Compression of FPGA Bitstreams Using Improved RLE Algorithm

P.Hemnath V.Prabhu

Abstract - FPGA are configured using bitstreams often loaded from memory. FPGA often called as reconfigurable design because it lower the memory requirements, reduces the bitstreamsize. Some techniques are not suitable for real time decompression. There is need to design a compression technique which efficiently reduces bitstream size meanwhile keeping decompression ratio minimum. In our technique there are some major part of work which are more important they are 1) smart arrangements of the compressed bits that can significantly decreases the overhead of decompression engine 2) combination of bitmask-based compression and run length encoding of repetitive patterns 3)selection of profitable parameter for bitstream compression.The proposed techniques outperforms the compression ratio of existing techniques by 5-15% and decompression hardware is capable of operating at 200M H Z.

Index Terms - Field-programmable gate array,Runlength encoding, Bitmask-based compression , bitstream compression,configurable logic block(CLB),Bitmask selection,Dictionary selection.

1.INTRODUCTION

Field programmable gate array (FPGA)-based embedded systems can sustain high processing rates while providing a high degree of flexibility required in dynamically changing environments. To measure the efficiency of bitstream compression, compression ratio (CR) is widely used parameter. Compression ratio is given as the ratio between the compressed bitstream size (CS) and the original bitstream size (OS) i.e., CR=CS/OS. Therefore, a smaller compression ratio implies a better compression technique. There are two major challenges in bitstream compression: 1) how to compress the bitstream as much as possible and 2) how to efficiently decompress the bitstream without affecting the reconfiguration time. Fig1 shows the traditional code compression and decompression flow where the compression is done off-line and the compressed program is loaded into the memory. The decompression is done during the program execution .The decompression hardware decodes and transfers the compressed bits from memory to configuration hardware which is then transferred to configurable logic blocks(CLB)memory. The existing bitstream compression techniques can be classified into two categories. first category have good compression ratio due to complex and variable-length coding. They also need expensive decompression hardware, which may not be acceptable for practical implementation. The second category of compression approaches accelerate decompression using fixed-length coding and therefore have very efficient decompression hardware. Here the compression ratios are usually compromised. Among the various compression techniques that has been proposed , application of bitmask-based compression are more attractive for bitstream compression, because of its good compression ratio and its simple decompression scheme.

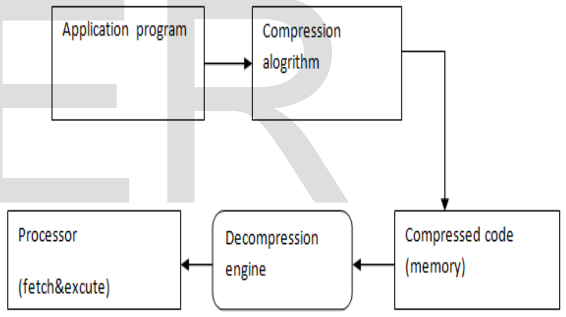

Fig1 Code compression overview

2. RELATED WORK

There are large number of compression algorithms that can be used to compress configuration bitstreams. These techniques can be classified into two categories based on how the redundancies are exploited: format specific compression and generic bitstream compression. The compression techniques in the first category exploit the local redundancies in a single or multiple bitstreams by reading back the configured data and storing the differences by performing exclusive-OR(XOR) operation. These algorithms requires FPGA to support partial reconfiguration and frame read back functionality. Pan et al. [1] uses frame reordering in the absence of read back facility on FPGA. In this technique frames are reordered such that the similarity between subsequent frames configured is maximum. The difference between consecutive frames is then encoded using either Huffman based run length encoding or LZSS based compression. Another method proposed in the same article organizes and read back the configured frames. The frames are organized such that compressed bitstream contains minimal number of difference vectors and maximal read back of configured frames thus reducing the compressed frames significantly. Such complex encoding schemes tend to produce excellent compression ratio.

