An enhanced FPGA-based Low-Cost Tester Platform exploiting effective Test Data Compression for SoCs*)

L. Ciganda
Universidad de la Republica Uruguay
Montevideo, Uruguay

F. Abate, P. Bernardi, M. Bruno, M. Sonza Reorda
Politecnico di Torino
Torino, Italy

Abstract— **Reducing the cost of test (in particular by reducing its duration and the cost of the required ATE) is a common goal which has largely been pursued in the past, mainly by introducing suitable on chip Design for Testability (DfT) circuitry. Today, the increasing popularity of sophisticated DfT architectures and the parallel emergence of new ATE families allow the identification of innovative solutions effectively facing that goal. In this paper we face the increasingly common situation of SoCs adopting the IEEE 1149.1 and 1500 standards for the test of the internal cores, and explore the idea of storing the test program on the tester in a compressed form, and decompressing it on-the-fly during test application.**

This paper proposes an improved version of an data compression/decompression technique which is well suited for reducing the size of test programs stored on the tester; this technique is particularly effective for very long sequential test vectors generated to test SoCs by means of low-cost test procedures; thus, the paper outlines the characteristics of an FPGA-based low-cost tester platform that takes advantage of the described compression schema.

The effectiveness of the proposed methodology was demonstrated by practically testing some SoCs equipped with suitable DfT for supporting low-cost testing resorting to a low-cost tester implementing the proposed architecture and the compression/decompression technique.

Keywords; FPGA based tester, test compression/decompression

I. INTRODUCTION

Nowadays, one of the major concerns in Systems-on-Chip (SoCs) testing is the application of test stimuli. This final step in the SoC production flow follows the test design phase, which includes the selection of Design-for-Testability (DfT) features to be added on-chip and the test pattern generation process. The generated test vectors are applied to the device-under-test (DUT) using suitable testers, and its responses observed for discriminating between good and faulty chips. The emergence of the SoC design paradigm, where several (possibly complex) cores are integrated in a single design coming from different sources, is pushing designers to move towards test paradigms where each core is tested by sending to it commands (instead of vectors), which activate suitable Built-In Self-Test (BIST) circuitry on it, and gather the corresponding results. These commands are often moved from the ATE to the DUT through an IEEE 1149 compliant interface at the device level, and an IEEE 1500 compliant interface at the core level. Even if in this flow the test length minimization may appear as a problem solely tackled during the test design steps, the elapsed time in the test stimuli application is a variable dependent on the test application phase characteristics, too. This dependency is due to:

1. the vector application rate of the tester
2. the memory space available on the tester.

The vector application rate is the number of vectors applied by the tester to the DUT in a time unit. Other than the maximum reachable tester-DUT communication frequency, this parameter may depend on the tester structure (e.g., the time required to prepare a vector for application may vary or be conditioned by the tester organization).

The tester memory space may impact on the time required to apply a test procedure as well. More in particular, the reader should distinguish between the fast memory in the tester in charge of directly feeding the DUT, and the large memory (normally composed of disks) that exists in the tester host unit. In case of very long test sequences, if not enough fast memory is available, the vector set has to be divided in many subsets to be sequentially uploaded from the much slower host memory. This means that for every tested chip, the application process is suspended many times to upload the fast memory from the host memory, with a consequent efficiency loss.

The basic idea proposed in this paper lies in complementing the fast memory of the tester with a suitable circuitry (mainly based on programmable devices) which is able to decompress on the fly the test set to be applied to the DUT, thus allowing to reduce the required amount of fast memory. A preliminary version of the automated data compression and decompression technique was introduced in [6]. The method we propose performs off-line compression and suits well for the very long sequential test vectors generated to test SoCs with low-cost test procedures such as Software-based Self-Test (SBST) and Built-In Self-Test (BIST). Moreover, our technique benefits of the specific structure of the typical test set for these circuits, which is composed of commands instead of vectors. Then, on-the-fly decompression is operated by an FPGA-based low-cost tester platform. The contributions introduced in the paper are:

- an automatic pattern compression methodology based on maximal-length repeated sequences search and ranking
- a suitable two-layer tester memory organization that takes advantage of the compression method to reduce the cost of the tester while enhancing its performance.

