and c_2 on TAM_1 and TAM_2. In this way, the merged test is shared. In order to separate the different PR from different cores the PR from each core is transported on separate TAM wires, TAM_3 to TAM_6. The shared TS and ER are stored in the ATE memory together with the masks M_1 and M_2.

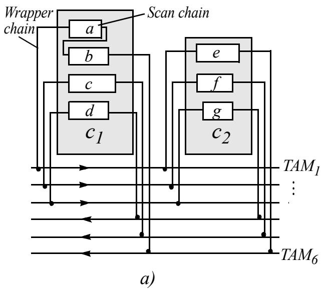
How two tests can be shared is illustrated in Figure 4, where the stimuli sequences TS_1 and TS_2 from Figure 3 have been partitioned into two wrapper chains, wr_1 and wr_2. In this example, scan chains *a, c,* and *e* have been assigned to wrapper chain wr_1 and scan chains *b, d, f,* and *g* to wrapper chain wr_2 that corresponds to the architecture in Figure 2 (b).

For balancing the wrapper chains before sharing, idle bits (don't-cares) are added such that all wrapper chains have equal length as illustrated in Figure 4(a). For ts_{11} three possible test sequence can potentially be overlapped, ts_{21}, ts_{22}, or ts_{23}. In Figure 4(b), ts_{11} and ts_{21}, are not overlapping (there are conflicting care bits that prohibits overlapping), hence, they cannot be shared. In Figure 4(c) ts_{11} and ts_{22}, are overlapping and a new, shared, TS sequence *ts_new* is generated.

We introduce a function called *share* that takes two tests[2] (TS and ER), T_i and T_j, as input and generates a new test, T_k:

$$share(T_i, T_j) \rightarrow T_k \qquad (1)$$

The *share* function, presented in Figure 5, consist of two steps. In Step1 (line 1 to 4) the TS are sorted according to the percentage of don't-care bits, such that the sequences with the most care bits are placed first in each test. The sorting is done in order to increase the utilization of the don't-care bits in each sequence. The test with the most patterns is selected as the reference, *ref_test*. In this example, test T_1 is selected. In Step2 (line 5 to 19), the sharing is performed by finding overlapping sequences. The size of the new shared test T_k will be equal to the size of the reference test if there exist an overlapping sequence for all sequences in the reference test. If an overlap is not found the size of T_k is increased.



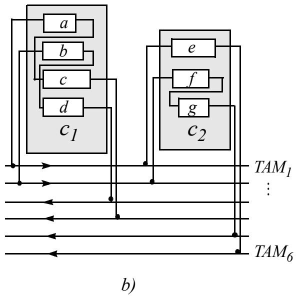


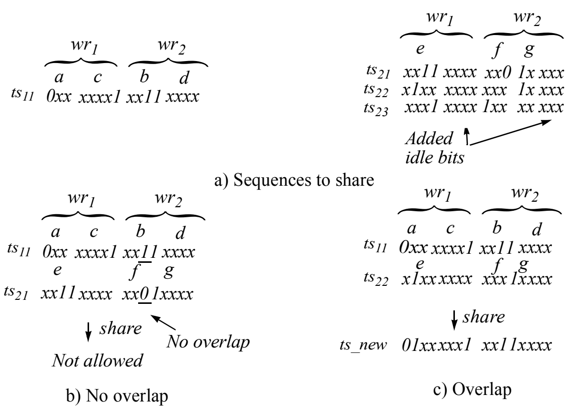
Figure 2. Test architecture (a) without sharing and (b) with sharing.

Scan chain

		a	b	c	d
T_1	TS_1	0xx	xx11	xxxx1	xxxx
		0xx	xxxx	xxx01	xxx1
		xx0	xxx0	xx0xx	xxxx
	ER_1	1xx	xxx1	xxxx0	xxxx
		11x	xxxx	xxxx1	xxxx
		xx0	xxxx	xxxxx	xxxx
	M_1	100	0001	00001	0000
		110	0000	00001	0000
		001	0000	00000	0000

		e	f	g
T_2	TS_2	xx11	xx0	1x
		x1xx	xxx	1x
		xxx1	1xx	xx
	ER_2	xxx0	xx0	x1
		xxxx	x1x	xx
		1xxx	x1x	xx
	M_2	0001	001	01
		0000	010	00
		1000	010	00

