Core-Based Testing of Multiprocessor System-on-Chips Utilizing Hierarchical Functional Buses

Fawnizu Azmadi Hussin[1], Tomokazu Yoneda[1], Alex Orailoglu[2], and Hideo Fujiwara[1]

[1]Graduate School of Information Science
Nara Institute of Science and Technology
Kansai Science City, 630-0192, Japan
{fawniz-h, yoneda, fujiwara}@is.naist.jp

[2]Computer Science and Engineering Department
University of California, San Diego
La Jolla, CA 92093
alex@cs.ucsd.edu

Abstract—An integrated test scheduling methodology for multiprocessor System-on-Chips (SOC) utilizing the functional buses for test data delivery is described. The proposed methodology handles both flat bus single processor SOC and hierarchical bus multiprocessor SOC. It is based on a resource graph manipulation and a packet-based *packet set scheduling methodology*. The resource graph is decomposed into a set of test configuration graphs, which are then used to determine the optimum test configurations and test delivery schedule under a given power constraint. In order to validate the effectiveness of the proposed methodology, a number of experiments are run on several modified benchmark circuits. The results clearly underscore the advantages of the proposed methodology.

I. Introduction

SOC design with multiple embedded processors is getting momentum due to the increasingly high demand on processing power. Due to the high bandwidth communication between the multiple embedded processors and the SOC cores, the use of a flat bus is no longer adequate. These recent advancements present new problems as well as opportunities to the test engineers in the form of multiple embedded processors and a complex network of communication channels between the SOC cores. The abundant resources of a communication network dramatically diminish the benefits of introducing extraneous Test Access Mechanism (TAM) for test purposes. Hence the use of functional on-chip resources for test purposes is more practical and more economical.

Amongst the earliest literature on the utilization of the functional interconnects for core-based testing of SOCs stands out the paper by Papachristou et al. in 1999 [1], in which, an embedded microprocessor is used as the test controller. The test data from an external tester are loaded into the embedded memories through a direct memory access controller (DMA) and delivered to the core under test by the embedded processor through the functional interconnect. The test responses are evaluated by embedded signature analyzers.

A more comprehensive methodology on the functional bus based SOC testing was discussed by Harrod [2]. The paper presented a test application strategy for various types of test requirements for the embedded Intellectual Property (IP) cores. Test access to the Core-Under-Test (CUT) is provided by the Test Interface Controller (TIC), which is part of the AMBA [3] specifications. An external ATE is used to deliver the test vectors through the TIC interface. At each core, a test wrapper is required to isolate the core from the surrounding logic.

Krstic et al. [4] presented a similar test methodology, where the embedded processor first tests itself by executing a set of instructions using a software-based self test (SBST) methodology [5, 6]. Subsequently, the processor tests the bus and the other IP cores. Huang et al. [7] presented a similar core-based test approach for PCI bus based SOCs. The test application is performed by the test support architecture at every core. The test patterns for each core under test are generated using software-based weighted random patterns [8]. Nahvi et al. [9] proposed a TAM architecture based on a packet switching communication network, called NIMA. NIMA provides access to the cores for test data delivery by means of multi-level routers and communication channels similar to the bus-based TAMs.

Larsson et al. [10] proposed a buffer-based test support architecture to enable parallel testing of core-based SOCs. As opposed to the embedded processor-based approach by [1, 4, 7], the test control is performed by an embedded finite state machine based controller which requires additional hardware overhead, proportional to the volume of the test data.

Our group recently [11] proposed a buffer-based test architecture similar to [10] for core-based test application utilizing the embedded processor and the functional bus. The test application time is minimized by optimizing the bus sharing between multiple CUTs using a novel scheduling methodology called *PAcket Set Scheduling (PASS)*. The applicability of the proposed methodology is, however, limited to flat bus and single processor SOC architectures.

In this paper, our *Integrated PAcket Set Scheduling (IPASS)* methodology targeting the hierarchical bus based multiprocessor SOCs is described. Scheduling core tests for a hierarchical bus and multiprocessor SOC involves the tasks of distributing the core tests to multiple processors, and allocating the time slots on the shared functional buses for the delivery of the test data to each CUT. The proposed *power-constrained* and *MultiProcessor PAcket Set Scheduling (MPPASS)* methodologies address the above issues in order to produce an efficient test data delivery schedule.

II. Scope and Problem Formulation

The design of a multiprocessor SOC (MPSOC) can be implemented using a wide range of architectures [12, 13, 14], depending on the exact design specifications. In addition, various test strategies can be adopted for each embedded IP core. In order to develop an effective SOC test scheduling methodology, a constricted scope of SOC architecture and test requirements are considered in this paper.