Guidelines introduced in the paper enable effective reduction of tester cost in terms of channel memory size while guaranteeing high frequency in the stimuli application.

The paper is structured as follows. Section 2 provides some background on the state of the art of data compression and low-cost test procedures for SoCs. Section 3 illustrates the proposed approach. Section 4 shows the results obtained on a sample SoC, and section 5 draws some conclusions.

II. BACKGROUNDS

A. Data compression

Recently, data compression techniques [1][2][10] have been effectively used to tackle the tester memory issue. The compression process is performed by suitable software tools and consists in encoding the generated test vectors using as few bits as possible. Compressed data are then reconstructed, or decompressed, by ad-hoc hardware decoders/decompressors placed on-chip or on-tester.

If on-chip decompression modules are used, other than test data volume reduction, a speed up in the test stimuli application may be achieved [1][2] by possibly leveraging on internal high frequency clock sources. Such a gain is obtained at the expense of additional silicon area overhead and demands for suitable tester-DUT communication schemas; in fact, even if the tester is demanded to send data to the DUT without performing any vector manipulation, it can be asked to manage synchronization [1] or implement handshake communication protocols [2].

On the contrary, if the decompression process is performed on-tester, no test time gain is obtained - the complete uncompressed vector set is sent to the DUT at the tester frequency - but the overall testing scenario is simplified:

- the tester communicates with the DUT without the need of synchronization or handshake schemas
- no modification is performed on the chip structure
- data compression is applicable to any manufactured chip and intellectual properties are protected, since from the tester point of view no information are required about the type of on-chip embedded structure.

On-chip and on-tester solutions for decompression are not mutually exclusive. In fact, a compressed set to be expanded on-chip can be further compressed to be expanded on-tester.

B. Application aspects of low-cost test procedures

With the advent of Systems-on-Chip, the Low-Cost concept is become a common denominator among test generation and test application. In fact, in the SoC terminology, the term Low-Cost is commonly used to classify a set of strategies [4] and equipments [6] that exploit Design-for-Testability features included on-chip for reducing test costs without impacting its effectiveness; the costs of SoC test procedure involve many factors, that are primarily test pins count and required application frequency.

The adoption of test access protocols to transport information inside the SoC architecture mainly addresses pin count reduction, often at the expense of the bandwidth. Indeed, autonomous test procedure execution addresses frequency requirements mitigation, since it normally exploits internal or independent clock supply resources that do not request any external intervention.

Concerning test data volume of procedures usually classified as Low-Cost, the overall number of test vectors to be applied to the DUT depends on the number of initialization and management operations required to activate the SoC test functionalities. Consequently, patterns describing such procedures finally reside on the tester memory and potentially impact on the test applicability.

With respect to other techniques, such as scan-based or combinational testing, for the considered procedures the pattern order cannot be modified. This constraint does not permit to optimize the pattern compression process as shown in many works [1][2].

III. THE PROPOSED APPROACH

The work presented herein aims at tackling two problematic issues related to the application of low-cost test procedures such as those described in section 2.2.

A primary effort is devised to reduce the memory space required to apply the vector set. In general terms, this issue is pursued by automatically identifying repetitive test vector segments and pruning them from the pattern set to be reproduced autonomously by the tester itself. In [6] the authors showed how an important gain could be obtained by leveraging on the identification of intensively repeated test segment parts. Anyway, the method in [6] was based on the test access protocol knowledge, while a new strategy is illustrated here, which is suitable to be applied to any chip without doing any assumption on the SoC test protocol.

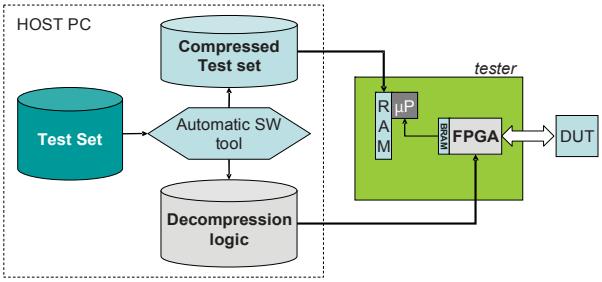

Figure 1. Working principle and HW/SW resource partitioning employed in the proposed approach.

