Time Domain Multiplexed TAM: Implementation and Comparison

Zahra sadat Ebadi and Andre Ivanov
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
{zahra, ivanov}@ece.ubc.ca

Abstract

One of the difficult problems which core-based system-on-chip (SoC) designs face is test access. For testing the cores in a SoC, a special mechanism is required, since they are not directly accessible via chip inputs and outputs. In this paper we introduce a novel Test Access Mechanism (TAM) based on time domain multiplexing (TDM-TAM). This TAM is P1500 compatible and uses a P1500 wrapper. The TAM characteristics are its flexibility, scalability, and reconfigurability. The proposed TAM is compared with two other approaches: a serial threading approach analogous to the IEEE1149.1 standard (Serial TAM)[7]and a packet-switching test network (NIMA)[9]. A network-processing engine SoC is used as a platform to compare the different TAMs [6]. Results show that in most cases, TDM is the most effective TAM in both test time and overhead area.

Keywords: *SoC testing, Embedded core testing, Test Access Mechanism (TAM), Time domain multiplexed TAM, Optimal test time.*

1 Introduction

Embedded cores are now being used in large SoC designs. They facilitate design and lead to shorter product development cycles. However, the manufacturing tests and debug of SoCs are major challenges. Since cores in an SoC are not directly accessible via chip inputs and outputs, special access mechanisms are required to test them at the system level. Zorian et al. proposed a generic conceptual test access architecture for embedded cores, with the following components: Source, Sink, and Test Access Mechanism. The test access mechanism (TAM) is used to deliver test stimuli from the source (which generates test stimuli) to cores and also to deliver responses from cores to the sink (which evaluates test responses).

There are several proposed TAM architectures in the literature. These architectures can be classified in four categories: 1) multiplexing, 2) serial connection, 3) indirect access and 4) bus-based connection.

In the first category, multiplexing is used to access the cores. The simplest method in this category directly multiplexes the test pins to the primary inputs/outputs (I/O). Another method modifies the cores such that each core has a transparent mode for testing [4]. There are several problems with the multiplexing TAM methods, such as limited scope of use for future complex SoCs, large overhead area, long test time, and non-scalability of the architecture.

TAMs in the serial connection category [10][7] use the established IEEE 1149.1 standard. For a few cores on an SoC, it may be possible to spend time transporting the test vectors serially to the cores. However, as the number and complexity of the cores increases, a serial solution based on the IEEE 1149.1 standard or its variants will prove too costly in terms of test time.

A Networked Indirect and Modular Architecture (NIMA) for TAM was proposed in [9], where emphasis is placed on modularity, generality, and configurability of the architecture to exploit the advantages offered by the reuse paradigm. A number of different variations of the bus-based connection schemes for TAM have been proposed. [8] and [11] are examples of this type of TAM connection. In terms of trading-off increased overhead area for reduced test access time, bus-based architectures are the most efficient TAM schemes suggested to date.

Here we present a novel bus-based TAM which not only has all the advantages of common bus-based TAMs like scalability, efficiency in time and area, but also it is flexible, reconfigurable and handles cores with BIST efficiently and needs less global routing than common bus-based. Using time domain multiplexing, our TAM uses test buses efficiently, so testing time decreases. Also using a method of dynamic masking gives flexibility to the tester to change the test scheduling and change the test strategy after fabrication.The proposed TAM is compared with two other TAM architectures for SoC design.

2 Time Domain Multiplexed TAM

TDM-TAM is a bus-based TAM that uses logic situated locally at each core to enable or disable the core such that bus contention not occur. This architecture eliminates the necessity of global address lines, which are normally required in a bus-based architecture. In our TDM-TAM, the data to be sent on the TAM bus is divided into frames. Each core is assigned a specific mask enabling the cores to extract the appropriate data bits from the frames.

