A new approach to compress the configuration information of programmable devices

M. Martina, G. Masera, A. Molino, F.Vacca
Politecnico di Torino
Dipartimento di Elettronica
I-10129 Torino, Italy

L. Sterpone, M. Violante
Politecnico di Torino
Dipartimento di Automatica ed Informatica
I-10129 Torino, Italy

Abstract

During the last decade programmable devices have gained an impressive diffusion, tackling some traditional ASIC marked domains. In particular, multi-million gate FPGAs have become a very appealing low-cost solution even for consumer applications. However, one of the big issues that can arise with modern FPGA devices is the need for large and expensive external non-volatile memory to keep the configuration data. In this work we developed an alternative technique to compress FPGA bitstreams based on the knowledge of the device internal structure. The proposed method performs a two-step coder: in the first step the bitstream is adaptively "filtered" to remove data redundancy, while in the second step an arithmetic coder is used to actually compress the information. The effectiveness of the proposed technique has been demonstrated on a set of case studies. As a result conventional approaches are outperformed reaching a compression ratio of 4.26 against 3.3 times.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are being increasingly used as parts of consumer products. They have gained a wide popularity due to their programmability together with the availability of high performance devices with millions of configurable blocks. In fact FPGAs offer a generic platform for hardware realization of application specific algorithms. In particular FPGAs are suited for accelerating compute intensive algorithms that can take advantage of massive hardware parallelism [1]. Moreover the possibility to add reconfigurability to processors or more in general to a System On a Chip (SOC), has fostered several research works in the field of embedded systems. FPGA-based embedded systems can sustain high processing rates while providing the high degree of flexibility required in dynamically changing environments.

As the number of configurable blocks and the complexity of the routing resources has increased, the amount of memory needed to store the configuration data grows accordingly. It is worth noticing that the configuration bitstream of the Xilinx Virtex FPGAs ranges from 0.6 Mbits to 16Mbits [2]. As a consequence, storing configuration bitstreams in an FPGA-based embedded system is a critical issue since it could tackle the system cost. From an other perspective the size of the bitstream has a negative impact on the configuration time. This can potentially compromize the performance of dynamically reconfigurable embedded systems.

The aim of this paper is to propose an effective algorithm to compress Xilinx Virtex bitstreams resorting to an adaptive binary arithmetic coder. In section 2 other works in the field of FPGA bitstream compression are summarized, whereas section 3 is devoted to emphasize the internal architecture of Xilinx Virtex FPGAs. The strength of the proposed algorithm stems from a deep knowledge of the FPGA architecture and of the bitstream syntax. These informations are exploited to feed the arithmetic coder with a filtered version of the bitstream as described in section 4.

In section 5 we show experimental results and comparisons compressing the configuration data with ZIP and with the compression tool provided by Xilinx [4]. Finally we study the feasibility of exploiting the proposed algorithm in a real embedded system based on an ARM processor. In particular in an embedded scenario it is crucial to understand the complexity of the decoder. In fact it is devoted to receive the compressed bitstream, decompress and send it to the FPGA configuration unit.

2. Related work

In order to reduce the memory footprint of the FPGA configuration data, Xilinx developed a bitstream compression algorithm based on LZ77 scheme [4]. LZ77 is a

3-9810801-0-6/DATE06 © 2006 EDAA

dictionary-based text compression scheme developed by Lempel and Ziv in the 1977 [12]. This scheme works by defining a fixed-size dictionary to hold bytes from an input source, referring to it when compressing the remainder of the input source to find existing pattern. As the compression progresses, the dictionary is updated by loading more bytes from the input source, subsequently forcing earlier entries out.

Similar works found in the literature until 2000 mainly address the problem of the compression techniques applied to software applications, where the size of the code for a given processor has to be minimized. Many results can be found in this framework, demonstrating that dictionary-based compression methods lead to best performance. The problem of FPGA bitstream compression is quite similar to this framework, both in terms of low-complexity requirements and statistical behavior of the compressed data stream. However, modern FPGAs show very complicate internal organization, and the bitstream organization is becoming quite complex. For this reason, simple compression schemes that work well in software-based applications shows poor results while working with configuration bitstreams.