Figure 1 shows the software and hardware resources required to support the illustrated compression process. Recurrent segments in the test set are first identified, and then mapped into hardware resources, so that they can be re-generated on the fly by the decompression logic. In particular, this corresponds to some ad hoc generated finite state machines (FSMs) implemented by the programmable devices existing on the tester and interacting with the tester software application tools [5]. The efficiency in the process of activating the FSMs is fundamental for guaranteeing the highest possible pattern set reconstruction frequency.

The second effort was therefore devoted to design a suitable FPGA-based low-cost tester platform implementing a two-layer tester memory organization including:

- a secondary, large and slow general purpose memory, which stores the complete compressed test transmitted by the host computer controlling the tester

*) This work was supported by the European Community in the Alpha-NICRON Framework.

- a primary, small and fast memory that receives at run-time the compressed test set to be decompressed by the FSM-based test head design.

This tester configuration perfectly combines with the proposed compression/decompression schema, which often asks the decompression logic to run autonomously, thus leaving time for even slower data transfer within the two memory layers.

A. Proposed algorithm for Low-Cost test procedure data compression

Differently from other approaches tackling test pattern compression, such as Run-length, Golomb and Huffman compression codes [2][3] or based on pattern value reuse [1], the proposed methodology does not encode separately each single signal to be provided to the DUT. On the contrary, in the explained approach a set of signals is jointly considered., taking also into account the considered test sets are mainly composed of commands, instead of vectors. This choice derives from two factors.

Firstly, access mechanisms to embedded test resources normally included in SoCs are based on a set of control signals that are opportunely sequenced to move data inside the device. Control signal waveforms are heavily recurrent, since they implement a translation of high level commands defined by the selected access protocol.

Secondly, test data for embedded test execution may be strongly repetitive; for instance, if an SBST test program is transmitted to an internal memory space to be later executed, the same instruction can be sent many times to different data locations. Similarly, if many commands are sent to internal decoding circuitries (e.g., for diagnostic BIST), not only control, but also data signals will show repeated occurrences.

The proposed algorithm is aimed at identifying two kinds of occurrences:

1. Vertical occurrences, a timing diagram slice that is repeated for many times in the vector set.
2. Horizontal occurrences, a timing diagram slice (longer than 1 clock cycle) during which one or more signal values are maintained stable at a certain value.

In a previous work [6], we showed that:

1. the identified vertical and horizontal pattern segments can effectively be translated into proper FSMs
2. the activation order for FSMs can be stored in a Test Segments Occurrence table (or TSO) file
3. the part of the test set that was not selected for compression can be stored in a Reduced Test Set description (or RTS) file.

For a meaningful stimuli reconstruction, vertical and horizontal occurrences have to be not overlapping. The memory gain obtained by adopting such codification method is maximized when many long vertical occurrences and few but extensively repeated cases of horizontal occurrences are identified. This hypothesis is pursued in the automatic identification algorithms detailed in next paragraphs.

Figure 2 exemplifies the vertical and horizontal occurrences concept; the shown example underlines the presence of two vertical and two horizontal occurrences.

```
I1:  0001001010011101001100000101
I2:  0010000100100000111011101001
O1:  1010110100101110101010101010
       V1 H1     V1 H1  V2   V2 V1
```

Figure 2. An example of vertical and horizontal occurrences.

In the proposed flow, firstly vertical occurrences are searched, then horizontal ones.

1) Identification and selection of vertical occurrences

The developed method for automatically identifying vertical occurrence is composed of three iteratively performed steps:

1. pattern discovery, aimed at identifying repeating occurrences in the pattern set
2. occurrence ranking, for populating the vertical occurrences set
3. pruning from the string of the selected vertical occurrence.

In a preliminary phase, for every time slice the signal values are firstly codified using a symbol as shown in figure 4; in our methodology, both input and output signals are considered.

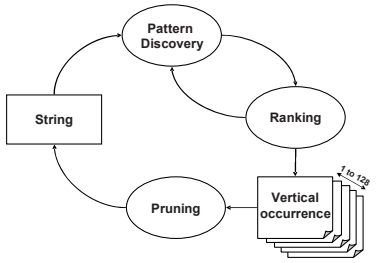

Figure 3. Vertical occurrence selection flow.

```
I1:   0001001010011101001100000101
I2:   0010000100100000111011101001
O1:   1010110100101110101010101010
-----------------------------------
 S:   1034114340345505236523303416
```

Figure 4. Example of symbol codification during pattern discovery on 3 signals.