The proposed test methodology is applicable to the multi-processor SOCs with hierarchical bus architecture like AMBA [3], CoreConnect, and some specialized MPSOC buses [12]. Given (i) a bus-based MPSOC with N_P embedded processors, N_B bridged buses, and N_M IP cores with the corresponding test power information, and (ii) the test requirements for all of the IP cores, find, for each processor, the optimum test delivery schedule (under the constraints of maximum power dissipation, P_{max}, and total buffer sizes, B_{max}) which utilizes the hierarchical functional buses under the assumptions that:

- Processors are not the target of testing; all the embedded processors are assumed tested by the SBST methodology [5] prior to the start of core testing.
- Each processor has a fault-free local memory, tested during the processor testing stage [15].
- The core scan frequency can be less than the maximum scan frequency, f_{max}.
- Deterministic full-scan tests are used for all CUTs.
- Test data are:
  - loaded into the corresponding memory location before core-testing begins, or
  - loaded as they are needed during the test application through DMA [1], or
  - generated by the embedded processors using the deterministic SBST [6].

III. Buffer-Based Test Architecture

In [10], the differences between three types of test access architecture are explained—dedicated TAM, functional bus, and functional bus with buffers similar to Fig. 1, for an n-bit bus and a CUT with m scan chains. The bit-width conversion is achieved by parallel-serial shifting within the buffer, controlled by the test controller [11]. The first n bits of the core's primary inputs (PIs) and primary outputs (POs) are connected to the data bus. The remaining u PIs and v POs are connected to other parts of the SOC.

To isolate the cores during testing, each PI/PO is connected to the bus through a boundary cell, similar to IEEE 1500's [16], which selects either the scan input (dotted line) or the functional input (solid line). The same control signal (T/N) is used for the boundary cells, the buffers, and the multiplexer to switch between test mode and normal mode. Wrapper scan

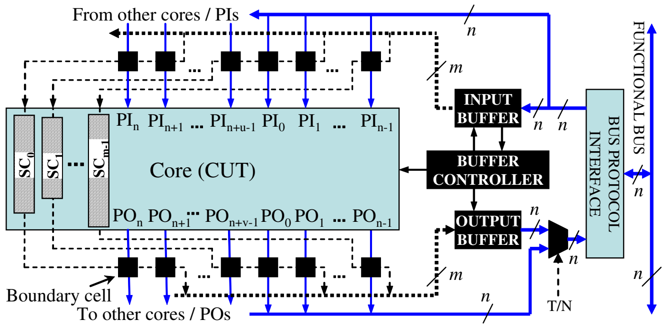


Fig. 1. Core test architecture

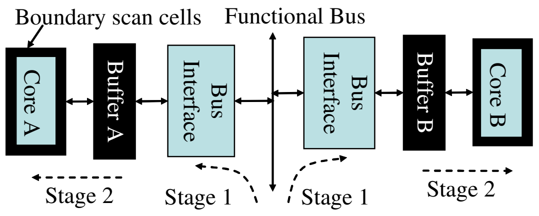


Fig. 2. Buffer-based test architecture

chains with equal scan-in/scan-out depths are formed by cascading PIs and POs to the internal scan chains (Fig. 1). Compared to a TAM-based architecture which utilizes the IEEE 1500 wrapper, the proposed buffer architecture incurs a single additional multiplexer delay overhead on the functional output path.

During the test application, the test data are delivered through the functional bus to the input buffer. Regardless of the data format on the bus, after passing through the functional bus protocol interface, decoded bit-level data is transferred to the input buffer. Test data decoding is handled by the existing functional interface. The dotted line shows the path taken by the test data from the input buffer into the scan chains. At the same time, the test responses are scanned out of the scan chains into the output buffer. The test responses are then read by the processor for analysis.

The introduction of buffers enables the test application to be scheduled concurrently [10, 11]. Figure 2 shows the simplified representation of the buffer architecture for two CUTs. The test data are delivered by an embedded processor to buffers A and B alternately in stage 1 (*data delivery stage*). In stage 2, the test data in the respective buffers are loaded into the scan chains (*test application stage*) of cores A and B simultaneously. Figure 3 shows the timing diagram of the data delivery on the functional bus (labeled *Bus*) and the test application at each core (labeled *Core A* and *Core B*).

IV. Test Scheduling Methodology

In this section, the scheduling methodology which consists of the resource graph manipulation, the power-constrained scheduling, and the multiprocessor packet set scheduling is explained. The packet set scheduling methodology [11], which is targeted for flat bus and single processor architectures, is also described in order to complete the flow of the algorithm description.

