# A Technique for Optimisation of SOC Test Data Transportation

Anders Larsson, Erik Larsson, Petru Eles and Zebo Peng

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

## Abstract

*We propose a Tabu-search-based technique for time-constrained SOC (System-on-Chip) test data transportation. The technique makes use of the existing bus structure, where the advantage is, compared to adding dedicated test buses, that no additional routing is needed. In order to speed up the testing and to fulfill the time constraint, we introduce a buffer at each core, which in combination with dividing tests into smaller packages allows concurrent application of tests on a sequential bus. Our technique minimizes the combined cost of the added buffers and the test control logic. We have implemented the technique, and experimental results indicate that it produces high quality results at low computational cost.*

## 1. Introduction

The increasing test application times for SOC (system-on-chip) designs is mainly due to the high amount of test data required for testing. The test application time can be minimized by an efficient organization of the test data transportation. In general, the test data is transported on a TAM (test access mechanism) which can be an added *dedicated* infrastructure for testing purpose only, or an *existing* functional structure, such as the system bus, for example.

Several approaches assuming a dedicated TAM for test data transportation have been proposed [1, 6, 9, 13]. Aerts and Marinissen proposed three TAM structures for scan-tested systems [1], and Varma and Bhatia [13], as well as Marinissen *et al*. [9] suggested dedicated TAM structures. Iyengar *et al.* defined a framework for TAM design and test scheduling on dedicated TAMs [6]. Cota *et al*. proposed a scheme for network-on-chip designs [3], and Nahvi and Ivanov proposed a packet based TAM scheme [11]. Harrod demonstrated the use of the AMBA-bus for test purpose [4].

We propose a buffer-based architecture that makes use of the existing bus structure for test data transportation. The advantage using the existing bus structure is that additional wire routing is not needed, and the advantage of introducing buffers is that we separate the transportation from the application of test data. The scheme adds buffers at each core and an additional controller. We, therefore, propose a Tabu-search-based algorithm to minimize the size of the buffers at each core, as well as the controller and, at the same time not violating the test time constraint. Experiments show that our approach produces results with high quality at low computational cost.

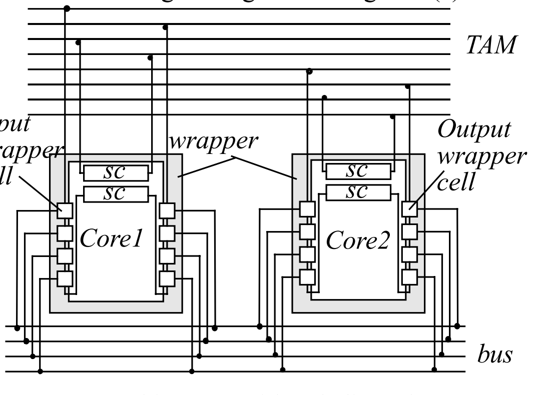
The rest of the paper is organized as follows. Section 2 gives an overview of SOC test architecture. The problem is formulated in Section 3, and the architecture is presented in Section 4. The proposed algorithm is discussed in Section 5. The experimental results are presented in Section 6, and conclusions are drawn in Section 7.

## 2. SOC Test Architectures

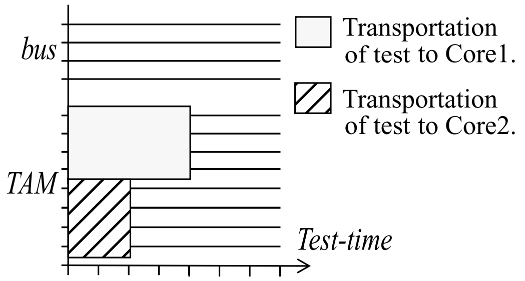
In this section we discuss different types of architectures for test data transportation.

A test architecture with a dedicated TAM is illustrated in Figure 1. The cores are scan-tested and to ease interfacing with the TAM, 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 TAM wires. A dedicated TAM for the transportation of test data has the advantage of high flexibility. It also offers the possibility of a trade-off between the test time and the number of wires used in the TAM. A high number of wires requires extra silicon area to the design, but it enables parallel transportation of test vectors, which will shorten the test time.