By leveraging on this codification schema, the pattern discovery analysis consists in identifying repeated sequences in the string S, basing on the following definitions.

A pair of substrings $R=((i_1,j_1),(i_2,j_2))$ is a repeated pair if and only if $(i_1,j_1)!=(i_2,j_2)$ and $S[i_1...j_1]=S[i_2...j_2]$. Therefore, the length L of R is j_1-i_1+1 and its frequency F is the number of times that it occurs in the string. A repeated pair $((i_1,j_1),(i_2,j_2))$ is called left maximal if $S[i_1-1]!=S[i_2-1]$ and right maximal if $S[j_1+1]!=S[j_2+1]$. Finally, a repeated pair is called maximal if it is both left and right maximal and a substring Ȧ of S is a maximal repeat if there is a maximal pair $((i_1,j_1),(i_2,j_2))$ such that Ȧ$=S[i_1...j_1]$.

During the pattern discovery phase, an Enhanced Suffix Array (ESA) is built [9]. ESA is obtained by using a commonly employed algorithm (such as SeqAn [5]) and is a fundamental indexing data structure using auxiliary tables to find all the maximal repeats in the string S.

Based on an existing ESA, the ranking phase aims at selecting the maximal repeat that provides the better reduction ratio with respect to the current string S. In this process, a trade off between vertical occurrence length and repetition frequency has to be considered, taking also into account the number of bits available for encoding [5].

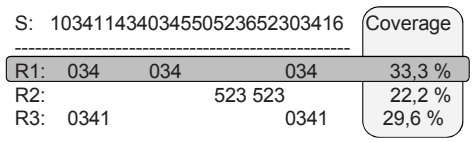

Figure 5. Trade off between vertical occurrence length and repetition frequency

Locally, a longer sequence provides a higher gain ratio than a shorter one because it codifies more time slices with only one occurrence. But globally, as exemplified in figure 5, the shorter sequence R1, repeating more times than R3, can permit a higher coverage with a resulting higher gain.

By these considerations, the ranking process selects the longer maximal repeat whose length is larger than a minimal threshold and whose frequency is greater than a parameter α,

$$\alpha = \frac{1}{n_{mr}} \int_{j=1}^{j=n_{mr}} F_j \qquad (1)$$

In equation (1) F_j is the frequency of the j^{th} maximal repeat and n_{mr} is the number of identified maximal repeats.

The selected maximal repeat is extracted from S during the pruning phase and the corresponding entry is added to the TSO file. The heuristic used to obtain vertical compression is described in the pseudo-code reported in figure 6.

Initially, (1) the test data volume is equal to the length of the string S multiplied by the number of signals n_S it is composed of. A first loop (2) is controlled by the value of a parameter K_V, which is the minimum admitted length for a vertical occurrence and ranges between L_{max} and L_{min}. Any time this loop is iterated, the pattern discovery phase is restarted by reducing the length threshold K_V. A second loop (6) is iterated until no maximal repeats are found larger than L_{min} or a maximum number of vertical occurrences i_{max} is obtained. In this nested loop, (7) pattern discovery resulting in ESA is operated on S_K limited to a threshold M, which was initially set equal to K_V (3), and ESA is submitted to ranking (8). If this sequence does not select any vertical occurrence, the threshold M is decreased by 1 (9), and S_{Kv} re-analyzed under less stringent parameters. Otherwise, (11) $S_{Kv}^{\ i}$ is pruned from the identified V_i. When exiting the internal loop, the obtained reduction ratio (RTDV) is evaluated, and if improved, the sequences configuration saved (14, 15).

```
vertical_compression(){
(1)RTDV_min = length(S)*n_S;
(2)for (K_v = L_Vmax ↷ L_vmin){
(3)       M = K_v;
(4)       i=1;
(5)       S_Kv = S;
(6)       while (M != L_vmin || i!=i_vmax){
(7)                 ESA=pattern_discovery(S_Kv^i,M);
(8)                 V_i = ranking(ESA)
(9)                 if (V_i == NULL) M--;
(10)                else{
(11)                          S_Kv^i+1 = pruning(S_Kv^i, V_i);
(12)                          i++;
                    }
          }
(13)      if (RTDV_Kv < RTDV_min){
(14)                RTDV_min = RTDV_Kv;
(15)                save(V_Kv);
          }
      }
}
```

Figure 6. Vertical occurrence set determination algorithm.