Figure 3. Initially given tests with don't-cares.



wr_1
wr_2
a c b d
ts_{11} 0xx xxxx1 xx11 xxxx
e f g
ts_{21} xx11 xxxx xx0 1x xxx
ts_{22} x1xx xxxx xxx 1x xxx
ts_{23} xxx1 xxxx 1xx xx xxx
Added idle bits
a) Sequences to share
ts_{21} xx11 xxxx xx0 1xxxx
share
No overlap
Not allowed
b) No overlap
ts_{22} x1xx xxxx xxx 1xxxx
share
ts_new 01xxxxx1 xx11xxxx
c) Overlap

Figure 4. Sharing using different sequences.

4.2 Compression

The objective of the compression is to generate a compressed test with minimal size. The compression function *compress* takes a test T_i as input and generates a new compressed test T_k:

$$compress(T_i) \rightarrow T_k \qquad (2)$$

```
    // Input: $T_1$ and $T_2$
    // Output: A new test, new_test
    // Step1:
1.  Sort($T_1$) // Sort $T_1$ according to % of don't-cares
2.  Sort($T_2$)
3.  ref_test = GetRefTest($T_1$, $T_2$)
4.  new_test = {}
    // Step2: Find overlapping sequences
5.  for each sequence i in ref_test begin // ref_test = $T_1$
6.      for each sequence j in $T_2$ begin
7.          if ts_new = Overlap($ts_{1i}$,$ts_{2j}$)
8.              new_test = new_test ∪ {ts_new}
9.              break
10.         end if
11.     end
12.     new_test = new_test ∪ {$ts_{1i}$}
13. end
14. for each sequence j in $T_2$ begin
15.     if $ts_{2j}$ is not previously added to new_test
16.         new_test = new_test ∪ {$ts_{2j}$}
17.     end if
18. end
19. return new_test
```

Figure 5. Share function.

2. From here and through the rest of the paper we use T_i to denote a given dedicated test or an alternative test.

We have made use of the fixed-length Nine-Coded Compression (9C) technique [16]. In the 9C coding the test sequence are divided into K-bits blocks. Each such block is then split into two halves and coded using nine different symbols (code words). This coding scheme enables test independent coding and it can be implemented using a small decoder.

The 9C coding technique is illustrated, with K=8 [16], in Figure 6 for different TS. In our work, each wrapper chain is coded separately. These examples also illustrate the trade-off between compression and sharing in terms of the test data volume. Using only compression for the initially given TS_1 and TS_2, the total number of bits to be stored in the ATE memory is 39 (20+19). This is less than the 49 bits needed to store the shared and compressed test alternative. This is due to the reduced amount of don't-care bits in the shared test that, for this case, leads to a poor compression.

5. Test application time minimization

In this section we describe the TAT minimization scheme, using integrated compression and sharing. We assume that the *share* and *compress* functions described in Section 4 have been used to generate a number of test alternatives per core. The TAT for the system is minimized by selecting one test for each core, defining the start time and TAM usage for each selected test.

We assume one test at a time on the TAM. Note, that due to sharing a test can be used to test one or more cores. Hence, the test transportation on the TAM is sequential, however, several cores may be tested concurrently. As described in Section 4, the cores that share a test will be connected to the same TAM wires for the TS and in order to separate the different test responses from different cores the PR from each core is transported on separate TAM wires.

The number of wrapper chains w_i for a test or alternative test T_i depends on the number of cores z that shares the test, and is given by:

$$w_i = \lfloor W_{TAM}/(z+1) \rfloor \qquad (3)$$

If no sharing is used ($z=1$) $w_i = W_{TAM}/2$, which means that half of the TAM width is used for the transportation of TS and the second half is used for PR as illustrated in Figure 2(a). In the case when two cores share a test ($z=2$) $w_i = W_{TAM}/3$; one third of the TAM width is occupied transporting the TS that are broadcasted to both cores and two thirds are used for the PR, one third for each core separately as illustrated in Figure 2(b).

