SOC Test Scheduling with Test Set Sharing and Broadcasting

Anders Larsson, Erik Larsson, Petru Eles and Zebo Peng

Embedded Systems Laboratory
Linköpings Universitet
SE-582 83 Linköping, Sweden
{anlar, erila, petel, zpe}@ida.liu.se

Abstract1[1]

Due to the increasing test data volume needed to test core-based System-on-Chip, several test scheduling techniques minimizing the test application time have been proposed. In contrast to approaches where a fixed test set for each core is assumed, we explore the possibility to use overlapping test patterns from the tests in the system. The overlapping tests serves as alternatives to the original dedicated test for the cores and, if selected, they are transported to the cores in a broadcasted manner so that several cores are tested concurrently. We have made use of a Constraint Logic Programming technique to select suitable tests for each core in the system and schedule the selected tests such that the test application time is minimized while designer-specified hardware constraints are satisfied. The experimental results indicate that we can on average reduce the test application time with 23%.

1. Introduction

The increasing test application time for SOC (system-on-chip) designs is due to the high amount of test data required for testing. The test application time can be decreased by an efficient organization of the test data transportation and concurrent test application.

It has been shown that the percentage of unspecified bits, so called don't cares, in test sets is high even for compacted tests [7], [8]. This high percentage of don't cares can be used to find overlapping test vectors from different tests, which can be merged [9] and broadcasted to multiple cores [6].

Several approaches assuming a dedicated test access mechanism (TAM) for test data transportation have been proposed [1, 3, 13]. Aerts and Marinissen proposed three TAM structures for scan tested systems [1]. Iyengar *et al.* defined a framework for TAM design and test scheduling on dedicated TAMs [3]. In [13] an addressable system bus for SOC testing is proposed.

The major contributions of this paper are twofold. First, we demonstrate the use of test sharing and broadcasting of test vectors for core-based SOCs, which means that the test set is not longer fixed for one core. One advantage with the proposed test architecture is that it offers a possibility to reuse on-chip functional connections, such as the system bus, for test transportation. Second, we solve the scheduling and test bus design problem while minimizing the overall test time. We have formulated the problem, solved it, and demonstrated its usefulness by experiments using Constraint Logic Programming (CLP) [5].

The rest of the paper is organized as follows. Section contains a description of the test set sharing. The SOC test architecture is described in Section 3, and is followed by a motivational example in Section 4. The problem is formulated in Section 5, and the CLP modelling is presented in Section 6. The experimental results are presented in Section 7, and conclusions are in Section 8.

2. Test set sharing

In this section, we describe how test set sharing and broadcasting of tests reduces the test application time, and how two different tests can be merged by using a pattern matching algorithm.

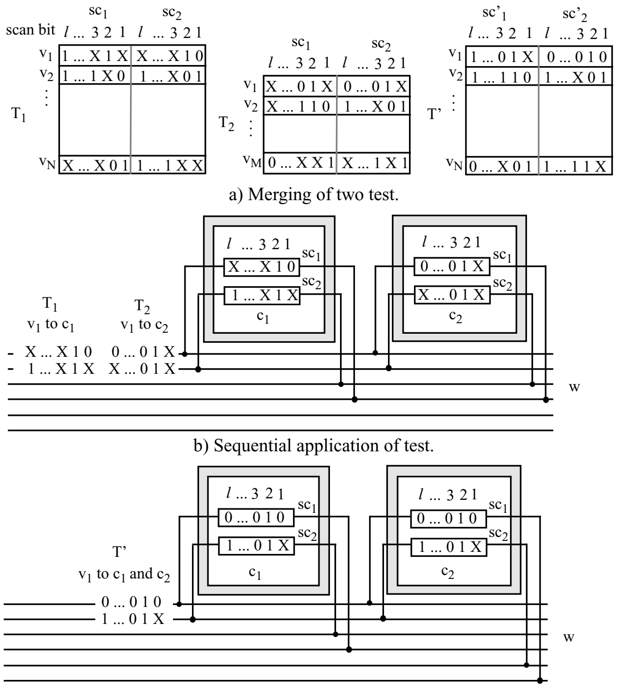
By sharing a test set among several cores it is possible to shorten the test application time significantly [6], since cores that share tests are tested concurrently. The efficiency of this test set sharing method depends on how large the merged test set is in comparison with the unmerged sets. It has been shown that the percentage of don't cares in the test sets is high, 78% for un-compacted or approximately 47% even for compacted tests [7], [8]. This high number of don't care bits provides the possibility of finding vectors from different sets that can be efficiently merged.

