Lossless Compression using Efficient Encoding of Bitmasks

Chetan Murthy and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA
{cmurthy, prabhat}@cise.ufl.edu

Abstract

Lossless compression is widely used to improve both memory requirement and communication bandwidth in embedded systems. Dictionary based compression techniques are very popular because of their good compression efficiency and fast decompression mechanism. Bitmask based compression improves the effectiveness of the dictionary based approaches by recording minor differences using bitmasks. This paper proposes an efficient encoding of bitmasks used in bitmask-based compression. We prove that a n-bit bitmask (records n differences) can be encoded using only $n-1$ bits. This encoding improves compression efficiency while reduces decompression hardware overhead. We have applied our approach in a wide a variety of domains including code compression, FPGA bitstream compression as well as control word compression. Our experimental results using a wide variety of benchmarks demonstrate that our approach improves the compression efficiency by 3 to 10% without adding any additional decompression overhead.

1 Introduction

Memory is one of the key driving factors in system design because a larger memory indicates an increased chip area, more power dissipation, and higher cost. As a result, memory imposes constraints on the size of application programs. Moreover, memory typically operates at lower speed than the processing elements and thereby brings down the overall system performance. Compression techniques address the problem of slow and limited memory by reducing the data size. Transmitting the compressed data improves the bandwidth between memory and processing elements. Compression ratio, widely accepted as a primary metric for measuring the efficiency of compression, is defined as:

$$\textit{Compression Ratio} = \frac{\textit{Compressed Size}}{\textit{Original Size}} \qquad (1)$$

Dictionary-based compression techniques are popular because they provide both good compression ratio and fast decompression mechanism. The basic idea is to take advantage of commonly occurring patterns by using a dictionary. Recently proposed techniques ([8, 12]) improve the dictionary-based compression by considering mismatches. The basic idea is to create matching patterns by remembering a few bit positions. The efficiency of these techniques is limited by the number and length of bit changes used during compression. It is obvious that if more bit changes are allowed, more matching sequences will be generated. However, the cost of storing the information for more bit positions offsets the advantage of generating more repeating word sequences. Studies [12] have shown that it is not profitable to consider more than three bit changes for compression of 32-bit vectors.

The bitmask based approach proposed by Seong et al. [20] compresses the data using bitmasks that can record differences between input words and dictionary entries. This paper proposes an efficient compression technique to further improve the compression ratio by reducing the bits needed to encode the bitmask information. Our approach is applicable in all compression scenarios where mismatch (correction) information is used but it is most effective in scenarios where small dictionaries are used for compression and most of the data patterns can be compressed using one or more bitmasks. This paper analyzes the application of this approach on compressing a wide variety of data patterns including no instruction set computer (NISC) control words, application programs, and FPGA bitstreams. We have used benchmark from various domains ([6], [11], [14], and [15]). Our experimental results demonstrate that our approach improves the compression ratio by an average of 3 to 10% over existing dictionary-based compression techniques without introducing any additional decompression overhead.

The rest of the paper is organized as follows. Section 2 presents related work on lossless compression. Section 3 describes the existing bitmask based compression. Section 4 presents our encoding technique followed by experimental results in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

The first compression technique for embedded processors was proposed by Wolfe and Chanin [1]. Their technique uses

Huffman coding, and the compressed program is stored in the main memory. The decompression unit is placed between the main memory and the instruction cache. They used a Line Address Table (LAT) to map original code addresses to compressed block addresses. Lekatsas et al. [7] proposed a dictionary based decompression prototype that is capable of decoding one instruction per cycle. The idea of using dictionary to store the frequently occurring instruction sequences has been explored by various researchers [3], [17]. The techniques discussed so far target reduced instruction set computer (RISC) processors. There has been a significant amount of research in the area of code compression for very long instruction word (VLIW) and and no instruction set computer (NISC) processors. The technique proposed by Ishiura and Yamaguchi [13] splits a VLIW instruction into multiple fields, and each field is compressed by using a dictionary-based scheme. Gorjiara et al. [5] applies similar approach in splitting the control words into different fields and compressing them using multiple dictionaries.

