In this paper, we propose a novel ant colony optimization (ACO)-based test scheduling method for testing network-on-chip (NoC)-based systems-on-chip (SoCs), on the assumption that the test platform, including specific methods and configurations such as test packet routing, generation, and absorption, is installed. The ACO metaheuristic model, inspired by the ant’s foraging behavior, can autonomously find better results by exploring more solution space. The proposed method efficiently combines the rectangle packing method with ACO and improves the scheduling results by dynamically choosing the test-access-mechanism widths for cores and changing the testing orders. The power dissipation and variable test clock mode are also considered. Experimental results using ITC’02 benchmark circuits show that the proposed algorithm can efficiently reduce overall test time. Moreover, the computation time of the algorithm is less than a few seconds in most cases.

Keywords: NoC-based SoC test scheduling, rectangle packing, ant colony optimization.

I. Introduction

To prepare for the deep submicron era, the trend in semiconductor design has moved toward system-on-chip (SoC). The modern SoC has been gaining faster market access through intellectual property reuse to reduce the gap between design complexity and production. Recently, highly efficient SoCs, such as network processors or media processors, which have separate intellectual properties for various individual functions and several processors, have been developed to increase price competitive power of chips as well as systems. Continuing this trend, it is expected that SoCs including hundreds of embedded cores will appear in the near future.

In a high-density SoC design, a communication scheme among built-in cores is a main design constraint and dominates issues of system architecture, performance, robustness, power consumption, and cost. The shared bus has generally been exploited for interconnection architectures within systems. However, due to deep submicron technologies and increases in the number of embedded cores and system frequencies, designers are contending with difficulties related to signal and power integrity within the shared bus architecture. For this reason, new models and templates suitable for future SoCs are needed. One such emerging approach is the network-on-chip (NoC)-based architecture and platform [1]. An NoC can be defined as an interconnection model implemented on a chip in the form of a micro-network [2]. Since NoCs have some outstanding features regarding design complexity, timing, scalability, and power dissipation, widespread interest in this novel paradigm is likely to grow.

Like all other SoCs, NoC-based SoCs must be tested for manufacturing defects. Today, testing SoCs has become one of the most expensive and time-consuming tasks of the circuit
design process, and the reduction of this cost is crucial for the electronic market. Considering the complexity of NoC-based SoCs, it is easily expected that their test cost will be much higher than that of typical SoCs. Therefore, test strategies are highly significant factors in the development of practical and feasible NoC-based systems.

In NoC-oriented systems, dedicated test logics are generally not suited to implementation due to their hardware overhead, such as the silicon area and pin count. Therefore, we focus on the test problems related to NoC characteristics and aim to make effective use of NoC as a test resource. For this purpose, we previously developed a test method to overcome the limitation of network-based test architectures and to improve test efficiency by considering test parallelism as well as test pipelining [3]. With the proposed test platform [3], many test scheduling ideas based on common SoC structures can be directly applied to testing NoC-based SoCs. In [3], a rectangle packing approach based on [4] was implemented to solve the NoC-based SoC test scheduling problem and produced good results. However, in the rectangle packing method, scheduling efficiency strongly relied on the preferred test access mechanism (TAM) width for each core, chosen heuristically and then fixed from start-up in most cases. The preferred TAM width means a pre-calculated value providing test efficiency similar to that of using the maximum TAM width.

This paper proposes an ant colony optimization (ACO)-based rectangle packing algorithm for test organization of NoC-based SoCs. The ACO model is a new approach inspired by the adaptive and social behavior of ants in finding the best route to a food source from their nest by indirect communication between ants using a chemical substance called a pheromone. The proposed method can dynamically choose the TAM widths and vary the testing order of cores through the actions of an artificial ant, which effectively handles the minimal cost problem. Consequently, this test solution could improve test parallelism despite preserving the test pipelining, and thus find better scheduling results. In addition, multiple test clocks and power dissipation are incorporated for actual testing conditions.

In the next section, we review prior studies and present the test platform proposed in [3]. The generic problem-solving procedure with ACO will be described in section III. The proposed scheduling method using rectangle packing based on ACO is presented with pseudo-code in section IV. The experimental results and comparisons with previous results are given in section V. Finally, conclusions are presented in section VI.

II. Related Research

The general concept of testing NoC-based SoCs first appeared in [5]. Before the built-in core test, the communication infrastructure of an NoC should be tested. The primary issues and methodologies regarding testing communication resources are introduced in [6]. Several approaches for testing an NoC are also presented in [7]-[10]. After verifying the communication resource, we can advance to the standard core test using an on-chip network as a TAM. An early approach on the subject of test architectures utilizing an on-chip network was proposed in [11]. The proposed network-oriented test architecture, novel indirect and modular architecture (NIMA), lays the groundwork for realizing a test architecture that benefits from the reuse of an NoC interconnect template.