The possibility of merging two tests is dependent on the density of don't cares present in the tests. If the tests are compressed so that only a small percent of don't cares remains, the merging capability will significantly decrease. How the compression ratio is affected by the density of don't cares has been studied in [14]. To cope with a situation where the compression reduces the number of don't cares so that it is not possible to merge tests, the test designer can set a limit on the amount of compression of a test so that it is still possible to use it for merging.

One important requirement that has to be fulfilled, in order for the test set sharing to be efficient and to reduce the test time, is that the size of the new merged test set is smaller than the sum of the original test sets used [10].

We have performed experiments to investigate the relationship between the number of don't care bits and the size of the merged test set. For this purpose we use a straightforward pattern matching algorithm that takes two test sets, T_1 and T_2, as input and generates a new test set T'. This is illustrated in Figure 1 where two cores, c_1 and c_2, are

[1] The research is partially supported by the Swedish National Program STRINGENT.


sc1
sc2
scan bit l ... 3 2 1 l ... 3 2 1
v1 1 ... X 1 X X ... X 1 0
v2 1 ... 1 X 0 1 ... X 0 1
T1
vN X ... X 0 1 1 ... 1 X X
v1 X ... 0 1 X 0 ... 0 1 X
v2 X ... 1 1 0 1 ... X 0 1
T2
vM 0 ... X X 1 X ... 1 X 1
sc'1
sc'2
v1 1 ... 0 1 X 0 ... 0 1 0
v2 1 ... 1 1 0 1 ... X 0 1
T'
vN 0 ... X 0 1 1 ... 1 1 X
T1
v1 to c1
T2
v1 to c2
X ... X 1 0 0 ... 0 1 X
1 ... X 1 X X ... 0 1 X
c1
c2
w
T'
v1 to c1 and c2
0 ... 0 1 0
1 ... 0 1 X

a) Merging of two test.

b) Sequential application of test.

c) Application of broadcasted test.

Figure 1. Merging and application of test.

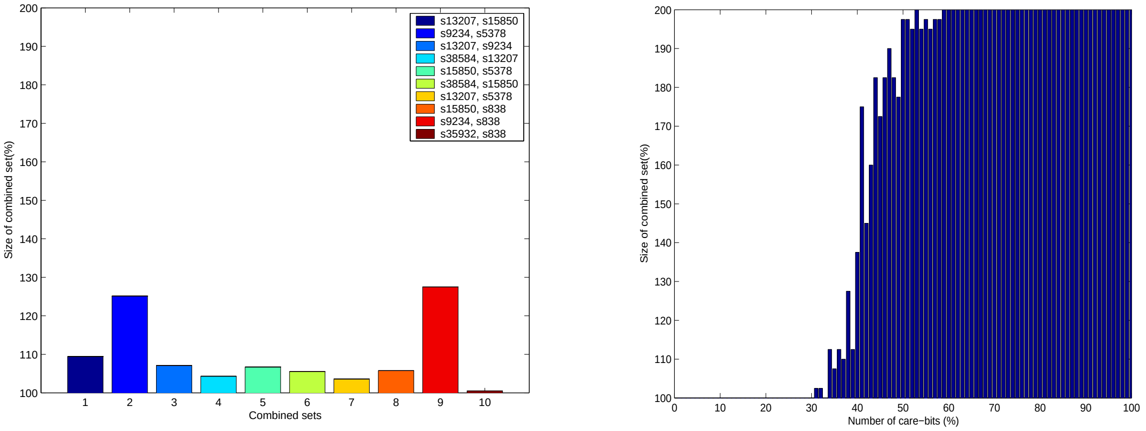
tested by two tests, T_1 and T_2, respectively. Test T_1 consists of N test vectors, $\{v_1, v_2,..., v_N\}$, and test T_2 of M test vectors, $\{v_1, v_2,..., v_M\}$. In this example, both cores have two scan chains, sc_1 and sc_2, with equal length, l, which are connected to a bus with the bandwidth, w.