A. Test Configuration Graph (TCG)

Figure 4 shows a resource graph [17] for the MPSOC in Fig. 5 consisting of two processors P_0 and P_1, two buses b_0 and b_1 (interfaced by a bridge B), and $(j+k)$ cores. Access to the buses is regulated by the arbiters A_i. The resource graph

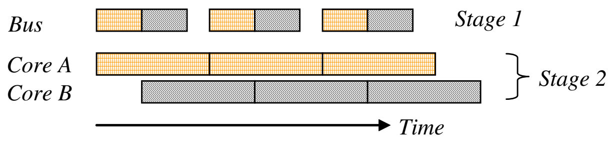


Fig. 3. Concurrent test application

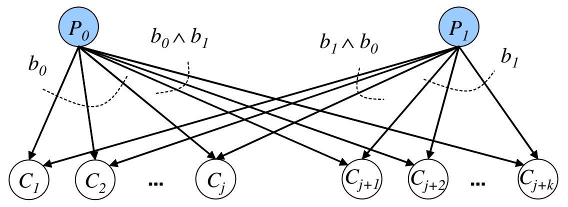


Fig. 4. MPSOC resource graph

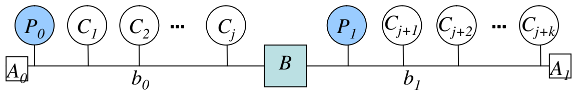


Fig. 5. Hierarchical bus based MPSOC

provides information about processor-core connectivity. For example, cores C_1 to C_j are connected to bus b_0 and can be reached by P_0 directly. The cores can also be reached by P_1 through a bus hierarchy $b_1 \wedge b_0$. The AND operator ($\wedge$) indicates that data delivery passes through a bridge.

Definition: A test configuration graph (TCG) is a processor [*test source*] - core [*test sink*] pair that specifies the delivery path on the functional bus(es) for test data delivery.

Figure 6 shows five types of basic TCGs (top half) and their corresponding bus architecture (bottom half). For each TCG, P_q is the test source, and C_i and C_j are the CUTs. Other secondary TCGs can be formed by expanding and merging the primary TCGs. Type IV and Type V are broadcast TCGs when C_i and C_j are identical cores with identical test requirements. For example, the resource graph in Fig. 4 can be decomposed into $2 \times (j + k)$ TCGs of Type I and Type II.

B. Power-Constrained Scheduling

Power constrained scheduling for core tests are performed using various methods, which include preemptive [17], 2D [18], and 3D bin packing [19] algorithms. The resulting test schedule is typically similar to Fig. 7(a). When we consider packet-based test delivery utilizing the functional bus (similar to Fig. 3), five distinct delivery patterns are required for every combination of cores tested concurrently for the schedule in Fig. 7(a). This is because at time t_0, C_1 and C_2 are scheduled. At t_1, the test application of C_2 is completed and C_3 is started. The schedule change also takes place at t_2 and t_3. The more frequent the schedule changes, the higher the complexity of the test program. To avoid the additional complexity, the cores are

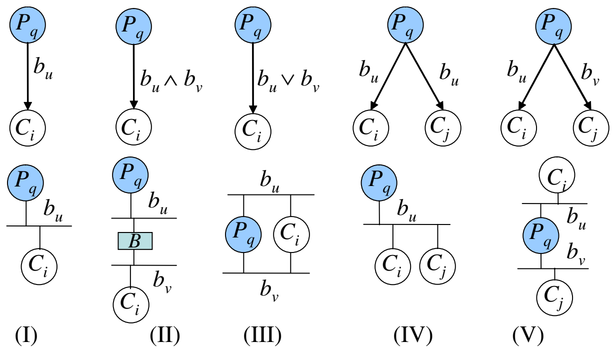


Fig. 6. Types of test resource graphs (TCG)

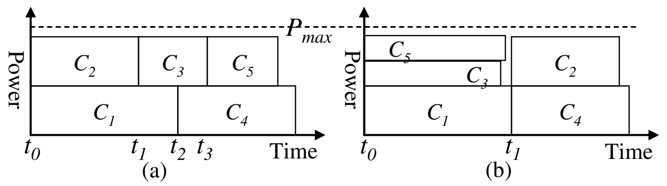


Fig. 7. Power-constrained test group assignment of CUTs

grouped into non-overlapping test groups as in Fig. 7(b). The scan frequencies for C_3 and C_5 are halved, in order to match the test application time (TAT) of core C_1 [11].

Figure 8 shows the power constrained scheduling, in which the test groups are formed. In step (1) of *FormTestGroups*, redundant TCGs are eliminated. A TCG of core C_i is said to be *redundant* if the set of buses in the TCG is a complete superset of another TCG of C_i. For example, in Fig. 6, the Type II TCG is a redundant TCG (relative to Type I) for C_i because the set $\{b_u, b_v\}$ is a complete superset of $\{b_u\}$. The redundant TCGs are eliminated from the list of candidate TCGs of C_i because their delivery costs are always greater and unnecessary.