2) Identification and selection of Horizontal occurrences

The process for identification of horizontal occurrences is executed on the string S_V pruned of the vertical occurrences. Similarly to figure 3, the method proposed for automatically identifying horizontal occurrences is composed of three iteratively performed steps:

1. statistical report, identifying for each signal how many times S_V show a constant value for an entire substring
2. statistical analysis, for populating the horizontal occurrence set
3. pruning from S_V of the selected horizontal occurrences.

Each signal is analyzed singularly, counting how many times it remains constant to 0 or 1 for an entire substring. The horizontal occurrences are then identified combining the n_H signal values that fully covers more segments, as described in the pseudo-code reported in figure 7.

```
horizontal_compression(){
(1)for( K_H = L_Hmin ↷ L_Hmax){
(2)       I=1;
(3)       S_KH = S_V;
(4)       n_H = n_s -1;
(5)       while(I < i_max && n_H > 0){
(6)         Stat=report(S_KH^i,K_H);
(7)         H_i=NULL;
(8)         while(H_i == NULL && n_H >0){
(9)                 H_i =analysis(Stat);
(10)                if(H_i == NULL)
(11)                      n_H--;
(12)                else{
(13)                       S_KH^i+1 = pruning(S_KH^i, H_i);
(14)                      i++;
(15)                        H_i=NULL;
                    }
           }
         }
(16)     if (RTDV_KH < RTDV_min){
(17)       RTDV_min = RTDV_KH;
(18)       save(H_KH);
       }
    }
}
```

Figure 7. Horizontal occurrence set determination algorithm.

The horizontal compression depends on the parameter K_H that ranges between L_{Hmin} and L_{Hmax} (1) and represents the admitted length range for a horizontal occurrence. At any iteration, the process is restarted by increasing the length

threshold K_H. An inner loop (5) goes on until no more repetitions are found, or the maximum number of horizontal occurrences i_{max} is obtained. In this loop, statistical analysis is performed on S_{KH} based on parameter K_H (6), and results are analyzed to select horizontal occurrence (10). If no horizontal occurrence exists (10), the number of signals on which occurrences are looked for is decreased by 1 (11), and statistical analysis is performed under less stringent parameters. Otherwise, (13) $S_{KH}{}^{i}$ is pruned from the identified H_i. When exiting the internal loop (5), the obtained reduction ratio is evaluated, and if improved, the horizontal sequences configuration saved (17,18).

B. Low-Cost Tester organization

A suitable Low-Cost tester organization designed to efficiently support the decompression process is shown in figure 8. This schema shows both HW and SW resources required to decompress and apply the vector set to the DUT:

- HW resources: they include the components needed both to manage and perform decompression
  - A FPGA device
    - ③ storing FSMs capable of autonomously reproducing the recurrent test segment parts pruned from the vector set [6]
    - ③ including small and fast dual-port BRAMs
  - A µprocessor in charge of managing the overall decompression procedure
  - A large stand-alone RAM
  - A DMA controller for system bus management
  - Some communication and mass storage peripherals required to transfer data internally
- SW resources: they include the compressed vector set, the tester Operative System and the SW application run by the processor to manage the decompression process.

The architecture of the stimuli generation block was already discussed in [6]. Such a tester architecture enables a very effective method for data management that exploits a two-layer memory organization. In principle,

- the compressed pattern set is completely stored on a secondary, large and slow memory belonging to the Stimuli Controller block
- from the secondary memory the compressed information is moved block-by-block, by means of the DMA controller, to smaller and faster primary RAMs, that in our schema correspond to the FPGA dual-port BRAM blocks [7] in the Stimuli Generator block.

