JOURNAL OF ELECTRONIC TESTING: Theory and Applications 18, 503–514, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Deterministic Test Vector Compression/Decompression for Systems-on-a-Chip Using an Embedded Processor

ABHIJIT JAS AND NUR A. TOUBA
Computer Engineering Research Center, Department of Electrical and Computer Engineering, Engineering Science Building, University of Texas at Austin, Austin, TX 78712-1084, USA
touba@ece.utexas.edu

Received August 31, 2001; Revised January 15, 2002

Editor: Krishnendu Chakrabarty

Abstract. A novel approach for using an embedded processor to aid in deterministic testing of the other components of a system-on-a-chip (SOC) is presented. The tester loads a program along with compressed test data into the processor's on-chip memory. The processor executes the program which decompresses the test data and applies it to scan chains in the other components of the SOC to test them. The program itself is very simple and compact, and the decompression is done very rapidly, hence this approach reduces both the amount of data that must be stored on the tester *and* reduces the test time. Moreover, it enables at-speed scan shifting even with a slow tester (i.e., a tester whose maximum clock rate is slower than the SOC's normal operating clock rate). A procedure is described for converting a set of test cubes (i.e., test vectors where the unspecified inputs are left as X's) into a compressed form. A program that can be run on an embedded processor is then given for decompressing the test cubes and applying them to scan chains on the chip. Experimental results indicate a significant amount of compression can be achieved resulting in less data that must be stored on the tester (i.e., smaller tester memory requirement) and less time to transfer the test data from the tester to the chip.

Keywords: system-on-chip testing, test data compression, deterministic testing

1. Introduction

Printed circuit board (PCB) based systems are being replaced by widespread use of systems-on-a-chip (SOC). All the functionality found in a complete system is put together on a single chip. A wide variety of electronic devices ranging from embedded processors, embedded memories, logic elements, communication peripherals, and various analog components can be integrated together on a single piece of silicon. The high level of integration has made the manufacturing process simpler thus driving down the manufacturing costs. However, the advent of SOC technology has rapidly increased the complexity of testing these chips. One of the increasingly difficult challenges in testing SOCs is dealing with the large amount of test data that must be transferred between the tester and the chip [25]. The entire set of test vectors for all components of the SOC must be stored on the tester and transferred to the chip during testing. This poses a serious problem in terms of ATE (automated test equipment) throughput because of the cost and limitations of ATE. Testers have limited speed, channel capacity, and memory. The amount of time required to test a chip depends on how much test data needs to be transferred to the chip and how fast the data can be transferred (i.e., the test data bandwidth). This depends on the speed and channel capacity of the tester and the organization of the scan chains on the chip. Both test time and test storage are major concerns for SOCs.

There are usually quite a few computing resources available on an SOC chip, and if this computing power is harnessed effectively, it could be used to greatly reduce the overhead of testing. As test cost is becoming a significant percentage of the manufacturing cost, chip vendors are looking into sophisticated DFT techniques to contain the exploding test cost.

This paper presents a novel approach for using an embedded processor to aid in deterministic testing of the other components of the SOC (preliminary results were published in [17]). The basic idea is that the tester loads a program along with compressed test data into the processor's on-chip memory. The processor executes the program which decompresses the test data and applies it to scan chains in the other components of the SOC to test them. This approach reduces both the amount of data that must be stored on the tester *and* reduces the test time. Moreover, it can enable at-speed shifting using a slow tester (i.e., a tester whose maximum clock rate is slower than the SOC's normal operating clock rate).

The compression/decompression scheme presented in this paper is very well suited for implementation on an embedded processor. The decompression process requires very few processor instructions and thus can be done very quickly. The decompression process is pipelined to maximize the throughput of decompressed test vectors and thereby minimize the test time.

The paper is organized as follows: Section 2 discusses previous research that relates to the work presented here. Section 3 describes the proposed compression scheme. Section 4 explains the implementation details for the proposed scheme. Section 5 discusses ordering heuristics for the test vectors to maximize compression. Section 6 discusses techniques for avoiding memory overflow. Section 7 analyzes the test time reduction that can be obtained using this scheme. Section 8 discusses hardware overhead for the proposed scheme. Section 9 presents experimental results for benchmark circuits. Section 10 is a conclusion.

2. Related Work

Previous research in using embedded processors to aid in testing has focused primarily on performing memory tests [19, 20], or pseudo-random built-in self-test (BIST). Techniques for generating pseudo-random patterns and compacting test responses using simple programs have been proposed in [1, 9, 18, 21–23]. Techniques for mixed-mode BIST using embedded processors have been described in [7] and [10]. In [19], an embedded microprocessor is used to perform embedded memory test using the march algorithm and also to test other embedded function specific cores like D/A converters (DACs). A self-test methodology for bus-based programmable SOCs is described in [11]. The approach presented here is a fully deterministic test approach which supports either external testing or mixed-mode BIST. The advantage of decompressing deterministic vectors is that a targeted fault coverage can be achieved with a short test time. The number of vectors that need to be applied to the circuit is much less than that required for pseudo-random BIST. Moreover, it supports structured delay fault testing and testing of intellectual property (IP) blocks. In some cases, an IP provider may not be willing to provide any information of the internal logic of an IP block and thus fault simulation is not possible thereby precluding pseudo-random BIST.