The timing of the test transportation is illustrated in Figure 7. The tests are stored in the ATE memory and the control signals, *Comp* and *Share*, are used to determine the operation of the decoder. For example, if a test is not compressed (*Comp = 0*) the decoder is bypassed. In this work the ATE synchronization problem is solved by using an acknowledgement signal *ACK* from the decoder to the ATE, holding the ATE from transporting the next codeword while the decoder is busy [16]. In the example in Figure 7, the TS sequence ts_{ij} is coded using three code words cw_1, cw_2, and cw_3. The code words are transported from the ATE to the decoder using the ATE frequency f_{ATE} and the decompressed stimuli are transported and applied to the wrapper chains using the scan frequency f_{scan}.

The decompressed ER and mask must be synchronized with PR such that the comparator receives the correct sequences at correct time. We have two cases; when compression is not used and used. In the case when compression is not used, the test data is arranged such that the ER and mask are placed after the TS (according to the length of the wrapper chains) in the ATE such that ER and mask arrive to the comparator when PR are ready. In the case when compression is used, decompression takes different time depending on code words. We assume the longest decompression time for each code word.

The synchronization when compression is used is solved by applying TS with a frequency f_{scan} that is lower than the frequency of the ATE, f_{ATE}. The value of f_{scan} is calculated using a constant, *9CConst*, which is multiplied with the value of f_{ATE}. The value of *9CConst* is given by the number of test bits that each codeword contain (K=8) [16], which is divided by he maximum number of clock cycles needed to apply the longest codeword that the 9C coding uses (12+8) [16]. When a test that is not compressed is applied, the decoder is bypassed and the scan frequency will be the same as the ATE frequency. The value of f_{scan} is then given as follows:

$$f_{scan} = \begin{cases} f_{ATE}, & \text{without using compression} \\ 9CConst \times f_{ATE}, & \text{with compression} \end{cases} \qquad (4)$$

The scan frequency f_{scan} is used to calculate the TAT τ_i for a test, T_i at core c_i as follows:

$$\tau_i = ((1 + max\{si_i, so_i\}) \times l + min\{si_i, so_i\})/f_{scan}, \qquad (5)$$

where si_i and so_i are the length of the longest wrapper scan-in and scan-out chain of core c_i respectively and l is the number of test sequences.

The TAT τ_{tot} for a schedule with n tests is given as:

$$\tau_{tot} = max((\forall i, i = \{1, 2, \dots, n\})(t_i + \tau_i)), \qquad (6)$$

where t_i is the start time when the test is applied to the core c_i.

Given above definitions, the TAT minimization problem can be regarded as a problem of selecting one test alternative, i.e., determine which test to compress and/or share, for each core in the system. In order to solve the test selection problem a number of H possible alternative tests is generated for the system using the initially given dedicated tests. To illustrate the complexity of the test selection problem, we consider a system consisting of n cores. The number of tests generated using the dedicated given tests and the *share* function equals the sum of the number of possible k-subset (where $k = \{1, 2, \dots, n\}$) of a set of size n. For example, a

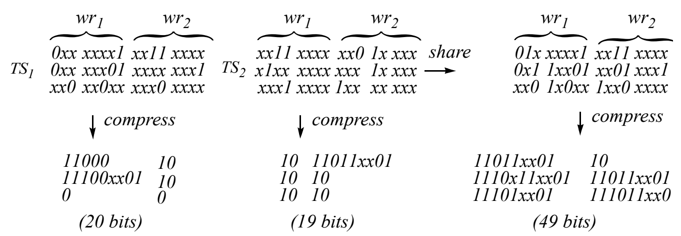


Figure 6. Sharing and compression of tests.

