International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Run length encoding and bit mask based Data Compression and Decompression Using Verilog

S.JAGADEESH [1], T.VENKATESWARLU [2], DR.M.ASHOK [3]

[1]*Associate Professor & HOD, Department of Electronics and Communication Engineering, Sri Sai Jyothi Engineering College, Gandipet, Hyderabad-75, (A. P.), India*

[2] *P.G. Student, M.Tech. (VLSI), Department of Electronics and Communication Engineering, Sri Sai Jyothi Engineering College, Gandipet, Hyderabad-75, (A. P.), India,*

[3] *Professor, Department of Computer Science and Engineering, Sri Sai Jyothi Engineering College, Gandipet, Hyderabad-75, (A. P.), India,*

Abstract

Higher circuit densities in system-on-chip designs have led to drastic increase in test data volume. Larger test data size demands not only higher memory requirements, but also an increase in testing time. Test data compression addresses this problem by reducing the test data volume without affecting the overall system performance. The major contributions of this paper are as follows: 1 it develops an efficient bitmask selection technique for test data in order to create maximum matching patterns; 2 it develops an efficient dictionary selection method which takes into account the bitmask based compression; and 3. it proposes a test compression technique using efficient dictionary and bitmask selection to significantly reduce the testing time and memory requirements. If the bit-stream contains consecutive repeating bit sequences, the bitmask-based compression encodes such patterns using same repeated compressed words, whereas our approach replaces such repetitions using a bitmask of "00". In this example, the first occurrence will be encoded as usual; whereas the remaining repetitions will be encoded using our method i.e. run length encoding of these sequences may yield a better compression result. Interestingly, to represent such encoding no extra bits are needed. Note that bitmask value 0 is never used, because this value means that it is an exact match and would have encoded using zero bitmasks. Using this as a special marker, these repetitions can be encoded without changing the code format of bitmask-based compression.

Keywords:bitmasking,dictionary,encoding,compression,decompression

1.Introduction

In system on chip designs, higher circuit den larger memory requirement in addition to an increased testing time. data compression plays a crucial role, reducing the testing time and memory requirements.it proposed a new run length encoding and bit mask based data compression and decompression .In this paper solve to the bit mask selection technique for test data.it develop dictionary selection method.by using these two techniques solve the data compression.in dictionary method we can solve only direct match valus. By using bit mask selection paper solve the dictionary not find out the valus.so tese paper using compression in this way.more number of bits repeated same.now we can modify the compress data.when it occurs we can solve one postion. Then it occurs all compressed data.remaining data not necessary.then totally data and decompressed. . Using this as a special marker, these repetitions can be encoded without changing the code format of bitmask-based compression.

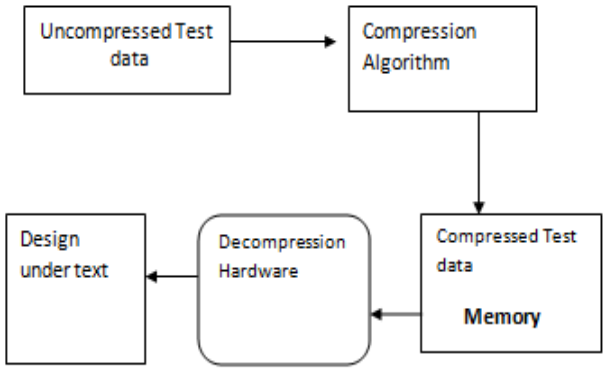


Figure 1.1 Block diagram for test data pattern.

2.BACKGROUND OF THE PROJECT

In this describes bitmask-based code compression,and highlight of the bitmasking As seen in Fig. 2, we can compress up to six data entries using bitmask based compression. The compressed data is represented as follows Those vectors that match directly are compressed with 3 bits. The first bit represents whether it is compressed (using 0) or not (using 1). The second bit indicates whether it is compressed using bitmask (using 0) or not (using 1).