Previous research has also been done in compressing/decompressing deterministic test data. Novel approaches for compressing test data using the Burrows-Wheeler transform and run-length coding were presented in [12, 24]. These schemes were developed for reducing the time to transfer test data from a workstation to a tester (not for use on chips). Iyengar et al. [13] presented the idea of statistically encoding test data. They described a BIST scheme for non-scan circuits based on statistical coding using comma codes (very similar to Huffman codes) and run-length coding. A scheme for compression/decompression of test data using cyclical scan chains is described in [16]. It uses careful ordering of the test set and formation of cyclical scan chains to achieve compression with run-length codes. A general technique for statistically encoding test vectors for full scan circuits using selective Huffman coding is presented in [14]. Chandra and Chakrabarty have proposed test vector compression techniques based on Golomb codes [3, 4] and frequency directed run-length (FDR) codes [5]. Test vector compression based on hybrid BIST techniques have been described in [6] and [15]. In [8], a test vector compression technique based on geometric primitives is proposed. In this technique, test vectors are optimally reordered and divided into blocks and then encoded based on geometric shapes. Although the technique achieves high compression, the authors have not discussed the implementation details of the scheme and assumed that an embedded processor will be used to decode the test vectors. However, the complexity of the decoding process in [8] is a severe limitation of the approach and may result in long test times. The compression/decompression scheme described in this paper can be very efficiently implemented using an embedded processor. The decompression program is very simple and compact and performs the decompression on the fly so that both the test data and test time are significantly reduced.

3. Proposed Scheme

The compression/decompression scheme described in this paper is based on generating the next test vector from the previous one by storing only the information about how the vectors differ. In this scheme, each test vector is divided into fixed length blocks as shown in Fig.1. The size of the test vector blocks depends on the word size of the processor. The way this block size is determined will be explained later. The next test vector is built from the previous test vector by replacing the blocks in which they differ. For example, in Fig. 1, the blocks in which test vector $t+1$ differs from test vector t are shaded. Hence test vector $t+1$ can be built from the t-th one by replacing only the shaded blocks.

Because of the structural relationship among faults in a circuit, there will be a lot of similarity between the test vectors. The test vectors can be ordered in an optimal way such that two successive test vectors will differ in a relatively fewer number of blocks. Hence the amount of information required to store these differences will be less than that required for storing the entire test vector. These differences are represented by “replacement words” which are encoded pieces of information that tell the processor how to build the next test vector from the previous one. Each replacement word has three fields as shown in Fig. 2. A single bit field called the *last flag*, a $\log_2 N$ bit field called the *block number* (where N is the number of blocks into which the test vector is divided) and a b bit field called the *new block pattern*. The block number field contains the address of the block that is to be replaced with the new pattern contained in the new block pattern field. If two successive test vectors differ in x blocks

Last Block	Block #	New Block Pattern
1	$m=\lceil \log_2 N \rceil$	b=block size

Fig. 2. Replacement words.

Last Flag	Block #	New Block Pattern
0	001	1011010011010011010001101010
0	010	1011010011010011010001101010
0	101	1011010011010011010001101010
1	111	1011010011010011010001101010

Word Size of Processor (e.g., 32 bits)

Fig. 3. Replacement words for example in Fig. 2.

then this information is represented as a sequence of x replacement words where the last flag field of the x-th replacement word (the last replacement word in the sequence) has its last flag bit set (1). All other replacement words have their last flags turned off (0). The sequence of replacement words for the example of Fig. 1 is shown in Fig. 3. The processor reads these replacement words and then replaces the appropriate blocks with the new block patterns. When it sees the last flag bit set, then it knows that the next test vector formation has been completed. It then shifts the test vector into the scan chain(s) and applies it to the core-under-test. The block size, b, is chosen in such a way that

$$1 + \lceil \log_2 N \rceil + b = W \tag{1}$$

where W is the word size of the processor.

There are several advantages to selecting the block size in the way as explained above. This leads to the most efficient packing of the replacement words in the memory as each replacement word fits completely into one memory word. Not only does it lead to the most efficient utilization of storage but also the decoding becomes easier as the processor has to just read one word, mask the index and the last flag bit and update the appropriate memory block with the block pattern. Choosing a block size that is too small would necessitate double packing of multiple replacement words into

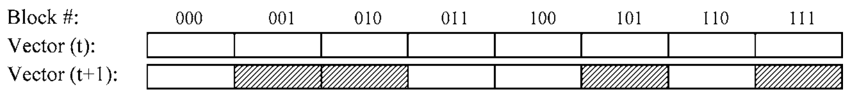

Fig. 1. Dividing test vector into blocks.

one memory location to save storage. Doing so would also complicate the decoding of the replacement words. If single replacement words are packed into each memory word then there will be waste of space as each replacement word will now be much less than W bits. On the other hand choosing a block size that is too big would necessitate a replacement word to be split across multiple memory words, which will make the decoding process more time consuming and complicated.

As mentioned before, if successive test vectors differ in a small number of blocks, then the total number of bits required for representing all the replacement words (to build the next test vector) will be less than that required for representing the entire test vector (which is the same as the number of bits in a test vector), thus resulting in compression. Note that the amount of compression depends on the ordering of the test vectors. The better the ordering, the fewer the number of blocks in which successive test vectors differ and consequently the fewer the number of replacement words. In Section 5, compression results are shown for different ordering heuristics to study the effect of ordering on the compression achieved.

4. Performing Decompression Using Embedded Processor

The compression/decompression technique described in the previous section can be implemented by using an embedded processor in an SOC. This section describes the implementation details.