For example in Fig. 1, a frame is assumed to consist of four bits. Core **A** uses the first bit of each frame, core **B** uses bits 2 and 4, while core **C** uses bit 3. For this specific configuration, assume that we wish to send the data “11” to core **A**, “00” to core **B** and don’t care bits “XX” to core **C**. The two required frames to be sent on the TAM bus in

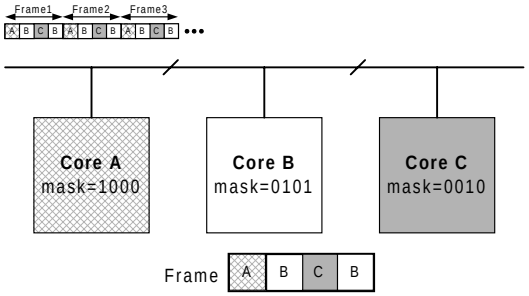

Figure 1. Example illustrating the Time Domain Multiplexing (TDM) TAM concept.

this case must therefore be "10X0" followed by "1XXX". Optimal frame and mask assignment is obviously critical for the efficiency of the scheme. This is addressed later in this paper.

In our case, to implement the TDM-TAM, a P1500 wrapper (Fig. 2) is used to wrap each core.The P1500 wrapper connects to one mandatory one-bit wide TAM, i.e., the Wrapper Serial Input/Output (*WSI/WSO*), and zero or more scalable-width TAMs (*TAM-in/TAM-out*) (TAM-in and TAM-out need not to be the same width).

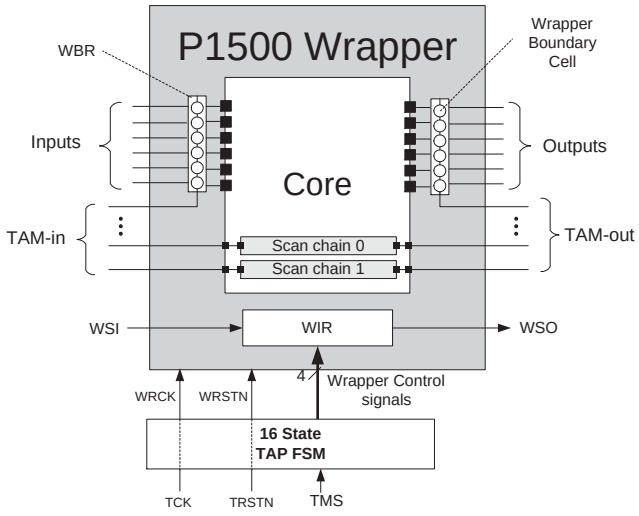

Figure 2. Conceptual view of the IEEE P1500 wrapper and TAP controller.

In the standard 1149.1 TAP controller, a 16-bit finite state machine generates the wrapper control signals from the serial TMS bit stream. For our TDM-TAM, we transformed the original 1149.1 TAP controller into a TDM-TAP controller (Fig. 3). This controller includes the original 1149.1 FSM as well as some minimal extra logic, referred to as a **Time Domain Multiplexer (TDM) Block**, which creates the *Enable* signal for the core according to a preassigned mask.

Here, we assume that the masks are assigned, before layout and fabrication. In this case, the mask for each core is said to be local. This means the mask lines for each core are hard-wired ground or Vdd. However, in Sec. 2.3, we show that it is useful to have the ability to change a mask after fabrication. We refer to this feature as Dynamic masking.

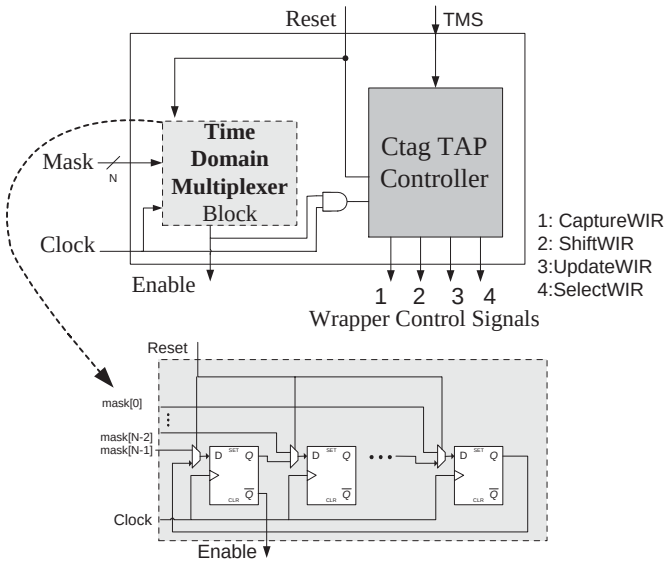

Figure 3. Block diagram for TDM-TAM TAP controller.

