IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 9, Issue 2, Ver. III (Mar - Apr. 2014), PP 82-86
www.iosrjournals.org

Reduction of Bitstream Transfer Time in FPGA

Snehal S. Khartad, Prof. Pravin Matte
Dept. of E & TC Engineering G.H.R.C.E.M., Wagholi Pune, India
Dept. of E & TC Engineering G.H.R.C.E.M., Wagholi Pune, India

Abstract: *In reconfigurable system, the size of bitstream and memory is reduced using bitstream compression. By reducing the reconfiguration time, the system can improve the communication bandwidth. Existing system shows that, at the cost of compression efficiency effective compression is achieved with slow or fast decompression rate. To improve both compression and decompression efficiencies, this paper proposes a new technique i.e., decode aware compression technique. Different approaches for the proposed system are i) A large set of matching pattern is created by using effective bitmask selection technique. ii) Using the bitmask and dictionary selection technique, a bitmask based compression is achieved because of which there is an efficient reduction in memory requirement. iii) Repetative patterns are generated by grouping bitmask based compression and run length encoding. Finally, using decompression engine the original bitstream can be generated.*
Index Terms: *Bitstream, Field Programmable Gate Array (FPGA), reduction, transfer time.*

I. INTRODUCTION

Field programmable gate array are revolutionary device which combine the benefits of hardware and software. FPGA are used to increase the effective size and increase the number of function which is commonly used in reconfigurable system and application specific integrated circuit (ASIC). It is used to stored configuration bitstream on memory. This configuration information has to be stored in internal or external memory of FPGA in the form of bitstream. This typically starts with an application program written in a HDL (Hardware Description Language) such as Verilog or VHDL. This design is further forward through a series of steps which are mentioned below.
Fig 1 shows the typical mapping flow of FPGA , that is how bitstream formation takes place.

First step is a logic synthesis which convert high level logic into gate level logic i.e.; structural and behavioral code converted into logic gate. Second step is mapping technology in which gates are separated and again grouped to get the best match of FPGA logic resources. The next step is placement which assign the logic grouping into specific logic block and routing determine the interconnect resources that will carry the user signal. Finally, system performs bitstream generation which creates binary files. These binary files are then compressed using various algorithms which help to solve the memory related issue. In the previous technique, if the compression is increased the decompression is lower and if we try to increase the decompression rate it result in slow compression. The proposed system try to balances both the compression and decompression. Thus the proposed compression technique is very efficient for bitstream compression [5] because of its simple decompression scheme and good compression ratio. The compression ratio is defined as ratio between compressed bitstream size and original bitstream size. Therefore, the propose technique compress the bitstream in a very efficient manner than the existing technique.

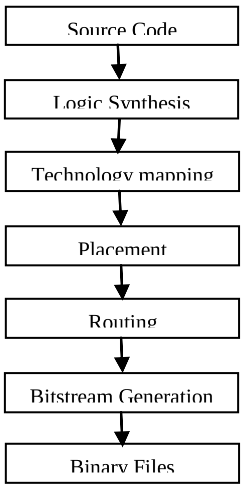

Fig. 1. Typical FPGA mapping flow.

Section 1 introduces the paper, Section 2 describes Literature Survey, Section 3 explains proposed system, Section 4 shows results and Section 5 and 6 are conclusion and references respectively.

II. LITERATURE SURVEY

To compress the configuration bitstream, numbers of compression algorithm are used. Previously, a number of studies focused on FPGA bitstream compression. One of the most notable is family of vertex product [6] .This family of FPGA examines an extensive range of technique, read back and frame recording and proposes a wildcard technique such as Huffman coding. But the Huffman coding is not appropriate for real time coding due to large amount of calculation. Also for computing of Huffman coding considerable time is required. The Xilinx XC6200 series of FPGA support wildcard compression scheme. The better redundancy is achieved by using frame reordering and active frame read back given by *Pan et al.* [1]. Using Huffman based run length encoding or LZSS based compression, the difference between consecutive frames is encoded. Such schemes demand excellent compression however the decompression overhead [1] is a major concern.

