Optimization of a Bus-based Test Data Transportation Mechanism in System-on-Chip

Anders Larsson, Erik Larsson, Petru Eles and Zebo Peng

Embedded Systems Laboratory
Linköpings Universitet
SE-581 83 Linköping, Sweden
{anlar, erila, petel, zpe}@ida.liu.se

Abstract

The increasing amount of test data needed to test SOC (System-on-Chip) entails efficient design of the TAM (test access mechanism), which is used to transport test data inside the chip. Having a powerful TAM will shorten the test time, but it costs large silicon area to implement it. Hence, it is important to have an efficient TAM with minimal required hardware overhead. We propose a technique that makes use of the existing bus structure with additional buffers inserted at each core to allow test application to the cores and test data transportation over the bus to be performed asynchronously. The non-synchronization of test data transportation and test application makes it possible to perform concurrent testing of cores while test data is transported in a sequence. We have implemented a Tabu search based technique to optimize our test architecture, and the experimental results indicate that it produces high quality results at low computational cost.

1. Introduction

The increasing test application time for SOC (system-on-chip) is mainly due to the high amount of test data required for testing. The time the chip spends in testing is an important factor for the cost of the production and hence, a maximum allowed test time is often set early in the design phase. However, the test application time can be maintained at a reasonable level by adopting 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 dedicated for test purpose [4].

The main disadvantage with a dedicated TAM is the additional routing it requires. Despite efficient scheduling techniques, usually a wide TAM is required to reach a low test time. On the other hand, the general disadvantage with using the existing bus for test data transportation is that the bus operates sequentially. It means the tests are executed one after the other and only one test is active at a time, which leads to long testing times.

We propose a buffer-based architecture that makes use of the existing bus structure for test data transportation. The buffers are placed between each core and the functional bus and they are used to temporarily store the test vectors while they are applied to the core. The advantage of using the existing bus structure is that additional wire routing is not needed, and the introduction of buffers make it possible to separate test data transportation from the application of test data. The application of test data usually takes longer time than the transportation on the bus since the scan-chains at the cores have to be loaded. In our scheme, we divide the test sets into smaller packages, which are sent on the bus. As soon as a package arrives at a core, testing can start. In our scheme, even if the bus is organized sequentially, the application of tests at cores is performed concurrently, leading to shorter testing times.

By adding buffers, we can send test data to a core and store it until the core can apply the test data. The scheme adds buffers at each core and an additional controller. Both the buffers and the controller contribute to an increased silicon area, however there is a trade-off between the size of the buffers and the complexity of the controller. Small buffers entail a larger set of packages and a complex controller with many control states. On the contrary, a small controller, with few control states, would require large buffers. The size of the packages also affects the overall test time for the system where small packages gives high flexibility to the schedule and a shorter test time as opposed to large packages. In the extreme case, each test is

transported in one package, which leads to a sequential application of the test and long test time. In order to explore this trade-off between the buffer size and controller complexity, an optimization algorithm is required.

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 given by the designer. 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.

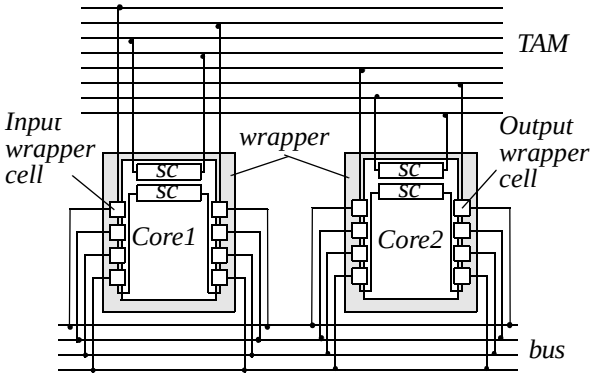
2. SOC Test Architectures

In this section we discuss different types of architectures for test data transportation. We make use of an example to illustrate the architectures for (1) dedicated TAM (Figure 1), (2) existing bus structure as TAM (Figure 2), and (3) our proposed buffer-based TAM scheme using the existing bus. Note, that in all cases, buses are required in normal operation.