A test schedule for the example design in Figure 1(a) is depicted in Figure 1(b). Note, that the bus in the system is not used at all during testing mode. Figure 2(a) shows an architecture where buffers are introduced between the existing bus and the cores, thus separating the transportation of test data from the application. The application of a test vector at a core, that is shift-in and capture, takes longer time than sending the test vector from the ATE to the core. The



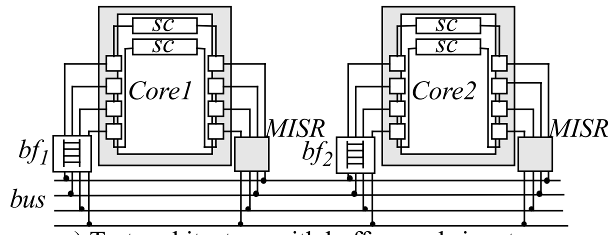
a) Test architecture with a dedicated TAM.


bus
TAM
Transportation of test to Core1.
Transportation of test to Core2.
Test-time

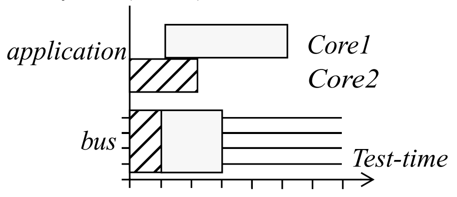
b) Test schedule using dedicated TAM

**Figure 1. A dedicated TAM and the test scheduling.**

idea with buffers at cores is combined with the division of the test set for each core into small packages of test vectors. It means that as soon as a package of test data has arrived to a core, testing at the core can start. Since the test application at a core takes longer than sending test data, the bus can be used to fed test data to other cores. And, hence, the cores are tested concurrently (Figure 2(b)), leading to a reduced test application time.



a) Test architecture with buffers and signature analyzers (MISR)



b) Test scheduling and application with buffers

**Figure 2. Functional bus and buffers**

## 3. Problem Formulation

In this section we formulate the problem. Given is a system consisting of a set of cores $C = \{c_1, c_2, ..., c_N\}$, where $N$ is the number of cores, and each core $c_i$, has a buffer $bf_i$ where $bs_i$ is the buffer size (initially $bs_i$ is not determined). The maximal allowed test time for the system $t_{max}$ is given as a constraint. Also given is the set of tests $T=\{T_1, T_2, ..., T_N\}$, where $T_i$ is a set of test vectors, which is to be applied to the core $c_i$. For each test $T_i$, the following information is given:

- the application-time $t_i^{appl}$ is the time needed to apply the test to core $c_i$,
- the transportation-time $t_i^{send}$ is the time needed to transport $T_i$ from the test source $SRC_T$ via the bus to core $c_i$,
- the size $s^{T_i}$ (the number of test vectors) of the test $T_i$.

A test $T_i$, is divided into $m_{T_i}$ packages, each of the same size $s^{T_i-p}$. The package size $s^{T_i-p}$ for a test $T_i$ is determined as follows[1]:

$$s^{T_i-p} = \left\lceil s^{T_i}/m_{T_i} \right\rceil \qquad (1)$$

The time $t_i^{appl-p}$ to apply a package belonging to test $T_i$ is calculated as:

$$t_i^{appl-p} = t_i^{appl}/m_{T_i} \qquad (2)$$

Associated to each package $p_{ij}$ of test $T_i$ where $j \in (1, m_{T_i})$, are three time points, $\tau_{start_{ij}}$, $\tau_{send_{ij}}$, and $\tau_{finish_{ij}}$. The time to send, $\tau_{send_{ij}}$, represents the start of the transmission of package, $p_{ij}$, on the bus. The time, $\tau_{start_{ij}}$, is the start time of the test at the core $c_i$. Finally, $\tau_{finish_{ij}}$ is the time when the whole package has been applied. The finish time, $\tau_{finish_{ij}}$, is given by the following formula:

$$\tau_{finish_{ij}} = \tau_{start_{ij}} + t_i^{appl-p} \qquad (3)$$

The objective of our technique is to find $\tau_{start_{ij}}$ and $\tau_{send_{ij}}$ for each package in such way that the total cost is minimised while satisfying the test time constraint, $t_{max}$. The total cost for the test is computed by a cost function, that consists of the system's total buffer size and the complexity of the controller given as follows:

$$Cost_{Tot} = \alpha \times Cost_{Controller} + \beta \times Cost_{Buffer} \qquad (4)$$

where α and β are two coefficients used to set the weight of the controller and the buffer cost. The cost of the buffers are given as:

$$Cost_{Buffer} = k_1^B + k_2^B \times BufferSize \qquad (5)$$

and the controller:

$$Cost_{Controller} = k_1^C + k_2^C \times CF1 \qquad (6)$$

where the constants $k_1^C$ and $k_1^B$ are constants reflecting the base cost, which is the basic cost for having a controller and buffers, respectively, and $k_2^C$ and $k_2^B$ are design-specific constants that represent the implementation cost parameters for the number of states and the buffer size.