Test scheduling algorithms for NoC-based SoCs can be grouped roughly into two main categories: packet-based and core-based scheduling. Packet-based scheduling determines the order of generation and transmission of test packets for cores according to the priority of each core [12], [13]. Test vectors and responses per core are represented as a set of packets to be transmitted throughout the network, and the packets are scheduled to minimize the total test time using test parallelism. In test parallelism, several cores are tested simultaneously, improving test efficiency through fully utilizing the network bandwidth. Core-based scheduling determines the test order of each core and the tested core is exclusively privileged to use all resources required for testing until the entire test is completed [14]-[17]. In this approach, the scheduler assigns each core a routing path, including an input port, an output port, and corresponding channels that transport test vectors from the input to the core and test responses from the core to the output. Once the core is scheduled on this path, all resources on the path are reserved for the core test until the entire test is completed. Because the proposed idea maintains test pipelining from a test vector input to a test response output for a core under test (CUT), it shows fairly good scheduling results. Test pipelining means the continuous, concurrent operation of scan input of a test vector and output of a response. As the collision among the routed test data causes test pipelining to be broken, the corresponding routing path for each CUT has to be reserved. In [15], the idle channel width of TAM, which cannot contribute to reducing the overall system test time, is efficiently utilized through a combination of on-chip clocking and parallel-serial conversion of test data. In order to preserve test pipelining, however, the test clock generated from an on-chip phase locked loop (PLL) for a CUT should increase in proportion to the number of test vectors transmitted together through a test packet. Also, a method of test scheduling that integrates router testing with embedded core testing is presented in [16]. Most recently, an improved version of [15] was proposed, which considers the thermal balance of an overall system to guarantee thermal safety during testing [17].
Unlike these scheduling approaches, our method reuses test scheduling heuristics originally proposed for testing traditional SoCs [3]. In [3], a new test platform which defines specific methods and configurations for testing NoC-based SoCs, such as test packet routing, test pattern generation, and absorption, was presented, and a rectangle packing heuristic was used for test scheduling on the test platform. The rectangle packing heuristic was first introduced in [18]. In [18], the SoC test scheduling problem was formulated as a 2-dimensional bin packing problem, and each core was represented by a rectangle, the width of which was the number of SoC pins allocated and the height of which was the core test time given the number of SoC pins. In [4], rectangle representation was used, and a technique based on rectangle packing was used for wrapper/TAM co-optimization and test scheduling for SoCs. A restricted 3-dimensional bin-packing model [19] was used for power constrained SoC test scheduling. In [19], a core testing was represented by a 3-D cube, where the TAM width, test time, and power consumption corresponded to each of the three dimensions, respectively. After choosing a cube for each core, the selected cube was packed to a 3-D bin implying the SoC testing. Packing a bin is a typical NP-hard problem. To find the optimal solution, all possible cases must be examined. However, as the search space grows exponentially with respect to the number of cases, it is crucial to develop an efficient exploration methodology. The previously mentioned bin-packing-based scheduling algorithms choose rectangles heuristically; however, they only consider some of the possible cases. Therefore, the scheduling results can be further improved.

Next, we briefly review the test platform. If the platform is used, the number of TAM widths can be set to \(K+1\), where \(2^k \leq W\) (\(W\) is the channel width of an NoC) despite the NoC-based TAM structure. For more details, refer to [3].

1. Test Configuration

In this study, as shown in Fig. 1, we assumed an NoC with a 2-D mesh topology, XY routing, and wormhole switching. In XY routing, a packet is first routed in the X direction and then in the Y direction before reaching its destination. In the case of wormhole switching, the routed packet is broken up into flits, units of flow control, and they are transported in a pipelined manner.

In Fig. 1, the test source can generate test vectors at a rate of up to one packet per network time step. The test sink can absorb test responses at the same rate as the test source, and the test controller assures that test sources and sinks satisfy test pipelining and parallelism by means of a predetermined test schedule. A test source and sink are attached directly to routers and the number of test sources and sinks in a multi-source/sink configuration type [12]-[17] corresponds to the difference in clock rate between an NoC and CUTs in this single source and sink type.

2. Generation and Routing of Test Packets

As seen in Fig. 2, a packet can transfer multiple test vectors if the packet size is several times larger than the vector size in order to efficiently utilize network bandwidth in the spatial domain [15]. Furthermore, to improve network bandwidth usage in the temporal domain, a time-division packet transmission is implemented to allow several cores to be tested concurrently without test clock multiplication. However, the number of test vectors contained in a test packet is restricted to powers of 2, that is, \(1, 2, \ldots, 2^k\) \((2^k \leq W, W\) is the channel width of a network\) for simplicity of calculating transmission cycles. In other words, as test vector size corresponds to the TAM width assigned to a core, the usable TAM widths are limited to powers of 2.

To improve the test scheduling results, test pipelining for each core is very important because it can minimize the test time of the core. Therefore, test packet collisions in a network should be avoided to preserve test pipelining. In the test packet routing process, there are three types of packet collisions. The first type is collision between test vector packets. The test vector packets are routed from the test source to CUTs. If all of them are uniformly generated in the test source one by one and move on paths in a deterministic way, such as XY routing, they will never meet each other. The next type is collision between test response packets. Unlike the first type, because test response packets are generated from several CUTs at variable times and all of them are routed toward the test sink, collision is inevitable. However, global combining used in [3] can solve this problem efficiently. Global combining eliminates packet collision by combining various packets into one packet.
example, suppose that there are two test response packets forwarding in the same direction in a router. Because a test scheduler maintains the total sum of TAM widths of CUTs simultaneously tested less than the channel width of the network, the sum of the valid data widths of the packets is within the channel width. Therefore, the router can combine the packets into one packet and avoid this type of collision. The last type is collision between test vector packets and response packets. This type of collision is avoidable if the test source and sink are put on the opposite sides of a row or column in an NoC and no core is installed in that row or column (see Fig. 1). This configuration can separate the routing path of the test vector packets from the test response packets. Figure 1 shows an example of the overall configuration to test an NoC-based d695 benchmark circuit. The numbers in Fig. 1 denote the core numbers of the d695 benchmark circuit.