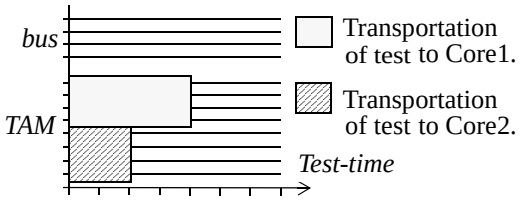
A test architecture with a dedicated TAM is illustrated in Figure 1. The cores, *Core1* and *Core2* are scan-tested and to ease interfacing with the TAM, each core has a wrapper. In the example, *Core1* have longer scan chains than *Core2*, thus *Core1* requires longer test-time time than *Core2* (Figure 1(b)). The scanned elements at a core (scan-chains and wrapper-cells) are connected into wrapper chains, which are connected to TAM wires. In normal operation the inputs and outputs of each core is connected to the bus, while in testing mode the test data is transported on the dedicated TAM. A dedicated TAM for the transportation of test data has the advantage of high flexibility and 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 with no dedicated TAM. The existing bus in the system is used in normal mode as well as for test data transportation in testing mode. The test schedule for the example is shown in Figure 2(b). The testing time is higher due to the bus architecture, which imposes sequential scheduling, and, hence, only one core is tested at a time. The example shows that the bus is the critical resource; it is fully occupied, but, the cores are only activated one after the other.

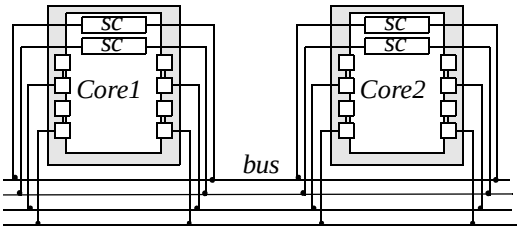


a) Test architecture with a dedicated TAM.



b) Test schedule using dedicated TAM.

Figure 1. A dedicated TAM and the test scheduling.



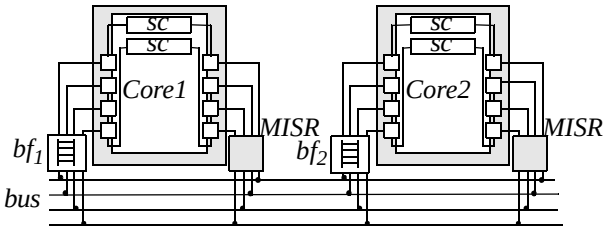
a) Test architecture with only the bus.

b) Test scheduling on the bus without buffers.

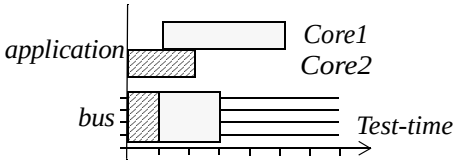
Figure 2. Test tranportation and scheduling using the functional bus.

Figure 3(a), shows an architecture where buffers are introduced between the existing bus and the cores. The advantage with such a scheme is that by inserting buffers, the transportation of test data is separated from the application of test data. The application of a test vector at a core, which is shifted-in and captured, takes longer time than sending the test vector from the ATE to the core over the bus. 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. In this way, the cores are tested concurrently (Figure 3(b)), leading to a reduced test application time.

The assumption that the application takes longer time



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



b) Test scheduling and application with buffers.

Figure 3. Functional bus and buffers.

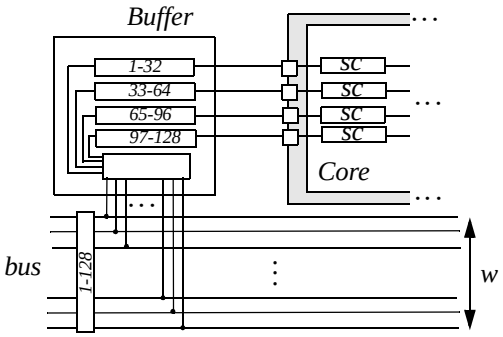


Figure 4. Bus and buffer connected to a core with four scan chains.

than the transportation is derived from the fact that the number of scan chains in the cores, usually is smaller than the bandwidth *w* of the functional bus, thus making it possible to transfer more test data per clock cycle on the bus than what can be applied to the core. We further illustrate this difference with a small example (Figure 4). Here a 128 bit wide functional bus is connected to a core *i*, with four scan chains through a buffer. In only one clock cycle of the bus, the buffer is fed with 128 bits of test data. This is partitioned through a parallel to serial converter, to four scan chains, each with the length of 32 bits. During the next cycle, the bus can transport data to another core *j* while core *i* is occupied for another 32 clock cycles with the shift-in of the scan chains.

One possible alternative to the buffers could be for example to dedicate a subset of the bus wires for testing core *i* and another subset for core *j*. However, this technique would require additional logic to partition the wires and

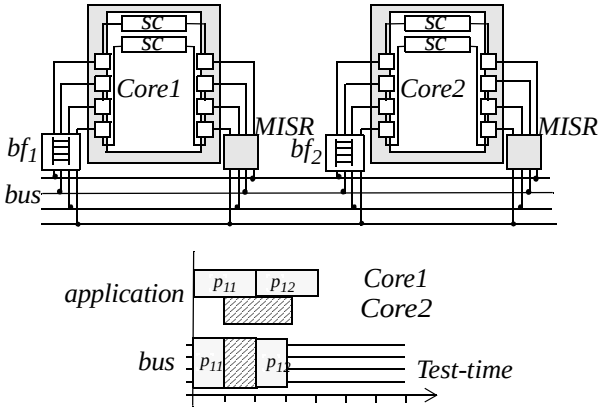


Figure 5. Test scheduling and application of test packages with buffers.

extensive test controller to manage the application of the tests, why this method should only be used together with a dedicated TAM.

