Test Compression and Hardware Decompression for Scan-based SoCs

Francis G. Wolff Chris Papachristou
Case Western Reserve University
Electrical Engineering and Computer Science
Cleveland, OH 44106
{fxw12,cap2}@po.cwru.edu

David R. McIntyre
Cleveland State University
Computer & Information Science
Cleveland, OH 44114
mcintyre@csuohio.edu

Abstract. *We present a new decompression architecture suitable for embedded cores in SoCs which focuses on improving the download time by avoiding higher internal-to-ATE clock ratios and by exploiting hardware parallelism. The Bounded Huffman compression facilitates decompression hardware tradeoffs. Our technique is scalable in that the downloadable RAM-based decode table and accommodates for different SoC cores with different characteristics such as the number of scan chains and test set data distributions.*

1 Introduction

Data compression techniques are used to alleviate the ATE test data volume problem. The idea is to compress the precomputed test set T_D provided by the vendor to a smaller test set T_E and store it in the ATE memory. An on-chip decoder is then used to decompress T_E and reproduce T_D. Huffman compression using synthesized or ROM-based decoders has been applied to test compression [2, 3]. Many parallel Huffman architectures have been developed [4] mainly for multimedia.

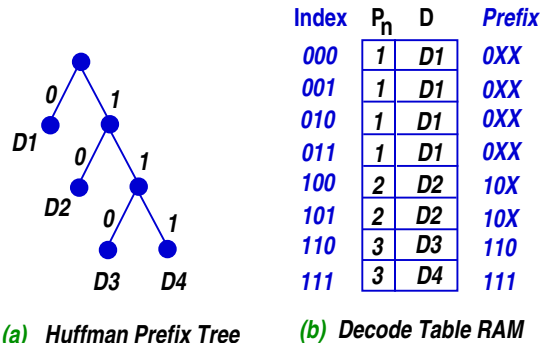
The focus of this work is on an efficient hardware architecture implementing the Bounded Huffman compression scheme [5] for test decompression and is used to reduce the decompression chip area overhead in particular the RAM-based (not ROM-based) decode table. Hardware parallelism is exploited in order to reduce the download time, feed multiple scan chains simultaneously and avoid high internal-to-external clock ratios. The decompressor can directly use the ATE clock in a one-to-one ratio, avoiding the complexity of the frequency multiplication of the internal clock.

We stress that the decompression time is just as important as the compression ratio in testing. Since ATEs test SoCs in real-time, compression is only useful if it reduces the download time or effective bandwidth and thereby reduce the SoC testing time.

2 Decompression Architecture

Our approach applies to embedded cores in a SoC. The major steps of our method are: a) Merging ATPG scan patterns into test character bitstreams; b) assigning Don't Care bits in test cubes using an entropy optimization technique favoring Huffman encoding; c) Huffman-based compression; d) downloading compressed patterns through ATPG channels into scan pins of the SoC; e) hardware Huffman decompression of downloaded patterns and interfacing to the scan chains of the core under test.

In order to decompress, the decompressor requires the decode table [1] which contains the mapping of compress codes to decompressed symbols, followed by the decompressed data. Fig. 1 shows an example of four characters decoded (i.e. D_1, D_2, D_3 and D_4) into four encoded prefix codes (i.e. 0, 10, 110, 111). For example, the prefix code '0' occupies four table locations. The decoder table contains two fields, P_n and D, which represent the prefix length and the decoded character, respectively. The height of the tree, h, represents the maximum prefix code length (i.e. 3).



Index	P_n	D	Prefix
000	1	D1	0XX
001	1	D1	0XX
010	1	D1	0XX
011	1	D1	0XX
100	2	D2	10X
101	2	D2	10X
110	3	D3	110
111	3	D4	111

Figure 1. Example of Decode Tree to RAM mapping

The tradeoff of the decode table is decode time versus table size. The address space of the decode table is equal to the maximum number of bits for the longest prefix code. This means that RAM size is proportional to 2^h and may be impractical for some designs. Bounded Huffman compression allows one to optimally length limit the prefix code whereby allowing the decode table to be smaller.

Fig. 2 shows the decompression architecture. The encoded test data from the ATE feeds the S_E register using k input scan pins. The external ATE clock, C_{ATE} drives the FSM, T_E and Prefix registers. The amount of shift is determined by the current FSM state and the

1530-1591/04 $20.00 (c) 2004 IEEE

currently decoded prefix length, P_n. The Prefix register, P, holds the currently encoded ATE data and must be large enough, P_w to contain the maximum prefix length, A_w, and also acts as buffer when the k is less than A_w. The shifter concatenates P and S_E registers and shifts left in order to advance the encoded data stream. The output of the shifter is fed to the decode table address, A via the mask register. The width of the decode table address, A_w, is selected by the designer to handle the maximum prefix code desired. The mask register, M, uses bitwise ANDing and masks all bits longer than h. This minimizes initialization time of the decode table The decode table RAM outputs the decompressed character, D which feeds data into n-scan chains, T_D and the FSM drives the scan clock, C_{TD}, when D is valid.