The **TDM** block operates as follows. When the *Reset* signal goes low, the mask is loaded into the flip flop chain. When *Reset* is high, the mask is shifted by one bit on every falling edge of the clock. Using the falling edge allows the *Enable* signal to be stable before the rising edge of the clock thus avoiding glitching when used to gate the clock. The *Enable* signal generated for each respective core is used to enable three different components. When a core's *Enable* is active (high), the TMS control signal is read by the core's TAP controller, the core's P1500 wrapper and the core is clocked allowing it to read data from the *WSI* and *TAM-in.* Moreover, the tri-state buffer is enabled, thereby allowing the core to write to the *WSO* and *TAM-out.* When *Enable* is deactivated (low) all these components are deactivated, thus allowing other cores on the same TAM bus to use the bus. Cores tested by a full BIST scheme may not be disabled such that their test be executed as quickly as possible, though their TAP controller, P1500, and tri-state buffer may still be deactivated. Test power considerations may however require that BISTed cores be disabled at times as well. Fig.4 illustrates what we refer to as a *single branch* TDM-TAM with multiple cores, where a branch includes the one-bit wide TMS, a one or more bit-wide *TAM-in* bus (to accommodate at least the *WSI* signal) and a one or more bit-wide *TAM-out* bus. Hence, the minimum data line width of a branch is three. Each branch also includes a global *Clock* and *Reset* signals. All the cores connected to a branch have a local TDM-TAP controller and associated logic, and a tri-state buffer, and all the cores share the *TMS, Clock, Reset, TAM-in,* and *TAM-out* lines.

Different SoCs can have different number of TDM-TAM branches. The case where all the cores are connected to the same branch constitutes the simplest, *single-branch* case illustrated in Fig. 4. The other extreme is to have as many branches as there are cores on the SoC, with each core con-

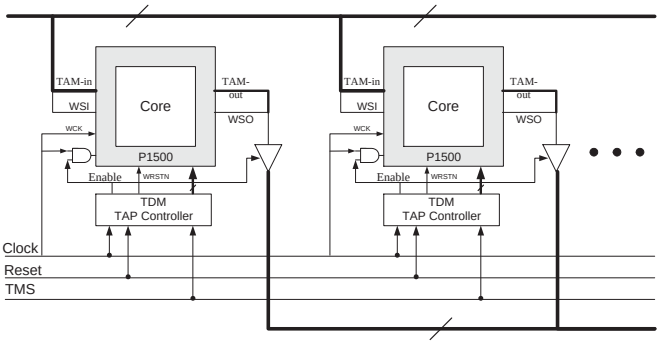

Figure 4. Single Branch TDM-TAM.

nected to its own private branch.

2.1 Timing Model

In this section, a model for test time for the TDM-TAM is developed. A core's test time t_c (in clock cycles) is determined by the scan-in cycles, s_i, the scan-out cycles, s_o, and the number of test patterns, p. according to the following equation [5].

$$t_c = \{1 + max(s_i, s_o)\} * p + min(s_i, s_o) \quad (1)$$

To refine the test time model, we consider the time required to send instructions to the wrapper and required to put the wrapper in test mode, as well as the time required for loading/capturing a signature (if required). We assume that the cores are tested using either test patterns provided externally via the TAM, or using patterns generated and evaluated within the core, i.e., using a form of built-in self test (BIST). For BISTed cores, the TAM is assumed to be used only to send a form of "start BIST" instruction, and to subsequently communicate a BIST result, e.g., signature. The cores that are not BISTed require not only test instructions but test pattern data to also be sent via the TAM. Based on the latter observations, we define two parameters for each core: t_I and t_D. t_I is the portion of the core test time during which the core does not need to control or use the TAM, and is therefore independent of the mask assignment, e.g., the time a BISTed core requires for the BIST test patterns to be generated, applied and evaluated. t_D is the part of a core's test time that requires the control/use of the TAM and which is therefore dependent on the mask assignment, e.g., the time required to send a specific test instruction to a core.