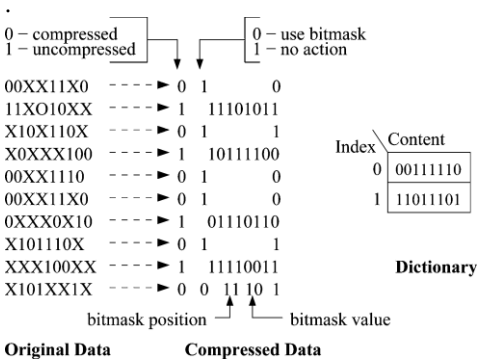


Figure 2 Bitmasked data compression

The last bit indicates the dictionary index. Data that are compressed. using bitmask requires 7 bits. The first two bits, as before, rep resent if the data is compressed, and whether the data is com pressed using bitmasks. The next two bits indicate the bitmask position and followed by two bits that indicate the bitmask pat tern. For example, the last data vector in Fig. 2 is compressed. using a bitmask. The bitmask position is 11, which indicates

fourth even bit position from left. For this case, we have as sumed fixed bitmasks, which are always employed on even-bit positions and hence only 2 bits are sufficient to represent the four positions in a 8-bit data. The last bit gives the dictionary index. The bitmask XOR with the dictionary entry produces Compressed Test data

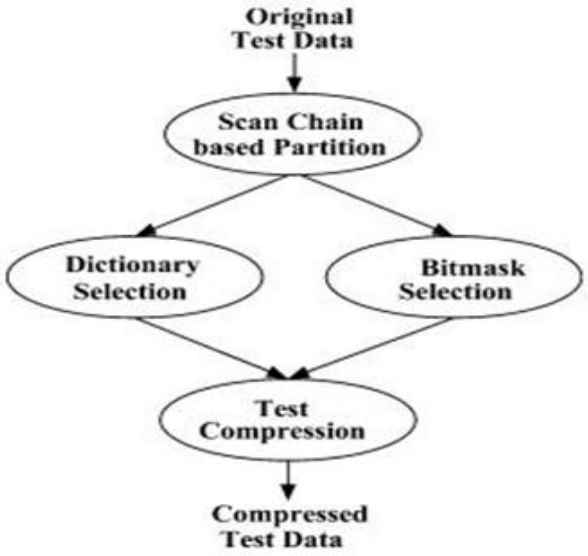


Fig. 3. Bitmask-based test data compression

Division of Test Data into Scan Chains

Once the input test data is considered, next task would be to divide them into scan chains of predetermined length. Let's assume that the test data consists of N test patterns. Divide the scan elements into m scan chains in the best balanced manner possible. This results in each vector being divided into m sub-vectors, each of length, say l. Dissimilarity in the lengths of the sub-vectors are resolved by padding don't cares at the end of the shorter sub-vectors. Thus, all the sub-vectors are of equal length. The m-bit data which is present at the same position of each sub-vector constitute an m-bit slice. If there are vectors at the beginning, a total of n*l m-bit slices is obtained, which is the uncompressed data set that needs to be compressed. Consider a simple example consisting of two test patterns 0011 and 1XXX for a design with two scan chains. Therefore, length of each sub-vector is l. In this case, padding of don't cares is not required. Figure 3.2 shows how four slices (XX, 1X, 01, and 01) can be formed with two vectors (001X and 11XX) obtained by scan chain based partitioning of the two original test patterns. These are the four slices that need to be compressed.