The block diagram in Fig. 4 gives an overview of the architectural set-up of this scheme. The processor

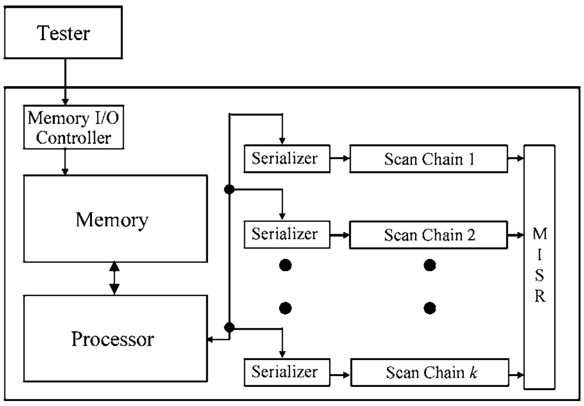

Fig. 4. Block diagram of test architecture with serializers as separate registers.

is used to concurrently load multiple scan chains on the chip. The on-chip memory holds the instructions that the processor executes and also the data on which the processor operates. The tester initially downloads a software program into the memory which the processor executes. The tester then supplies the processor with a stream of encoded data (replacement words) about blocks that are to be replaced with new patterns to produce the next test vector. The processor just decodes the replacement words and acts accordingly.

In the initialization phase (before testing begins) the tester downloads the software program (compiled machine code) into the on-chip memory. Then a fixed set of locations are reserved in the memory for the tester to continuously download the encoded data (replacement words) which the program uses. For example, let us assume that memory locations $0, 1, \ldots, M-1$ are M memory locations which are reserved for the tester to write encoded data into in a modulo M manner, i.e., the tester first writes encoded data into locations $0, 1, \ldots, M-1$ and then again starts from 0. After the initialization phase is over, the tester starts loading replacement words into the specified memory locations. The processor now starts running the program. The program is very simple. It directs the processor to routinely read the replacement words from the memory locations $0, 1, \ldots, M-1$ in a modulo M manner and act accordingly. While the processor is running the program, which results in test vectors being generated on the chip and being applied to the scan chains, the tester continues to load the memory with new replacement words for the processor to work on.

Note that, in general, there will be a memory I/O controller on the SOC for interfacing with the outside world during normal system operation. This memory I/O controller is used by the tester to load the data into the memory during testing. The memory I/O controller is typically capable of handling a slower clock rate when interfacing with the outside world. Thus, the tester can run at a slower clock rate than the normal system clock rate of the processor. Thus the processor can be operating at-speed even though the tester may be slower. Because of the tremendous cost of high-speed ATE equipment, this is a major advantage because it allows at-speed scan shifting while using slower (and cheaper) ATE equipment.

The current set of blocks is stored in the memory along with the replacement words. The program that runs on the processor causes the processor to execute a while loop until the end of the test data. In each iteration

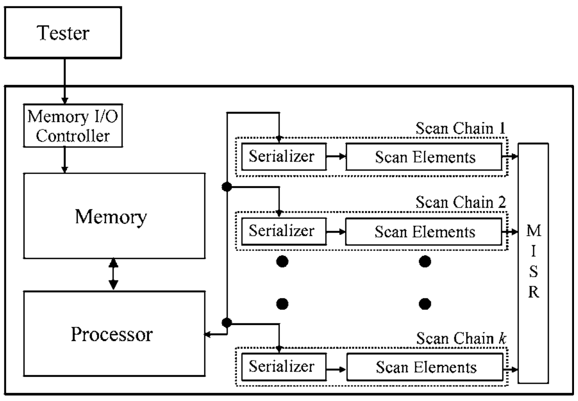

Fig. 5. Block diagram of test architecture with serializers as part of scan chains.

of the loop, it fetches a replacement word from the next memory location from which it is supposed to read the data and replaces the appropriate block. When the processor sees that the last flag bit is set, it knows that it has seen the last replacement word for that particular test vector and now needs to apply the test vector to the scan chain. The mechanism for applying a test vector to a scan chain is controlled by a *serializer*. The processor downloads each block to the appropriate serializer one at a time and the serializer shifts the block into the scan chain. The serializer is a register with a small finite state machine (FSM) controller which shifts in one bit per clock cycle into the scan chain and stops the shifting when all the bits have been shifted out. The shift register portion of each serializer can either be a separate entity that is not part of the scan chain (as shown in Fig. 4), or it can be formed from the scan elements in the scan chain (i.e., be part of the scan chain itself, as shown in Fig. 5). When the entire test vector has been loaded into the scan chain, the system clock is applied and the response is loaded back into the scan chain. The response is shifted out into a multi-input signature register (MISR) for compaction as the next test vector is shifted into the scan chain. If the serializers are part of the scan chains, then the contents of the serializer after scan capture must first be shifted out before the serializer is loaded in parallel with the next block of data.

As shown in Fig. 6, a test vector is stored as N words in the memory (where N is the number of blocks into which the test vectors have been divided). Note that there are two distinct areas (sequence of addresses) in the memory from which the processor reads/writes data. There is one area from which the processor only reads the data. This is one area into which the tester writes the replacement words and the processor reads it to update the blocks. The other area is where the patterns for the blocks of the test vector are stored. The processor both reads and writes data from and to this area. It writes data into this area when it is replacing a block with a new pattern, and it reads data from this area when it downloads them to the serializers for shifting them into the scan chains. When the processor sees the *last flag* bit set, it begins the process of applying the vector by downloading the 0th block into the serializer for the 0th scan chain and starts the serializer. The serializer then starts shifting the block into the scan chain. While it is doing so the processor continues to download the 1st block into the serializer