III. ACO Metaheuristic

Social insect behavior can be characterized as the collective behavior of individuals. Insects make swarms and perform tasks or make decisions as if they have intelligence. Though individual insects can only perform simple tasks, the sum of individual skills can result in the solution of more complicated problems. These mechanisms have been reverse engineered to give rise to a multitude of ant algorithms, based on stigmergetic communications and controls [20]-[21]. The ACO model is population-based and imitates the adaptive and social behavior of ants foraging for food. At first, ants navigate randomly to find food. Ants that find food return to their nest. Whenever ants move, they deposit pheromone on the path traversed. Later, other ants detect pheromone deposited by previous ants, and are inclined to follow the path with more pheromone. Over time, most ants move along the same path, and this path corresponds to an optimal solution for finding the best route to the food source.

In order to solve an NP problem using ACO, the problem should be defined as a graph of sets of components and transitions to simulate an ant path. Furthermore, as artificial ants stochastically build new solutions using a combination of heuristic information and artificial pheromone trails, it is necessary to define the pheromone trails and the heuristic preference properly in order to converge on good solutions.

We explain the basic ant mechanisms through simple-ACO, or S-ACO [21], operation. In S-ACO, two types of mobile agents, forward and backward ants, mimic ant behavior when traveling on paths between source and destination nodes. Forward ants move from the source nodes toward the destinations and backward ants travel back to the sources. At source node $s$, a forward ant, $F_{s\rightarrow d}$, is regularly created to go to destination node $d$ in a search to discover a low-cost path between the nodes. At each node, the forward ant locally selects a next neighbor node to hop to in accordance with probability values of all links connecting the current node to its neighbor nodes. These values are stored in pheromone trails previously deposited on the graph by other ants. Pheromone trails in each node indicate the estimation values about the quality of the corresponding paths passing to the next node. For example, let $\tau(i,j)$ be the intensity of the pheromone trail on the path between nodes $i$ and $j$. The probability, $p(i,j)$, that an ant will choose node $j$ as the next node at node $i$ is given by

$$p(i,j) = \begin{cases} \frac{\tau^\alpha(i,j)}{\sum_{g\in S} \tau^\alpha(i,g)}, & \text{if } j \in S, \\ 0, & \text{otherwise}, \end{cases} \quad (1)$$

where $S$ is the set of nodes not visited yet by the current ant, and $\alpha$ is the parameter which defines the relative importance of $\tau$. Therefore, the path with the highest probability of destination success has the best chance to be selected as the next node to be visited on the path toward the destination. This does not guarantee that the best edge will be chosen every single time. This allows ants to adapt their exploration activity to a wider solution space. At each node visited, the forward ant chooses a next neighboring node, which has not been visited previously during the journey. However, when an ant is forced to return to an already visited node or a cycle occurs, the ant is destroyed or continues on its journey toward the destination by choosing a different next node. The forward ants memorize the paths they have visited and some pieces of information while searching for the destination node.

When the destination is reached, the ant $F_{s\rightarrow d}$ creates a backward ant, $B_{d\rightarrow s}$. which carries all the information gathered by the forward ant back to the source node and dies. On the way back to the source node from the destination node, the backward ant takes the same route as the forward ant did, but in...
the opposite direction, by using the information (gathered by the forward ant) to know which node to hop to next. While returning home, the backward ant evaluates the cost of the paths the forward ant has traversed and updates the pheromone trails stored at each node on its journey without gathering further information. Here, pheromone is updated according to the following equation:

$$\tau_{ij}(t + 1) = \tau_{ij}(t) + \Delta \tau_{ij}(t),$$  \hspace{1cm} (2)

where $\Delta \tau_{ij}(t)$ is the amount of pheromone deposited or the reinforcement value made by an ant on the basis of the quality of its routing results, such as travel time and the rate of updates. Reinforcement is a critical quantity [22], which must be assigned by considering three aspects. First, paths should receive an increment in their selection probability proportional to their quality. Second, quality is a relative measure, which depends on the routing conditions and can be estimated by means of the local statistical model. Third, it is important not to follow all the routing fluctuations. To follow all fluctuations would be a serious mistake because the uncontrolled oscillation in pheromone values is one of the main problems with ACO.

Pheromone trails can also be varied according to static effects, regardless of an ant’s action. One of those effects is evaporation, influencing pheromone values in the following fashion:

$$\tau_{ij}(t + 1) = \rho \cdot \tau_{ij}(t),$$  \hspace{1cm} (3)

where $\rho$ is the evaporation parameter, which determines the speed of decay of pheromone and $\rho$ mimics the volatile chemical characteristic of pheromone. If the evaporation parameter is used, all pheromone gradually fades, except pheromone trails that are reinforced. These steps taken by the backward ant repeat until pheromone trails at the source node are updated and the ant dies. For more details, refer to [21] and [23].