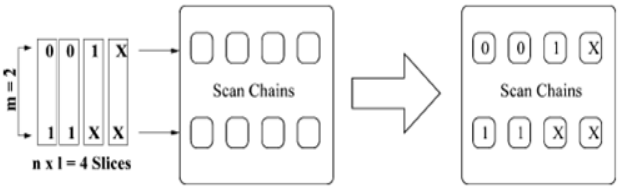


Figure 4: Division of Test Data into Scan Chains

Bit Mask Selection

A "fixed" bit-mask pattern implies that the pattern can be applied (starting position) only on fixed locations. For example, an 8-bit fixed mask (referred as 8f) is applicable on 4 fixed locations (byte boundaries) in a 32-bit vector. A "sliding" bit-mask pattern can be applied anywhere. For example, an 8-bit sliding mask (referred as 8s) can be applied in all locations on a 32-bit vector. There is no difference between fixed and sliding for a 1-bit mask. In this thesis, use of 1-bit sliding mask (referred as 1s) for uniformity.The number of bits needed to indicate a location will depend on the mask size and the type of the mask. A fixed mask of size s can be applied on (32 ÷ s) number of places. For example, an 8-bit fixed mask can be applied only on four places (byte boundaries) and requires 2 bits.

Table 1: Various Bit-Mask Patterns

Bit-mask	Fixed	Sliding
1 bit		X
2 bit	X	X
3 bit		X
4 bits	X	X
5 bits		X
6 bit		X
7 bit		X
8 bit	X	X

Similarly, a 4-bit fixed mask can be applied on eight places (byte and half-byte boundaries) and requires 3 bits for its position. A sliding pattern will require 5 bits to locate the position regardless of its size. For instance, a 4-bit sliding mask requires 5 bits for location and 4 bits for the mask itself

Table 2 Profitable Bit-Mask Patterns

Bit-Mask	Fixed	Sliding
1 bit		X
2bits	X	X
8bit	X	x

A careful study of the combinations of up to two bit-masks using various applications compiled on a wide variety of architectures. Analyzed result of compression ratios on various mask combinations and observed that 8f and 8s are not helpful and also observed that 4s does not perform better than 4f. The final set of bit-mask patterns are shown in Table 3.3.

Table 3 Final Bit-Mask Sets

Bit-mask	Fixed	Sliding
1 bit		X
2 bits	X	X
4bits	x	

Figure 3.3 shows the encoding format for considering mismatches. A compressed data stores information regarding the mask type, mask location and the mask pattern itself. The

mask can be applied on different places on a vector and the number of bits required for indicating the position varies depending on the mask type. For instance, consider a 32-bit vector, an 8-bit mask applied on only byte boundaries requires 2-bits, since it can be applied on four locations. With no restrict, the placement of the mask, it will require 5 bits to indicate any starting position on a 32-bit vector

Itmasks may be sliding or fixed. A fixed bit mask always operates on half-byte boundaries while a sliding bitmask can operate anywhere in the data. It is obvious that generally sliding bitmasks require more bits to represent themselves compared to fixed bitmasks. In this thesis, the alphabets `s` and `f` are used to represent sliding and fixed bitmasks respectively. The optimum bitmasks to be selected for test data compression are 2s, 2f, 4s and 4f. However, in the last two need not be considered. This is because as per Lemma 1, the probability that 4 corresponding contiguous bits will differ in a set of test data is only 0.02%, which can easily be neglected. Thus, the compression is performed by using only 2s and 2f bitmasks. The number of masks selected depends on the word length and the dictionary entries and is found out using Lemma 2.

Lemma 1: The probability that 4 corresponding contiguous bits differ in two test data is 0.2 %.

Proof: For two corresponding bits to differ in a set of test data, none of them should be don't cares. Consider the scenario in which they really differ, and find out the probability of such an event. It can beseen that any position in a test data can be occupied by 3 different symbols, 0, 1 and X. However, as already mentioned, to differ, the positions should be filled up with 0 or 1.Hence, the probability that a certain portion is occupied by either 0 or 1 is 2/3 = 0.67. Therefore, the probability that all the four positions have either 0 or 1 is P1 = (0.67)4 = 0.20.

For the other vector, the same rule applies. The additional constraint here is that the bits in the corresponding positions are fixed due to difference in the two vectors, that is, the bits in the second vector has to be exact complement of those of the first vector.