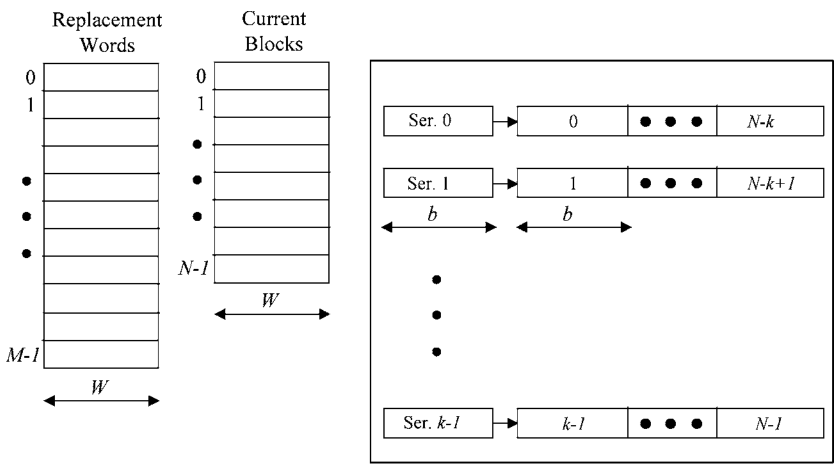

Fig. 6. Organization of data in memory.

```
void test (void)
{
  int i;  // looping variable
  int k; // number of scan chains
  int N; // total number of blocks in a test vector
  unsigned int mem_index = MEM_START;
  int last_block, block_to_replace, pattern;

  // continue until read sees "end_of_test" instruction and returns FALSE
 while (read_memory(MEM_START+mem_index, &last_block, &block_to_update, &pattern))
 {
   // replace appropriate block with new block pattern
   write_memory(base_address+block_to_replace, pattern);
   // load test vector into scan chains
   if (last_block)
   {
     for (i=0; i < N; i++)
     {
       while (serializer_busy(i %k));  // wait until serializer is ready (if necessary)
       load_serializer_from_memory(&serializer[i%k], base_address+i);
       start_serializer(i%k);
     }
   }
   // point to next memory location for reading replacement word
   mem_index = (mem_index+1) % MEM_SIZE;
 }
}
```

Fig. 7. Pseudo-code for program running on processor.

for the 1st scan chain and so on and so forth. Thus if there are k scan chains and N blocks ($k < N$) then the i-th block gets downloaded into the ($i\,\%\,k$)-th serializer, where % denotes the *modulo* operator. When the processor has finished downloading the block into the k-th serializer, it comes back to the 0th serializer. Now there can be two situations. If the 0th serializer has finished shifting in the earlier block the processor can immediately download the next block and start the serializer again. Otherwise it has to wait for the serializer to finish shifting before it can download another block into it. The FSM controller associated with the serializer controls this mechanism through a "*ready*" signal, which is checked by the processor before it downloads a particular block. Once the last block has been downloaded and shifted in, the test vector is applied to the scan chains and the processor again continues to read replacement words from the memory and update the blocks.

In Fig. 7, '*C*' like pseudo-code for the decompression program that runs on the processor is given. It is a high level view of how the assembly code for the program running on the processor may look like. Note that it is just an abstraction and tries to convey the basic algorithm for the decoding. The macro calls in most cases will be just a few processor instructions. For example, the *write_memory* is actually a representative of a LOAD instruction with indirect register addressing (base and offset register). The variable *mem_index* is an offset that points to the address from which the processor should get the next data to decode. MEM_START is defined to be the starting location for reading the replacement words. It is assumed that the very last instruction of the test session causes the processor to set some kind of a flag which is abstracted by the condition of the *while* loop. MEM_SIZE is the total size of the address space reserved for the tester to write the replacement words i.e., the tester writes to locations MEM_START, (MEM_START + 1), ..., (MEM_START + MEM_SIZE − 1). The *read_memory* macro implements reading a replacement block from a memory location and obtains the information about the address of the block to replace, pattern with which to replace, and whether it is the last block for that test vector. The *load_serializer_from_memory* macro downloads a block from the memory location addressed by the block number as offset from some base address, into the serializer for that scan chain and starts the serializer (by setting an indicator line abstracted by the *start _serializer* routine) to shift that into the scan chain.

5. Ordering Heuristics for Test Vectors to Maximize Compression

As mentioned earlier, the amount of compression achieved in the proposed scheme depends on the ordering of the test vectors (which determines the number of replacement words). There are $n!$ different ways of ordering a set of n test vectors. Hence it is impractical to examine all possible orderings for large values of n. Thus some heuristics must be used in selecting the ordering. In this section, we describe several ordering heuristics that we studied and show experimental results (number of replacement words) for each. We tried three different initial orderings, and then for each initial ordering, we tried using a greedy reordering heuristic. The first initial ordering we tried is simply the default order (which is the order produced by the ATPG tool). The second initial ordering we tried is having the test vector set ordered in ascending number of specified bits, i.e., test vectors with fewer specified bits (i.e., more don't care bits) precede those with more specified bits. The third initial ordering we tried is having the test vector set ordered in descending number of specified bits, i.e., test vectors with more specified bits precede those with fewer specified bits. For each of these three initial orderings, we computed the number of replacement words that would be required. This is shown in Table 1 under the subheading "No Reorder."

We then tried reordering the test vectors with a greedy heuristic. We started with the first test vector in the initial ordering and computed the number of replacement words that would result if each of the other $n - 1$ vectors were placed after it. The test vector that resulted in the fewest number of replacement words was then placed after the first vector. For each of the non-replaced words between the first and second test vector, the don't care bits were specified as necessary so that the two vectors matched. This process was then repeated by comparing the second test vector with each of the remaining $n-2$ vectors, and so forth. The results for using this greedy reordering procedure on each of the three initial orderings are shown in Table 1 under the subheading "Greedy Reorder."