In addition to this basic procedure, there are various heuristic methods to improve ACO performance. One method is a mixture of the iteration-best solution ($S^i$) and the global-best solution ($S^g$). It contributes to solving the premature convergence or stagnation problem of the search [24], [25]. The best solution found since the start of the algorithm is denoted by $S^g$, and the best solution in the current iteration of an ant colony is denoted by $S^i$. To balance exploration of the solution space and exploitation of the best result found previously, $S^g$ is mainly used to update the pheromone trail, and $S^i$ is often used periodically. The period is a parameter to tune ACO performance. A limit on the amount of pheromone is also a well-known heuristic to enhance the ant’s exploration [25].

Since the ant-based approach can find high-quality solutions, provided that the problem is well defined on the basis of ACO design policy, it promises to be a good solution for the test scheduling problem of NoC-based systems, which corresponds to a static NP problem.

IV. ACO-Based Test Scheduling

1. Rectangle Packing Representation of Test Scheduling Problem

An NoC-based SoC test scheduling technique can be formulated in terms of a rectangle packing problem if the test platform summarized in sections II.1 and II.2 is used. The test time varies with TAM width in a staircase pattern, and the testing of a core is represented as a rectangle whose height indicates the TAM width assigned to that core and whose width denotes the test time of the core for the corresponding value of the TAM width. Thus, we can obtain a number of TAM width and test time combinations for the same core. Taken as a whole, a test scheduler chooses just one rectangle from the candidate rectangle set of each core and then packs it into a thin of a fixed height and unlimited width until the bin is filled with rectangles of all cores embedded in SoC, while minimizing the overall width of the bin without overflowing the bin’s height.

For example, as seen in Fig. 3, let NoC-based SoC $N$ consist of 10 cores. The channel width of the NoC is $W$, and the system clock frequency of the NoC, $f_N$, is three times faster than test clock frequencies, $f_i$, of embedded cores. We assume that the test speeds of all cores are uniform. A scheduler selects a rectangle, $r_i$, from $R_i$ ($1 \leq i \leq 10$), using heuristic methods. Here, $R_i$ denotes a rectangle set representing TAM width and test time combinations for core $i$. Eventually, the test scheduling problem of $N$ becomes finding a geometric packing method to minimize the longest width among three bins.
without overflowing each bin’s height. In Fig. 3, the channel width of the NoC, \( W \), corresponds to each bin’s height; the TAM width assigned to core \( i \), \( w_i \), corresponds to the height of \( r_i \); the relative speed between \( f_0 \) and \( f_r \), \( n \), corresponds to the number of bins; and the total test time of SoC \( N, T \), corresponds to the longest width among bins.

2. ACO-Based Rectangle Packing

Here, we describe how the ACO algorithm can be implemented for a rectangle packing solution. A rectangle packing problem consists of two sub-parts: rectangle selection and rectangle packing. The ACO algorithm can be applied to both parts or either part. Through some experiments, we determined that ACO should be used only to select rectangles due to the computation time. Therefore, the rectangle packing method is based on the procedure used in [3].

Before describing the implementation of ACO for the rectangle packing solution, some features related to ACO need to be clarified.

A. Definition of Pheromone Trail \( \tau(k) \)

We define the pheromone trail, \( \tau(k) \), as the favorability of choosing \( k \) as the TAM width assigned to core \( i \) and calculate it as

\[
\tau(k) = \sum_{j \in S} \tau(k_i, g_j), \quad 1 \leq i, \quad j \leq m, \quad 1 \leq 2^k, \quad 2^k \leq W, \quad (4)
\]

where \( S \) denotes the cores already visited by the current ant, \( g_j \) is the TAM width selected for core \( j \), \( m \) is the number of embedded cores, and \( W \) is the channel width of an NoC. Consequently, \( (k_i, g_j) \) is the favorability of choosing \( k \) as the TAM width for core \( i \) when the TAM width assigned to core \( j \) is \( g \).

B. Heuristic Variable \( \eta_i \)

The ACO algorithm combines pheromone information with heuristic values to find solutions, since heuristic parameters can help ACO be applicable to various conditions. Here, we use a preferred TAM width for core \( i \), \( w_{prefer}(i) \), as the heuristic favorability, \( \eta_i \). The calculation flow to seek \( w_{prefer}(i) \) is shown in [3].

C. Stochastic Decision with \( \tau(k) \) and \( \eta_i \)

The probability \( p_i(k) \) that an ant will choose \( k \) for core \( i \) is given by

\[
p_i(k) = \frac{\tau(k_i) \cdot \beta}{\sum_{j=1}^{m} \tau(k_x) \cdot \beta}, \quad (5)
\]

where \( \beta \geq 1 \) if \( w_{prefer}(i) = k \) and 1 otherwise.