The idea with buffers at cores is combined with the division of the test set for each core into small packages of test vectors as illustrated in Figure 5. Here the test to *Core1* has been divided into two separate packages, p_{11} and p_{12}, which then are scheduled in order but without a fixed interval between each package. This leads to a more flexible schedule, which also contributes to a possible decrease of the test application time.

3. Problem Formulation

We now formulate our problem precisely as follows. 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} is 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]:

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

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

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

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

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:

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

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 minimized 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:

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

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:

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

and the controller:

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

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 buffer size is translated into estimated silicon area expressed by the number of NAND gates used.

The total buffer size in the system is given by:

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

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

$$CF1 = (N_i + N_o + 2 \times \lceil \log_2 N_s \rceil) \times N_t + 5 \times \lceil \log_2 N_s \rceil \} \quad (8)$$

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. The formula estimates the complexity of a finite state machine in equivalent two-input NAND gates. In this work the number of inputs N_i and outputs N_o is equivalent to the number of cores and the number of states N_s is equal to the number of transitions N_t. Our problem is similar to the NP-complete multiprocessor resource constrained scheduling problem [14].

4. The Buffer-based Architecture

In Section 2, we described test data transportation using the existing functional bus. By inserting buffers at each core and dividing the test set into smaller packages, we showed that the testing can be performed concurrently even with a bus based on sequential ordering. In Section 3, we introduced the problem formulation and the notations. In this section, we further illustrate the problem.

The example in Figure 6 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. Information needed for the final test result evaluation is also sent via the bus.

Each test T_i can be divided into m_{T_i} packages (a set of test vectors). There are two reasons for dividing tests into packages. As mentioned earlier, the transportation-time $t_i^{send\text{-}p}$ for a package on the bus is shorter than the application-time $t_i^{appl\text{-}p}$. The size of the buffer, however, 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:

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

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

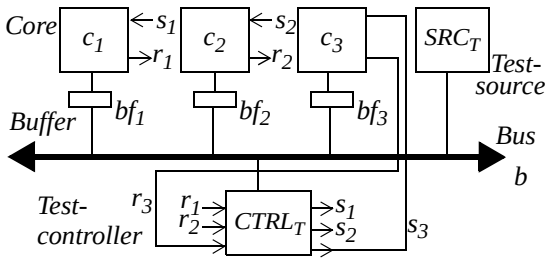


Figure 6. Bus-based architecture.

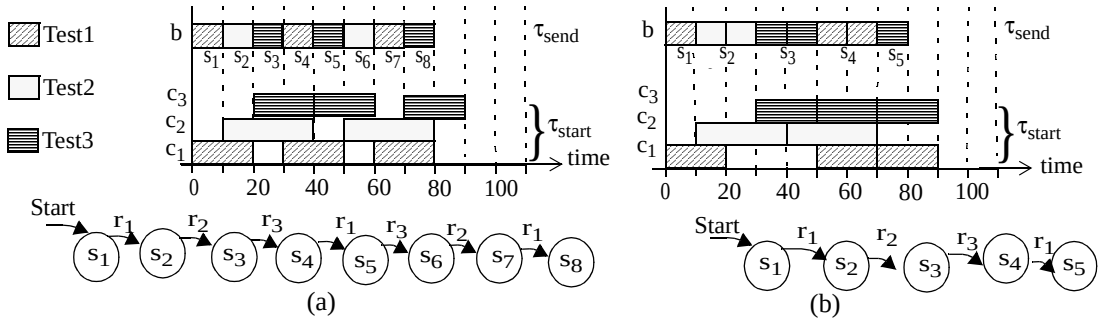


Figure 8. Scheduling examples (a) Schedule with small buffers but high number of control states, (b) Schedule with large buffers and few control states.

bus. The constant Δ_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 Δ_i is determined by the difference between $t_i^{appl\text{-}p}$ and $t_i^{send\text{-}p}$, which is multiplied by the constant k_i.

The calculation of the buffer size is illustrated in Figure 7, which shows the bus schedule and the application of a test T_1 to core c_1, with t_1^{appl}=60, t_1^{send}=30, m_{T_1}=3, and k_1=1. In this example the core has not finished the testing using the package sent at time point $\tau_{send_{1,2}}$=10 before the package sent at $\tau_{send_{1,3}}$=20 arrives at the core. This forces the buffer size to be increased. For this example the buffer size will be equal to 1×(40-20)+10=30, which is the difference between the termination of applying the last test package and the end point of transporting the corresponding package.