When forming the test groups, the core with the largest TAT amongst the remaining unscheduled cores is first scheduled to the group and assigned the maximum scan frequency, f_{max}. The TAT (at f_{max}) of the remaining unscheduled cores is less than the TAT of the first core in the test group. When scheduling the subsequent cores, the core scan frequencies are reassigned (step 4c) to the highest frequency less than f_{max} such that the TAT of the new group member is less than or equal to the TAT of the first member of the group. The choices of frequency values are constrained by the maximum scan frequency, f_{max}, and the resolution of the frequency divider [11].

In step 4a of *FormTestGroups*, for each TCG of C_i, the maximum resulting bus utilization of all buses in the TCG is determined. The TCG which results in the minimum of the maximum total bus utilization is chosen. Step 4 seeks to evenly distribute the bus utilization. This is achieved by dynamically selecting the best TCG for each core during scheduling based on the current bus utilization by other cores.

```
Function: FormTestGroups
  1. For each $C_i$, eliminate all redundant TCGs
  2. Set the total power, $P_{total}$ = 0 for current group
  3. Amongst unscheduled cores, select a core with maximum TAT
     and not yet attempted for scheduling in current group
  4. For the currently selected $C_i$
     (a) If $C_i$ has multiple TCGs, select the TCG which results
         in min{max{bus utilization of all buses in TCG}}
     (b) If $C_i$ has unique TCG, schedule $C_i$ in current group if
         • at least one of the buses in TCG has minimum utilization, and
         • $P_{total} \leq P_{max}$ after scheduling $C_i$
     (c) If $C_i$ is scheduled,
         • reassign the scan frequency for $C_i$
         • update $P_{total}$ and bus utilization of current group
     (d) If $C_i$ cannot be scheduled,
         • if all cores are attempted in the current group, create a new group and go to step (2)
         • else, go to step (3)
Note: $C_i \equiv C_i$ and all broadcast pairs (if any) sharing a TCG
```

Fig. 8. Assigning cores to test groups

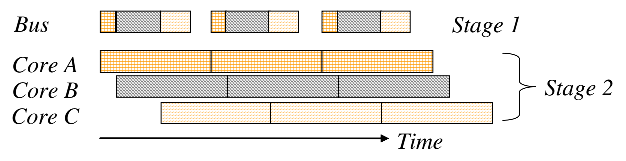


Fig. 9. Variable packet sizes

C. Packet Set Scheduling (PASS) [11]

The main objective of PASS methodology [11] is to find, for a processor, a repetitive delivery sequence and the corresponding packet sizes for each core (for a given maximum total buffer size) that minimize the test application time. The limitation of such a cyclic delivery sequence is that the job of all members in the repetitive group must be completed before the next (identical) job can be started. In other words, if there is one member of the group who is late in completing the current job, all other members stall while waiting for the slowest member to complete, prior to initiating a new group delivery sequence.

In the scan-based testing perspective, the cores with the smaller number of scan chains (N_s) or those tested at lower scan frequency (f_s) take longer time to complete, for the same amount of test data. Therefore, cores with larger $N_s \times f_s$ require a larger test packet (therefore, larger buffer) in order for the test application of each packet of each core to have equal scan time (Stage 2) as shown in Fig. 9. Equal scan time is necessary to avoid stalling.

To reduce the total buffer size required, the packet size for core B can be halved and the delivery sequence changed from A-B-C (Fig. 9) to B-A-B-C (Fig. 10). Both figures show three repetitions of the smallest subset of delivery sequence, called *packet set*. In Fig. 10, cores A and C are said to belong to the *split-1* group because only one packet is delivered in the packet set. Similarly, core B is said to belong to the *split-2* group [11].

If three different split groups are used, the delivery sequence (or packet set schedule, PASS) can be specified as in Fig. 11, for k, $d \times k/r$, and q cores assigned to *split-1*, *split-r*, and *split-$2k$* respectively. Each entry represents a time slot on the bus allocated for the delivery of packet $p_{i,j}^{g}$, where

g = module number (1 to n) from split-i group
i = split group to which module g belongs
j = packet sequence number for module g, and $j \leq i$

D. Optimizing PASS for Hierarchical Bus MPSOC (MPPASS)

The delivery sequence specified by Fig. 11 is optimized for a flat-bus architecture. For a hierarchical bus, the delivery time of some packets may be longer than others. Strictly following the delivery sequence in Fig. 11 may result in stalling the delivery of some packets, as shown in [11]. Due to the bus contention and the store-and-forward operation by the bridges, the completion time of a packet delivery cannot be independently calculated without considering the events on other processors and bridges. Fig. 12 shows the possible contention scenario on bus b_1 between data packets originating from processors on buses b_0 and b_2.