Merging one test vector v_i from Test T_1 with a test vector v_j from T_2, where $i \in (1, N)$ and $j \in (1, M)$, is done by comparing each position in the two test vectors. As long as both have the same value or one is marked as a don´t care (x), the vectors can be merged as illustrated in Figure 1(a). If it is not possible to merge v_i with v_j the next test vector, v_{j+1}, is tried until all test vectors have been investigated, i.e. when $j=M$. The test vectors, which are not possible to merge are kept intact and the size of the new test set T' is increased. If the cores have scan chains with different length, the one with shorter length will be filled with don't cares. Figure 1(b) shows how the cores are connected to a test bus. In this example the tests T_1 and T_2 are transported and applied sequentially. How the merged test, T' is transported and applied is illustrated in Figure 1(c) and since it consists of vectors from test T_1 and T_2, both core c_1 and c_2 will be tested concurrently.

This pattern matching algorithm has been applied to the benchmark design d695 [4]. The test vectors (with don't cares marked) have been extracted by Kajihara and Miyase in [7]. The results from this experiment are presented in Figure 2(a), which shows 10 different combinations of the tests and the sizes of the merged tests. The sizes of the merged tests are compared with the size of the largest test used and the result shows an increase in size with on average only 10.94%. To illustrate the relationship between the number of don´t cares and the size of the merged test set, a number of randomly generated tests where created and merged. The results depicted in Figure 2 (b) show that when the number of care bits is in the range of 0 to 45 % the size of the merged set is still reasonable small, within an increase of 50%.

In order to apply the to merged tests, it is required that the cores, which shares the tests, are connected in such a way that the tests can be broadcasted to the cores. How this broadcast of test patterns can be done has been shown in [6] and [11]. In [11], a method for generating common tests for multiple cores directly is proposed. A fault simulator is used for evaluating the fault coverage. The tests generated by the method in [11] can also be utilized by the technique described in this paper.

One additional problem that occurs when using a broadcast method is how to transport the test responses. The responses from different cores cannot be merged in the same way as the input stimuli since the opportunity to detect a fault may be lost. In [6] and [11] this problem has been solved by compressing the test response by using a multiple input signature register (MISR). This leads to increased hardware overhead. In this work the test responses from each core will be transported on separate wires (described in the following section).



a) Different combinations from design d695

b) Random sets with increasing number of don't cares

Figure 2. Merging of test sets.

3. SOC test architecture

In this section we discuss the architecture that is used for the test data transportation. First we introduce the general architecture consisting of cores (e.g. c_1, c_2, c_3, and c_4 in Figure 3) which are connected to buses (bf_1 and bt_1 in Figure 3.) The bus wires are connected to an ATE, which is used to apply test vectors and analyze the responses of the tests. A connector consisting of logic needed for the communication and application of test data is introduced between each core and the bus. For example, $o_{4,1}$ is the connector connecting core c_4 with bus bf_1, as shown in Figure 3. In this work it is assumed that the buses are connected to the input and output pins of the chip, and hence, directly accessible and controlled from the ATE.

The functional bus, bf_1 in Figure 3, is used to transport test data from the ATE to the cores. However, if the bandwidth of the functional bus is not enough for transporting the test data in reasonable time, one or several dedicated test buses may be added to the design. A dedicated test bus for the transportation of test data will increase the transportation capacity and shorten the test time. It also offers the possibility of a trade-off between the test time and the number of wires used. A high number of wires require however a large silicon area to implement.