Let the SoC design consist of N_C cores and N_B branches, and assume that core j, $1 \leq j \leq N_C$ is assigned to branch k, $1 \leq k \leq N_B$. The t_I portion of the test time of core j, t_{I_j}, does not depend on the mask assignment. Hence the core-branch connections do not affect this part of the core test time. On the other hand, the t_D portion of the core test times depend on the cores' mask assignments and frame lengths, i.e., t_{D_j} depends on mask assignments and frame lengths. From the example of Fig. 1, it is obvious that as the number of ones in a core's mask increases, the proportion of data from each frame used by the corresponding core increases accordingly. Hence, the test time for core j when assigned to branch k is:

$$T_{jk} = \lceil \frac{t_{D_j}}{\|mask_j\|} \rceil * M_k + t_{I_j} \quad [1] \quad (2)$$

where, M_k is the number of bits in a frame for branch k and $\|mask_j\|$ is the number of ones in the mask assigned to core j. To illustrate, reconsider the example in Fig. 1 and assume that core **B** is not BISTed and that the TAM-independent test time is 20 cycles, while $t_{D_B} = 15$. From the same Fig., M for the bus is 4, $\|mask_A\| = \|mask_C\| = 1$ and $\|mask_B\| = 2$. Core **B** can use two bits of every 4-bit frames. Therefore, testing core **B** requires $\lceil \frac{15}{2} \rceil$ time frames. In turn, this implies a total of 8*4 clock cycles where core **B** is connected to the TAM bus.

Let X_{ij} be a 0-1 variable defined as follows:

$$X_{jk} = \begin{cases} 1, & core\ j\ is\ assigned\ to\ bus\ k \\ 0, & else \end{cases} \quad (3)$$

Using Equation 2, the test time for each core can be determined such that the total test time for testing all the cores assigned to bus k amounts to $T_k = max_{j=1}^{N_C}(X_{jk} * T_{jk})$ since all the cores can be tested concurrently due to *time division multiplexing*. Also, assuming that all the TAM branches can be exercised simultaneously, then the total test time amounts to $T = max_{j=1}^{N_B}(T_k)$ or

$$T = Max_{j=1}^{N_B}(max_{j=1}^{N_C}(X_{jk} * T_{jk}) \quad (4)$$

2.2 Area Overhead Model

Modeling area for the TDM-TAM is straightforward since the same additional logic is required for each core. In our area modeling, we neglect wiring area. Overhead for each core is due to the TDM-TAP controller and a tristate buffer. Hence, for an SoC with N_C cores, the total area overhead $A_{TDM-TAM} = N_C * (A_{buffer} + A_{TDM-TAP\,controller})$ where A_{buffer} and $A_{TDM-TAPcontroller}$ correspond to the area of the buffer and TDM-TAP controller, respectively. The TDM-TAP controller, illustrated in Fig. 3, includes a **CTAG TAP Controller** block and a **Time Domain Multiplexer (TDM)** block. Therefore, the area required is $A_{TDM-TAM} = N_C(A_{buffer} + A_{TDM-TAP}) = N_C(A_{buffer} + A_{CTAG-TAP} + A_{TDM\,block})$.

In the TDM block, there are M flip flops and M 2×1 multiplexers. Therefore, $A_{TDM\,block} = M * (A_{FF} + A_{2\times1MUX})$, where A_{FF} and $A_{2\times1MUX}$ is the area of the flip flop and multiplexer, respectively. From the above, the area overhead for the scheme, neglecting wiring area, can be expressed as follows:

$$A = N_C(A_{buffer} + A_{CTAG-TAP} + A_{TDM\,block}) \quad (5)$$

Estimates of actual area for the constituent blocks appear in Table 2.2 (for a 0.18μm CMOS technology).From Equation 5, the area overhead is proportional to the number of cores in the SoC and proportional to the length of the data frames, M (also corresponding to the number of bits required to encode the mask associated with each core). As the frame length increases, the area overhead increases.