D. Policy for Pheromone Updating

According to test scheduling results, pheromone trails are updated as follows:

\[
\tau(k_i, g_j) = \rho \cdot \tau(k_i, g_j) + \Delta \tau(k_i, g_j), \quad (6)
\]

where \( \rho \) is the evaporation parameter, which determines the speed of decay of pheromone; \( \Delta \) is the amount of pheromone deposited by the ant on the basis of the quality of the scheduling result and is defined as

\[
\Delta \tau(k_i, g_j) = N_{k, g_j} \cdot \lambda, \quad (7)
\]

where \( N_{k, g_j} \) indicates how many times \( k \) of core \( i \) and \( g \) of core \( j \) go together in the best scheduling result \( S_{best} \), and \( \lambda \) is the constant value to weigh \( N_{k, g_j} \).

The method of rectangle selection using ACO is explained in the following example. As shown in Fig. 4, let an SoC include three cores. The rectangle sets of cores, \( R_i \) (\( 1 \leq i \leq 3 \)), can be represented as nodes linked with one another in the graph. At first, an ant randomly selects a starting node and a rectangle in that node. In Fig. 4, we assume that the ant chooses \( R_1 \) as a starting node and rectangle \( r_{11} \). Then, the ant can select \( R_2 \) or \( R_3 \) as a next node. After arbitrarily selecting a next node, according to pheromone trails \( (\text{table}) \) in the figure stored in \( R_1 \) and heuristic information, \( w_{prefer} \), the ant chooses a rectangle in that node. For example, if \( R_2 \) becomes the next node, two pheromone trails, \( \tau(w_{21}, w_{11}) \) and \( \tau(w_{22}, w_{11}) \), and one heuristic value, \( \eta_{R_2} \), are used to choose a rectangle in \( R_2 \) by (5). As
previously mentioned, $\tau(w_{21}(or \ w_{22}), \ w_{11})$ is the favorability of choosing $w_{21}(or \ w_{22})$ as the TAM width for core 2 when the TAM width assigned to core 1 is $w_{11}$, and $\eta_{pfr}$ is the preferred TAM width of core 2. In this example, we assumed that $r_{21}$ in $R_2$ is selected by (5). The ant continues this process until all nodes are visited just once. After choosing all rectangles in $R_n$, the ant packs them using the method used in [3] and gets a scheduling result. Figure 4 illustrates a scheduling result with three rectangles, $r_{11}$, $r_{21}$, and $r_{22}$. Finally, the result is based on updating pheromone values. If the result is better than the current best result, pheromone values, such as $\tau(w_{21}, \ w_{11})$ in $R_1$ and $\tau(w_{22}, \ w_{11})$ in $R_2$, can be reinforced.

In addition to basic ACO methods, we adopt several heuristic tips previously used, such as iteration-best ant (S$^i$), global-best ant (S$^{gb}$), and lower limit of the pheromone values ($\tau^{min}$) [23], [25]. These methods are efficient means to balance exploitation of the best solution found and exploration of the solution space. Furthermore, we assume that just one ant per colony can form a pheromone trail.

3. Power Consumption Model

During testing, the peak power cannot be over the power limit of the system. In this paper, we added a power dissipation constraint to the test scheduling procedure. Generally, there are two sources of power consumption in an NoC-based SoC: the communication resources, such as routers and channels, and the cores. We limited ourselves to considering the power consumption at the cores, as the power consumption of communication resources is difficult to estimate at this time.

The power consumption model [13] of a core $i$, $P_i$, can be represented as

$$P_i = C_i \cdot V_{dd}^2 \cdot \frac{1}{T} \left\{ (\sigma_{ff} + 1) \cdot nb_{ff} + \sigma_{gg} \cdot nb_{gg} \right\},$$  

(8)

where $C_i$ is the load capacitance, $V_{dd}$ is the voltage applied to the core, $T$ is the test clock period, $\sigma$ is the switching factor, and $nb_{ff}$ and $nb_{gg}$ denote the number of active flip-flops and gates in the core, respectively. To simplify the power calculation, we assume that $P_i$ does not vary with the number of wrapper scan chains and is the same over the entire time that the core is tested. The scheduler always checks whether the overall power consumption of cores tested at a given time is over the power budget before packing rectangles. As the power consumption is directly proportional to the test clock frequency, the two constraints should be considered together.

4. Overall Test Scheduling Procedure

The ACO-based test scheduling procedure is shown in Fig. 5.

In the procedure, $N_{colony}$ denotes the number of ants in one colony, $N_p$ is the iteration-best ant number, $\gamma$ is the waiting number until $S^{gb}$ is used again, and $ant\_exploration$ means an ant’s action. The ACO parameters include the variables previously mentioned, such as $N_{colony}$, $\gamma$, $\beta$, $\rho$, $\tau^{min}$, and so on.

Before beginning the scheduling, we design test wrappers for embedded cores and found $R_1$, (line 1), using the algorithm proposed in [26]. For reference, we only consider the balanced wrapper designs. Next, at line 2, heuristic variables, $w_{frfr}(i)$, are computed as shown in [3]. Then, ACO parameters are set. As it is impossible to calculate the parameters theoretically, we set them according to experimental analysis and recommendations from prior works [25], [27]. After doing so, pheromone table entries are initialized (line 4). Each core contains scheduling information as shown in Fig. 6, where $w_{frfr}$ is the preferred TAM width, $w_{selected}$ is the TAM width from the ACO-based rectangle selection process, and $w_{assigned}$ is the TAM width assigned at the end after rectangle packing finishes. If the packing result is excellent, $w_{assigned}$ will be reinforced. The estimated peak power information in this core is denoted by $pwr$, the position of the bin including this core is denoted by $bpos$, and $clk\_mode$ is the test clock rate of the core relative to the normal test clock.