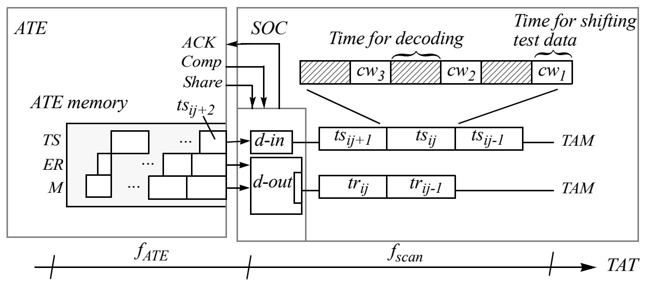


Figure 7. Test application using compression.

system consisting of 2 cores (*n=2*) has the following *k*-subsets: {*1*}, {*2*}, and {*1, 2*}, i.e., three different alternative tests. Since each such alternative test can be either compressed or not compressed the sum is multiplied by two. The value of *H* is then given as:

$$H = \left(\sum_{k=1}^{n} \binom{n}{k} \right) \times 2 \qquad (7)$$

In order to avoid excessively large optimization times due to the large number of possible test alternatives *H* we limit the number of possible alternative tests for the system by restricting the number of possible permutations for the sharing. In this work we restrict the generation of new tests by only considering possible 2-subset combinations during the sharing of tests. In addition, we do not consider those permutations that have minor effect on the ATE memory requirement. We only share those tests that have similar size in term of number of sequences and scanned elements. We, therefore, define a maximum share ratio MSR as follows:

$$MSR = 100 - \left(\frac{max(size(T_i), size(T_j))}{size(T_i) + size(T_j)} \right) \times 100, \qquad (8)$$

where the *size*(T_i) is given as the number of bits in the TS and ER, which is considered during the sharing. This ratio is further explained by the example presented in Figure 8, where *MSR* is calculated for two different combinations of tests. Merging test T_i ($size(T_i) = 100$) with T_j ($size(T_j)=20$) will lead to a maximum decrease of only 17% of the memory, while merging T_i with T_k ($size(T_k) = 90$) potentially reduces the size with 47%. By setting a limit on the *MSR* during the pre-process stage it is possible to avoid those alternatives that have little possibility to be part of the optimal solution and therefore will not be explored during the optimization process.

The test selection, scheduling and architecture design problem have been formulated as a CLP problem [8]. The CLP-tool CHIP [17] has been used for the implementation and the built in predicates *labeling* and *min_max* are used for the enumeration and search for the optimal solution. A short description of the program is depicted in Figure 9. The variables such as the TAT τ_{tot} and used memory M_{tot} are defined (line 1 to 5) and two new predicates, *sum_test_time* and *sum_test_mem* (line 6 and 7), have been implemented that calculate the TAT and the required memory for a specific solution. On line 8 the constraint expressing that the memory used is less than the size of the ATE memory is defined, and finally the optimization is done by using *labeling* for the enumeration inside the *min_max* predicate (line 9).

6. Experimental results

In this section the significance of integrating both test sharing and compression in one framework is demonstrated by experiments. The TAT is minimized under ATE memory constraints using the following four techniques: no compression and no sharing (NC, NS), only compression (C), only sharing (S) and both sharing and compression (C, S).

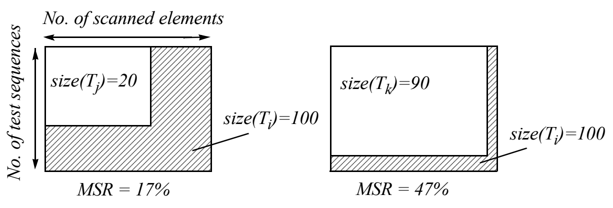


Figure 8. Maximum share ratio for different tests.

```
    // Define variables
1.  $\tau_{tot}$::0..100000,
2.  $M_{tot}$::0..100000,
3.  get_max_mem($M_{ATE}$),
4.  get_input_tests(InputTests),
5.  get_input_cores(InputCores),
    // Define constraint
6.  sum_test_time(InputTests, InputCores, $\tau_{tot}$),
7.  sum_test_mem(InputTests,InputCores,$M_{tot}$),
8.  $M_{tot}$ #<=$M_{ATE}$,
    // Search for the optimal solution
9.  min_max((labeling(InputCores)), $\tau_{tot}$).
```

Figure 9. CLP formulation in CHIP for test application time minimization.

For the experiments we have used the following three ITC'02 benchmark designs: *d695*, *g1023*, and *p34395* [18]. The input characteristics for the designs are collected in Table 2 where Column one contains the name of the design and Column two the number of tests given as input. Column three lists the amount of memory required to store the original TS and ER for the given tests. The last column, Column four, contains the number of TAM wires, which is specified by us.