The transportation and application of tests to the cores are illustrated in Figure 4 by considering cores c_2 and c_3 from Figure 3. The cores are scan tested and to facilitate interfacing with the bus, each core has a wrapper. The scanned elements at a core (scan chains and wrapper cells) are connected into wrapper chains, which are connected to bus wires. It is assumed that both cores have three wrapper chains each, which are connected to the bus as illustrated in Figure 4(a). The test stimuli are transported from the ATE on the bus to the core through the input test pins, t_{in}. When the test stimuli have been applied the test responses are transported back to the ATE through the outputs, t_{out}. When the system is in functional mode, the functional inputs and outputs at each core are connected to the functional bus. When the system is in testing mode the connectors will receive control signals, t_{ctrl}, indicating when a pattern should be applied.

Short application time of scan patterns entails a concurrent scan-in and scan-out phase, that is, when one scan pattern is shifted out, a new pattern is shifted in. Therefore it is not possible to share the same bus wires for the test stimuli and test responses of one core; the total number of bus wires used to test one core is twice the number of scan chains of the core as illustrated in Figure 4(a).

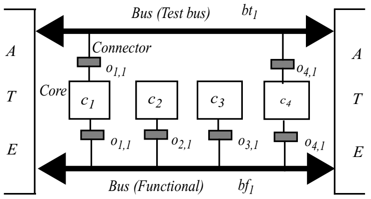
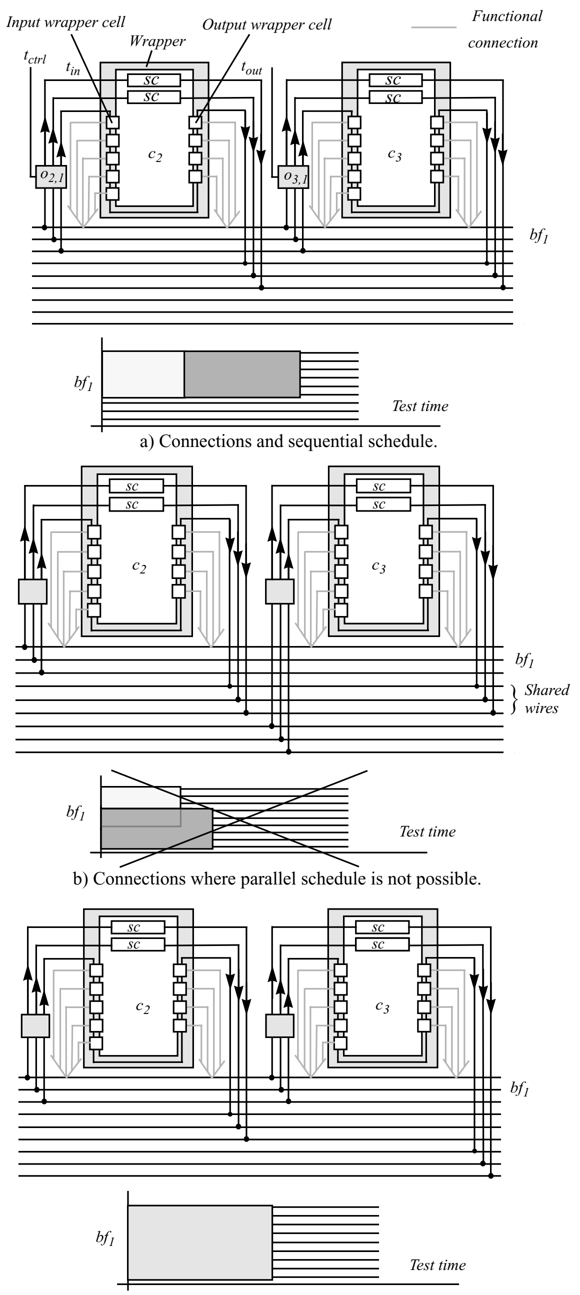


Figure 3. Bus-based architecture.

The application of tests in Figure 4(a) can only be done sequentially, as long as the cores are connected to the same bus wires. To give exclusive access to the bus for both cores at the same time, and hence make it possible to apply the test to the cores concurrently, would significantly decrease the test time. However, in the example in Figure 4(b), the bandwidth of the bus is not enough and the two cores are forced to use the same wires for several connections. If the tests are applied in parallel under this situation they will overlap and hence give an impossible schedule as illustrated in Figure 4(b). If we consider that the two cores, c_2 and c_3,



a) Connections and sequential schedule.

b) Connections where parallel schedule is not possible.

c) Connections and broadcasted test.