In terms of tester costs versus capabilities, the benefit stemming from the usage of this particular memory organization and from the compression method explained in the previous sections is twofold. A former advantage is that a large and fast, and therefore expensive, primary memory is not required, since data are transmitted block-by-block to be decompressed. A second benefit stems from the fact that the decompression logic mapped on the FPGA can usually run autonomously for many clock cycles without interactions with the rest of the tester. This decompression schema characteristic implies that the frequency required to move compressed data from the secondary to the primary memory may be substantially lower than the decompression frequency.

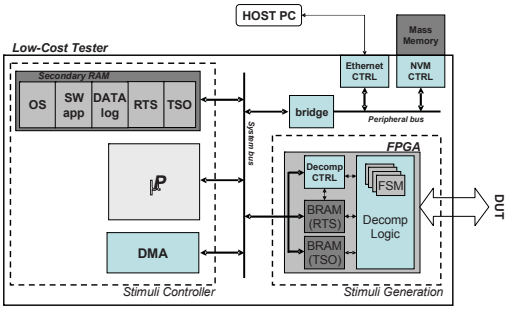

Figure 8. Low-Cost tester architecture.

As an example, let us consider an explanatory scenario where the primary RAM storing TSO words has only one 8 bit location; if this location currently contains a word describing a vertical occurrence, an FSM is activated that reproduces an n clock cycles long sequence at the f frequency. That means the transfer data frequency can be reduced to f/n. Similarly, if a primary RAM storing RTS data has only one 8 bits location, and if 2 bits per fast clock cycle are used to complete an horizontal occurrence, then the requested secondary to primary memory transfer frequency is a quarter of the generation frequency.

More in general, the required average transfer frequency F_{trans} is function of R (compression ratio), B_v (number of bits used to codify a vertical occurrence), R_v (compression ratio obtained only by vertical occurrence pruning), L_v (average length of the vertical occurrences), B_h (number of bits used to codify a horizontal occurrence), L_h (average length of the horizontal occurrences), B (number of bits transmitted from secondary to primary memory per transfer clock cycle), F_{app} (vector application frequency to DUT) and S (number of input and output signal to and from the DUT).

$$F_{trans} = \frac{F_{app}}{B}\left(1-R\right)+\frac{B_v R_V}{L_v}+\frac{B_h(1-R_V)}{L_h} \quad (2)$$

The obtained reduction in the required transfer frequency permits to physically separate the stimuli generator from its controller; this aspect fits the case of probe cards that actually can include FPGA cores. Our technique enables augmenting their ability in terms of stimuli application frequency while mitigating the test driver to probe card communication frequency. Moreover, it allows many stimuli generator blocks to be managed by a single controller.

IV. EXPERIMENTAL RESULTS

The illustrated architectural principles for the FPGA-based low-cost tester were experimentally proved by mapping them on a Digilent XUP development board [8]. This commercial product is equipped with two PowerPC processors connected to a 256MB std-alone DRAM (secondary RAM); the board also includes a FPGA containing about 30k logic cells and 2Mb BRAM blocks (primary RAM) [7]. The system bus available for connecting processors, DRAM and FPGA

resources permits transferring 64 data bits per system clock cycle. The DMA controller was included in the system as a soft-core mapped in the FPGA.

We used the Linux kernel 2.6 as operative system and wrote an assembly procedure to manage/monitor the compressed data to be transferred from the host PC to the secondary RAM via Ethernet connection. The length of this program is about 200 code lines.

Table I: compression ratio

Test program	Original size	Compression ratio
test_01	525 K	77.5%
test_02	551 K	74.2%
test_03	610 K	74.6%
test_04	620 K	73.7%
test_05	634 K	73.9%
test_06	8.1 M	66.5%
test_07	122 K	80.0%
test_08	2.4 M	64.0%
test_09	5.9 M	67.6%
test_10	112 K	82.2%
test_11	133 K	79.1%
test_12	200 K	74.6%
test_13	146 K	78.4%
test_14	419 K	79.6%
test_15	190 K	74.1%
test_16	171 K	76.4%
test_17	236 K	75.2%
test_18	184 K	93.5%
test_complete	**20,75 M**	**69.8%**

The SoC considered as a case study contains several cores, including a processor core, some memory cores equipped with programmable BIST circuitry, and several peripherals. The considered test set, devised to cover stuck-at and transition faults in the processor and peripherals via SBST procedures and exploiting BIST to test memories, consists in 18 test programs managed through an IEEE 1500 driven through JTAG; moreover, the test procedures are additionally supported by a free-running clock. Table I shows the compression ratio [1] obtained on the 18 tests, plus another obtained by collapsing all the test programs into a single one, which finally accounts for 3MVectors, or 20.75 MB.