Therefore, the probability of occupancy of a single position is 1/3 = 0.33 Therefore, the probability of 4 mismatches in the second vector P2 = (0.33)4 = 0.01 The cumulative probability of the 4-bit mismatch is a product of the two probabilities P1 and P2 and is given by:P = P1 X P2 = 0.2 %

Lemma 2: The number of masks used is dependent on the word length and dictionary entries.

Proof: Let L be the number of dictionary entries and N be the word length. If y is the number of masks allowed, then in the worst case (when all the masks are 2s), the number of bits required is,and this should be less than N. The first two bits are required to check whether the data is compressed or not, and if

compressed, mask is used or not. So, the maximum number of bitmasks allowed is

$$y = \frac{N-2-\log(L)}{2+\frac{\log(N)}{\log(2)}} - \frac{\frac{\log(y)}{\log(2)}}{2+\frac{\log(N)}{\log(2)}}$$

It is not easy to compute y from here since both sides of the equation contain y related terms. To ease our calculation, we can replace the y-related term on the right hand side of the equation with a constant. It is to be noted that since y<N, a safe measure would be to use 1 as this constant. Therefore, the final equation for y is

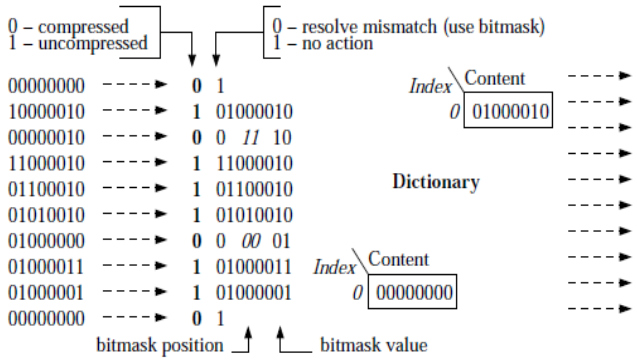
$$y = \left(\frac{N-2-\log(L)}{2+\frac{\log(N)}{\log(2)}} - 1\right)$$

:

Dictionary Selection Method

Dictio nary selection is a major challenge in code compression

Figure 5 compression using bitmasks



A graph is drawn with nodes, where each node signifies a -bit test vector. An edge is drawn between two nodes when they are compatible. Two nodes are said to be compatible if they meet any one of the following two requirements: For all positions, the corresponding characters in the two vectors are either equal or one of them is a don't care; or 2) Two vectors can be matched by predetermined profitable bit masks. Each edge also contains weight information. The weight is determined based on the number of bits that can be saved by using that edge (direct or bit mask-based matching). Based on this graph model**, three dictionary selection techniques are developed**: Two-step method (TSM); Method using compatible edges (MCE) without edge weights;MCE with edge weights (MEW).

Each of these techniques uses a variant of well-known clique partitioning algorithm. The remainder of this section describes these three techniques in detail.**1.Two-Step Method:** In TSM, only edges that are formed by direct matching. In other words are considered, the graph will not have any edges corresponding to bit mask based matching. Then a clique partitioning algorithm is performed on the graph. This is a heuristics-based procedure that selects the node with the largest connectivity and is entered as the first entry to a clique. Now, the nodes connected with it are analyzed, and the node having the largest connectivity among these (and not in the entire graph) is selected. This process is repeated until no node remains to be selected. The entries of the clique are deleted from the graph. The algorithm 1 is repeated until the graph becomes empty. The clique partitioning algorithm is used in MCE and MEW as well..**Method Using Compatible Edges (MCE) Without Edge Weights**: In MCE, weight of all the edges (direct or bit mask- based match) are considered equal. A clique selection algorithm is then performed in the same way as discussed..**MCE With Edge Weights (MEW):** MEW is same as MCE except that consider edge weights are taken. As indicated earlier, the edge weight is determined based on the number of bits saved if that edge is used for direct or bit mask-based matching. Since a predefined number of dictionary entries are taken, two possibilities may arise. The number of cliques selected may be greater than the predefined number of entries or vice versa. In the latter case, it is just need to fill in the dictionary entries with those obtained from clique partitioning. However, if the number of cliques is larger, it is to select the best dictionary entries as illustrated in Algorithm 1 by considering maximum overall savings using both frequency and bit masks.The three methods are illustrated by using the example test data set in Table 3-4. The resultant graph is shown in Figure 3.4. The straight lines in the graph indicate a direct match while the dotted lines signify a match by applying one bit mask. Obviously, the dotted lines will be absent in case of TSM. The dotted lines will have the same weight as the straight lines for MCE. However, they will have different weights in case of MEW. In case of MEW, the weight is determined based on the number of bits saved by using that edge (direct or bit mask match). TABLE.4: Test Data Sets