The total buffer size in the system is given by:

$$BufferSize = \sum_{i=1}^{N} bs_i \qquad (7)$$

The complexity of the test-controller $CTRL_T$ in equivalent two-input NAND gates is given by the following formula described in[10]:

$$\begin{aligned} CF1 = {} & (N_i + N_o + 2 \times \lceil \log_2 N_s \rceil) \times N_t \\ & + 5 \times \lceil \log_2 N_s \rceil \} \end{aligned} \qquad (8)$$

where $N_i$ is the number of inputs, $N_o$ the number of outputs, $N_s$ the number of states and $N_t$ the number of transitions.

## 4. The Buffer-based Architecture

The example in Figure 3 shows a system consisting of three cores, $c_1$, $c_2$, and $c_3$, all connected to the bus $b$. Each core $c_i$ is associated with a buffer $bf_i$ placed between the core and the bus. Also connected to the bus are two test components, $SRC_T$ and $CTRL_T$. We assume that the tests are all produced in the test-source $SRC_T$ and the test-controller $CTRL_T$ is responsible for the invocation of transmissions of the tests on the bus. The test-controller consists of a finite state machine sending a signal $s_i$ to each core indicating when it will receive a package of test data. When the core has received the package, it sends a signal $r_i$ to the controller, indicating that the controller can continue to transmit packages to another core. We assume that the core itself handles the evaluation of the test results, by, for example, a signature analyser.

Each test $T_i$ can be divided into $m_{T_i}$ packages (a set of test vectors). The size of the buffer does not have to be equal to the size of the packages. This is explained by the fact that the test data in a package can be applied immediately when it arrives at the core. The buffer size $bs_i$, associated to a core $c_i$, is calculated with the following formula:

$$bs_i = max(k_i \times (\tau_{start_{ij}} - \tau_{send_{ij}}) + \Delta_i),\ j \in (1, m_{T_i})$$

1. This means that the last test package may have a smaller number of test vectors than $t_i^{size-p}$. We assume that this package is filled up with empty vectors.

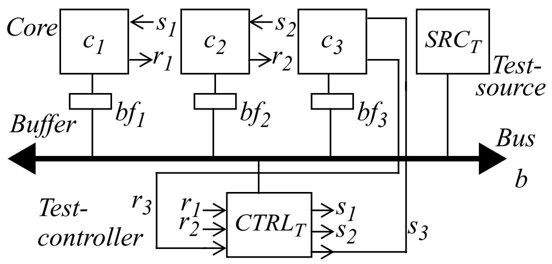
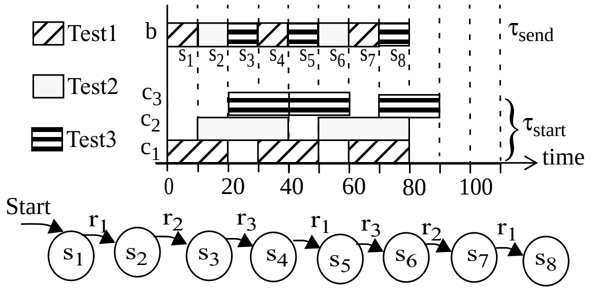


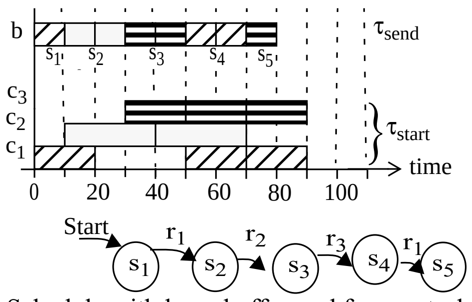
**Figure 3. Bus-based architecture.**

where the constant $k_i$ represents the rate at which the core can apply the test, the time $\tau_{start_{ij}}$ is the scheduled start time of the application of the package $j$ from test $T_i$ at the core, and $\tau_{send_{ij}}$ is the start time for sending the package on the bus. The constant $\Delta_i$ represents the leftover package size, which is the size of the test vectors that remain in the buffer after the transportation of the package terminates. This constant $\Delta_i$ is determined by the difference between $t_i^{appl-p}$ and $t_i^{send-p}$, which is multiplied by the constant $k_i$.