3. RUNLENGTH ENCODING

A variation of Run-Length encoding perfectly meets the requirements for the address compression. A series of addresses with a common offset can be compressed into a code word of the form: base, offset, length. Base is the base address, offset is the offset between addresses, and length is the number of addresses beyond the base with the given offset. For example, the following sequence of addresses:100, 103, 106, 109, 112 can be compressed into the codeword: base = 100, offset = 3, and length = 4. This compression technique does not require repetitive data, and will take advantage of the sequences of addresses sharing a common offset. The configuration data sometimes repeats data values many times. For this reason, we will attempt to compress the data streams with Run-Length encoding as well, although the compression may not be as great as that achieved with the addresses.

3.1 Runlength hardware

The Run-Length hardware is shown in Fig 2. It consists of a register to hold the current address to output; a down counter to count the length; an adder, to add the offsets to the previous value; and a mux to choose between a previous valued added to the offset and the new base value. when the down-counter equals zero mux chooses the output of the base register. When a new code word arrives, the base value is written into the address register at the same time that the length is written into the down-counter. The down-counter then counts down until zero, while the address register captures its previous value plus the offset.

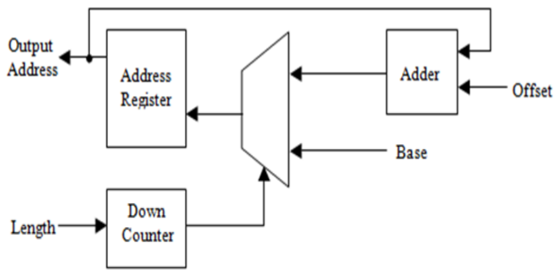

Fig 2 Run-Length hardware support

4.DECODE-AWAREBITSTREAM COMPRESSION

Fig. 3 shows our decode-aware bitstream compression framework. By combination of the bitmask-based compression and run length encoding (RLE) the compressed bitstream is obtained. Next step to decode, our decode-aware placement algorithm is used to place the compressed bitstream in the memory for efficient decompression. When running, the compressed bitstream is transmitted from the memory to the decompression engine, finally original bitstream is produced by decompression.

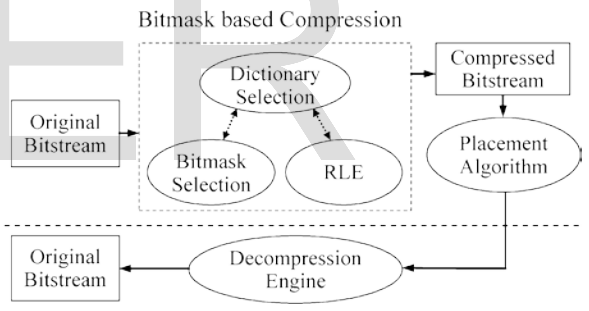

Fig3 Decode-aware bitstream compression framework

There are mainly four important steps in our decode-aware compression framework as shown in algorithm1 1) bitmask selection; 2) dictionary selection; 3) RLE compression; and 4) decode-aware placement. The input bitstream is first divided into a sequence of symbols with length of w. Then bitmask patterns and dictionary entries used for bitmask-based compression are selected as described in Section 4-1 and Section 4-2. Next, the symbol sequence is compressed using bitmask and RLE. The RLE compression in our algorithm is discussed in Section 4-3. Finally, we place the compressed bitstream into a decode friendly layout within the memory using placement algorithm in Section 4-4.

Algorithm 1 Decode-Aware Bitstream Compression
Input: bitstream

Output: Compressed Bitstream placed in memory

Step 1 : Divide input bit stream in Fixed size symbols

Step 2 : Perform Bitmask based pattern selection

Step 3 : Perform Dictionary Selection

Step 4 : Compress symbol into code sequence using bitmask and RLE

Step 5 : Perform decode aware placement of code

4.1 Bitmask selection

There are mainly three types of encoding formats are used in our compression . Fig. 4 shows the formats in these cases: without compression compression, compression using dictionary, and compression using bitmask. The selection of bitmask plays an important role in bitmask-based compression. Generally, there are two types of bitmask patterns. First One fixed bitmask, which can only be applied on fixed positions in a symbol. The other one is "sliding" bitmask, which can be applied at any position. For example, a 2-bit fixed bitmask ("2f" bitmask) is restricted to be used on even locations, but a 2-bit sliding bitmask ("2s" bitmask) can be used anywhere. Clearly, fixed bitmasks require less bits to encode its location, but they can only match bit changes at fixed positions. In other words, only a few number of bitmask patterns or their combina- tions are profitable for compression.The sliding bitmasks are more flexible, but consume more bits to encode. The profitable bitmask patterns that we use in our compression is(1s,2s,2f,3s,3f,4f,4s).