Several techniques ([8], [12]) have been proposed to improve the standard dictionary based code compression by considering mismatches. Seong et al. [18, 19, 20] improve the compression efficiency further by using bitmasks. Figure 3 shows an example of the bitmask-based code compression. The basic idea is to create repeating patterns from mismatches by storing the differences during compression using bitmasks. We try to encode the differences (bitmask) using minimal number of bits to improve the compression efficiency without introducing any additional decompression overhead.

3 Background: Bitmask-Based Compression

This section describes the existing bitmask based compression using illustrative examples. First, we describe the standard dictionary-based approach. Next, we describe the bitmask based compression that improve the standard approach by considering mismatches (using bitmasks).

3.1 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 2 shows an example of dictionary based code compression using a simple program binary and encoding format shown in Figure 1. 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% (using Equation (1)).

isCompressed *(1−bit)*	Dictionary Index *($log_2(d)$ bits)*

(a) Compressed with dictionary index

isCompressed *(1−bit)*	UnCompressed Word *(w−bits)*

(b) Uncompressed word

Figure 1. Format for dictionary-based compression

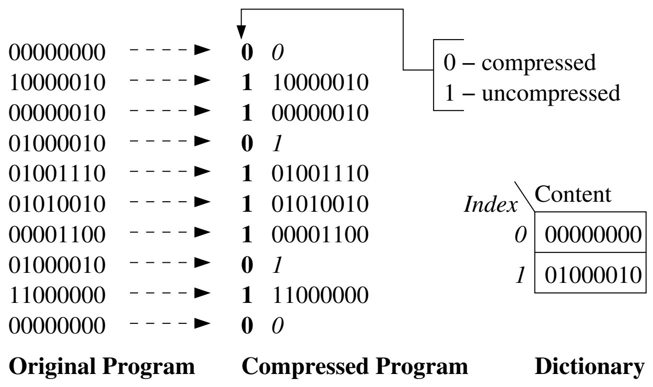

Figure 2. Dictionary-based compression example

3.2 Bitmask-based Compression (BMC)

Seong et al. [20] improve the standard dictionary based compression techniques by considering mismatches. The basic idea is to find the data sequences that are different in few consecutive bit positions and store that information in the compressed program. Compression ratio will depend on how many bit changes (and length of each consecutive change) are considered during compression. Figure 3 shows the same example (80-bit binary in Figure 2) compressed using one bitmask allowing 2 consecutive bit changes starting at even locations. In this example, we are able to compress all the mismatched words using smaller number of bits and achieve the compression ratio of 87.5%.

Figure 4 shows the encoding format used by these techniques for a w-bit program. In general, the compression ratio depends mainly on the word width, dictionary size and the number of bitmasks used. A smaller word size results in more direct matches whereas it increases the number of words needed to be compressed. A larger dictionary size can match more words replacing them with dictionary index but at the cost of increased dictionary index bits. More number of bitmasks results in more compressed words but requires more bits to encode the bitmask information. Our approach reduces the number of bits required for encoding bitmasks without adding any extra overhead during decompression.

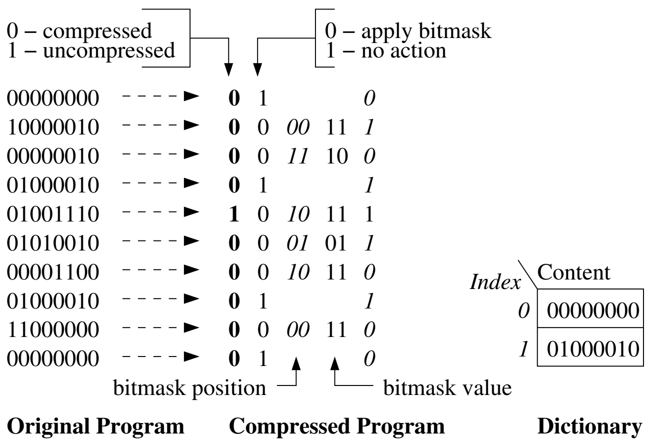

Figure 3. Bitmask-based compression example

Bitmask-based compression has been successfully used in different domains including code compression [18, 19, 20], manufacturing test compression [9], and NISC control word compression [4].