Figure 4. Connection between core and bus with different schedules.

use a shared test set, as described in Section 2, which is broadcasted. In this case the input wires will be shared by the cores and only the output responses require dedicated wires for each core as shown in Figure 4(c).

This example illustrates that by using shared test sets, which are broadcasted, it is possible to get a shorter test time compared to a sequential application and still use a smaller amount of wires compared with a parallel application.

4. Motivational example

Let us consider an example design consisting of four cores, c_1, c_2, c_3, and c_4, connected to one functional bus bf_1 and added test connectors ($o_{1,1}$, $o_{1,2}$, $o_{1,3}$, $o_{1,4}$,) between the bus and the cores as shown in Figure 5. Each core has one test, c_1 is tested by T_1, c_2 is tested by T_2, c_3 is tested by T_3, and c_4 is tested by T_4. The dedicated test set to these cores has been extended with one additional test T'_5, which is a combination of the tests T_3 and T_4 for core c_3 and c_4. If T'_5 is selected, it is broadcasted to c_3 and c_4.

For the sake of readability it is assumed that each test will occupy the whole bandwidth of the bus, which means that a single bus configuration will lead to a sequential application of the tests. In the first schedule shown in Figure 5(a) the shared test (T'_5) is not used, while in the second schedule, Figure 5(b), T'_5 is introduced and since it can be applied to the two cores c_3 and c_4 concurrently, the test time is decreased.

The test time may be further decreased if a dedicated test bus, bt_1 is introduced as illustrated in Figure 5(c and d). It will enable concurrent application of tests. Figure 5(c) shows an example of a mapping of cores to buses. In this example only one core is tested through the test bus but the test time has decreased compared with the example where only one bus was used. Since core c_4 is tested through the test bus it is not possible to make use of the broadcast capability between c_3 and c_4. However, by connecting c_3 and c_4 to the same bus as shown in Figure 5(d) the test time can be further reduced.

5. Problem formulation

For the modelling we assume the following. Given is a system consisting of:

- a set of N cores $C = \{c_1, c_2, ..., c_N\}$,
- a set of M functional buses $'_F = \{bf_1, bf_2, ..., bf_M\}$, where each bus bf_i has the bandwidth, w^{bf_i},

and the following constraints:

- w_{ATE} the bandwidth of the ATE and
- K_{max} the designer-specified hardware-overhead.

Given for each core c_i is:

- u^i the number of wrapper-chains,
- l^i the length of the longest wrapper chain,
- a set of test sets $\Psi_i = \{\Gamma^i_1, \Gamma^i_2, ..., \Gamma^i_X\}$, where each test set $\Gamma^i_j = \{T_1, T_2, ..., T_H\}$.

Given for a test T_k is the number of patterns s^{T_k}.

Consider the following example where core c_1 and c_2 have two associated test sets each, $\Psi_1 = \{\{T_1, T_2\}, \{T_3\}\}$, and $\Psi_2 = \{\{T_4, T_5\}, \{T_3, T_5\}\}$. Core c_1 is fully tested by either applying both T_1 and T_2 or only T_3. T_3 is used to test both c_1 and c_2, which means that it is shared and should be broadcasted.

The test time, τ_k for applying a test, T_k at core c_i is:

$$\tau_k = (1 + l^i) \times s^{T_k} + l^i \qquad (1)$$

The number of wires w_k that a test makes use of when transported on a bus depends on the number of cores z that shares the test. If no sharing is used (z=1), w^i wires are used to transport test stimuli to the core, and w^i wires are used to transport test response from the core. In total $2 \times w^i$ wires are needed when z=1. In the case of broadcasting, w^i wires are used to transport test stimuli to the cores, but each core requires w^i wires to transport test response from the core. In total $w^i + w^i \times z$. This is given by the following formula:

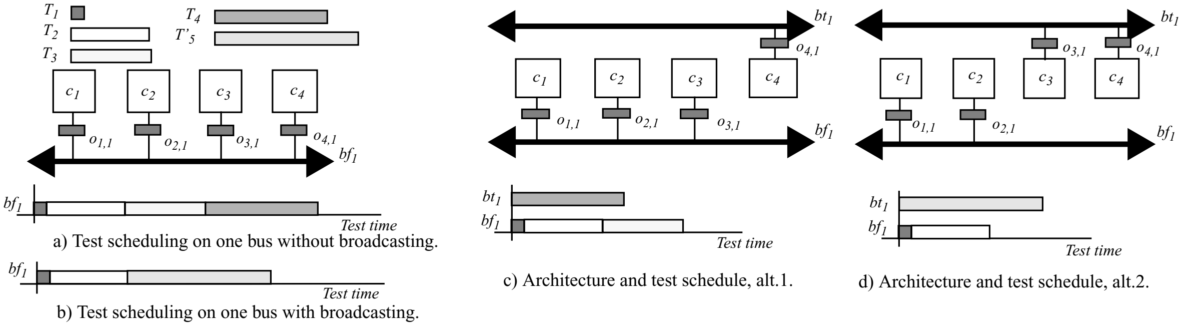
$$w_k = \begin{cases} 2 \times w^i, \text{when z} = 1 \\ w^i + w^i \times z, \text{when z } \geq 2 \end{cases} \qquad (2)$$

In order to apply tests, the tests are transported from the ATE to the cores. For this purpose, the existing functional bus structure can be used or added test bus structure. If a test shall make use of the functional bus, a connector must be inserted between the bus and the core. If a dedicated test bus is used, the bus must have been inserted, and a connector is added between the test bus and the core. We introduce test bus connectors:

- a set of G test buses $B_T = \{bt_1, bt_2, ..., bt_G\}$where bus bt_i has the bandwidth w^{bt_i},
- test connectors $o_{i,j}$ between core c_i and bus bf_j (or bt_j).

The following hardware cost factors are considered:

- $kf_{i,j}$ is the cost of inserting a connector $o_{i,j}$ between core c_i and functional bus bf_j,
- $kt_{i,j}$ is the cost of inserting a connector $o_{i,j}$ between core



a) Test scheduling on one bus without broadcasting.

b) Test scheduling on one bus with broadcasting.

c) Architecture and test schedule, alt.1.

d) Architecture and test schedule, alt.2.

Figure 5. Test scheduling for different bus architecture.

c_i and test bus bt_j,

- k^{bt_i} is the base cost of inserting test bus bt_i.

The total hardware cost HW_{Tot} is given by:

$$HW_{Tot} = \sum_{i=1}^{N} \sum_{j=1}^{M} kf_{i,j} + \sum_{i=1}^{N} \sum_{j=1}^{G} kt_{i,j} + \sum_{j=1}^{G} k^{bt_j} \quad (3)$$

The optimization objective is to:

- select tests for each core,
- insert test buses (if required),
- insert connectors between cores and buses,
- and schedule the selected tests on the buses

in such a way that each core is fully tested and the test application time is minimized without:

- inserting test buses such that the ATE bandwidth w_{ATE} is violated, that is;

$$\sum_{i=1}^{M} w^{bf_i} + \sum_{j=1}^{G} w^{bt_j} \leq w_{ATE} \quad (4)$$

  where M is the number of functional buses and G the number of added test buses, and
- exceeding the given total hardware cost constraint.

$$HW_{Tot} \leq K_{max} \quad (5)$$

6. CLP modelling

Constraint Logic Programming (CLP) [5] is a combination of logic programming and constraint solving and is suitable for solving problems like scheduling, resource planning, and layout assignment.