In Table 1, the lowest number of replacement words (i.e., the best compression) for each of the benchmark circuits is shown in bold. As can be seen from the results, the greedy reordering procedure improved the results in all cases. The block size b in each case was chosen according to Eq. (1) as explained earlier in Section 3.

Note that this technique is primarily targeted for stuck-at scan vectors where the ordering of the test vectors does not matter. However, for test sets where the ATPG generated ordering needs to be maintained, this technique could still be used at the cost of some loss in compression. On an average the compression produced by the "default order—no reorder" case is 6.46% worse than the corresponding best case for each of the circuits above.

6. Avoiding Memory Overflow

Since the tester is constantly transferring replacement words to the on-chip memory, one potential problem is that if the processor falls too far behind in processing the replacement words, there could be a "memory overflow." This would result in the tester overwriting a replacement word in the memory that has not yet been processed. Care must be taken to ensure that no memory overflow will occur.

Table 1. Number of replacement words for different ordering heuristics.

	Default order		Least specified first		Most specified first	
Circuit	No reorder	Greedy reorder	No reorder	Greedy reorder	No reorder	Greedy reorder
s5378	494	**398**	491	411	494	401
s9234	730	572	722	596	722	**568**
s13207	801	**737**	803	749	801	741
s15850	837	751	841	**742**	843	758
s38417	3177	2983	3206	2995	3209	**2980**
s38584	2901	2717	2901	2737	2898	**2716**

The possibility for a memory overflow depends on the relative speed of the processor to the tester. The rate at which the tester puts data into the memory depends on the tester clock rate and the number of channels. The rate at which the processor processes data in the memory depends on its clock rate, word size, and instruction set. The instruction set determines how many clock cycles are required to process the replacement words.

The tester continuously writes test data in a cyclic fashion into locations $0, 1, \ldots, M-1$ of the memory. So after every MW (where W is the word size) bits, each location of the memory gets overwritten by the tester. Hence the processor should finish processing all the replacement words in locations $0, 1, \ldots, M-1$ within the time taken by the tester to shift in MW bits to the scan chains. If the tester has n scan channels and has a clock period T_T, the time taken by the tester to shift in MW bits is $[WM/n]T_T$. Hence the processor should process all M replacement words within this time. All the processor does for each replacement word is to read the new pattern and the block address and replace the appropriate block. However for the blocks which have the last flag set, the processor has to download all the blocks into the serializers to shift them into the scan chains. This is the most time consuming part for the processor. So the speed at which the processor can process M replacement words depends on how many of them have their last flag set. Let this be denoted by e. Then the time taken by the processor to process M replacement words is $[Mu + (eNb/k)]T_P$ where T_P is the clock period of the processor, and u is the number of cycles taken by the processor to read a replacement word and replace the appropriate block. The value of u depends on the instruction set architecture of the processor. No memory overflow will occur if the following condition is satisfied:

$$[Mu + (eNb/k)]T_P < [WM/n]T_T \qquad (2)$$

If the condition above is not satisfied, then for a given compressed test set, a quick check can be made to see if a memory overflow will occur. If a memory overflow would occur, then something must be done to avoid this. One solution would be to reorder the test vectors when constructing the compressed test set so that e will become smaller. This will likely result in less test data compression, but it will avoid memory overflow. Another solution may be to insert NOP's (no operation instructions) in the tester program at carefully selected locations to slow it down so that no memory overflow will occur, or to simply run the tester at a slower clock rate that ensures no memory overflow. These solutions would not reduce the amount of test data compression (and thus would still minimize tester memory requirements), however they could potentially result in less test time reduction.

Note that if the processor is sufficiently faster than the tester, then the above condition will be satisfied and the maximal reduction of both test data and test time can be achieved. Note also that as the number of scan chains (i.e., k) is increased, the condition becomes easier to satisfy. Lastly, if there is enough memory on the chip to store all the replacement words then no memory overflow will occur irrespective of all the parameters in Eq. (2) above. The results shown in the following paragraph help to understand the effect of all the parameters in the context of memory overflow.

Equation (2) can be alternatively represented as:

$$F_p/F_t > n/W[u + e/M(S/k)] \qquad (3)$$

where F_p ($=1/T_p$) is the processor frequency, F_t ($= 1/T_t$) is the tester frequency and $S = Nb$ is the length of the scan vector. Equation (3) establishes a lower bound for the processor frequency to tester frequency ratio to prevent any memory overflow. Table 2 shows the F_p/F_t ratio for the benchmark circuits for different values of M. In each case, values of the other parameters are kept constant. The k/n value for each circuit has been carefully chosen to prevent a very high value of the F_p/F_t ratio. Also the ordered test set chosen in each case is the one that produces the best compression as shown in Table 1. The values of the parameters that are held constant for all the circuits are $W = 32$, $u = 3$. The value of e depends on the ordered test set and M. The scan length S depends on the benchmark. Note that W and u

Table 2. Variation of F_p/F_t for different values of M.

		M				
Circuit	k/n	256	512	1024	2048	4096
s5378	8/4	1.62	NA	NA	NA	NA
s9234	8/4	1.82	1.42	NA	NA	NA
s13207	8/1	2.26	1.31	NA	NA	NA
s15850	8/2	1.74	1.17	NA	NA	NA
s38417	8/4	2.10	1.90	1.62	1.34	NA
s38584	8/2	2.24	1.64	1.20	0.85	NA

are two parameters on which the system integrator will most likely have the least amount of control (as they are tightly integrated to the processor/memory micro-architecture). In Table 2 their values have been fixed for all the circuits at realistic values. The k/n ratio chosen for each circuit is also shown in the table with the actual values of k and n. The results of Table 2 illustrate how by carefully tuning the different parameters on which the system designer has control, the memory overflow can be avoided.