The compression process applied to this large test set reduced the test data volume to 6.3 MB (2.4 MB for the TSO and 3,9 MB for the RTS file), which corresponds to a compression ratio of 69.8%. In particular, the process identified 128 vertical occurrences (covering 48.5% of the original test set) and 7 horizontal occurrences completing the compression process. Parameter values adopted in the vertical occurrences search are K_V in the 32 to 5 range and 64 for the maximal length of a single vertical segment. For the horizontal occurrences, instead, K_H ranges from 10 to 1.

For the sake of completeness, we compared this result with that obtained by using a Golomb code compression mechanism [2] (using groups of size 16); through this experiment we obtained a compression ratio of 66.5% that is slightly less than the result obtained with our technique. However, it is worth noting that the Golomb code based compression can not be supported by the low-cost tester architecture described above.

The decompression logic obtained automatically at the end of the compression phase was synthesized using Xilinx ISE v.9.0 and mapped on the FPGA of the development board. Its final occupation is 7,079 LUTs and 4,815 FFs and the achieved frequency is 220 MHz. Because of device specifications and FPGA to DUT communication constraints (mainly due to connection within developed tester and daughter board) a final communication frequency of 50MHz was used. With this communication frequency, the average secondary to primary RAM transfer frequency required is 4.4 MHz (19.4 MHz for a 220MHz communication frequency).

This case study thus shows how the size of a large test procedure (~17MB) was significantly reduced (~5MB). The procedure is therefore applied at 50 MHz without any interruption for pattern reloading on a low-cost tester with reduced fast channel memory space (~2MB).

V. CONCLUSIONS

In this paper we proposed an automated data compression/decompression technique suitable for very long sequential test vectors generated to test SoCs composed of cores supporting a BIST-based test by means of low-cost test procedures. The described method performs compression off-line and on-the-fly decompression on an FPGA-based low-cost tester platform. Based on the illustrated compression schema, the characteristics of an FPGA-based low-cost tester platform were detailed.

Feasibility and effectiveness of the described methodology was demonstrated by applying it to a SoC tested by means of SBST and BIST procedures and using a commercial development board including processors, RAM and FPGA resources to implement the tester.

REFERENCES

[1] F. Karimi, et al., "Using data compression in automatic test equipment for system-on-chip testing", IEEE Transactions on Instrumentation and Measurement, Volume 53, Issue 2, April 2004 Page(s):308 – 317

[2] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, No. 3, pp. 355-368, March 2001.

[3] P. T. Gonciari, et al., "Variable-Length Input Huffman Coding for System-on-a-Chip Test," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22, No. 6, pp. 783-796, June 2003.

[4] M. Psarakis, et al., "Systematic software-based self-test for pipelined processors", ACM/IEEE Design Automation Conference, 2006, pp. 393 – 398

[5] A. Döring, D. Weese, T. Rausch, K. Reinert, "SeqAn – An efficient, generic C++ library for sequence analysis", BMC Bioinformatics 2008

[6] P. Bernardi, M. Sonza Reorda, "A novel Methodology for Reducing SoC Test Data Volume on FPGA-based Testers", Design, Automation and Test in Europe, 2008. DATE '08, 2008, pp. 194 - 199

[7] Virtex II Pro Platform FPGAs Complete Datasheet. Available: http://www.xilinx.com

[8] XUP Virtex II Pro Development System HW Reference Manual. Available: http://digilentinc.com

[9] M. I. Abouelhoda, S. Kurtz, E. Ohlebusch, "The Enhanced Suffix Array and its application to Genome Analysis", WABI September 17-21, 2002, pp. 449-463

[10] N.A. Touba, "Survey of Test Vector Compression Techniques", IEEE Design & Test of Computers, July-August 2008, pp. 294-30