We have formulated our test scheduling problem as a CLP problem (Figure 6). The cores, their connections and information about the tests regarding the size and number of wrapper chains, are first given as input (line 2 and 3 in Figure 6). A number of variables used to describe a solution is then defined, (4..10). In order to find a feasible solution that minimizes the total test time (16) the program ensures that the following constraints are fulfilled (11..15):

- Each core must be connected to at least one bus (11).
- Each core must be fully tested (12).
- The hardware cost does not exceed the given maximum hardware cost (14), Eq.(5).
- The total number of wires does not exceed the given maximum limit Eq.(4) and tests do not make use of the same wires concurrently (15).

```
(1) run:-
(2)     Cores({1,2,3,... ,NrCores}),  % Get input data
(3)     Tests({1,2,3,... ,NrTests}),
(4)     NrBuses::1..MaxNrBuses,  % Define variables
(5)     Cost::1..MaxCost,
(6)     TestTime::1..MaxTestTime,
(7)     ListOfTests::0..NrTests,
(8)     ListOfCores::0..NrCores,
(9)     Schedule::0..NrTests*NrBuses,
(10)    Tam::1..MaxTam,
(11)    connect_all(Cores),  % Set up constraints
(12)    complete_cores(Cores,Tests),
(13)    count_costs(Cores,Costs,Cost),
(14)    Cost #< MaxCost,
(15)    cumulative (Schedule, Duration, Resource, Tam, TestTime),
(16)    min_max((labeling(Schedule)),TestTime).  % Find optimal solution
```

Figure 6. CLP formulation in CHIP for test time minimization.

We have used the following built in predicates in the CLP tool CHIP [2, 12] to ensure that all constraints are satisfied and the optimal solution is found:

- Cumulative (15), ensures that, at any given time, the total amount of resources does not exceed a given limit.
- Min_max (16), implements a depth first branch and bound search for a solution with the minimal cost.
- Labeling (16), is used to assign values to variables.

Since a test T_i can be listed for several cores, a special constraint is implemented so that T_i is not scheduled more than one time as long as the cores shares the same bus.

7. Experimental results

In our experiments we have used the following eight designs; SOC_(1..7), which are randomly generated, and the benchmark design d695 [4]. The main characteristics of the eight designs can be found in Table 1. This table contains information about the number of cores, tests, and the minimum required hardware constraint needed. The minimum hardware constraint is the hardware cost needed in order to connect each core to a functional bus. If this constraint is not satisfied the core will not be fully accessed or tested.

The hardware cost, such as wiring and control logic needed to connect a core to a bus or to add a test bus, is assumed to be given by the designer. In the experiments the cost of connecting a core to a functional bus is set to 10 units, the cost to connect a core to a test bus to 20 units, and the cost of adding a test bus to the system is set to 100 units. This means that adding one test bus and connect one core to it will be associated with a hardware cost of 120 units. In these designs it is assumed that each system has a 64 bit wide functional bus and that each test bus, if added to the system, has a width of 32 bits.

We have used the CLP tool CHIP (V 5.2.1) [12] for the implementation and we have compared when broadcasting is not used and when broadcasting is used.

The results are collected in Table 3. Column 1 lists the eight different designs. In Column 2 the hardware constraints are listed. These constraints have been set so that it is possible to add at least one test bus for each design. The following three columns, three to five, contain the results from the first approach where no broadcasting is used; the number of test patterns used, the minimized test time, and the optimization time. The following columns, fiveo eight, contain the results

Table 1. Design characteristics

Design	No. of Cores	No. of Tests	Min. HW constraint
SOC_1	4	5	40
SOC_2	7	9	70
SOC_3	10	12	100
SOC_4	12	15	120
SOC_5	18	20	180
SOC_6	24	28	240
SOC_7	30	34	300
d695	10	12	100

when broadcasting is used. The last column shows the comparison in test time between the two approaches. The experiments show that broadcasting of tests between cores can shorten the test time. The test time could be decreased with 23.72% on average.

Experiments have also been made to show the impact on the test time at different hardware constraints. The test time minimization has been made with different values of the hardware constraint for two examples, SOC_1 and the benchmark design d695. The results collected in Table 2 show how the test time for the different designs decreases as additional test buses are added. For SOC_1 a saturation point is met when all cores can be tested concurrently (HW constraint = 400).