The following example illustrates the minimization of the buffer size and the test controller complexity. We make use of the example system in Figure 6, which consists of three cores c_1, c_2, and c_3 which are tested with three tests, T_1, T_2, and T_3, respectively. We have divided the tests into a total number of 8 packages, all with the same application-time and minimum package size, but different transportation-times (Table 1). We assume that the minimal test time for the system is given by the designer. In this example the time is 90 time units, which is the minimal time for applying these tests. This is the sum of the transportation times plus the smallest value of all Δ_i.

Two different schedules for the 8 packages derived from the three tests are illustrated in Figure 8(a) and Figure 8(b).

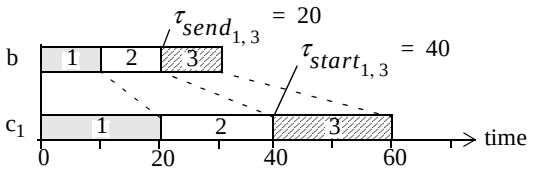


Figure 7. Example to illustrate time to transport and time to apply test.

In Figure 8(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 8(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.

5. The Tabu Search Based Algorithm

We have implemented a Tabu search based optimization heuristic for the problem described in Section 3. The algorithm (Figure 9), 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 theoretical 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).

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

Test	Nr packages	Application-time (t_i^{appl})	Transportation-time (t_i^{send})	Δ_i
T_1	3	60	30	10
T_2	2	60	20	10
T_3	3	60	30	10

Table 1. Test characteristics.

```
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 9. Algorithm for test cost minimization.

packages, hence, decreasing the buffer size. After this step the slack is zero. The schedule is then further improved in step three, were a Tabu search based strategy is used to find the best solution. 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. Unlike local search, which stops if no improved new solution is found, tabu search continues the search from the best solution in the neighborhood even if that solution is worse than the current. To prevent cycling the most recently visited solutions are marked as tabus meaning that they are not allowed to be repeated until the tabu has expired.

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 it has been applied, is marked as a tabu. The tabu tenure, that is the number of iterations when a move is kept as tabu, is set to seven. This value has to be long enough to prevent cycling without driving the solution away from the global optimum. Extensive experiments where carried out to find this value of the tenure. The tabu is aspirated if the cost of the obtained schedule is the best obtained so far. In order to find a good solution an outer loop iterates until no further improvement is made for 10 consecutive tries. Also this number as been set on the basis of extensive experiments. 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 for these benchmarks can be found in [8].

In order to estimate the quality of the results produced by our heuristic we have compared them with those generated by solving the same optimization 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 majority of our examples (Ex1, Kime, and Asic Z). For the last, and largest example (System L) the optimization was not able to find the optimal solution within reasonable time.

The experimental results are collected in Table 3. Column 1 lists the four designs. The results obtained from the CLP solver are shown in column 4 and 5, while the results produced by our heuristic can be found in column 6 and 7. As can be seen from the last column, our heuristic produced results which were only less then 6.1% 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 18 hours and produced results that, on average, were only 3.8% better.

We have also compared our results with the results produced by the CLP solver after the same time as our proposed algorithm needed, i.e. 1s for design Ex1, 2s for

Design	Nr Tests	Nr Packages	Min Buffer size	Max Buffer size
Ex1	3	8	80	200
Kime	6	20	186	680
Asic Z	9	38	222	838
System L	13	39	560	1976

Table 2. Design characteristics.

Design	*Nr Cores*	t_{max}	Constraint Logic Programming		Proposed Algorithm		Cost Comparison
			CPU time (s)	Total cost (No. of NAND gates)	CPU time (s)	Total cost (No. of NAND gates)	
Ex1	3	111	160	92	<1	92	0%
Kime	6	257	27375	460	2	486	+5,7%
Asic Z	9	294	47088	319	2	330	+3,4%
System L	14	623	64841	1182	3	1254	+6.1%

Table 3. Experimental results.

design Kime and Asic Z, and finally 3s for System L. This comparison showed that our Tabu search based algorithm on average produced solutions that were 10.2% better.

7. Conclusions

Efficient test data transportation for SOC is becoming an important issue due to the increasing amount of test data that cause long test times. The test application time is an important factor for the cost of the production and hence, a maximum allowed test time is often set early in the design phase. 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 without exceeding the given maximum test time. 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.

8. 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, M. Kreutz , C.A. Zeferino, L. Carro, M. Lubaszewski, and A. Susin, “The Impact of NoC Reuse on the Testing of Core-based Systems”, *Proceedings of VLSI Test Symposium*, pp. 128-133, 2003.

[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.

[14] M. R. Garey and D. S. Johnson, ”Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman And Company, New York, 1979.