[1] $\lceil . \rceil$ denotes rounding to the closest larger integer

Table 1. Area Estimates for TDM-TAM Controller Block

Circuit	Area(μm^2)	Description
Buffer	37	tri-state buffer
CTAG-TAP	1142	CTAG TAP controller
FF+MUX	123	Flip Flop+2 $\times$ 1 MuX

However, test time will generally decrease with increased frame length, thereby resulting in a usual tradeoff between area and test time.

2.3 Dynamic Masking

The preceding discussion includes an underlying assumption that the mask associated with each core is hardwired prior to fabrication by making appropriate ground and power-line connections. However, it is possible to implement the scheme such that the masks be programmed at arbitrary times post-fabrication. We refer to this scenario as *Dynamic Masking*. Dynamic Masking can be realized at the expense of little additional logic over that required for the *Static* case. Dynamic masking offers many potential advantages, primarily that of flexibility allowing for better and more effective post fabrication test resource optimization.

One example of an advantage offered by dynamic masking is that of increased core diagnostics, deemed necessary only post fabrication and possibly subsequently to an initial test phase. For example, a given core under test may require further diagnosability. This may be achieved a posteriori by modifying the core's mask to allow for more test data to reach and leave the core per given test time. Also, dynamic masking can accommodate test data/pattern alterations that occur post fabrication and therefore allowing for better test resource optimization, e.g., test time minimization, subsequently to such changes and fabrication. Dynamic masking can also be exploited to modify the mask assignments to accommodate different core test times/data requirements for cores connected on the same or different branches. Finally, power dissipation, coupling, and other noise or performance related post-fabrication effects, can be more easily handled by virtue of the versatility introduced by dynamic masking.

Dynamic masking can be implemented by adding a new state in the **TDM TAP Controller** that controls the shifting in of a mask into the flip flop chain of the **Time Domain Multiplexer Block** in Fig. 3. Note that for realizing dynamic masking, the only additional logic required in the **Time Domain Multiplexer Block** is a multiplexer. This multiplexer needs to be controlled by the **TAP Controller** to select between the WSI (for shifting in a new mask) and the feedback signal (for normal operation).

3 Optimization

In Section 2, we described the basic concept of TDM-TAM, as well as a developed a test time model and a model for area overhead. One of the most important issues associated with TAM architectures is SoC test time minimization. This section focuses on this issue assuming that core designs and their test requirements are fixed, i.e., we focus on the optimal assignment of cores to test buses and the optimal assignment of masks to individual cores assuming the TDM-TAM architecture.

Table 2. U226 from ITC SoC'02 Benchmarks

Core	No. of primary I/Os	No. of test patterns	No. of scan chains	Scan chain lengths	TAM use
1, 2, 3	2/1	1363968	0	-	n
4, 5, 6	3/17	2666	0	-	y
7	97/64	76	20	52	y
8	34/32	1048576	0	-	n
9	17/10	15	0	-	y

More specifically, we address the following problems: *Mask Assignment*: assuming an SoC using the TDM-TAM scheme, assuming N_C cores assigned to N_B branches, determine the optimal mask assignment for each of the N_C cores; and *Core-Branch Pairings*: assuming an SoC using the TDM-TAM scheme, assuming a total of N_C cores having to be assigned to N_B branches, determine the optimal core-branch pairing for each core. These problems are in fact revised problems from earlier TAM optimization works [3] where the objectives are to find the optimum configuration, for a specific TAM, [8], to achieve a minimum test time. In our special TDM-TAM, the problem is not only (1), finding the best assignment of cores to buses, but also (2) finding the best mask assignments for each of the cores.

Toward solving the aforementioned optimization problems, here we used a Genetic Algorithm (GA)-based method. Our program requires the following information as input: number of cores, number of branches, the test strategy for each core and whether any functional (non-scan) test patterns need to be applied, the number and length of the core scan chains, and the number of core input/outputs. Our program outputs an optimal branch configuration and core mask assignments.