The entries "NA" in the above table corresponds to cases where there will be no memory overflow because of enough memory being present to store all the replacement words. Also in most cases if the processor is sufficiently faster than the tester as given by the table entries above, the memory overflow can be avoided.

7. Test Time Reduction

The total reduction in test time that can be obtained using the proposed scheme can be analytically estimated given the ordering of the test set and the other parameters that define the test architecture. In conventional scan based testing, the total test time consists of mostly the scan shifting time since the test application time (launch and capture cycle) is usually negligible compared to the scan shifting time. The scan shifting time on the other hand depends on the total amount of test data, the number of tester channels and the speed of the tester. Following the notations that are being used in this paper this time can be denoted by $T_{\text{conventional}} = (SVT_T)/n$ where V is the number of test vectors. Let T_{scheme} denote the total test time in the proposed scheme. Thus the percentage reduction in total test time using this scheme is given by

$$(1 - T_{\text{scheme}}/T_{\text{conventional}}) \times 100.$$

The following paragraphs explain how T_{scheme} can be calculated based on the test architecture. T_{scheme} consists of two components, T_{transfer} and $T_{\text{application}}$. T_{transfer} is the time taken by the tester to transfer the compressed data and $T_{\text{application}}$ is the time taken by the processor to apply the test vectors to the core-under-test. Unlike conventional scan testing in this case $T_{\text{application}}$ is not negligible and contributes significantly towards the total test application time. However, the effect is not additive because of the inherent parallelism in the operations of the tester and the processor. This means that $T_{\text{scheme}} < T_{\text{transfer}} + T_{\text{application}}$. It is obvious that $T_{\text{transfer}} < T_{\text{conventional}}$ (since the tester has to transfer less test data in this scheme) with the percentage reduction being directly proportional to the amount of test data compression. Hence the total reduction in test data is directly related to how much parallelism can be obtained between the operations of the tester and the processor. Note that T_{transfer} is affected by the tester frequency whereas $T_{\text{application}}$ is affected by the processor frequency. The following paragraph shows how the parallelism can be maximized and also derives some formulas for calculating T_{scheme}.

Let R be the total number of replacement words. Starting at time $t = 0$, let $W_1, W_2, \ldots, W_i, \ldots, W_R$ denote the times when the tester finishes writing the first, second, *i-th* and the *R-th* replacement words respectively. Maximum parallelism can be achieved if the processor can process the *i-th* replacement word as soon as the tester writes it. The processor can process a replacement word only after the tester has written that replacement word and the processor has finished processing all the previous replacement words. The time taken by the processor to process a replacement word is uT_p if the replacement word does not have the last flag bit set (because in this case all the processor has to do is to read it and update the appropriate block in memory), and $[u + S/k]T_p$ if the replacement word has the last flag bit set (because in this case besides updating the necessary block in memory it has to download the blocks into the serializer and apply them). Let this be denoted by c, a constant for a given test architecture. Starting at time $t = 0$, let $P_1, P_2, \ldots, P_i, \ldots, P_R$ denote the earliest possible times at which the processor can start processing a replacement word. The P_i's can be easily computed using the following iterative formula.

$$P_1 = (S/k)T_p \tag{4}$$

$$\text{and} \quad P_i = \max\{W_i, P_{i-1} + c\} \tag{5}$$

Equation (5) basically means that the processor can start processing the *i-th* replacement word only after the tester has written it AND it has finished processing all the previous replacement words. Equation (4) is determined by the fact that the processor can start processing the first replacement word after the very first test vector has been applied to the core-under-test. It is very easy to compute the W_i's as it depends on n, T_T and W and is given by

$$W_1 = (S/n)T_t \tag{6}$$

$$\text{and} \quad W_i = W_{i-1} + (W/n)T_t \tag{7}$$

Equation (6) is determined by the fact that the tester has to first write the memory block patterns required for the very first test vector and subsequently a constant amount of time for every other replacement word determined by the number of tester channels and its frequency. Note that each replacement word is W bits (this is how the block size b is chosen, so that each replacement word completely fits within a memory word). Assuming that both the processor and the tester is ready to start operation at $t = 0$, the time at which the testing process is completed is given by $P_R + c$, which is when the final replacement word is processed by the processor. Hence, $T_{\text{scheme}} = P_R + c$. Once T_{scheme} has been determined, the percentage reduction in test time can be determined from the formula mentioned above.

8. Hardware Overhead of the Scheme

In terms of the extra hardware that is required, this scheme is very efficient requiring only a small area overhead due only to the serializers. As mentioned earlier, the serializers consist of two components. One is a shift register and the other is a finite state machine controller that controls the operation of the shift registers along with the clocking mechanism. Part of the scan chains can be configured as the shift register (as shown in Fig. 5). The complexity of the finite state controller is independent of the size of the design or the size of the test vector set. The complexity of the controller is totally determined by the block size. This is because the controller has to keep track of the number of bits being shifted out and hence has a counter associated with it whose size depends on the block size. This means that the percentage of hardware overhead due to the serializers becomes less as the design size increases. Note that the processor and the memory are not being considered as part of the test hardware as they are assumed to be already present in the system-on-chip. As mentioned earlier, the main objective of the paper is to show how existing hardware in a system-on-chip (processor and memory) can be utilized to reduce the cost of testing other cores on the chip.