Next, the ACO-based test scheduling procedure is advanced (lines 5 to 9 in Fig. 5). Unless the condition to terminate the

---

**Fig. 5. ACO-based test scheduling procedure.**

1. $w_{frfr}$ //Preferred TAM width from heuristic estimation
2. $w_{selected}$ //Selected TAM width from ACO rectangle selection procedure
3. $w_{assigned}$ //Assigned TAM width after rectangle packing procedure ends
4. $begin\_time$ //Test start time
5. $end\_time$ //Test end time
6. $scheduled$ //Flag which indicates this core has been scheduled
7. $completed$ //Flag which indicates testing has completed
8. $bpos$ //Position of bin that this core is scheduled
9. $clk\_mode$ //Test clock rates
10. $pwr$ //Peak power information
11. $bpos$ //Pheromone trails

---

**Fig. 6. Scheduling information of a core.**
simulation is satisfied, an ant will search the solution space continuously with new \( r \) combinations. Currently, the termination condition is whether \( S_T \) is uniform for the predetermined iterations and pheromone trails are biased sufficiently.

As mentioned in section IV.2, after choosing \( r \) for all cores, the ant packs them while minimizing the idle space. As in [3], we use the packing method, which is based on TAM optimizer [4] for its simplicity and feasibility. Finally, if the packing result of the ant is better than the current best result, the scheduler updates the best test time. The pseudo-code of an ant’s action is shown in Fig. 7.

V. Experimental Results

We simulated four ITC’02 benchmark circuits [28] to evaluate the proposed scheduling algorithm. All of our simulations were conducted on a SUN UltraSPARC III with 1.2-GHz processors. The final results reported in this section are based on the following ACO parameter values:

\[
N_{\text{column}}: 10, \gamma: 10, \rho: 0.95, r_{\text{init}}: 2.0, r(0): 20.0, \beta: 10, \lambda: 0.2,
\]

where \( r(0) \) is an initial value of a pheromone trail. To obtain parameter values that achieve good performance, tests using circuits with different sizes and structures are required. As previously mentioned, the parameter values used here are chosen on the recommendations of [25] and [27] and the consideration of balancing the test application times with the calculation times through some experiments.

Table 1 displays the results of an experiment in which various test clock modes were used for the core test. Column 2 of the table shows the test configuration method related to the number of test sources and sinks. In order to make comparisons with previous results, we assume that the network clock frequency, \( f_c \), could increase up to four times as fast as the core test clock frequency \( f_T \), that is, \( n = f_c / f_T \), and \( n \) is an integer. In Table 1, we compare the test times of the proposed method with those of the method presented in [3]. The network channel width is either 32 or 16 bits. With two multiple test clock modes, we confined the test clocks of all cores to \( f_T \) or \( 2f_T \). In case 1, we assigned a \( 2f_T \) clock to core \( i \) if \( T(W) = T(W/2) \). On the other hand, \( f_T \) or \( 2f_T \) was dynamically chosen according to the TAM width available in case 2. Consequently, our algorithm improves on earlier methods in most cases, regardless of circuit size, channel width, \( W \), and network speed, \( n \). Compared with [3], the test time reduction ratio increases by up to 2.1\% at \( W=32 \) and 2.5\% at \( W=16 \) on average. The improvement in \( W=16 \) is more general than that in \( W=32 \) because the number of TAM width and test time combinations in \( W=16 \) is smaller. This means that the proposed method can search the solution space more broadly. Also, though the average reduction ratio is the highest in the single-clock mode, the test time efficiency uniformly increases in all circuits and conditions.

Next, we show the experimental results incorporating power constraints in Table 2. As power consumption information on benchmark circuits is not available, we replace \( n_{\text{by}} \) in (8) by the sum of the numbers of scan flip-flops, inputs, outputs, and bidirectional ports for power calculation. To evaluate the scheduling results including power constraints, we assume that a system has an overall power limit, \( P_{\text{max}} \), and that each core \( i, 1 \leq i \leq m \), has a separate power limit, \( P_i \). We set the power limit, \( P_{\text{max}} \), to 50\% and 30\% of \( \Sigma P_i \) and set \( W \) to 32-bit to compare with the previous results in [15]. As the results demonstrate, the proposed algorithm shows good performance in most cases. When \( P_{\text{max}} \) is set to 50\%, the test time reduction ratio goes up to 26\% on average. However, when \( P_{\text{max}} \) is set to 30\%, the reduction ratio is somewhat variable according to the
The reason for inefficiency in scheduling $g_{1023}$ and $p_{22810}$ under used circuits and experimental conditions. In particular, the imbalance between the fixed test-clock rates and the power consumption of the embedded cores of those circuits. Therefore, it is important to dynamically select test clocks in circuits including cores whose test times vary extensively depending on the assigned TAM widths and whose power dissipation is greater than that of others. Moreover, though it can be longer according to ACO parameters, the computation time of our algorithm is no more than a few seconds in most cases, regardless of power constraints. This is a very encouraging indicator for the practicality and feasibility of the proposed algorithm.