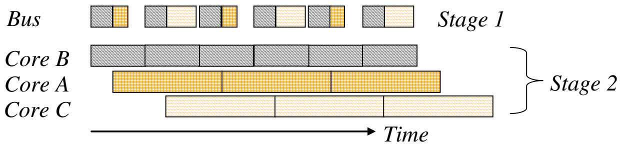


Fig. 10. Improved delivery sequence for packet size minimization

$$
\begin{array}{cccccccc}
p_{2k,1}^{1} & p_{2k,1}^{2} & \cdots & p_{2k,1}^{q} & p_{r,1}^{1} & p_{r,1}^{1+k/r} & \cdots & p_{r,1}^{1+(d-1)k/r} \\
p_{2k,2}^{1} & p_{2k,2}^{2} & \cdots & p_{2k,2}^{q} & p_{1,1}^{1} & & & \\
p_{2k,3}^{1} & p_{2k,3}^{2} & \cdots & p_{2k,3}^{q} & p_{r,1}^{2} & p_{r,1}^{2+k/r} & \cdots & p_{r,1}^{2+(d-1)k/r} \\
p_{2k,4}^{1} & p_{2k,4}^{2} & \cdots & p_{2k,4}^{q} & p_{1,1}^{2} & & & \\
\vdots & \vdots & & \vdots & \vdots & & & \\
p_{2k,2k-1}^{1} & p_{2k,2k-1}^{2} & \cdots & p_{2k,2k-1}^{q} & p_{r,r}^{k/r} & p_{r,r}^{k/r+k/r} & \cdots & p_{r,r}^{k/r+(d-1)k/r} \\
p_{2k,2k}^{1} & p_{2k,2k}^{2} & \cdots & p_{2k,2k}^{q} & p_{1,1}^{k} & & &
\end{array}
$$

Fig. 11. Packet delivery sequence for three split groups [11]

The delivery sequence of Fig. 11 only requires that, for each time slot, the packet belong to a specific split group, not a specific module. For example, the delivery sequence of B-A-B-C (Fig. 10) can as well be changed to B-C-B-A without affecting the scan in operation at each core.

The flowchart in Fig. 13 shows the process of determining the best PASS (i.e. B-A-B-C or B-C-B-A) for a hierarchical MPSOC by simulating all delivery sequences that do not violate the group delivery sequence in Fig. 11 as specified by the index i in each $p_{i,j}^{g}$. The PASS can be formed by randomly permuting the delivery ordering within each split group. For every permutation of PASS, the test application is simulated (explained in Sect. V). If the new PASS returns a smaller TAT, it is recorded as the current best PASS.

In order to minimize the simulation time, each PASS is simulated only for w packet sets (i.e. w repetitions of B-A-B-C). Furthermore, if $N_{PASS,max}$ consecutive PASS is tried without any improvement, the simulation is stopped, and the current best PASS is returned. The value of w is chosen under the assumption that the interaction between the data packets is cyclic after several repetitions of packet set delivery. The values for w and $N_{PASS,max}$ of 5 and 10, respectively, were used for the results presented in Sect. VI.

V. Simulating the Test Application

In order to determine the test application time for each PASS, an event-driven MPSOC simulation environment (Fig. 14) was implemented in C++ under the following constraints:

- The delivery sequence by each processor follows the PASS repeatedly. A new test packet for core C_i is delivered by the processor only when the test response of the previous packet of C_i is received.
- Bus arbitration follows the functional arbitration scheme.
- Every event requires a separate bus arbitration.

In Fig. 14, the middle (shaded) blocks are the main simulation engine, which keeps track of the simulation time and pro-

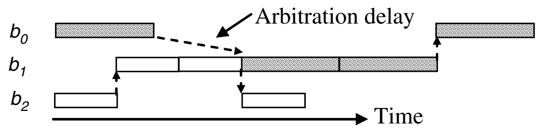
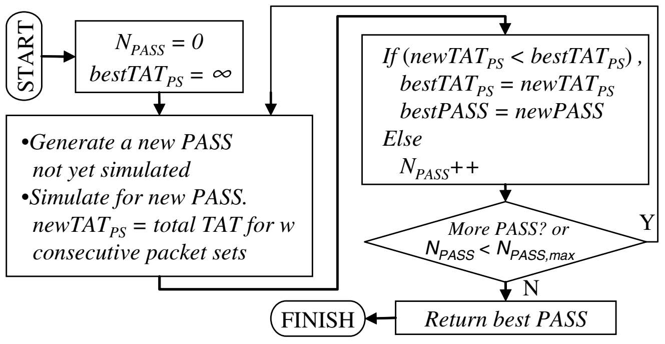


Fig. 12. Bus contention in an MPSOC



START
$N_{PASS} = 0$
$bestTAT_{PS} = \infty$
•Generate a new PASS not yet simulated
•Simulate for new PASS. $newTAT_{PS}$ = total TAT for w consecutive packet sets
If ($newTAT_{PS} < bestTAT_{PS}$), $bestTAT_{PS} = newTAT_{PS}$ bestPASS = newPASS Else N_{PASS}++
More PASS? or $N_{PASS} < N_{PASS,max}$
Y
N
Return best PASS
FINISH

Fig. 13. Optimizing the packet delivery sequence (MPPASS)

cesses the events based on their time stamp. The left and right sections show the steps for handling the processor-initiated events and bridge-initiated events respectively. For both types of events, the packets are forwarded to either the next bridge (vector or response packets), directly to the core (vector packets), or back to the processor (response packet). The next destination of each packet is determined by the TCG chosen for the packet owner (the core/CUT).

