GOLOMB Compression Techniqe For FPGA Configuration

P.Hema
Assistant Professor,EEE
Jay Shriram Group Of Institutions

ABSTRACT

Bit stream compression is important in reconfigurable system design since it reduces the bit streams size and the memory requirement. It also improves the communication bandwidth and thereby decreases the reconfiguration time. Existing research in this field has explored two directions: efficient compression with slow decompression or fast decompression at the cost of compression efficiency. This project proposes a novel decode-aware compression technique to improve both compression and decompression efficiencies. The proposed work combines golomb codes which is efficient for both variable and fixed length parameters with dictionary and bit mask selection methods for improving compression efficiency. To reduce the hardware overhead during decompression, a smart placement of compressed bitstreams which enables the compressed variable-length coding bit streams to be stored and buffered in the form of multiple fixed length coding bit streams. So that, the decompression hardware for variable-length coding is capable of operating at the speed closest to the best known field-programmable gate array-based decoder for fixed-length coding. The area and configuration delay of the decompression engine are reduced significantly.

I. INTRODUCTION

FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs) are widely used in reconfigurable systems. Since the configuration information for FPGA has to be stored in internal or external memory as bitstreams, the limited memory size, and access bandwidth become the key factors in determining the different functionalities that a system can be configured and how quickly the configuration can be performed. While it is quite costly to employ memory with more capacity and access bandwidth, bitstream compression technique alleviates the memory constraint by reducing the size of the bitstreams. With compressed bitstreams, more configuration information can be stored using the same memory. The access delay is also reduced, because less bits need to be transferred through the memory interface. To measure the efficiency of bitstream compression, *compression ratio* (CR) is widely used as a metric. It is defined as the ratio between the compressed bitstream size

(CS) and the original bitstream size (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.

Our approach combines the advantages of previous compression techniques with good compression ratio and those with fast decompression. This paper makes three important contributions. First, it performs smart placement of compressed bitstreams to enable fast decompression of variable-length coding. Next, it selects bitmask-based compression parameters suitable for bitstream compression. Finally, it efficiently combines run lengthencoding and bitmask-based compression to obtain better compression and faster decompression.

II BLOCK DIAGRAM

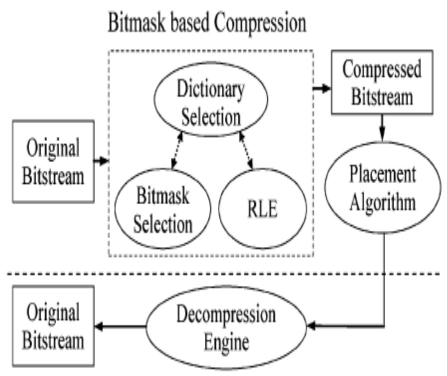

Fig.1 Block Diagram

The figure 1 shows our decode-aware bitstream compression framework. On the compression side, FPGA configuration bitstream is analyzed for selection of profitable dictionary entries and bitmask patterns. Our decode-aware placement algorithm is employed to place the compressed bitstream in the memory for efficient decompression. During run-time, the compressed bitstream is transmitted from the memory to the decompression engine, and the original configuration bitstream is produced by Decompression. Since memory and communication bus are designed in multiple of bytes (8 bits), storing dictionaries or transmitting data other than multiple of byte size is not efficient. Thus, we restrict the symbol length to be multiples of eight in our current implementation. Since the dictionary for bitstream compression is smaller compared to the size of the bitstream itself, we use d=2^i to fully utilize the bits for dictionary indexing, where i is the number of indexing bits.

III DICTIONARY BASED APPROACH

Dictionary-based code compression techniques provide compression efficiency as well as fast decompression mechanism. The basic idea is to take advantage of commonly occurring instruction sequences by using a dictionary. The repeating occurrences are replaced with a codeword that points to the index of the dictionary that contains the pattern. The compressed program consists of both codewords and uncompressed instructions. Figure 3 shows an example of dictionary based code compression using a simple program binary and encoding format shown in Figure 2. The binary consists of ten 8-bit patterns i.e., total 80 bits. The dictionary has two 8-bit entries. The compressed program requires 62 bits and the dictionary requires 16 bits. In this case, the compression ratio (CR) is 97.5%