The following example illustrates the minimisation of the buffer size and the test controller complexity. We make use of the example system in Figure 3,which are tested with three tests, $T_1$, $T_2$, and $T_3$ (Table 1). In this example the time $t_{max}$ is set to 90 time units. Two different schedules for the 8 packages derived from the three tests are illustrated in Figure 4(a) and Figure 4(b). In Figure 4(a) the packages are sent in such a way that the application of the previous package has finished before a new one arrives. This leads to small buffers since every package can be applied immediately as they arrive, that is $\tau_{start_{ij}}-\tau_{send_{ij}}=0$ for all packages. The buffer sizes for this schedule are, $bs_1$ =10 ( $bs1_1 = 1 \times (0) + 10$ ), $bs_2$ =20, and $bs_3$ =10. In the second schedule, Figure 4(b), some packages are grouped together in pairs, which will produce larger buffers, $bs_1$ =20 ($bs1_1 = 1 \times (70 - 60) + 10$ ), $bs_2$ =40, and $bs_3$ =20.



a) Schedule with small buffers but high number of control states



b) Schedule with large buffersand few control states

**Figure 4. Scheduling examples**

| Test | Nr packages | Application-time ($T_i^{appl}$) | Transportation-time ($T_i^{send}$) | $\Delta_i$ |
|---|---|---|---|---|
| $T_1$ | 3 | 60 | 30 | 10 |
| $T_2$ | 2 | 60 | 20 | 10 |
| $T_3$ | 3 | 60 | 30 | 10 |

**Table 1. Test characteristics**

## 5. The Tabu-search-based Algorithm

We have implemented a Tabu search based optimisation heuristic for the problem described in Section 3. The algorithm (Figure 5), consists of three steps: in step one an initial schedule is built, which is further improved in step two and step three. The algorithm takes as inputs the tests $T$, a minimal test time possible for the tests, $t_{min}$, and the maximum allowed test time, $t_{max}$. $t_{min}$ is the theoritical minimal time needed for transportation and application of tests $T$, with unlimited buffer and controller cost. This value can be computed by a CLP (Constraint Logic Programming) model in very short time (none of the experiments we have used needed more than 700 ms for computing this value)

```
Step1: if $t_{max} < t_{min}$ return Not schedulable
sort the tests T in increasing order of $t_i^{appl}$
until all packages are applied do
  apply package from $T_i$
  until time < $t_i^{appl-p}$ do
    apply package from $T_{i+1}$
    time = time +$t_{i+1}^{send-p}$
  repeat
repeat
Step2:doMark()
do until Slack is 0
  Delay package from MarkList
repeat
best_cost = compCost(Sched$_0$)
Step3:start:
doMark()
for each pos in MarkList
  build new schedule Sched$_i$
  delta_cost$_i$ = best_cost - compCost(Sched$_i$)
repeat
for each delta_cost$_i$< 0, in increasing order
of delta_cost$_i$ do
  if not tabu(pos) or tabu_aspirated(pos)
    Sched$_0$ = Sched$_i$
    goto accept
  end if
repeat
for each pos in MarkList
  delta_cost$_i$' = delta_cost$_i$ + penalty(pos)
repeat
for each delta_cost$_i$' in increasing order of delta_cost$_i$'
do
  if not tabu(pos) goto accept
repeat
accept:
if iterations since previous best solution < 10 goto start
return Sched$_0$
```

**Figure 5. Algorithm for test cost minimisation**

In the initial step, the tests are sorted according to their application time, $t_i^{appl}$, and then the initial schedule is built. The slack, which is the difference between the end time of the schedule and $t_{max}$, is calculated. In step two, the initial schedule is improved by distributing the slack between the packages, hence, decreasing the buffer size. The schedule is then further improved in step three, were a Tabu search-based strategy is used. Tabu search [12] is a form of local neighborhood search, which at each step evaluates the neighborhood of the current solution and the best solution is selected as the new current solution. In our algorithm the neighborhood is determined by the possible points of improvements in the schedule. These can be points which decrease the buffer size by splitting a package, or decrease the controller cost by merging packages. Each improvement point is defined as a move, which, after is has been applied, is marked as a tabu. When the Tabu search terminates, the solution with the lowest cost is returned.