In 2001 Dandalis and Prasanna proposed a compression scheme [2] for the configuration of SRAM-based FPGA. In that work a modified version of the original Lempel-Ziv algorithm is used. Compression rates are improved by reducing the dictionary size in memory, by selectively decomposing strings on it. The achieved memory savings for a set of bitstreams have been shown to be in the range of 11 - 41%.

An accurate benchmarking of existing compression methods can be found in [5]. In the same paper a novel scheme is proposed based on LZ, where a modified Frame Data input Register (FDR) is proposed and a factor of 2.5 improvement over [2] is reached. The FDR register is the on-chip Serial In Parallel Out (SIPO) input buffer for the configuration stream and, if modified, can act as analysis window in the dictionary update process. The simulation results that where presents on a wide range of applications are useful, and demonstrate that Arithmetic compression [9] [6] works better than Huffman codes[3] on configuration bitstreams.

Recently, Pan et al. [7] proposed a new compression technique exploiting the intra-bitstream correlation, and extended it looking at the inter-bitstream redundancies. They found that a good way to compress bitstreams is to work with the differences between consecutive frames. They also demonstrated that Run Length Encoding (RLE) techniques (e.g. Huffman) works better than dictionary-based ones (e.g. LZ). This mainly depends on the fact that *stream regularities may be too fine-grained to be captured by the LZSS method*. The inter-bitstream correlation is a interesting topic, and can be exploited to reduce configuration data during consecutive reconfigurations.

3. Background

The partially reconfigurable SRAM-based FPGAs manufactured by Xilinx comprise an array of configurable logic blocks (CLBs) surrounded by input-output blocks (IOBs), block RAMs (BRAMs), clock resources, configuration circuitry and several programmable interconnection points (PIPs). All these resources are programmed by a configuration memory based on several SRAM cells.

Configuration bitstreams contain different command and data that can be read and written through one of the device configuration interfaces. The configuration memory of a Xilinx SRAM-based FPGA has a regular structure composed of several rectangular arrays of bits. The bits are grouped into one-bit wide vertical frames and extend from the top to the bottom of the array. Frames are grouped together into larger units, called columns. There are several types of columns composed of a specific number of frames: a central column of 4 frames that control the clock, two IOB columns, multiple block RAM columns of 64 frames, and multiple CLB columns of 48 frames for each one. For each frame, the first 18 bits control the two IOBs on the top of the frame, then 18 bits are allocated for each CLB row and another 18 bits control the two IOBs at the bottom of the frame [11]. The frame is padded with bits to make it an exact multiple of 32 bits. Thus, the configuration memory can be visualized as a rectangular array of bits, divided into several sub-rectangular arrays. In particular, the sub-rectangular arrays related to the CLBs 1. They correspond to more than the 90% of the FPGA area, and are programmed by a matrix, called Tile, of 864 bits divided in 48 columns and 18 rows of bit. The current Xilinx Virtex configuration interface can load whole frames of data at a time. Because of the regular structure of the resources in the array, each Tile configuring common structures among the FPGA arrays may share high regularity. By encoding the bit differences between different tiles and between the CLB's location within the FPGA's matrix array, we can reduce the redundancies in storing identical information repeatedly for similar resources. The purpose is to assign a set of suitable reference sub-tile that can construct a difference tile matrix. Given that the bit-flips obtained between two tiles tend to be either few in numbers and scattered or clustered in bands, we observe long sequences of 0s in the difference vector with 1s occurring in shorted sequences. Therefore, the Arithmetic compression can be used to effectively compress sequences in the difference matrix array.