Data set	Entry
1	11X001XX01100110
2	1X00X10101011100
3	0X1010101X0110X1
4	XXXXX1110010011
5	101X101X0101XX1
6	1111100000XXXX1
7	1100XXXX10001X1
8	111000XXXX1XX11
9	1010X101X101X100
10	1100X010X1010X11

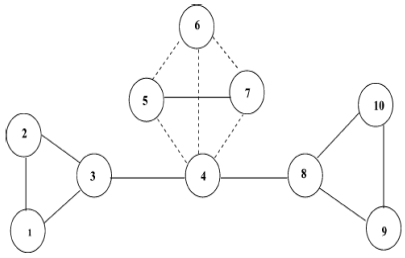


Figure 6 Graph model for test data in TABLE 3.4

C. Run Length Encoding of Compressed WordsThe configuration bitstream usually contains consecutive repeating bit sequences. Although the bitmask-based compression encodes such patterns using same repeated compressed words, it is suggested in [2] and [4] that run length encoding of these sequences may yield a better compression result. Interestingly, to represent such encoding no extra bits are needed. Note that bitmask value 0 is never used, because this value means that it is an exact match and would have encoded using zero bitmasks. Using this as a special marker, these repetitionscan be encoded without changing the code format of bitmask-based compression. Fig. 4 illustrates the bitmask-based RLE. The input contains word "00000000" repeating five times. In normal bitmask-based compression these words will be compressed with repeated compressed words, whereas our approach replaces such repetitionsusing a bitmask of "00". In this example, the first occurrence will be encoded as usual, whereas the remaining 4repetitions will be encoded using RLE. The number of repetition is encoded as bitmask offset and dictionary bits
combined together. In this example, the bitmask offset is "10"and dictionary index is "0". Therefore, the number of repetition will be "100" (i.e., 4). The compressed words are run length encoded only if the RLE yields a shorter code length than the original bitmask encoding. In other words, if there are repetitions of code with length and the number of bits required to encode them using RLE is bits, RLE is used only if bits. Since RLE is performed independently, the bit savings calculation during dictionary selection (see Section IV-B) should be modified accordinglyto model the effect of RLE.D. Decode-Aware Placement of Compressed BitstreamsThe placement algorithm places all bitmask codes in the memory so that they can be decompressed using the efficient

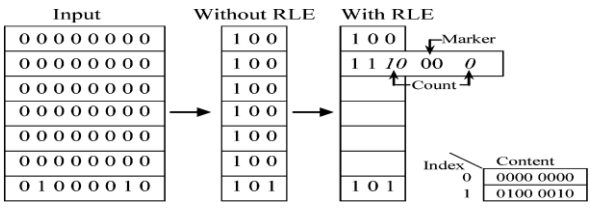


Figure 7 modified data

3.2.4 Run Length Encoding

If the bit-stream contains consecutive repeating bit sequences, the bitmask-based compression encodes such patterns using same repeated compressed words, whereas our approach replaces such repetitions using a bitmask of "00". In this example, the first occurrence will be encoded as usual; whereas the remaining repetitions will be encoded using our method i.e. run length encoding of these sequences may yield a better compression result. Interestingly, to represent such encoding no extra bits are needed. Note that bitmask value 0 is never used, because this value means that it is an exact match and would have encoded using zero bitmasks. Using this as a special marker, these repetitions can be encoded without changing the code format of bitmask-based compression.