We also compared the proposed ACO-based method with a simulated annealing (SA)-based algorithm proposed in [29].

Table 1. Experimental results with no power constraint ($P_{\text{max}} = 100\%$).

<table>
<thead>
<tr>
<th>SoC name</th>
<th>n</th>
<th>$W = 32$</th>
<th>$W = 16$</th>
<th>$W = 32$</th>
<th>$W = 16$</th>
<th>$W = 32$</th>
<th>$W = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D695</td>
<td>2</td>
<td>13732</td>
<td>11978</td>
<td>13</td>
<td>23193</td>
<td>21437</td>
<td>7.6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9869</td>
<td>9869</td>
<td>0.0</td>
<td>16197</td>
<td>15124</td>
<td>6.6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9869</td>
<td>9869</td>
<td>0.0</td>
<td>12192</td>
<td>12192</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
<td>17798</td>
<td>17596</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
</tr>
<tr>
<td>G1023</td>
<td>2</td>
<td>138990</td>
<td>131299</td>
<td>3.5</td>
<td>249164</td>
<td>239530</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102965</td>
<td>102965</td>
<td>0.0</td>
<td>177009</td>
<td>164065</td>
<td>7.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>102965</td>
<td>102965</td>
<td>0.0</td>
<td>145417</td>
<td>145417</td>
<td>0.0</td>
</tr>
<tr>
<td>P22810</td>
<td>2</td>
<td>470011</td>
<td>459381</td>
<td>2.3</td>
<td>932380</td>
<td>901508</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>341858</td>
<td>332356</td>
<td>2.8</td>
<td>623715</td>
<td>604923</td>
<td>4.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>259263</td>
<td>258482</td>
<td>0.3</td>
<td>470011</td>
<td>458526</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 2. Experimental results with power constraint ($P_{\text{max}} = 50\%$ or $30\%$).

<table>
<thead>
<tr>
<th>SoC name</th>
<th>n</th>
<th>$P_{\text{max}}=50%$</th>
<th>$P_{\text{max}}=30%$</th>
<th>$P_{\text{max}}=50%$</th>
<th>$P_{\text{max}}=30%$</th>
<th>$P_{\text{max}}=50%$</th>
<th>$P_{\text{max}}=30%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D695</td>
<td>2</td>
<td>18869</td>
<td>11978</td>
<td>37</td>
<td>25024</td>
<td>20774</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>16771</td>
<td>10705</td>
<td>36</td>
<td>24039</td>
<td>20774</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15156</td>
<td>10705</td>
<td>29</td>
<td>24039</td>
<td>20774</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25062</td>
<td>14794</td>
<td>41</td>
<td>25062</td>
<td>17110</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>17925</td>
<td>14794</td>
<td>18</td>
<td>22847</td>
<td>17110</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16489</td>
<td>14794</td>
<td>10</td>
<td>22847</td>
<td>17110</td>
<td>25</td>
</tr>
<tr>
<td>P22810</td>
<td>2</td>
<td>271384</td>
<td>34081</td>
<td>51</td>
<td>271384</td>
<td>175946</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>181169</td>
<td>19852</td>
<td>34</td>
<td>204724</td>
<td>175946</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>156810</td>
<td>11756</td>
<td>25</td>
<td>205506</td>
<td>157946</td>
<td>14</td>
</tr>
<tr>
<td>P93791</td>
<td>2</td>
<td>611991</td>
<td>462412</td>
<td>24</td>
<td>620585</td>
<td>540781</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>482352</td>
<td>348532</td>
<td>28</td>
<td>572327</td>
<td>512330</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>389935</td>
<td>292871</td>
<td>25</td>
<td>537237</td>
<td>477344</td>
<td>11</td>
</tr>
</tbody>
</table>

ETRI Journal, Volume 30, Number 1, February 2008  
Jin-Ho Ahn et al.  
137
### Table 3. Comparison of ACO-based and SA-based test scheduling in single test clock mode.

<table>
<thead>
<tr>
<th>SoC name</th>
<th>n</th>
<th>$W = 32$</th>
<th></th>
<th>$W = 16$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SA [29]</td>
<td>ACO’s improvement over SAbest (%)</td>
<td>SA [29]</td>
<td>ACO’s improvement over SAbest (%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Converged</td>
<td>Best</td>
<td></td>
<td>Converged</td>
</tr>
<tr>
<td>D695</td>
<td>2</td>
<td>16075</td>
<td>12050</td>
<td>11978</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10069</td>
<td>9869</td>
<td>9869</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9869</td>
<td>9869</td>
<td>9869</td>
<td>0.0</td>
</tr>
<tr>
<td>G1023</td>
<td>2</td>
<td>14794</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14794</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>14794</td>
<td>14794</td>
<td>14794</td>
<td>0.0</td>
</tr>
<tr>
<td>P22810</td>
<td>2</td>
<td>173705</td>
<td>138990</td>
<td>131299</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102965</td>
<td>102965</td>
<td>102965</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>102965</td>
<td>102965</td>
<td>102965</td>
<td>0.0</td>
</tr>
<tr>
<td>P93791</td>
<td>2</td>
<td>527587</td>
<td>470011</td>
<td>459381</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>363131</td>
<td>341858</td>
<td>332356</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>294200</td>
<td>259263</td>
<td>258482</td>
<td>0.3</td>
</tr>
</tbody>
</table>