8. Conclusions

Decreasing the test application time for SOCs entails an efficient test data transportation and concurrent test application. We propose a scheme to explore the high amount of don't cares present in the test sets in order to merge different tests, which can be used as alternative to the original dedicated test for the cores. The proposed method allows the existing functional bus structure to be reused for the test data transportation. However, in order to decrease the test time, dedicated test buses may be added to the design. The problem formulated, is to select appropriate tests for each core, insert test buses (if required), and schedule the selected tests on the buses in such way that the test application time is minimized without exceeding the given hardware cost constraints.

Table 2. Test time for different hardware constraints

Design	HW constraint	Nr. added test wires	Test time
SOC_1	40	0	7514
	150	32	6421
	200	32	6221
	300	64	6155
	400	96	4329
	500	96	4329
d695	100	0	26071
	250	32	22718
	300	64	20382
	400	64	18522
	500	96	13712
	600	128	12633
	700	160	11791

We have modelled the problem and implemented it using CLP and experiments show that the overall test time can be significantly reduced when broadcasting of tests is used. Since CLP uses an exhaustive search approach, execution times can become large for complex examples. A natural extension of the work is therefore to find a heuristic that would work for larger designs.

References

[1] J. Aerts and E. J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-Based ICs," *Proceedings of the International Test Conference (ITC)*, pp. 448-457, 1998.

[2] P. Van Hentenryck, "The CLP language CHIP: constraint solving and applications," *Compcon Spring '91. Digest of Papers* , 25 Feb-1 Mar 1991 , pp. 382 -387, 1991.

[3] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Test Data Volume Reduction for System-on-Chip", *Transactions on Computer*, Vol. 52, No. 12, Dec. 2003.

[4] K. Chakrabarty, "Optimal Test Access Architectures for System-on-a-Chip," *ACM Transactions on Design automation of Electronic Systems*, vol. 6, pp. 26-49, 2001.

[5] 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.

[6] K-J. Lee, J-J. Chen, C-H. Huang, "Broadcasting Test Patterns to Multiple Circuits*," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.18, No.12, pp.1793-1802, 1999.

[7] 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.

[8] A. Chandra, and K. Chakrabarty, "A unified approach to reduce SOC test data volume, scan power and testing time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22 , Issue 3, pp352-363, 2003.

[9] 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.

[10] K-J. Lee, J-J. Chen, C-H. Huang, "Using a Single Input to Support Multiple Scan Chains," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 74 - 78, 1998.

[11] J.H. Jiang, W-B. Jone, S-C. Chang, and S. Ghosh, "Embedded core test generation using broadcast test architecture and netlist scrambling," IEEE Transactions on Reliability, Vol. 52, Issue 4, pp.435 - 443, 2003.

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

[13] S. Hwang, J.A. Abraham, "Reuse of addressable system bus for SOC testing," *IEEE International ASIC/SOC Conference*, pp. 215 - 219, 2001.

[14] A. B. Kinsman, and N. Nicolici, "Time-Multiplexed Test Data Decompression Architecture for Core-Based SOCs with Improved Utilization of Tester Channels," *Proceedings of the 10th IEEE European Test Symposium (ETS)*, pp. 196 - 201, 2005.

Table 3. Experimental results

Design	*Hardware Constraint*	*Nr. Added Test buses*	Without Broadcasting			With Broadcasting			Test time Comparison
			Tot. Nr. Patterns Used	Test time	CPU time (s)	Tot. Nr. Patterns Used	Test time	CPU time (s)	
SOC_1	250	1	275	8271	14	209	6755	76	-18.33%
SOC_2	350	1	440	22361	76	297	18421	132	-17.62%
SOC_3	400	2	539	37943	391	423	29364	572	-22.61%
SOC_4	450	2	753	51946	624	687	37526	1517	-27.76%
SOC_5	500	2	2316	86301	1028	1723	61730	39843	-28.47%
SOC_6	500	3	15017	1167250	2734	11483	869195	62087	-25.53%
SOC_7	600	3	18779	1602862	4893	14021	1187512	95274	-25.91%
d695	350	2	881	24219	235	802	18522	586	-23.52%