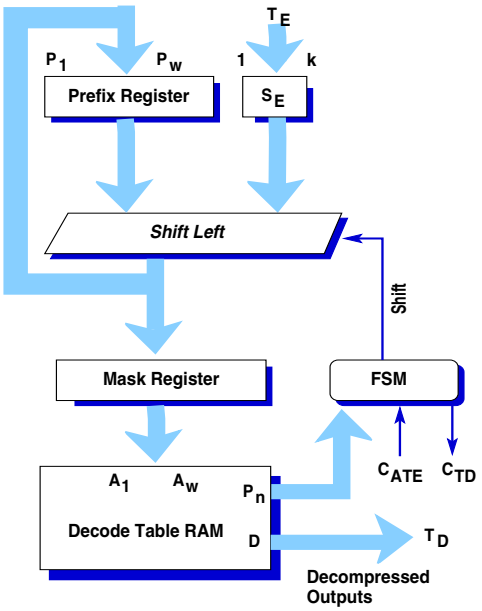


Figure 2. Decompression Architecture

3 Results

Table 1 shows the download times for ITC99 b14 benchmark. The four rows that show the download time in milliseconds for $f_{ATE} = 20MHz$ as a function of k and n. The download time improvements from 51.9ms to 9.8ms as k and n are increased. Clearly, the range of the download time in clock cycles is, $T_E \geq t_{decode} \geq T_E/k$. $T_{compression}$ shows the compression ratio (i.e. $(T_D - T_E)/T_D$) achieved through the combination of Don't Care minimization and Huffman compression. $T_{compression}$ is a function of n but not k. T_E is the total encoded data sent to the decompressor and includes the the decode table overhead. The overhead relative to T_E is shown in the next row. Clearly, the overhead is very small to the size of the data.

The longest encoded character as a result of the optimal Huffman compression algorithm is shown in the row as A_w. The total size of decode table RAM is $2^{A_w} \times (A_w + D_w)$ bits. The RAM word consists of P_n and D and their bits sizes are $(A_w + D_w)$, respectively. Increasing n benefits the compression ratio at the cost of increased area size of decode table RAM.

	Decoded Character Size, D_w			
	$n = 2$	$n = 4$	$n = 8$	$n = 12$
$t_{decode}(k = 1)$	**51.9 ms**	30.8	23.5	18.4
$t_{decode}(k = 2)$	48.2	26.1	18.4	13.6
$t_{decode}(k = 4)$	46.8	24.4	16.4	10.9
$t_{decode}(k = 8)$	46.8	23.4	15.3	**9.8 ms**
$T_{compression}$	45%	67%	78%	82%
T_E	1037264	616622	404804	334796
T_E overhead	0.00%	0.02%	0.46%	2.04%
A_w	3	9	17	16

Table 1. ITC99 b14 benchmark download times

By using Bounded Huffman we can tradeoff a small loss of compression for a reduction of RAM overhead. For example, Table 2 shows the effect of limiting the huffman code length for the case $n = 4$ by using a length-limited huffman algorithm [5]. For the $A_w = 5$ case, we can reduce the RAM size requirements from 1024 words to 64 words. Thus, it is possible to reduce the RAM requirements or at the loss of some compression ratio match the existing RAM size for all. This ability to limited the Huffman length is due to the non-uniform distribution of the test set codes themselves.

	Bounded Huffman length, A_w				
	9	8	7	6	5
T_{comp}	67.06%	67.05%	66.97%	66.42%	64.52%
RAM	1024	512	256	128	64
t_{decode}	23.4	23.406	23.407	23.408	23.41

Table 2. Decode table RAM size reduction for $n = 4$ and $k = 8$

References

[1] D. S. Hirschberg and D. A. Lelewer. Efficient decoding of prefix codes. *Commun. ACM*, 33(4):449–459, 1990.

[2] H. Ichihara, A. Ogawa, T. Inoue, and A. Tamura. Dynamic test compression using statisical coding. In *Proc. of Asian Test Symposium*, pages 143–148, 2001.

[3] A. Jas, J. Ghosh-Dastidar, Ng, and N. A. Touba. An efficient test vector compression scheme using selective huffman coding. In *IEEE Trans. on CAD*, number 6, pages 797–806, June 2003.

[4] B.-J. Shie and C.-Y. Lee. An efficient vlc decompression scheme for user-defined coding tables. Vol. 4, pp. 1961-1964, *ICASSP '99*.

[5] A. Turpin and A. Moffat. Practical length-limited coding for large alphabets. *The Computer Journal*, 38(5):339–347, 1995.