## 6. Experimental results

In our experiments we have used the following four designs; Ex1, ASIC Z, Kime and System L. The main characteristics of the four designs are presented in Table 2, and detailed information can be found in [8].

| Design | Nr Tests | Nr Packages | Min Buffer size | Max Buffer size |
|---|---|---|---|---|
| Ex1 | 3 | 8 | 40 | 100 |
| Kime | 6 | 20 | 93 | 340 |
| ASIC Z | 9 | 38 | 111 | 419 |
| System L | 13 | 39 | 280 | 988 |

**Table 2. Design characteristics**

We have compared the results produced by our heuristic with those generated by solving the same optimisation problem using a Constraint Logic Programming formulation. Such a formulation has been given by us in [7]. Using a CLP solver [5] we were able to obtain the theoretical optimum for the smallest of our examples (Ex1). For the other three examples the optimisation was terminated after 5 hours without verifying the optimum. For these cases we considered the best result so far produced by the CLP solver.

The experimental results are collected in Table 3. As can be seen from the last column, our heuristic produced results which were only less then 8.6% worse then those produced by the CLP-based approach. However, the heuristic proposed in this paper take 3s for the largest example, while the CLP-based solver was running up to 5 hours and produced results that, on average, were only 4.4% better

## 7. Conclusions

Efficient test data transportation for SOC is becoming an important problem to focus on due to the increasing amount of test data. We propose a technique to make use of the existing bus structure in the system for test data transportation. The advantage is that a dedicated bus for test purpose is not needed, hence we reduce the routing costs. We insert buffers and divide the tests into packages, which means that tests can be scheduled concurrently even if the bus only allows sequential transportation. We have proposed a Tabu-search-based algorithm where the test cost, given by the controller and buffer cost, is minimized. We have implemented and compared our technique with the results from an CLP approach. The results indicate that our technique produces high quality solutions at low computational cost.

## 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] AMBA Specification Rev 2.0, ARM Ltd., 1999

[3] E. Cota, L. Carro, A. Orailoglu, and M. Lubaszewski, "Test planning and Design Space Exploration in a Core-based Environment," *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 478-485, 2001.

[4] P. Harrod, "Testing Reusable IP-a Case Study," *Proceedings of International Test Conference (ITC),* pp. 493-498, 1999.

[5] 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.

[6] 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.

[7] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Buffer and Controller Minimisation for time-Constrained Testing of System-on-Chip," *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 385 - 392, 2003.

[8] E. Larsson, A. Larsson, and Z. Peng, "Linkoping University SOC Test Site," *http://www.ida.liu.se/labs/eslab/soctest/*, 2003.

[9] E. J. Marinissen, R. G. J. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable cores," *Proceedings of International Test Conference ,* pp. 284-293, 1998.

[10] B. Mitra, P.R. Panda, and P.P Chaudhuri, "Estimating the Complexity of Synthesized Designs from FSM Specifications," *Design & Test of Computers,* Vol 10*,* pp. 30-35, 1993.

[11] M. Nahvi and A. Ivanov, "A Packet Switching Communication-based Test Access Mechanism for System Chips," *Digest of Papers of European Test Workshop (ETW),* pp. 81-86, 2001.

[12] C. R. Reeves (Editor), "Modern Heuristic Techniques for Combinatorial Problems", *Blackwell Scientific Publications*, ISBN 0-470-22079-1, 1993.

[13] P. Varma and B. Bhatia, "A Structured Test Re-use Methodology for Core-based System Chips," *Proceedings of International Test Conference (ITC)*, pp. 294-302, 1998.

| Design | $t_{max}$ | Constraint Logic Programming | | Proposed Algorithm | | Cost Comparison |
|---|---|---|---|---|---|---|
| | | CPU time (s) | Total cost | CPU time (s) | Total cost | |
| Ex1 | 111 | 160 | 62(opt) | <1 | 62 | 0% |
| Kime | 257 | 18000 | 275 | 2 | 290 | +5,5% |
| Asic Z | 294 | 18000 | 208 | 2 | 215 | +3,4% |
| System L | 623 | 18000 | 747 | 3 | 811 | +8,6% |

**Table 3. Experimental results.**