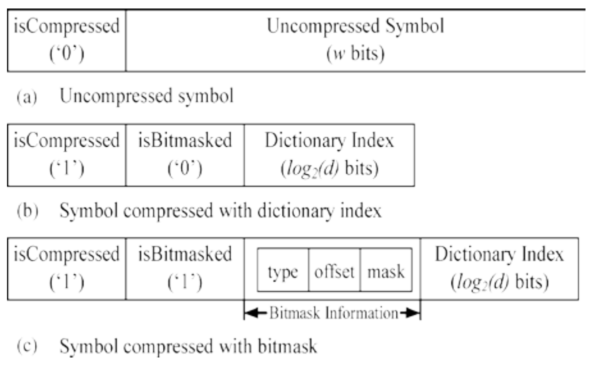
isCompressed ('0')
Uncompressed Symbol (w bits)
(a) Uncompressed symbol
isCompressed ('1')
isBitmasked ('0')
Dictionary Index ($log_2(d)$ bits)
(b) Symbol compressed with dictionary index
isCompressed ('1')
isBitmasked ('1')
type
offset
mask
Dictionary Index ($log_2(d)$ bits)
Bitmask Information
(c) Symbol compressed with bitmask

Fig4 Encoding formats in bitmask-based compression. (a) Uncompressed symbol. (b) Symbol compressed with dictionary index. (c) Symbol compressed with bitmask

4.2 Dictionary selection

The dictionary selection is majorally governed by a words capability to match other words using minimal number of bit masks and covers as most of the input words. The input is divided into unique words with each word associated with frequency (fi). A graph (G) is created in which each vertex represents word with frequencies as its weight. Two vertices are connected via an edge if the two words represented by them can be bit masked with using at most all the bitmasks in B. Each edge (u, v) will have the number of bitmasks used to match vertex u and vertex v as its weight. The savings made for each vertex is calculated based on the sum of savings made by itself in the dictionary and savings made by bitmask matching with other vertices indicated by the incident edges on it.

4.3 Run-length encoding of compressed word

Analysis of the bitstream pattern revealed that the input bitstream contained consecutive repeating patterns of words. The algorithm proposed in previous section will encode such patterns using same repeated compressed words. Instead we use a method in which repetition of such words are run length encoded (RLE). Such repetition encoding will result in an improvement in compression performance by around 10-15%.To represent such encoding no extra bits are needed; another interesting observation leads to the conclusion that bitmask 0 is never used, because this value means that it was an exact match and would have encoded using zero bitmasks. Using this as a special marker, these repetitions can be encoded. This smart encoding will reduce the extra bit that is required to indicate on all the compressed words otherwise.Another advantage of such run length encoding is that it alleviates the decompression overhead by providing the decompressed word instantaneously to the decoder to sendit to the configuration hardware in the same cycle. This ensures the full utilization of the configuration hardware bandwidth and reduces the bottleneck on communication channel between memory and decoder. Fig 5: illustrates the RLE bitmask in use. The compressed words are run length encoded only if the savings made by RLE word encoding is greater than the actual encoding. That is if there are r repetition of compressed words and cost of representing each word is x bits and the number of bits required to encode run length is l bits then RLE is used only if $x * r < l$ bits.