Example 1: Consider the U226 SoC benchmark from the ITC'02 SoC test benchmarks [2]. The characteristics of this benchmark are reported in Table 2. We assume a TDM-TAM design for this SoC and the application of our optimization program to find the optimal parameters for the TDM-TAM. We assume two branches with minimum width, i.e., $N_B = 2$, and the number of bits/frame to be sixteen in both cases, i.e., $M_1 = M_2 = 16$. As five of the nine constituent cores are assumed to be connected to the TAM for this benchmark, the problem is optimally assigning these five cores to two branches and making optimal mask assignments for each of the cores. Applying our optimization program yields the assignment of cores 4, 5 and 9 to a first branch, and cores 6 and 7 to the second, as illustrated in Fig. 5 (a). For the first branch, the optimal mask assignment is such that $\|mask_4\| = \|mask_5\| = 7$ bits, and $\|mask_9\| = 2$ bits, while those for the second branch are such that $\|mask_6\| = 6$ bits and $\|mask_7\| = 10$ bits. This configuration and assignment yields a total test time

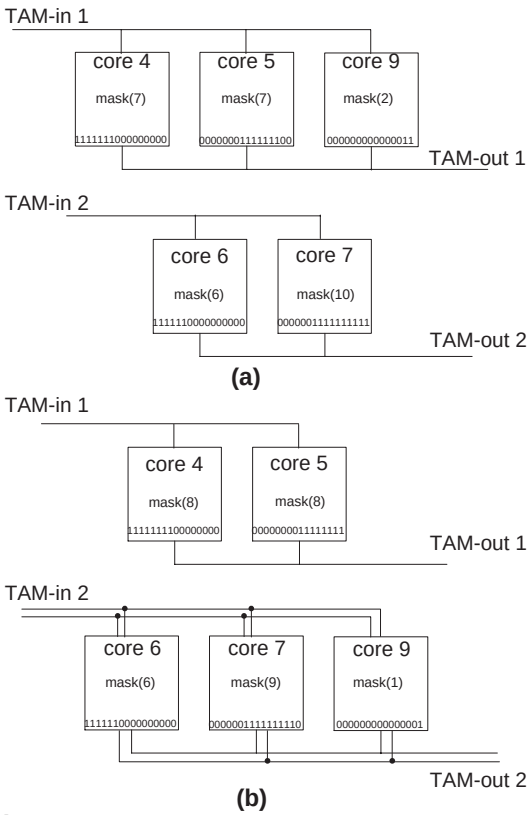

Figure 5. Optimal configurations for SoC U266 assuming two branches, for total effective TDM-TAM width of (a) 2 and (b) 3.

of 140160 cycles. This test time results for the case where the branches are both of minimal width, i.e., such that each TAM branch consists of only one available data line and hence the total effective *TDM-TAM width = 2. (Note that we define the total effective TDM-TAM width* as the total number of input/output TAM lines excluding the necessary control lines like TMS, Clock and Reset.) However, when the total effective *TDM-TAM width = 3* the optimal configuration differs. For a total effective *TDM-TAM width = 3* , the optimal configuration is as illustrated in Fig. 5 (b), i.e., branch one is attributed an effective width of one bit (minimal width), with cores 4 and 5 are assigned to it and such that $\|mask_4\| = \|mask_5\| = 8$; while branch two is attributed an effective width of two bits, with cores 6, 7 and 9 are assigned to it and such that $\|mask_6 = 6\|$, $\|mask_7\| = 10$ and $\|mask_9\| = 2$. The total test time for this configuration amounts to 95984 cycles. In Fig. 6, the total test time (cycles) versus total effective TDM-TAM width, assuming two-branch configurations is reported for benchmark U226.