During decompression, the bitstream compression techniques access the configuration memory which is applicable to all FPGA's. This technique based on divide and conqueror strategy i.e., the entire bitstream is first divided into small words which is then compressed using various algorithm like Huffman coding, arithmetic coding or dictionary based compression. Furthermore a bitstream compression algorithm which is based on LZ77 is proposed by Xilinx [9]. LZSS based technique for Xilinx vertex XCV2000E FPGA is proposed by *Huebner et al.* To achieve fast decompression, the decompression engine is carefully design. The algorithm like LZSS help us to maintain decompression overhead but the compression efficiency is very poor. On the other hand complex algorithm achieves better compression but it is very difficult to maintained decompression overhead.

The decompression through put of complex compression algorithm is increased using parallel decompression. To enable parallel decompression *Qin et al* introduced a placement technique of compressed bitstream. However the area overhead is also multiplied because of not a single change in decoder and buffering circuitry. In contrast propose technique try to balance both compression and decompression. This is done by effectively addressing the buffering circuitry problem to improve the maximum operating frequency of the system.

III. PROPOSED SYSTEM: IMPLEMENTATION DETAILS

The block diagram of proposed system is shown in fig.2. In this system, the input is applied to the compression algorithm. Before proceeding for compression algorithm, it is first converted into binary files i.e., bitstream. Afterward the compressed bitstream are loaded into memory. During the program execution decompression is done. The function of decompression hardware is to decode and transfer the bits from memory which are already compressed and then transferred to the configurable logic blocks (CLB) of memory. As an output we get the original signal similar to input. The major contributions of proposed system are as follows.
i) The bitstream are compressed as much as possible.
ii) Without affecting the reconfiguration time, the effective decompression is achieved.

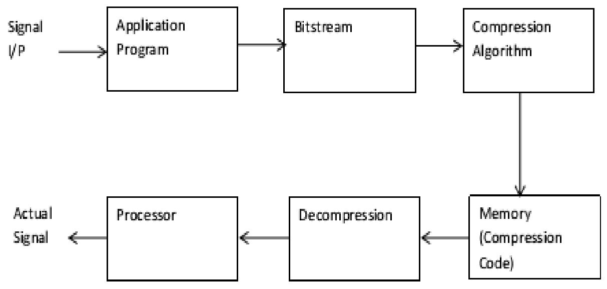

Fig. 2. Architecture of proposed system

Algorithm for Decode Aware Bitstream Compression
Input : Bitstream
Output : Compressed Bitstream placed in memory.
Step 1 : Divide input bitstream 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.

IV. DECODE AWARE COMPRESSION TECHNIQUE

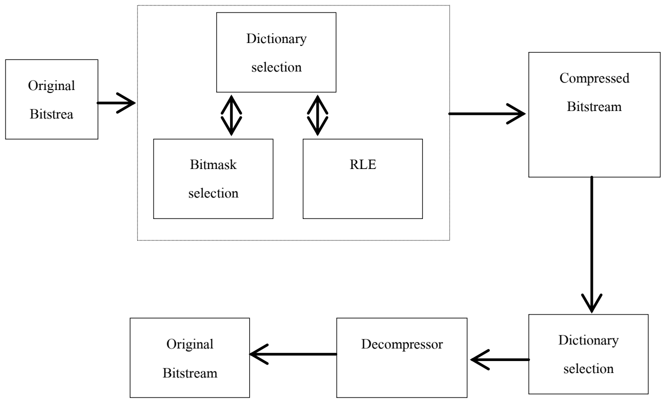

Fig. 3. Decode aware compression

Figure 3 shows the block diagram of decode aware compression technique [9]. In this technique the original bitstream is applied to the compression algorithm consist of dictionary selection, bitmask selection and RLE algorithm to get the compressed bitstream. To place this compressed bitstream in the memory, placement algorithm is used. This compressed bitstream from the memory are further transmitted to the decompressor to get the original bitstream.