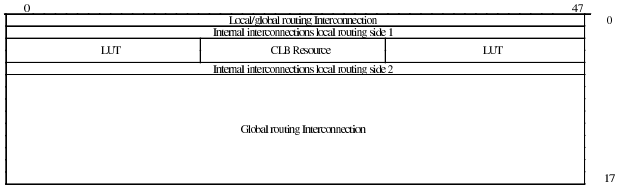

Figure 1. Xilinx CLB logical organization

4. Proposed algorithm

As described in section 3 the proposed algorithm has been developed around the Xilinx Virtex architecture. The bitstream's organization of this device is composed of several regions concerning CLBs, IOBs and BRAMs. Furthermore in every bit-region other sub-regions can be identified, for instance for storage and logic elements such as Look-Up Tables (LUTs), or for interconnection resources such as local routing (LR) or global routing (GR). Since each region and subregion is referred to different parts of the FPGA architecture, they exhibit different statistical properties. As an example not configured CLBs are characterized by the same configuration pattern into the bitstream. Our consideration leads to the idea of performing the bitstream compression exploiting inter bitstream diversity. This can be accomplished only through a deep knowledge of the FPGA architecture and of the bitstream syntax. In particular it is crucial to know which bits into the bitstream are related to the different regions and sub-regions. Thus the first operation required to exploit inter bitstream diversity is to parse the bitstream and to build a map of the regions and subresions. Once this preliminary operation is completed the algorithm evolves through three steps, as shown in figure 2. Given a certain bitstream:

1. For each region (or subregion) find its most probable pattern. This pattern is stored as the *region filter* (or *subregion filter*).

2. Filter each region into the bitstream with its filter mask (i.e. perform a bitwise exclusive or operation). This operation "smooths" the bitstream as it tries to remove the interdependence among symbols.

3. Apply an adaptive binary arithmetic encoder to compress the filtered bitstream. The arithmetic encoder takes advantage of the strongly skewed distribution of the filtered bitstream approaching the source entropy [6].

The high compression ratio achievable with the proposed approach stems from the binary arithmetic encoder. The basic idea behind arithmetic coding is to exploit symbols probability and to represent them through unambiguous values in the range [0,1]. The encoding task is based on the recursive probability interval partition, known as Elias coding; at each iteration the interval is split into two sub-intervals, and the code string C is adjusted so as to point to the base of the sub-interval that corresponds to the input symbol d_i [6].

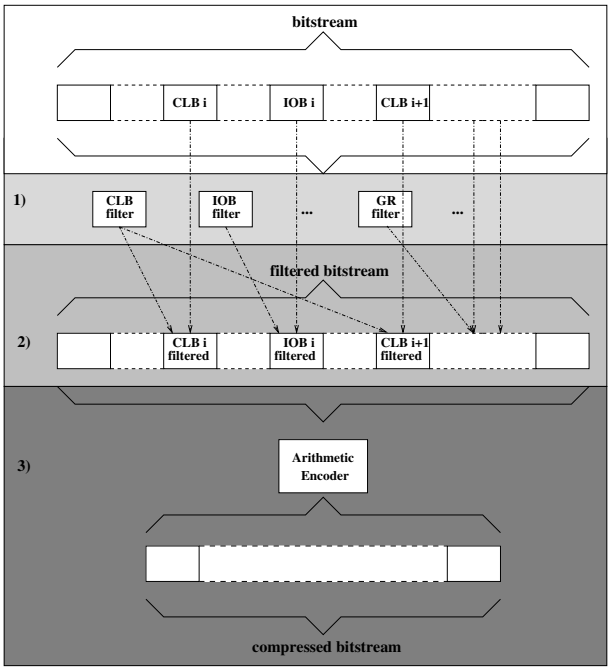

Figure 2. Proposed compression algorithm steps

As far the decoder side is concerned, the decompression scheme is dual with respect to the encoder. So that the decoder first performs the arithmetic decoding then given the *region* and *subregion* filters, it applies the same filtering algorithm employed by the decoder and achieves the original bitstream.