For the *d695* design the TS and ER (with don't-care bits marked) are given [1]. We have randomly generated TS and ER for designs *g1023* and *p34395* such that the amount of don't-care bits is 95% [19]. We assume, in these experiments that the designs are tested using an ATE with a frequency f_{ATE} of 100MHz and after running extensive experiments, the *MSR* threshold is set to 35%. The memory constraint M_{ATE} has been determined by multiplying the amount of memory required for each design (given in Table 2 Column three) with a constant, *MConst.* In total, three experiments has been performed each with different ATE memory constraint, $MConst = 1$, $MConst = 2/3$, and $MConst = 1/3$.

The experimental results are collected in Table 3. Column one lists the different designs and Column two the different techniques for each design respectively. Column three lists the total number of test alternatives considered during the optimization. The following columns, column four to twelve lists the memory constraint, the TAT, and the optimization time (CPU time) for each of the three experiments.

The results obtained using our integrated approach, using both compression and sharing (denoted C, S in Table 3), are compared to the following three techniques. First, using no compression and no sharing (NC, NS), i.e., only the dedicated, initially given, tests are used to test the system. Second, using only compression (C) and third, using only sharing (S).

Without using either sharing nor compression, results are only obtained when the ATE memory is large enough as in Experiment 1 ($MConst = 1$). When reducing the ATE memory size as in Experiment 2 ($MConst = 2/3$), sharing only is sufficient to decrease the amount of memory used for the design

Table 2. Input characteristics

Design	No. of input tests	Memory requirement (kbit)	No. of TAM wires
d695	10	3398	48
g1023	14	4789	60
p34392	19	125332	60

d695 and *p34392*, for the design *g1023* the compression technique must be applied to obtain a solution. The compression technique is required for all three designs in Experiment 3 ($MConst = 1/3$) since only sharing does not decrease the required memory sufficiently.

When using compression the test size is reduced and less ATE memory is used but the TAT is increased due to the slower scan frequency, (Equation (4)). For all three experiments, the results show a decrease in the TAT when sharing is used. Experiment 3 show that our proposed technique, using both sharing and compression, is able to considerably reduce the TAT when using a small ATE memory. When using a large ATE memory such that sharing only (S) is able to obtain a solution our method is not able to further decrease the TAT, however, our proposed integrated technique always produces solutions that are equal or better compared to when sharing or compression is used separately.

The results also show the trade-off between the TAT obtained using the proposed approach, and the amount of optimization time needed. In general, the optimization time is increased using our approach since the complexity (the number of test alternatives) is increased.

7. Conclusions

The high test data volume needed for testing modern chips leads to long test application times and high ATE memory requirements. In this paper we propose a technique where we integrated test data compression, test sharing, test architecture design and test scheduling with the objective to minimize the test application time under ATE memory constraint. We assume a core-based system with tests per module and we define techniques for test data compression and test sharing to find the best test alternatives for the testing of each core. We make use of a Constraint Logic Programming for selection of test for each module, scheduling and architecture design for the selected tests. The efficiency of our approach has been demonstrated with experiments on several ITC'02 designs.

References

[1] S. Kajihara and K. Miyase, "On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits," *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 364–369, 2001.

[2] A. Chandra and K. Chakrabarty, "A unified approach to reduce SOC test data volume, scan power and testing time," *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 22, Issue 3, pp. 352–363, 2003.

[3] A. Jas, J. Ghosh-Dastidar, M. Ng, and N. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding," *IEEE Transaction on Computer-Aided Design (TCAD)*, Vol. 22, pp. 797–806, 2003.

[4] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Colomb Codes," *IEEE Transaction on CAD of Integrated Circuits and Systems*, Vol. 20, Issue 3, pp. 355–368, 2001.

[5] T. Shinogi, Y. Yamada, T. Hayashi, T. Yoshikawa, and S. Tsuruoka, "Test Vector Overlapping for Test Cost Reduction in Parallel Testing of Cores with Multiple Scan Chains," *Digest of Papers of Workshop on RTL and High Level Testing (WRTLT)*, pp. 117–122, 2004.