A. *Dictionary Based Compression*

This technique provides better compression output as well as fast decompression rate [10]. In dictionary selection commonly occurring instruction sequences is taken [9]. If the pattern contains the repeating occurrences then the sequences are replaced with a code word which point to the index of the dictionary. Thus the compressed code (bitstream) consists of both code word and uncompressed instruction. Fig.4 shows an example of dictionary based code compression using a simple binary program. The dictionary file has two 8-b entries and the binary consist of 10 8-b pattern i.e., 80-b. The dictionary required 16-b entries and the compressed bitstream require 62-b entries. Thus the compression ratio is 97.5%. As the compression ratio for the dictionary selection is large enough it is not a good compression technique. Therefore the bitstream which cannot be compressed by using dictionary selection that can be compressed that bitmask selection which results in smaller compression ratio.

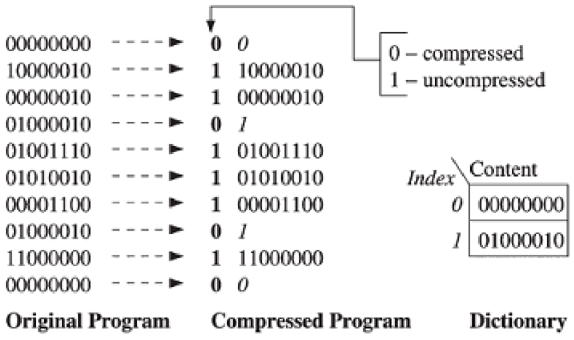

Fig. 4. Bitstream compression using dictionary selection.

B. *Bitmask Based Compression*

This technique is used to set certain bits using bitwise OR and invert them by using bitwise XOR. Bitmask based technique consist of a pattern of binary values combine with some value, where the mask is zero or also set to zero using bitwise AND. Without adding the significant cost, this approach helps to improve the compression ratio. In bitstream compression the selection of bitmask plays a vital role as it helps to compress the bitstream which cannot be compressed using dictionary selection. Figure 5 below shows the compression using bitmask selection [5]. By using this technique the compression ratio is improved , by keeping the decompression efficiency same as compared to the existing technique.

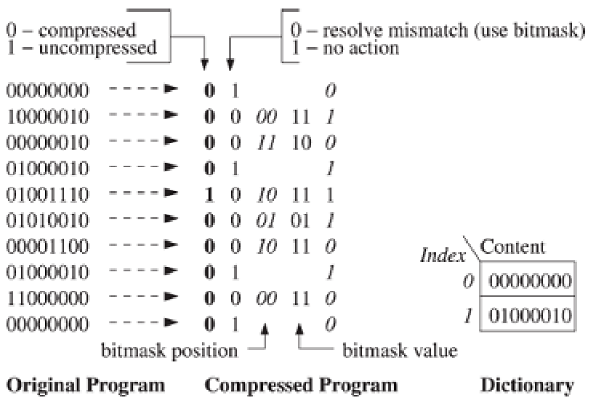

Fig. 5. Bitstream compression using bitmask selection.

C. Run Length Encoding Technique

Repeating bit sequences usually found in configuration bitstream. Run length encoding of such sequences may yield a better compression result [2],[4], though such patterns encoded using bitmask based compression. Such encoding doesn't require any extra bits [5] .Thus the bits are saved using run length encoding. In this technique zero is never used, because it means an exact match which must be encoded using zero bitmask. Considering this bitmask as a special marker, this repetition can be encoded without changing the code format of bit masked based compression. Without changing the code format of bitmask based compression, the repetition of sequences can be encoded and RLE yield a shorter code length than the original bitmask encoding.

Figure 6 shows the RLE based compression [9]. Consider an input sequence "00000000" which is repeating five times. This input will be compressed in normal bitmask based compression. Our approach of RLE technique replaces such repetition using a bitmask of "00". In this example, the four repetition will be encoded using RLE and the first occurrence will be encoded as usual. By combining the dictionary bits and bitmask offset the number of repetition is encoded. For this example, the dictionary index is "0" and the bitmask offset is "10". Thus the number of repetition will be "100" i.e., 4.