4 Case Study

To evaluate the effectiveness and tradeoffs associated with the proposed TDM-TAM, a network processor engine (NPE) design was developed and used as a target test vehicle. Our NPE is an OSI Layer 3 device that forwards IPv4 packets [6]. The NPE's major blocks include a pre-

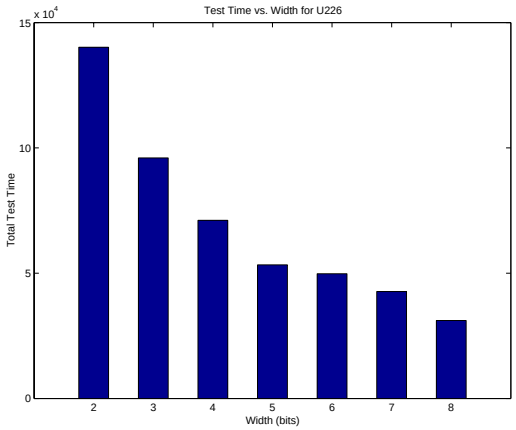

Figure 6. Benchmark U226: Test time (cycles) vs. total effective TDM-TAM width, $N_B = 2$.

processing unit, a classifier, an embedded processor, several post-processing units, and various memory components.

In regards to the core-level test methodology, a full-scan test methodology is assumed for the preprocessing and the post-processing units. To emulate a heterogeneous test methodology environment, the classifier and embedded processor blocks are assumed to be tested using a logic BIST methodology. The logic BIST uses a 32-bit linear feedback shift register (LFSR) to generate pseudo-random test vectors and uses a signature analyzer to compact the test result. The memory modules use memory BIST that runs a Marching C algorithm. All the blocks and the associated test structures are encapsulated with P1500-compliant wrappers [1].

As illustrated in Fig. 7, we compared three different TAM architectures using the NPE as a target design. A first TAM that we investigated in our comparison, referred to as *Serial P1500*, leverages the new P1500-compliant wrappers. The proposed P1500 standard architecture resembles the STD 1149.1 Test Access Port and Boundary Scan architecture. The second TAM that we included in our comparison is NIMA [9]. The basis of NIMA is the establishment of indirect digital communication paths between cores through the use of a packet-switching connections. In our specific case, we use the on-chip network for test purposes. Finally, the third TAM used in our comparison is the TDM-TAM.

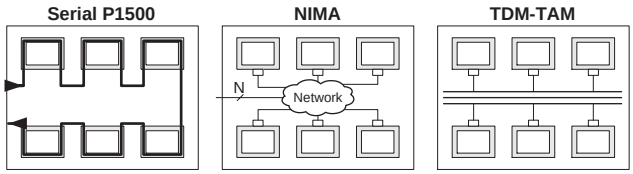

Figure 7. TAM architectures (in comparison).

We derived specific test time and area overhead models for the TDM-TAM when the latter is applied to the NPE design described above. For test time, we derived expressions that depend on a core's test methodology, i.e., whether full scan or BIST. For BISTed cores, we assume that a given set of instructions are required to initiate and complete the BIST while the actual test (application of test patterns and

Table 3. Overhead area comparison.

TAM Type	Overhead Area	
	μm^2	%
TDM (with 16 bits mask)	18882	0.75
TDM (with 32 bits mask)	30690	1.22
NIMA	1343236	53.3
Serial	5041	0.2

signature generation) can proceed independently from the specific TAM architecture. Hence, for a BISTed core, t_d is the time required to send the BIST instructions while t_i is the core test time. Cores tested using a full-scan test methodology must use the TAM throughout the entire test session. Hence, $t_i = 0$ for such cores and t_d is the time to send all the test patterns, run the test and get the result. For the area overhead associated with TDM-TAM when applied to our NPE, we used the area model described in Section 2.2. Results for the frame lengths of 16 and 32 are reported in Table 3, assuming the NPE design implemented in a .18μm CMOS technology. The area overhead for different TAMs are compared in Table 3. The serial TAM has the minimum area overhead, and that for TDM is very close, while the test time for TDM is much shorter than that of the serial TAM. The area overhead for the NIMA is very large.

For the test time, the three TAMs were compared for six different scenarios with each new scenario corresponding to an increased number of cores tested using a full scan test methodology, thereby corresponding to increased test data volumes for each new scenario. Fig. 8 illustrates the total test time (measured in clock cycles) required for testing the NPE assuming full-scan DFT for NIMA and TDM of different widths (from 2 bit to 5 bits) and the serial TAM. The horizontal axis indicates the total test data (in bits) transferred. The BIST execution time is approximately 530,000 cycles for all six scenarios, and it is chosen as the threshold for total test time. From Fig. 8, it is obvious that the serial TAM has the worst test time and TDM with width 5 has the minimum test time. The test time of NIMA is very close to that of the TDM, but the area overhead for NIMA is much larger than for TDM-TAM.