5. Experimental results

The proposed approach has been tested on the bitstream of different designs and compared with the standard ZIP algorithm and the Xilinx tool [4]. As a case of study we concentrate on the Xilinx Virtex XCV300 FPGA. For this device we generated six bitstreams [10] for six different designs, namely: Cordic, DCT, Div1, Mac1, Mac3 and 8051. These designs differ in terms of percentage of logic and memory occupation so that to be a significant set (see table 1). The results of our experiments are shown in table 2. As it can be observed the proposed algorithm shows better performance than ZIP and the Xilinx compression tool.

Finally we run the proposed decoder on a cycle accu-

Circuit	Uncompressed [byte]	**Proposed [byte]**	ZIP [byte]	Xilinx compression tool [byte]
Cordic	219052	**56751**	59801	209134
DCT	219052	**66412**	92587	210391
Div1	219052	**70433**	86143	209368
Mac1	219052	**12386**	13415	126884
Mac3	219052	**35479**	38487	169448
8051	219052	**67071**	97019	218986

Table 2. Comparison among different compression strategies

Circuit	Description	Logic	Memory
Cordic	Pipelined 14 iter. cordic	24%	0%
DCT	8×8 2D-DCT core	51%	12%
Div1	28 bit divider	82%	0%
Mac1	12 bit MAC (26 bit result)	3%	0%
Mac3	32 bit MAC (68 bit result)	19%	0%
8051	8 bit microcontroller	45%	37%

Table 1. Bitstream benchmark designs description

rate ARM model [8] in order to estimate execution times on a processor suitable for embedded applications. It is worth noticing that the obtained decompression times are near 10% faster than compression ones. This fact is due to the filter search procedure required only by the compressor. The overall decompression procedure for each of the benchmarked bitstreams takes about 1.91 s on a ARM7 running at 206 MHz.

6. Conclusions

As described in the previous sections, the aim of this work is twofold: on one hand we wanted to devise a novel method to efficiently compress FPGA bitstreams, while on the other hand we were interested in a preliminary complexity evaluation.

As far as the first goal is concerned, the presented method is able to outperform ZIP–like algorithm as well as the Xilinx compression tool, achieving an average compression ratio of 4.26 times, compared to 3.3 times of ZIP. Such a figure can be of particular interest in reconfigurable system since it enables a sensible reduction of bitstream size, reducing configuration times and bitstream memory requirements. On the complexity side, preliminary analysis shows that arithmetic coding tends to adsorb the greatest part of the computation time. This is particularly true since the adopted arithmetic coder requires divisions and multiplications to support adaptation. Encoder performance can benefit from the use of semi–adaptive or multiplications–free arithmetic coders, as in [6]. Another future direction will be to implement dedicated accelerator cores able to improve codec performance for the presented approach.

References

[1] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, Jun 2002.

[2] A. Dandalis and V. K. Prasanna. Configuration compression for FPGA-based embedded systems. In *ACM International Symposium on Field Programmable Gate Arrays*, pages 173–182, 2001.

[3] D. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.

[4] A. Khu. *Xilinx FPGA Configuration Data Compression and Decompression*. Xilinx - WP152, Sep 2001.

[5] Z. Li and S. Hauck. Configuration compression for Virtex FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 111–119, 2001.

[6] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, Jul 1998.

[7] J. Pan, T. Mitra, and W. Wong. Configuration bitstream compression for dynamically reconfigurable FPGAs. In *IEEE/ACM International Conference on Computer Aided Design*, 2004.

[8] W. Qin. http://www.princeton.edu/∼wqin/armsim.htm.

[9] J. Rissanen. Generalised Kraft inequality and arithmetic coding. *IBM J. Res. Dev.*, 20:198–203, 1976.

[10] Web. http://www.vlsilab.polito.it/ molino/scientific/bitstream.zip.

[11] Xilinx. *Virtex Series Configuration Architecture User Guide*. Xilinx - XAPP151, Oct 2004.

[12] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Tran. on Information Theory*, 23(3):337–343, May 1977.