1. Build a test schedule by the rectangle packing method proposed in [3].
2. Transform the above result into a sequence pair by the method presented in [29], and set it to the initial solution ($S_{init}$).
3. Set the initial temperature ($T_{init}$), the number of iteration ($N_{iter}$), stopping temperature ($T_{stop}$), and temperature reduction ratio ($K$).
4. Let the current solution be $S_{cur} = S_{init}$ and the current temperature be $T = T_{init}$.
5. While ($T$ is higher than $T_{stop}$) {
   1. For i from 1 to $N_{iter}$ loop {
      2. Generate a neighboring solution ($S_n$) and the test time by $S_n$, $T(S_n)$.
      3. Compute the difference between $T(S_n)$ and $T(S_{cur})$, $D(S_n - S_{cur})$.
      4. If $D(S_n - S_{cur}) < 0$ then $S_{cur} = S_n$.
      Else {
         1. Generate one random value $q$, $0 < q < 1$;
         2. If $q < \exp(-D(S_n - S_{cur})/T)$ then $S_{cur} = S_n$.
      }
   }
   6. Set new temperature $T = K\cdot T$.
   }

The basic scheduling procedure using SA is shown in Fig. 8. The neighboring solution $S_n$ was produced by random moves over the sequence pair. Four types of movements were implemented, and three of them are the same as those implemented in [29]:

- Choose a bin and exchange the positions of two rectangles in the plus sequence ($\Gamma^+$)
- Choose a bin and exchange the positions of two rectangles in both the plus sequence ($\Gamma^+$) and the minus sequence ($\Gamma^-$)
- Choose a rectangle and change the width and height of the rectangle.
- Choose two bins, select a rectangle in each bin, and swap them.

The occurrence ratios of all movements were equally distributed. Simulation results for the single test clock mode are shown in Table 3. As in [29], the SA parameters for Table 3 were set to $T_{init}=4000$, $N_{iter}=400\cdot N_{core}$ (the number of cores), $T_{stop}=0.001$, and $K=0.98$. After monitoring the temporary solutions generated by the SA scheduler, we found the best value and the converged value were often not the same. The best value is the local optimal value the scheduler generates, and the converged value is the last value the scheduler chooses. However, even considering the best values, the ACO-based method gave better results in all cases. Furthermore, the computation times of the SA-based approach were up to 10 times longer than those of ACO.

As previously mentioned, we tuned the ACO parameters according to recommendations from previous papers and experiments using benchmark circuits. Now, we will look into the influence of ACO parameters, $N_{colony}$, $\gamma$, $\rho$, and $\beta$, with respect to the test time. To understand the effect of each parameter, only the monitored parameter was varied, while the other parameters were fixed to default values, chosen heuristically. Default values were given as
Experimental results shown in Fig. 9 demonstrate that most parameters barely influence the overall test time, except in the initial stages. The variance is between 0.2 and 0.3%, at most, in the steady state. After analyzing the results carefully, we found that, with rectangle-packing methods, multiple combinations of rectangles produce the best scheduling result. Scheduling results are affected by the combination of rectangles, as well as their packing order. In other words, as $S_{gb}$ was generated in several ways, ACO parameters had less effect on the test scheduling results than would be expected. Thus, we have only to set the parameters to avoid transient ranges.

VI. Conclusion

In this paper, we proposed a new NoC-based SoC test scheduling method with an ACO solution. The proposed algorithm formulates the test scheduling problem as a rectangle bin packing problem and uses ACO to search more solution space to increase the probability of finding optimal solutions in shorter periods. Experimental results obtained using ITC’02 benchmark circuits demonstrate that the proposed algorithm is very efficient and feasible for various conditions. As many testing-related questions of NoC-based systems are still open, the future of NoC-based systems is unclear. However, since studies for testing NoC-based systems are just emerging, it is expected that the proposed test method will become a promising and competitive approach, stimulating further related research.

References


Jin-Ho Ahn received the BS and MS degrees in electrical engineering from Yonsei University, Seoul, Rep. of Korea, in 1995 and 1997, respectively. From 1997 to 2002, he worked as a research engineer at LG Electronics. He received the PhD degree in electrical and electronic engineering from Yonsei University in 2006, and joined the Department of Electronic engineering at Hoseo University, Asan, Chungnam, Rep. of Korea, as a lecturer in 2007. His research interests focus on design and testing of NoC-based SoCs.

Sungho Kang received the BS degree from Seoul National University, Seoul, Rep. of Korea, and the MS and PhD degrees in electrical and computer engineering from the University of Texas at Austin in 1992. He was a research scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc. and a senior staff engineer with the Semiconductor Systems Design Technology, Motorola Inc. Since 1994, he has been a professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea. His main research interests include VLSI design and testing, design for testability, BIST, defect diagnosis, and design for manufacturability.