To give an idea of the hardware overhead, Table 3 shows the area overhead as a percentage of the area of the benchmark circuits. The serializers were designed in verilog and synthesized along with the benchmark circuits. The resulting area (in terms of standard library gate equivalents) was then compared to the synthesized area of the benchmarks without the serializers. The resulting area overhead is shown in Table 3 below. The area overhead is calculated as $(A_{\text{scheme}}/A_{\text{benchmark}} - 1) * 100$ where A_{scheme} is the area of the benchmark circuits with the serializers and $A_{\text{benchmark}}$ is the area of the benchmark circuits without the serializers. Also the amount of memory required to store all the replacement words for the best results obtained in each case (based on Table 1) is shown. Since each replacement word is W bits and in all the experiments W has been assumed to be 32, the memory required is four times the number of replacement words (in bytes).

Table 3. Hardware requirements of the proposed scheme.

Circuit	Percentage overhead due to serializer	Memory required for replacement words (Kbytes)
s5378	5.15	1.592
s9234	3.96	2.272
s13207	1.60	2.948
s15850	1.69	2.968
s38417	0.53	11.980
s38584	0.58	10.948

As can be seen from the results in Table 3, the area overhead of a single serializer is very small compared to the benchmark circuits. In fact, the area overhead becomes almost negligible for the largest benchmark circuits.

9. Experimental Results

The proposed scheme was used to compress test sets for the largest ISCAS 89 [2] circuits. A commercial ATPG tool was used to generate test cubes that provided 100% coverage of detectable faults in each circuit. Unspecified input assignments were left as X's to enable better compression. The block size in each case was derived according to Eq. (1). The results obtained in Table 4 are for $W = 32$.

For each circuit, Table 4 shows the size of the scan chain for each circuit and the number of bits in each block. The number of test vectors and total number of bits of test data is shown for the original test set. For the compressed test data, the number of replacement words that are required is shown along with the total number of bits of test data. The second to last column shows the percentage reduction in test data (amount of compression) that is achieved which is computed as:

$$(1 - \text{Compressed Bits/Original Bits}) \times 100$$

Table 4. Compression obtained for ISCAS benchmark circuits using proposed scheme.

			Original test data		Compressed test data			
Circuit	Scan size	Block size (bits)	Num. vectors	Num. bits	Rep. words	Num. bits	% Reduction of test data	% Reduction of test time
s5378	214	28	119	25466	398	12950	49.15	48.75
s9234	247	27	147	36309	568	18423	49.26	48.87
s13207	700	26	239	167300	737	24284	85.48	85.48
s15850	611	26	120	73320	742	24355	66.78	66.46
s38417	1664	24	95	158080	2995	97024	38.62	37.99
s38584	1464	25	131	191784	2737	88376	53.92	53.60

The last column shows the percentage reduction in test time that can be achieved using this scheme. The results are based on the calculations shown in Section 7. The test architecture used to compute the test time reduction is given by the parameters $M = 3072$ (12 KB Memory), $F_P/F_T = 2$, $u = 3$, and k/n ratios based on Table 2.

The results in Table 4 are the best results obtained from the ordering heuristics as shown in Table 1. As can be seen from the results, a significant amount of compression can be achieved with the proposed scheme. Also note that it is possible to achieve an almost equal amount of test time reduction by tuning the test architecture.

In order to see the effect of block size on the percentage of compression, experiments were done with varying block sizes on the four largest benchmark circuits. In each case the same test set was used as the one in Table 4 above. The results are shown in Table 5. As can be seen, there is a clear trend of decreasing compression with increasing block size. This is expected. With increasing block size, the number of blocks that are similar between two test vectors decreases. Also from the results in Table 4 it is evident that there is not much loss in compression in our present scheme due to the restrictions imposed on the block size because of implementation.

To give an idea of how the amount of test data compression for the proposed scheme compares with other schemes, results are shown in Table 6 comparing the final test data volume for the proposed scheme with the results in [5] (compression obtained using T_d). Note that this is not really a direct comparison as different test sets were used. As can be seen, the final results are fairly similar. The advantage of the proposed scheme is that it can be efficiently implemented using an embedded processor.

Table 5. Variation of percentage compression with block size.

	Block size					
Circuit	8	16	24	32	40	48
s13207	87.20	87.06	85.67	84.32	82.53	82.16
s15850	69.68	68.52	66.92	64.49	63.63	61.72
s38417	41.86	42.86	38.62	38.77	34.34	29.81
s38584	53.30	53.22	53.44	50.39	48.61	46.95

Table 6. Comparison of final test data volume.

Circuit	Test data in [5] (bits)	Test data in proposed scheme (bits)
s5378	12306	12950
s9234	22152	18423
s13207	30880	24284
s15850	26000	24355
s38417	93466	97024
s38584	77812	88376

10. Conclusion

The proposed approach supports external testing of embedded cores using deterministic test vectors. It harnesses the computational power of an embedded processor (already present for functional purposes) to perform test data compression/decompression in software. The decompression processes is pipelined so that the latency is minimized. Hence, it reduces both the amount of test storage *and* test time, thereby reducing the tester memory and channel capacity requirements. Such techniques are needed to keep down the cost of the ATE equipment needed to test future SOC's.

This paper presents the basic framework for how an embedded processor can be used for decompressing test data. One specific compression/decompression algorithm, which gives good results, is described. However, there is a lot of scope for future research in other compression/decompression algorithms for test data. The ability to use a processor to perform the decompression in software opens the door to many possible techniques.