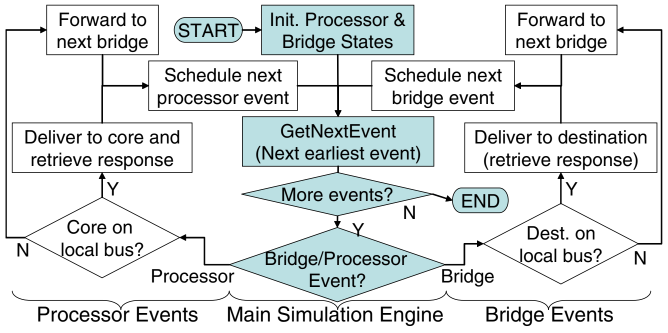
VI. Experimental Results

In order to evaluate the effectiveness of the proposed methodology, we have conducted experiments on several modified ITC'02 benchmark circuits [20]. The power dissipation information for the selected circuit is obtained from [18, 19][1]. We have additionally added the functional bus information to the selected circuits as follows:

- A bus b_0 is added to connect all the cores. A processor core is assumed attached to b_0. This modified circuit is named *NXh1*, where *NX* is the original circuit name.
- For circuits which have level-2 hierarchy, a single shared bus b_0 is added to connect all the level-1 cores. The definition for *level* is defined in the benchmark suite [20]. Additionally, a local bus b_i is added within each hierarchical level-1 core. Bus b_i is interfaced to bus b_0 through a bridge. This modified circuit is named *NXh2*.

When comparing the test application time with a TAM-based approach, the processor cores are assumed to be tested using software-based self-test prior to the test application of other

[1]The unit for maximum power, P_{max}, is based on an estimate given by [18]. We utilized the same power values in order to offer a comparison with TAM-based scheduling approaches.



Forward to next bridge
START
Init. Processor & Bridge States
Forward to next bridge
Schedule next processor event
Schedule next bridge event
Deliver to core and retrieve response
GetNextEvent (Next earliest event)
Deliver to destination (retrieve response)
Y
More events?
N
END
Y
Y
Core on local bus?
N
Dest. on local bus?
N
Processor
Bridge/Processor Event?
Bridge
Processor Events
Main Simulation Engine
Bridge Events

Fig. 14. MPSOC event-driven simulation flowchart

TABLE I
Flat bus SOC (p93791h1)

p93791h1	BW = 32			BW = 64		
flat-bus	*fb = fs*		*fb=2*fs*	*fb = fs*		*fb=2*fs*
Pmax	Pouget	IPASS	IPASS	Pouget	IPASS	IPASS
10,000	18.28	18.44	9.04	11.17	8.94	5.34
15,000	18.28	17.34	8.85	10.15	8.85	4.70
20,000	18.28	17.35	8.89	9.58	8.93	4.59
25,000	18.28	17.63	9.07	9.65	9.05	4.75
30,000	18.28	17.78	9.08	9.45	9.07	4.67

TABLE II
Hierarchical bus MPSOC (p93791h2)

p93791h2	BW = 32					BW = 64				
hierarchy	*fb = fs*			*fb = 2*fs*		*fb = fs*			*fb = 2*fs*	
	Pouget	IPASS		IPASS		Pouget	IPASS		IPASS	
Pmax		P@b0	P@All	P@b0	P@All		P@b0	P@All	P@b0	P@All
10,000	18.28	26.97	15.51	13.47	7.83	11.17	13.47	7.83	7.13	5.69
15,000	18.28	20.15	9.51	10.07	4.83	10.15	10.07	4.83	5.05	3.79
20,000	18.28	20.39	7.37	10.20	4.27	9.58	10.21	4.23	5.11	3.51
25,000	18.28	18.95	5.31	9.47	3.24	9.65	9.50	3.20	4.72	2.82
30,000	18.28	18.89	5.31	9.44	3.24	9.45	9.44	3.20	4.78	2.82

non-processor cores. Therefore, the TATs for the processor cores are assumed to be equal for both TAM-based and our IPASS approach. The TAT for the processors is therefore not included in the results presented in this section. IPASS dynamically chooses the PASS algorithm [11] for scheduling, when the target system is a single processor SOC with a flat bus architecture. Alternatively, it will use MPPASS algorithm for multiprocessor SOCs with hierarchical buses.