[6] K-J. Lee, J-J. Chen, C-H. Huang, "Broadcasting Test Patterns to Multiple Circuits*," IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 18, Issue. 12, pp. 1793–1802, 1999.

[7] G. Zeng and H. Ito, "Concurrent Core Test for SOC Using Shared Test Set and Scan Chain Disable," *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 1045–1050, 2006.

[8] J. Jaffar and J.-L. Lassez, "Constraint Logic Programming," *Proceedings of the 14th. ACM Symposium on Principles of Programming Languages (POPL),* pp. 111–119, 1987.

[9] A. Larsson, E. Larsson, P. Eles, and Z. Peng , "SOC Test Scheduling with Test Set Sharing and Broadcasting," *Proceedings of IEEE Asian Test Symposium*, pp. 162–169, 2005.

[10] S. Mitra, M. Mitzenmacher, S. S. Lumetta, and N. Patil, "X-Tolerant Test Response Compaction," *IEEE Design & Test of Computers*, Vol. 22, Issue 6, pp. 566–574, 2005.

[11] E. Larsson and J. Persson, "An Architecture for Combined Test Data Compression and Abort-on-Fail Test," *Asia and South Pacific Design Conference (ASP-DAC)*, Accepted for publication, 2007.

[12] F. Poehl, J. Rzeha, M. Beck, M. Goessel. R. "Arnold, and P. Ossimitz, "On-Chip Evaluation, Compensation, and Storage of Scan Diagnosis Data - A Test Time Efficient Scan Diagnosis Architecture," *Proceedings of IEEE European Test Symposium (ETS)*, pp. 239–244, 2006.

[13] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test," *Proceedings of International Test Conference,* pp. 911-920, 2000.

[14] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," *Proceedings of International Test Conference (ITC)*, pp. 1023–1032, 2001.

[15] U. Ingelsson, S. K. Goel, E. Larsson, and E. J. Marinissen, "Test Scheduling for Modular SOCs in an Abort-on-Fail Environment," *Proceedings of IEEE European Test Symposium (ETS),* pp. 8–13, 2005.

[16] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Technique for Tesing Embedded Cores in SoCs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* ,Vol. 13, Issue 6, pp. 719–731, 2005.

[17] CHIP, System Documentation, COSYTEC, 1996.

[18] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs," *Proceedings of the IEEE International Test Conference (ITC)*, pp. 519–528, 2002.

[19] E. Larsson, A. Larsson, and Z. Peng, "Linköping University SOC Test Site," *http://www.ida.liu.se/labs/eslab/soctest*, 2006.

Table 3. Experimental results

Design	Technique (No compression (NC), No sharing (NS), Compression (C), Sharing (S))	No. of test alternative	Experiment 1 (MConst = 1)			Experiment 2 (MConst = 2/3)			Experiment 3 (MConst = 1/3)		
			Memory constraint (kbit)	TAT (ms)	CPU time (s)	Memory constraint (kbit)	TAT (ms)	CPU time (s)	Memory constraint (kbit)	TAT (ms)	CPU time (s)
d695	NC, NS	10	3398	0.49	0.2	2265	n.s.	n.s.	1132	n.s.	n.s.
	C	20		0.49	0.2		0.82	0.8		1.22	0.3
	S	20		0.36	1.0		0.36	0.9		n.s.	n.s.
	C, S	40		0.36	1.0		0.36	0.9		0.44	1.0
g1023	NC, NS	14	4789	0.56	0.2	3193	n.s.	n.s.	1596	n.s.	n.s.
	C	28		0.56	0.3		0.90	1.6		1.28	0.5
	S	35		0.47	15.9		n.s.	n.s.		n.s.	n.s.
	C, S	70		0.47	51.9		0.55	29.8		0.94	40.4
p34392	NC, NS	19	125332	14.72	0.3	83551	n.s.	n.s.	41784	n.s.	n.s.
	C	38		14.72	216.2		23.68	204.4		33.30	46.9
	S	37		10.93	313.3		10.93	5.8		n.s.	n.s.
	C, S	74		10.93	2255.6		10.93	1902.8		16.43	437.7