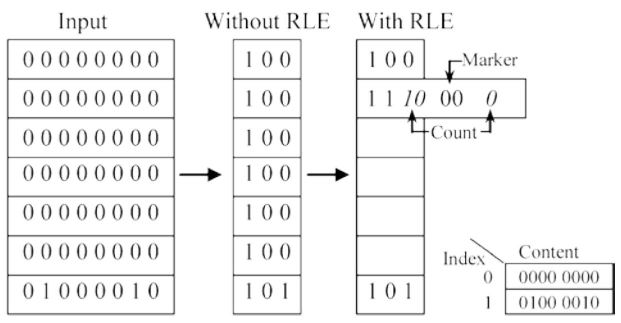

Fig 5 RLE-basedcompression

4.4 Decode-aware placement of compressed bitstreams

The basic idea is to split the original single VLC bitstream into multiple FLC bitstreams for storage. During decompression, these FLC bitstreams are buffered separately, then used to reconstruct the original bitstream by bitmask decoding. Since the buffering circuitry for FLC bitstreams is much simpler than that of VLC bitstreams, the overall decompression performance will be improved even when multiple FLC buffers are used. We will use “power-two n-bit stream,” in remaining sections. Then we describe how we split the original compressed bitstream into multiple FLC bitstreams and how to place these FLC bitstreams into the memory in such a way that the original bitstream can be reconstructed during decompression.

5. DECOMPRESSION ENGINE

The diagram of our decompression engine for 8-bit memory is shown in Fig. 6 Barrel Shifter (BBS) is replaced by an Assemble Buffer with a Left Shifter Array (ABLSA) .The basic working principle of ABLSA is to use an array of left shift registers to buffer the power-two bit streams separately. Since the code length in bitmask-based compression is uniquely determined by the first two bits of a code , we can easily get the length of a code by checking of front bits of stream CS and BS. Then, the shift register that hold bits of the code is identified based on the binary representation of the code length. Finally, the original code is assembled in the assemble buffer and fed to the RLE decoders. When some other shifter becomes empty it is to be loaded correctly by our decompression algorithm.

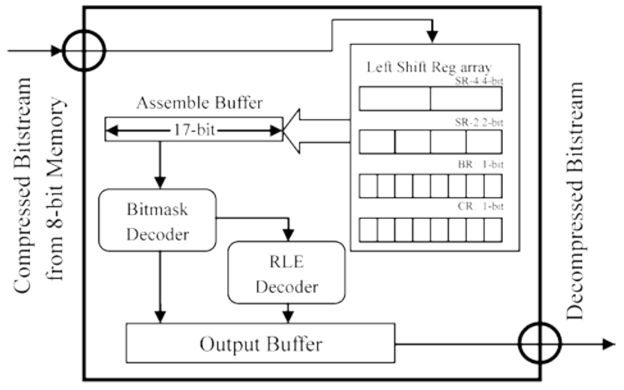

Fig 6 Decompression Engine

6. COMPRESSSION EFFICIENCY

In our system there two sets of hard to compress IP core bitstreams that are being taken from image processing and encryption domain to compare compression and decompression efficiencies. We used Xilinx Virtex-II family IP core benchmarks to analyze the results. The same results are applicable to other families and vendors too. We compared our approach with existing best known distance vector (DV)-based bitstream compression technique proposed by Pan et and best known parameterized LZSS based decompression accelerator proposed by Koch et al. In our experiments, Pan et al benchmarks are compressed with 32 bit symbols, 512 entrydictionary entries and two sliding 2- and 3-bit bitmask for storing bitmask differences. Koch et al benchmarks are compressed using 16 bit symbols, with 16 entry dictionary and a 2-bit sliding bitmask.

we compared BMC and our approach with LZSS, which also employs RLE. The results are given in Fig 7: It can be observed that pure LZSS is quite effective on Koch benchmarks, because these benchmarks have large amount of repetitive patterns, which are suitable for run length encoding. Nevertheless, LZSS is not able to reduce the bitstream size significantly