The following tables show the TATs (in millisecond) when the maximum scan frequency is set to f_{max} = 100 MHz. TAM-based approaches make use of the f_{max} for all cores. They also disregard the functional buses; their TATs are the same for an SOC with either a flat bus or a hierarchical bus implementation.

Table I shows the TAT for a single processor SOC with a flat functional bus and bus widths (BW) of 32 and 64 bits. The test applications are simulated for several values of P_{max}. In the table, *fb=fs* represents the circuit configurations when the maximum scan frequency (f_s) and the bus frequency (f_b) are set to 100 MHz. The second and fifth columns show the TATs for a TAM-based approach [18]. The third and sixth columns show the results of our IPASS approach. The TATs of our approach and of the TAM-based approach are comparable when the bus frequency is the same as the scan frequency (i.e. same as TAM approach). However, much shorter TATs are achieved when we allow the bus frequency to be higher than the scan frequency, which is not possible for the TAM-based approach, without adding similar buffers, in addition to the TAMs. The fourth and seventh columns (*fb=2*fs*) illustrate this advantage.

In Table II, *P@b0* represents a hierarchical bus SOC with one processor. The elevated TATs are due to the hierarchy overhead in the delivery time of each test packet. The effect of hierarchical buses is evident when comparing p93791h1 and p93791h2 since in both cases, there is one test processor on the level-0 bus (b_0). In order to show the benefit of using multiple processors, we analyzed the best case scenario, where there is a processor in every isolated bus region (*P@All*). The bus hierarchy allows simultaneous delivery of the test data and reduces contention on the bus access. For typical MPSOC configurations, the TAT is expected to be between the best case (*P@All*) and the worst case (*P@b0*). Simulation results for

TABLE III
FLAT BUS SOC (P22810H1)

p22810h1	BW = 32			BW = 64		
flat-bus	*fs = fb*		*fb=2*fs*	*fs = fb*		*fb=2*fs*
Pmax	Pouget	IPASS	IPASS	Pouget	IPASS	IPASS
3,000	4.83	4.34	3.06	3.09	3.06	2.93
4,000	4.80	4.34	2.93	3.24	2.94	2.65
5,000	4.72	4.53	2.74	3.22	2.74	2.33
6,000	4.76	4.67	2.46	2.50	2.49	1.88
10,000	4.73	4.32	2.21	2.36	2.20	1.36

TABLE IV
HIERARCHICAL BUS MPSOC (P22810H2)

p22810h2	BW = 32					BW = 64				
hierarchy	*fs = fb*			*fb = 2*fs*		*fs = fb*			*fb = 2*fs*	
	Pouget	IPASS		IPASS		Pouget	IPASS		IPASS	
Pmax		P@b0	P@All	P@b0	P@All		P@b0	P@All	P@b0	P@All
3,000	4.83	4.89	4.23	3.79	3.66	3.09	3.59	3.72	3.54	3.72
4,000	4.80	4.61	3.06	2.65	2.67	3.24	3.16	2.67	3.05	2.67
5,000	4.72	4.81	2.88	2.47	1.87	3.22	2.48	1.87	1.93	1.87
6,000	4.76	4.65	2.96	2.35	1.87	2.50	2.35	1.87	1.68	1.82
10,000	4.73	4.69	2.70	2.30	1.37	2.36	2.37	1.37	1.30	1.33

some randomly assigned number and locations of processors demonstrate this expected trend.

Table III and Table IV show the TATs for flat-bus p22810h1 and hierarchical bus p22810h2 SOC, respectively. Similar trends are observed for this circuit. However, smaller variations are observed between *P@b0* and *P@All* because p22810h2 has only three bus regions, as compared to eight bus regions for p93791h2. Correspondingly small variations are also observed between the flat and hierarchical bus SOCs for the same reason.

The area overhead of the proposed buffer-based test architecture is estimated in terms of the number of flip-flops for the buffers. The overhead on the controller and the boundary scan cells are comparable to the IEEE 1500 wrapper architecture [11]; therefore, it is not included. For all the circuits in Table I to Table IV, the buffer sizes per core (for BW=32), averaged over all P_{max}, are shown in Table V.

VII. CONCLUSION

We have proposed a test scheduling methodology for core-based testing of SOCs based on the utilization of the functional buses. The proposed method can handle both flat bus single processor architectures and hierarchical bus multiprocessor architectures. The bus hierarchy information is efficiently incorporated into the methodology by representing the resource graph with the *test configuration graphs*—a concept introduced in this paper.