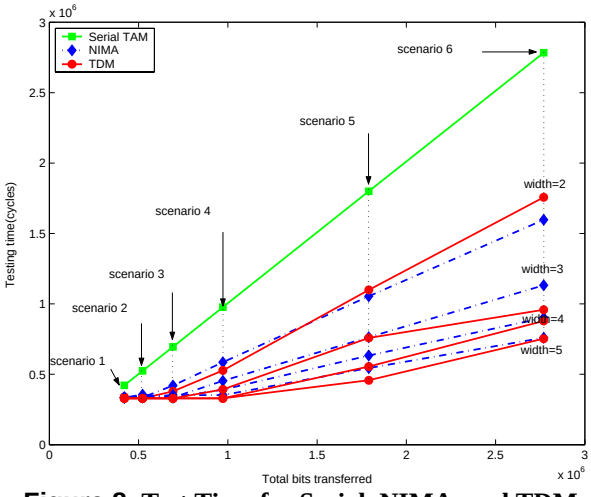

Figure 8. Test Time for Serial, NIMA, and TDM

5 Conclusion

We proposed a new bus-based TAM which is scalable and that compares favorably to other proposed TAMs in terms of test time and area requirements. The proposed TAM uses the concept of time domain multiplexing (TDM) to effectively reduce TAM area requirements while still achieving good test time performance. This is possible by making optimal assignments of cores to buses and optimal assignment of time slots in the time domain multiplexing scheme. We illustrated the use of a genetic algorithm-based methodology to achieve such optimal assignments. We presented test time and area requirement models for our TDM-TAM.

TDM-TAM not only generally offers excellent time and area performance, but can also be implemented to enable optimal reconfiguration of test resources *post* fabrication using *dynamic masking.* This feature is particularly attractive for addressing test requirements that cannot be anticipated *pre* fabrication, for example in cases requiring increased diagnostics due to one or more faulty cores on an SoC. We implemented the TDM-TAM on a network processor engine design, and compared area and test time of TDM-TAM to other proposed TAMs. We illustrated how TDM-TAM offers an attractive alternative to such TAMs, because of its smaller area requirements and comparatively smaller test time.

References

[1] IEEE P1500 web site. http://grouper.ieee.org/groups/p1500/.

[2] ITC'02 SoC Test Benchmarks. http://www.extra.research.philips.com/itc02socbenchm/.

[3] Z. Ebadi and A. Ivanov. Design of an optimal test access architecture using genetic algorithm. In *Proc. of Asian Test Symposium*, pages 205–210, 2001.

[4] I. Ghosh, N. Jha, and S. Dey. A low overhead design for testability and test generation technique for core-based systems. In *Proc. of International Test Conference*, pages 50–59, 1999.

[5] S. Goel and E. Marinissen. TAM architecture and thier implication on test application time. In *Proc. of International workshop on Test Embedded Core-based Systems*, pages 3.3–1–10, 2001.

[6] L. Hong, M. Nahvi, R. Fung, A. Ivanov, and R. Saleh. Novel test methodologies for soc/ip design implementation and comparison. In *Proc. of IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2002.

[7] L.Whetsel. An IEEE 1149.1 based test access architecture for ic with embedded cores. In *Proc. of International Test Conference*, pages 69–78, 1997.

[8] E. Marinissen. A structures and scalable mechanism for test access to embedded reusable cores. In *Proc. of International Test Conference*, pages 284–293, 1998.

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

[10] N. Touba and B. Pouya. Testing embedded cores using partial isolation rings. In *Proc. of International Test Conference*, pages 10–16, 1997.

[11] L. Whetsel. Addressable test ports: an approach to testing embedded cores. In *Proc. of International Test Conference*, pages 1055–1064, 1999.