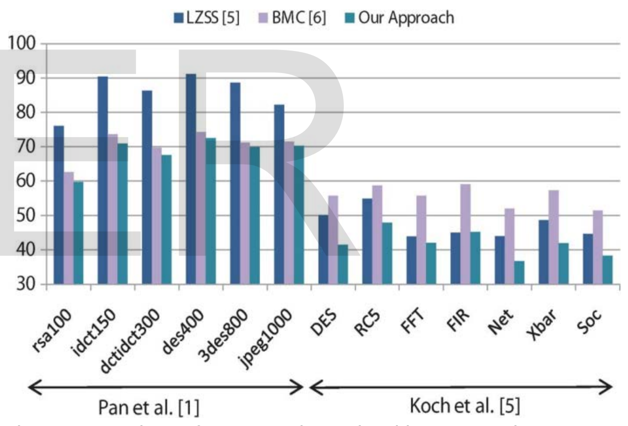

Fig 7 Comparison of compression ratio with LZSS and BMC

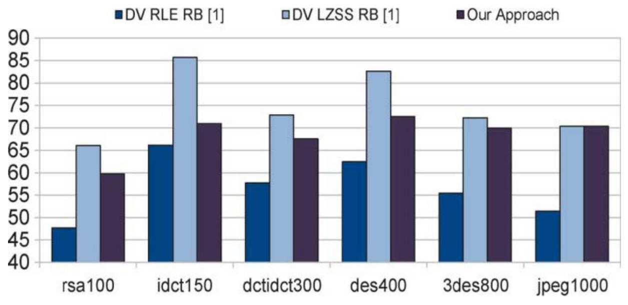

FIG 8 Comparison with difference vector compression.

The difference vector is encoded using Huffman based RLE with frame readback , and LZSS with frame readback (DV LZSS RB). Fig8: shows the results.The Huffman based method achieves the best com- pression (10% to 15% better than our approach) using shorter variable length encodings. Study reveals that encoding requires complex and large hardware to handle variable-length Huffman codes and has slower operating speed. Thus Huffman-based decoder inspite of its good compression ratio is not suitable for realtime decompression. Compared with DV LZSS RB approach, which is suitable for practical implementation, our method has better (10% to 15%) compression performance

Fig:9 shows the results. Our approach outperforms X-Match PRO. Since software compressors like PPMZ and bz2 have complex compression algorithm and almost unlimited resources, it is excepted to generate near optimal results

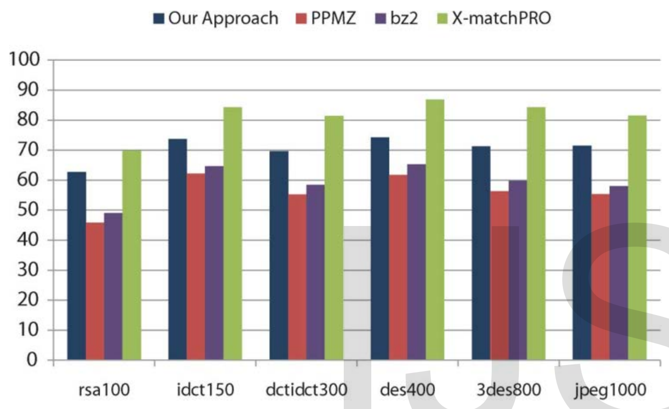

Fig 9 Comparison with other compression techniques**.**

6.1 Decode Aware Vs Bmc

Bitmaskbasedcompression technique proposed is compared with all other main techniques and it is shown in fig 10: The four different type of compression techniques that are compared; i) BMC - bit mask compression technique- , ii) BMC DC - bit mask compression along with new dictionary selection technique, iii) pBMC DC - our proposed decode aware bit mask compression and iv) pBMC+RLE - our proposed decode aware bitmask compression combined with run length encoding. The dictionary generated by our algorithm improves the com- pression ratio by 4% to 5%. The reason is that other benchmark requires large dictionaries for better compression ratio of size untill 1k entries. we were unable to find the threshold value manually for each bitstream, our algorithm adaptively finds the most suitable dictionary entries for each bitstream. The experimental results also illustrate the improvement of compression ratio due to the run length encoding used in our technique. The column pBMC+RLE in Fig. 10 shows improvement on all the benchmarks. On an average we found 5% to 7% reduction over pure bitmask-based compression for Pan *et al.* [1] benchmarks and 15% improvement on Koch *et al.* [4] benchmarks. This is due to the fact that FPGA configuration bitstreams usually have many repetitive patterns. Our RLE en-coding technique adaptively compresses these patterns without compromising the effectiveness of bitmask-based compression technique.