Acknowledgments

This material is based on work supported in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Technology Program under Grant No. 003658-0644-1999.

References

1. D. Bakalis, D. Nikolos, and X. Kavousianos, "Test Response Compaction by an Accumulator Behaving as a Multiple Input Non-Linear Feedback Shift Register," in *Proc. of International Test Conference*, 2000, pp. 804–811.
2. F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in *Proc. of International Symposium on Circuits and Systems*, 1989, pp. 1929–1934.
3. A. Chandra and K. Chakrabarty, "Test Data Compression for System-on-a-Chip Using Golomb Codes," in *Proc. of VLSI Test Symposium*, 2000, pp. 113–120.
4. A. Chandra and K. Chakrabarty, "Efficient Test Data Compression and Decompression for System-on-a-Chip Using Internal Scan Chains and Golomb Coding," in *Proc. of Design, Automation, and Test in Europe (DATE)*, 2001.
5. A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length Codes with Application to System-on-a-Chip Test Data Compression," in *Proc. of VLSI Test Symposium*, 2001, pp. 42–47.
6. D. Das and N.A. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid Test Patterns," in *Proc. of International Test Conference*, 2000, pp. 115–122.
7. R. Dorsch and H.-J. Wunderlich, "Accumulator Based Deterministic BIST," in *Proc. of International Test Conference*, 1998, pp. 412–421.
8. A. El-Maleh, S. al Zahir, and E. Khan, "A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," in *Proc. of VLSI Test Symposium*, 2001, pp. 54–59.
9. S. Gupta, J. Rajski, and J. Tyszer, "Test Pattern Generation Based on Arithmetic Operations," in *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 1994, pp. 117–124.
10. S. Hellebrand, H.-J. Wunderlich, and A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," in *Proc. of International Test Conference*, 1996, pp. 195–204.
11. J.-R. Huang, M.K. Iyer, and K.-T. Cheng, "A Self-Test Methodology for IP Cores in Bus-Based Programmable SOCs," in *Proc. of VLSI Test Symposium*, 2001, pp. 198–203.
12. M. Ishida, D.S. Ha, and T. Yamaguchi, "COMPACT: A Hybrid Method for Compressing Test Data," in *Proc. of VLSI Test Symposium*, 1998, pp. 62–69.
13. V. Iyengar, K. Chakraborty, and B.T. Murray, "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," in *Proc. of VLSI Test Symposium*, 1998, pp. 418–423.
14. A. Jas, J. Ghosh-Dastidar, and N.A. Touba, "ScanVector Compression/Decompression Using Statistical Coding," in *Proc. of VLSI Test Symposium*, 1999, pp. 114–120.
15. A. Jas, C.V. Krishna, and N.A. Touba, "Hybrid BIST Based on Weighted Pseudo-Random Testing: A New Test Resource Partitioning Scheme," in *Proc. of VLSI Test Symposium*, 2001, pp. 114–120.
16. A. Jas and N.A. Touba, "Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs," in *Proc. of International Test Conference*, 1998, pp. 458–464.
17. A. Jas and N.A. Touba, "Using an Embedded Processor for Efficient Deterministic Testing of Systems-on-a-Chip," in *Proc. of International Conference on Computer Design*, 1999, pp. 418–423.
18. J. Rajski and J. Tyszer, "Accumulator-Based Compaction of Test Responses," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 643–650, June 1993.
19. R. Rajsuman, "Testing a System-on-a-Chip with Embedded Microprocessor," in *Proc. of International Test Conference*, 1999, pp. 499–508.
20. J. Saxena, P. Ploicke, K. Cyr, A. Benavides, and M. Malpass, "Test Strategy for TI's TMS320AV7100 Device," in *IEEE Int. Workshop on Testing Embedded Core Based Systems*, 1998.
21. A.P. Stroele, "A Self-Test Approach Using Accumulators as Test Pattern Generators," in *Proc. of International Symposium on Circuits and Systems*, 1995, pp. 2010–2013.
22. A.P. Stroele, "Test Response Compaction Using Arithmetic Functions," in *Proc. of VLSI Test Symposium*, 1996, pp. 380–386.
23. A.P. Stroele, "Bit Serial Pattern Generation and Response Compaction Using Arithmetic Functions," in *Proc. of VLSI Test Symposium*, 1998, pp. 78–84.
24. T. Yamaguchi, M. Tilgner, M. Ishida, and D.S. Ha, "An Efficient Method for Compressing Test Data," in *Proc. of International Test Conference*, 1996, pp. 191–199.
25. Y. Zorian, "Test Requirements for Embedded Core-Based Systems and IEEE P1500," in *Proc. of International Test Conference*, 1996, pp. 191–199.

Abhijit Jas received the B.E. degree in computer science and engineering from Jadavpur University, Calcutta, India in 1996. He received the university gold medal for standing first in the college of engineering. He received the M.S. and Ph.D. degrees in electrical engineering from the University of Texas at Austin in 1999 and 2001, respectively. He currently works as a Sr. CAD Engineer at Intel Corporation in Austin, TX. His research interests are in VLSI testing, computer-aided design, and formal hardware verification.

Nur A. Touba received the B.S. degree in electrical engineering from the University of Minnesota where he graduated summa cum laude in 1990, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1991 and 1996, respectively. He is currently an Associate Professor at the University of Texas at Austin. His research interests are in VLSI testing, computer-aided design, and fault-tolerant computing. He received a National Science Foundation (NSF) Early Faculty CAREER Award in 1997. He serves on the Technical Program Committees of the International Test Conference, International Conference on Computer Design, International Test Synthesis Workshop, and International On-Line Testing Workshop.