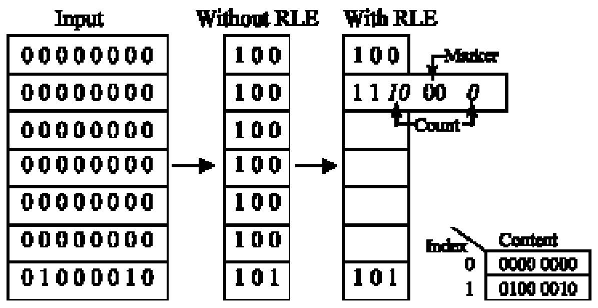

Fig. 6. Run length encoding based compression.

If the Run length encoding yield a shorter code length than the original bitmask encoding then the compressed word are run length encoded. RLE is used only if $r*l>l'$ bits, where r represents the repetition of code with length l and l' represents the number of bits required to encode them using RLE. Thus the bits are saved using RLE algorithm.

D. Decompression Engine

The original bitstream are regenerated by using the decompression engine to decode the compressed configuration and to feed the uncompressed bitstream to the configuration unit in FPGA's, the decompression engine is used. A decompression engine is divided into two parts: the buffering circuitry and decoder. The function of the buffering circuitry is to buffer and align codes fetch from the memory whereas decoder is used to perform decompression operation to generate original bitstream. Thus, to get the original bitstream back decompression engine is used.

V. EXPECTED RESULT AND DISCUSSION

The propose system is expected to give better compression ratio than the existing system. The system reduces the memory requirement by balancing both compression and decompression and also decreases the reconfiguration time in FPGA.

VI. CONCLUDING REMARKS

The existing technique either provides good compression with slow decompression or fast decompression at the cost of compression efficiency. In this paper, a new technique is proposed i.e., decode aware compression technique which is design to produce the compressed bitstream in an efficient manner with fast decompression performance. This technique helps us to reduce the transfer time in FPGA with improved compression ratio. The combination of bitmask based compression and run length encoding provide an efficient compression bitstream so that more configuration information can be stored using the same memory. In the future, we planned to investigate more placement algorithm that are compatible with other compression technique such as Huffman coding, Goulomb coding and arithmetic coding.

.

REFERENCES

[1] J. H. Pan, T. Mitra, and W. F. Wong, “Configuration bitstream compression for dynamically reconfigurable FPGAs,” in Proc. Int. Conf. Comput.-Aided Des., 2004, pp. 766-773.

[2] S. Hauck and W. D. Wilson, “Runlength compression techniques for FPGA configurations,” in Proc. IEEE Symp. Field-Program. Custom Comput. Mach., 1999, pp. 286-287.

[3] A. Dandalis and V. K. Prasanna, “Configuration compression for FPGA-based embedded systems,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 13, no. 12, pp. 1394-1398, Dec. 2005.

[4] D. Koch, C. Beckhoff, and J. Teich, “Bitstream decompression for high speed FPGA configuration from slow memories,” in Proc. Int. Conf. Field-Program. Technol., 2007, pp. 161-168.

[5] S. Seong and P. Mishra, “Bitmask-based code compression for embedded systems,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 4, pp. 673-685, Apr. 2008.

[6] S. Hauck, Z. Li, and E. Schwabe, “Configuration compression for the Xilinx XC6200 FPGA,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 18, no. 8, pp. 1107-1113, Aug. 1999.

[7] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” Proc. IRE, vol. 40, no. 9, pp. 1098-1101, 1952.

[8] A. Moffat, R. Neal, and I. H. Written, “Arithmetic coding revisited,” in Proc. Data Compression Conf., 1995, pp. 202-211.

[9] Xiaoke Qin, Chetan Muthry, and Prabhat Mishra, “Decoding Aware Compression of FPGA Bitstreams,” in Proc. Data Compression Conf., 2011, pp. 411-419.

[10] C. S. Manikandababu and P. M. Sandeep “FPGA Bitstream Compression using Run length Encoding” IJECCT, Volume 3- Issue 2 (March 2013).