It was shown that the hierarchical bus architecture introduces a delay overhead in the delivery of a test packet, which prolongs the overall test application time. Subsequently, the use of multiple processors embedded within the bus hierarchy annuls the negative effects of the hierarchical bus architecture on the overall test application time. Incorporating the bus hierarchy in the test scheduling algorithm is an important step in promoting the use of the functional bus based test methodology.

ACKNOWLEDGEMENTS

This work was supported in part by 21st Century COE Program (Ubiquitous Networked Media Computing) and in part by Japan Society for the Promotion of Science (JSPS) under Grants-in-Aid for Scientific Research B (No.15300018) and for Young Scientists B (No.18700046).

TABLE V
AVERAGE BUFFER SIZES PER CORE

Circuit	p93791h1	p93791h2	p22810h1	p22810h2
Min	99.20	89.79	106.06	107.65
Max	99.39	98.00	112.00	113.15

REFERENCES

[1] C. A. Papachristou, F. Martin, and M. Nourani, "Microprocessor based testing for core-based system on chip", In Proc. IEEE Design Automation Conference, 1999, pp. 586-591.

[2] P. Harrod, "Testing reusable IP - A case study", In Proc. International Test Conference, 1999, pp. 493-498.

[3] D. Flynn, "AMBA: Enabling reusable on-chip designs", *IEEE Micro*, Vol. 17, No. 4, July/Aug. 1997, pp. 20-27.

[4] A. Krstic, L. Chen, W-C. Lai, K-T. Cheng, and S. Dey, "Embedded software-based self-test for programmable core-based designs", *IEEE Design & Test of Computers*, Vol. 19, Issue 4, Jul/Aug. 2002, pp. 18-27.

[5] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores", *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, Vol. 20, No. 3, March 2001, pp. 369-380.

[6] A. M. Paschalis, D. Gizopoulos, N. Kranitis, M. Psarakis, and Y. Zorian, "Deterministic software-based self-testing of embedded processor cores", In Proc. Design, Automation & Test in Europe, 2001, pp. 92-96.

[7] J-R. Huang, M. K. Iyer, and K-T. Cheng, "A self-test methodology for IP cores in bus-based programmable SOCs", In Proc. IEEE VLSI Test Symposium, 2001, pp. 198-203.

[8] M. K. Iyer and K-T. Cheng, "Software-based weighted random testing for IP cores in bus-based programmable ICs", In Proc. IEEE VLSI Test Symposium, 2002, pp. 139-144.

[9] M. Nahvi and A. Ivanov, "A packet switching communication-based test access mechanism for system chips", In Proc. IEEE European Test Workshop, 2001, pp. 81-86.

[10] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Optimization of a bus-based test data transportation mechanism in system-on-chip", In Proc. Euromicro Conference on Digital Systems Design, 2005, pp. 403-411.

[11] F. A. Hussin, T. Yoneda, A. Orailoglu, and H. Fujiwara, "Power-constrained SOC test schedules through utilization of functional buses", In Proc. IEEE International Conference on Computer Design, 2006, pp. 230-236.

[12] K. K. Ryu, E. Shin, and V. J. Mooney, "A comparison of five different multiprocessor SOC bus architectures", In Proc. Digital Systems Design Conference, 2001, pp. 202-209.

[13] W. O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SOC platforms: A component-based design approach", *IEEE Design & Test of Computers*, Vol. 19, Issue 6, Nov/Dec. 2002, pp. 52-63.

[14] V. Salapura, C. J. Georgiou, and I. Nair, "An efficient system-on-a-chip design methodology for networking applications", In Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2004, pp. 212-219.

[15] R. Rajsuman, "Design and test of large embedded memories: An overview", *IEEE Design & Test of Computers*, Vol. 18, Issue 3, May/June 2001, pp. 16-27.

[16] E. J. Marinissen, R. Kapur, M. Lousberg, T. McLaurin, M. Ricchetti, and Y. Zorian, "On IEEE P1500 standard for embedded core test", *Journal of Electronic Testing: Theory and Applications*, Aug. 2002, pp. 365-383.

[17] E. Larsson and H. Fujiwara, "System-on-chip test scheduling with reconfigurable core wrappers", *IEEE Trans. on VLSI Systems*, Vol. 14, No. 3, March 2006, pp. 305-309.

[18] J. Pouget, E. Larsson, and Z. Peng, "Multiple-constraint driven system-on-chip test time optimization", *Journal of Electronic Testing: Theory and Applications*, Vol. 21, 2005, pp. 599-611.

[19] Y. Huang, S. M. Reddy, W-T. Cheng, P. Reuter, N. Mukherjee, C-C. Tsai, O. Samman, and Y. Zaidan, "Optimal core wrapper width selection and SOC test scheduling based on 3-D bin packing algorithm", In Proc. International Test Conference, 2002, pp. 74-82.

[20] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs", In Proc. International Test Conference, 2002, pp. 519-528.