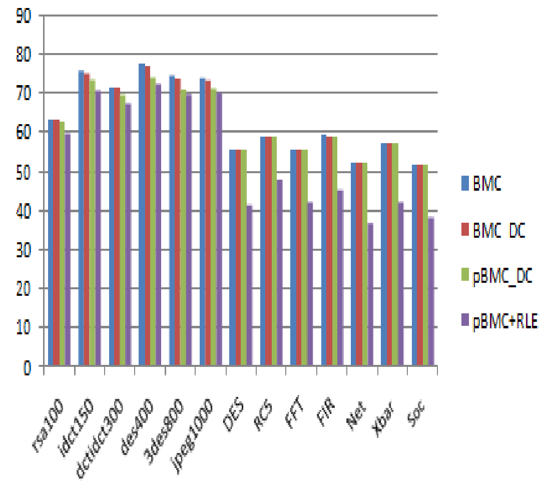

Fig10 Comparison of compression ratio with bit mask based code compression technique

7. DECOMPRESSION EFFICIENCY

We measured the decompression efficiency using the time required to reconfigure a compressed bitstream, the resource usage and maximum operating frequency of the decompression engine. The reconfiguration time is calculated using the product of number of cycles required to decode the compressed bit- stream and operating clock speed. We have synthesized decompression units for variable-length bitmask-based compression, difference vector-based compression (DV RLE RB), and our proposed approach on Xilinx Virtex II family XC2v40 device FG356 package using ISE9.2.04i to measure the decompression efficiency. The results are given in Table I. Observation shows that our approach can operate at a much higher frequency and occupies only 60% area compared to original bit- mask-based decompression engine. Since our approach has the identical bitmask decoding circuit of the original one, the improvement is due to our ABLSA as we expected we have achieved 15%–20% better compression which means we can decompress more configuration information during the same amount of time.

Table 1 Operating speed and lookup table usage of decoders

Type	Speed(MHz)	Slice usage

Bitmasdecoder	130	250
Lzss	198	45
Our approach	195	130

8. CONCLUSION

In this paper we have presented the efficient novel decode-aware compression technique to improve both compression and decompression efficiencies. The combinations of bitmask based compression and run length encoding provide an efficient compressed bitstream, so that more configuration information can be stored using the same memory. The proposed technique to compress reconfiguration bitstream is found to improve compression ratio by around 10-15% and the decompression engine capable of operating at around 200MHZ. The reconfiguration time is reduced by around 15-20% compared to nearest decompression accelerator.

Acknowledgment

Iwould like to thank my guide and my department staff for providing me an opportunity to solve complex problems and encourage me to learn new technologies.

References

[1] J. H. Pan, T. Mitra, and W. F. Wong,2004 Configuration bitstream com- pression for dynamically reconfigurable FPGAs.
[2] S. Hauck and W. D. Wilson, 1999 Runlength compression techniques for FPGA configurations in *Process.*

[3] A. Dandalis and V. K. Prasanna, 2005 Configuration compression forFPGA-based embedded systems.

[4] D. Koch, C. Beckhoff, and J. Teich, 2007 Bitstream decompression for high speed FPGA configuration from slow memories.

[5] S. Seong and P. Mishra, 2008 Bitmask-based code compression for em- bedded systems.

[6] S. Hauck, Z. Li, and E. Schwabe,1999 "Configuration compression for the Xilinx XC6200 FPGA.

[7] D. A. Huffman,1952 A method for the construction of minimum-redun-dancy codes.

[8] A. Moffat, R. Neal, and I. H. Witten, 1995 Arithmetic coding revisited.

[9] A. Khu,2001 Xilinx FPGA configuration data compression and decompression.

[10] M. Huebner, M. Ullmann, F. Weissel, and J. Becker, 2008 Real-time config-uration code decompression for dynamic FPGA self-reconfiguration

P.HEMNATH is currently pursuing master degree in vlsi design from vel tech multi tech DR.Rangarajan DR.Sakunthala engineering college,avadi,Chennai.

E-mail: hembakkiya@gmail.com

V.PRABHU is assistant professor in the department of ECE in vel tech multi tech DR.Rangarajan DR.Sakunthala engineering college,avadi,Chennai.

E-mail:prabhu.cvj@gmail.com