3.5 Efficiency For Compression and Decompression

Input	previous	modified
00XX11X0	0 1 0	0 1 0
11X010XX	1 11X010XX	1 11X010XX
X00X110X	0 1 1	0 1 1
00XX1110	0 1 0	0 1 0
X0XXX100	0 0 11 10 0	0 0 11 10 0
X0XXX100	0 0 11 10 0	0 0 01 00 1
X0XXX100	0 0 11 10 0	
X0XXX100	0 0 11 10 0	
X0XXX100	0 0 11 10 0	
X001XX1X	0 0 01 10 0	0 0 01 10 0

Figure 8: Run Length Encoding For compression

The above figure shows Efficiency for compression and decompression.when we are given 10 vectors. And each vector length 8 bits.so totally you are given 80 bit input data. Using previous method we can reduced only 20 bits i.e 60 bits are there. The same data repeted contuniesly then we need not check each and every thing.Using modified

method we can reduced half of the input data. when efficiency high the data size automatically decresed.

Decompression Technique.

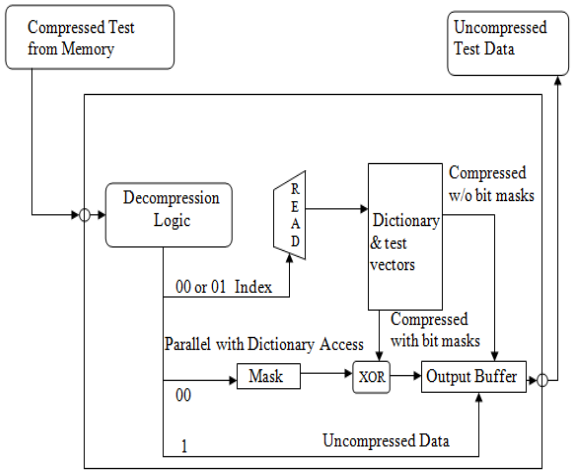


Figure 9 Decompression engine

The design of a decompression engine (DCE), shown in Figure.3.7, that can easily handle bit masks and provide fast decompression. The design of our engine is based on the one cycle decompression engine proposed by Seong et al. The most important feature isthe introduction of XOR gate in addition to the decompression scheme for dictionary based compression. The decompression engine generates a test data length bit mask, which is then XOR ed with the dictionary entry. The test data length bit mask is created by applying the bit mask on the specified position in the encoding. The generation of bit mask is done in parallel with dictionary access, thus reducing additional penalty. The DCE can decode more than one compressed data in one cycle.The decompression engine takes the compressed vector as input. It checks the first bit to see whether the data is compressed. If the first bit is "1" (implies uncompressed), it directly sends the uncompressed data to the output buffer. On the other hand, if the first bit is a "0", it implies this is a compressed data. Now, there are two possibilities in this scenario. The data may be compressed directly using dictionary entry or may have use bit masks. The decompression engine will operate differently in these two cases.

State machine control

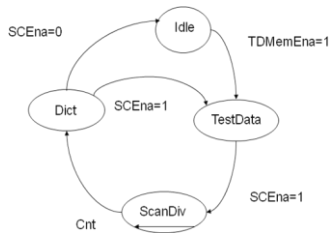


Figure 10 state machine control

Figure 10shows the state diagram of Compression Technique. Idle state is the initial state. When the rst, clk signal are active, then for the next state transition it checks the test data memory is enable, when TDMemEna is high then the test vectors from a text file is copied to a buffer. When SCEna is high, the vectors are fed to the scan chain div block. Until Cnt is less than or equal to number of test vectors, it will be in this state only.