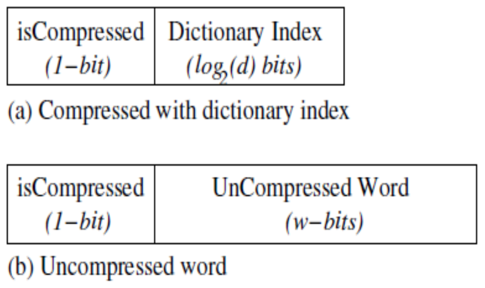
isCompressed (1−bit)
Dictionary Index (log2(d) bits)
(a) Compressed with dictionary index
isCompressed (1−bit)
UnCompressed Word (w−bits)
(b) Uncompressed word

Fig.2 Format for Dictionary Based Compression

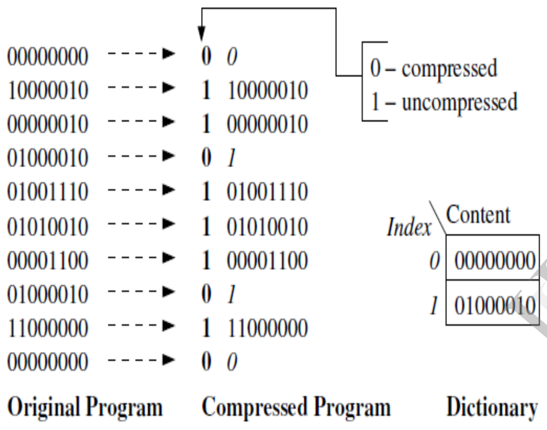

Fig.3 Dictionary Based Compression Example

IV BITMASK BASED COMPRESSSION

Bitmask-based compression is an enhancement on the dictionary-based compression scheme, which helps us to get more matching patterns. In dictionary- based compression, each vector is compressed only if it completely matches with a dictionary entry. Figure 4 shows an example of bitmask based code compression using a simple program binary and encoding format shown in Figure 5.

The 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).The last bit indicates the dictionary index.

Data that are compressed using bitmask requires 7 bits. The first two bits, as before, represent if the data is compressed, and whether the data is compressed using bitmasks. The next two bits indicate the bitmask position and followed by two bits that indicate the bitmask pattern.

The data which is different for more than 1 bit is left uncompressed.

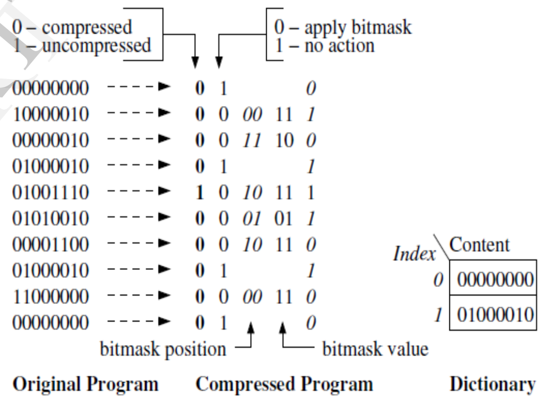

Fig.4 Bitmask Based Compression Example

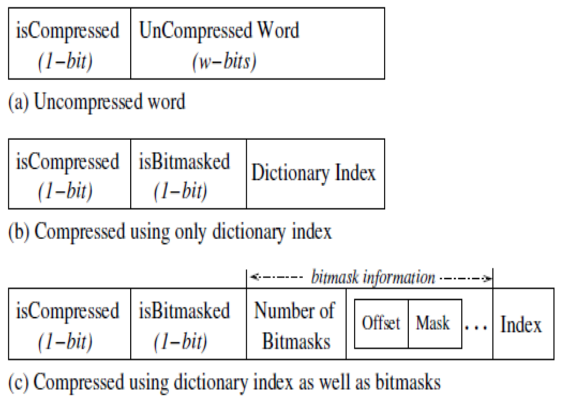

Fig.5 Format for Bitmask Based Compression

V. GOLOMB CODING

In Golomb Coding, the group size, *m*, defines the code structure. Thus, choosing the *m* parameter decides variable length code structure which will have direct impact on the compression efficiency.

Once the parameter *m* is decided, a table which maps the runs of zeros until the code is ended with a one is created. Determination of the run length is shown as in Figure 6. A run length of multiples of *m* are grouped into *Ak* and given the same prefix, which is (*k* – 1) number of ones followed by a zero. A tail is given for each members of the group, which is the binary representation of zero until (*m* – 1). The codeword is then produced by combining the prefix and the tail. An example of the table is in Figure 7.

Data set	01 0000001 0001 000001 001 1					
Subset	01	0000001	0001	000001	001	1
Run-length	1	6	3	5	2	0

Fig.6 Determination of Run-Length

Using Figure 7, binary strings can be divided into subsets of binary strings and replacing the subsets with the equivalent codeword as shown in Figure 8.