Experimental results

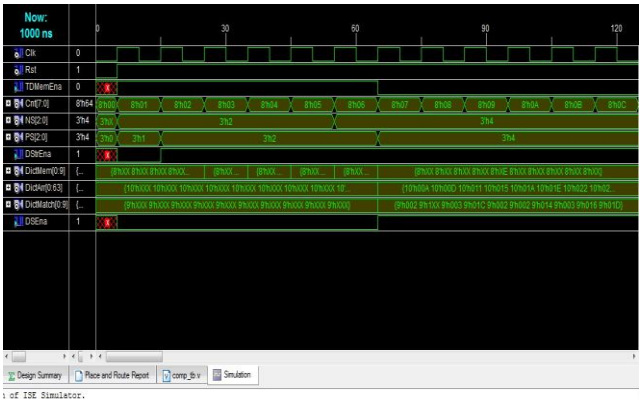


Figure 11:**Simulation for compression**

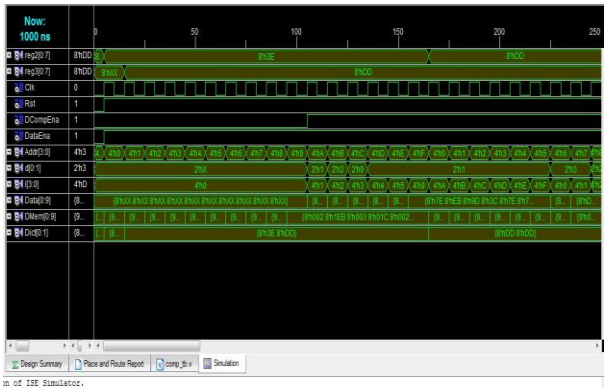


Figure 11 **Simulation for decompression**

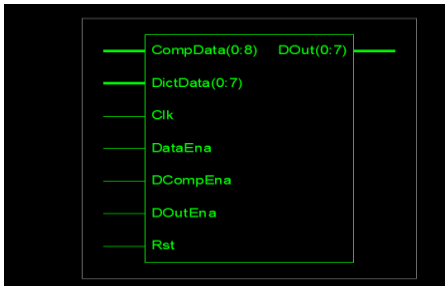
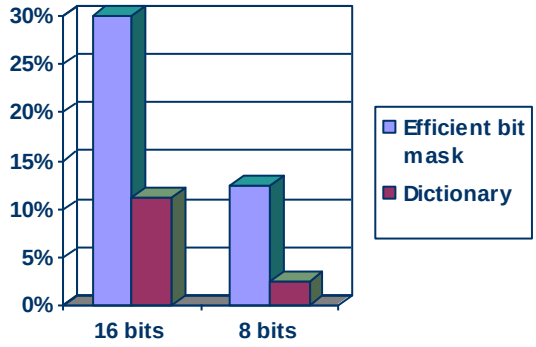


Figure 12 RTL view for decompression

Efficiency for compression and decompression result



CONCLUSION

Using dictionary and bit masking methods we compress the data and also incress the efficiency of data i.e,data will be compressed.we have applied our algoritham on various benchmarks and compared our results with existing test compression technique.

References

[1] S. Hauck and W. D. Wilson, "Runlength compression techniques forFPGA configurations," in *Proc. IEEE Symp. Field-Program. CustomComput. Mach.*, 1999, pp. 286–287

[2] L. Li, K. Chakrabarty, and N. A. Touba, "Test data compression usingdictionaries with selective entries and fixed-length indices," *ACMTrans. Des. Autom. Electron. Syst.*, vol. 8, no. 4, pp. 470–490, 2003.

[3] J. Nikara, S. Vassiliadis, J. Takala, and P. Liuha, "Multiple-symbol paralleldecoding for variable length codes," *IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst.*, vol. 12, no. 7, pp. 676–685, Jul. 2004.

[5] S. Seong and P. Mishra, "Bitmask-based code compression for embeddedsystems," *IEEE Trans. Comput.-Aided Des. Integr. CircuitsSyst.*, vol. 27, no. 4, pp. 673–685, Apr. 2008.