Group	Run-length	Group prefix	Tail	Codeword
A1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011

Fig.7 Golomb Coding Example With Parameter m=4

Data set	01 0000001 0001 000001 001 1					
Subset	01	0000001	0001	000001	001	1
Encoded	001	1010	011	1001	010	000

Fig.8 Golomb Coding Example With Parameter m=4

VI MOTIVATION OF WORK

In this section, we briefly analyze the decompression hardware complexity of common variable-length compression techniques. This analysis forms the basis of our approach. In the following discussion, we use the term symbol to refer to a sequence of uncompressed bits and code to refer to the compression result (of a symbol) produced by the compression algorithm.While compression efficiency is straightforward and widely used criteria to evaluate compression techniques, the complexity of decompression hardware determines whether an algorithm with promising compression ratio can be applied to commercial FPGAs. Interestingly, our study shows that the complexity of the decompression algorithm is not the only determining factor of the hardware complexity. When variable- length coding is employed, the hardware complexity is also determined by the complex buffering circuitry, which is overlooked by previous bitstream compression approaches.

Since each code has different length, we have to use barrel shifter to align the input buffer in each cycle. Theoretically, a barrel shifter operating on a n-bit buffer needs nlog(n) multiplexers (MUXes) organized into log(n) layers. When implemented in modern FPGAs (Xilinx Virtex II or Virtex 4), the barrel shifter for an input buffer with typical size of 32–64 bits consumes 200– 400 fourinput lookup tables (LUTs), which is similar to the total area of a typical bitmask or Huffman decoder. Moreover, barrel shifter will increase the latency remarkably by introducing several layers of combinational logic in the critical path. Therefore, the buffering circuitry is a major bottleneck in a decompression engine for variable-length coding both in terms of area and performance.

VI ALGORITHM

Input: Input bitstream

Output: Compressed bitstream placed in memory

Step 1: Divide input bitstream into symbol sequence SL.

Step 2: Perform bitmask pattern selection.

Step 3: Perform dictionary selection.

Step 4: Compress symbol SL into code sequence CL using bitmask and Golomb coding.

Step 5: Perform placement of CL using power two-streams.

VII CONCLUSION

The existing compression algorithms either provide good compression with slow decompression or fast decompression at the cost of compression efficiency. In this paper, we proposed a decoding-aware compression technique that tries to obtain both best possible compression and fast decompression performance. The proposed compression technique analyzes the effect of parameters on compression ratio and chooses the optimal ones automatically. We also exploit golomb encoding efficiently combined with bitmask-based compression to further improve both compression ratio and decompression efficiency.

VIII REFERENCES

1. Xiaoke Qin, *Member, IEEE*, Chetan Muthry, and Prabhat Mishra, *Senior Member, IEEE(March 2011)* "Decoding-Aware Compression of FPGA Bitstreams", IEEE transactions on Very Large Scale Integration (VLSI) systems, vol. 19, no. 3.

2. Chandra.A and Chakrabarty.K,(March 2001) "System on-a-chip test data compression and decompression architectures based on Golomb codes," *IEEE Trans Computer-Aided Design*, vol. 20, pp.355-368.

3. Dandalis.A and Prasanna.V.K,(December 2005) "Configuration compression for FPGA-based embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 12, pp. 1394–1398.

4. Hauck.S and Wilson.W.D,(March 1999) "Runlength compression techniques for FPGA configurations," in *Proc. IEEE Symp. Field-Program. Custom Comput.* , pp. 286–287.

5. Hauck.S, Li.Z, and Schwabe.E,(August 1999) "Configuration compression for the Xilinx XC6200 FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 8, pp. 1107–1113.

6. Koch.D, Beckhoff.C, and Teich.J,(2007) "Bitstream decompression for high speed FPGA configuration from slow memories," in *Proc. Int. Conf. Field-Program. Technol.*, pp. 161–168.
7. Pan.H.J, Mitra.T, and Wong.W.F,(December 2004) "Configuration bitstream compression for dynamically reconfigurable FPGAs," in *Proc. Int. Conf. Comput.-Aided Des.*, pp. 766–773.
8. Seong.S and Mishra.P,(April 2008) "Bitmask-based code compression for embedded systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 4, pp. 673–685.
9. Stefan.R and Cotofana.S ,(2008) "Bitstream compression techniques for Virtex 4 FPGAs," in Proc. Int. Conf. Field Program. Logic Appl.,