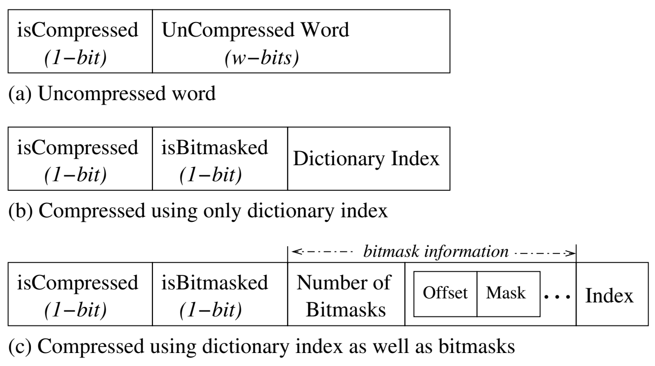

Figure 4. Format for bitmask-based compression

4 Efficient Representation of Bitmasks

We describe our approach using illustrative examples. Figure 5 shows compression using a two-entry dictionary and one bitmask that records 1-bit change. It is clear that if we know the location, additional bitmask information is redundant. Thus it can be removed from the bitmask encoding. For example, the second pattern in the Figure 5 is differing at location '000' (from left) with bit difference of 1 (XOR operation will give us the required bitmask). Our approach removes this bit from encoding.

Figure 6 shows another example that uses a bitmask to encode 2 bit differences. In this example, four entries (second, third, fourth and fifth input patterns) are compressed using one bitmask (of size 2). Note that the second pattern is compressed using bitmask '01' applied at location 0. We can also encode the same difference using bitmask '10' applied at location 1. Thus we have only two unique differences to be

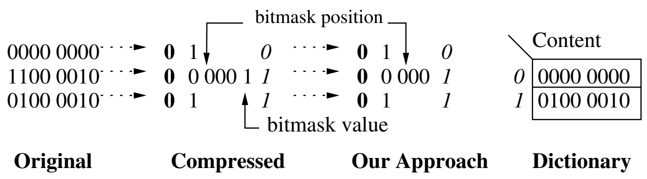

Figure 5. Encoding 1-bit change using bitmasks

encoded ('10' and '11'). These two encodings require only one bit as shown in Figure 6 (third column – Our Approach).

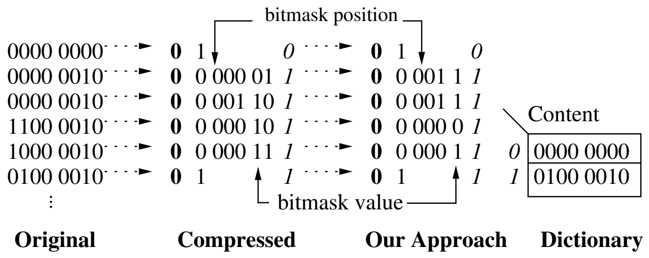

Figure 6. Encoding 2-bit changes using bitmasks

In general with a 2-bit bitmask, there are four different bitmask values possible { '00', '01', '10', '11' } as shown in Figure 7. Out of these possibilities the first pattern ('00') never occurs as this indicates that there are no differences. The second and third bitmasks are equivalent except that offset of these differ by one. Hence both can be represented using '10' bitmask. Thus there are only two bitmasks ('10' and '11') that needs to be encoded. Hence a single bit is sufficient to represent any 2-bit bitmasks.

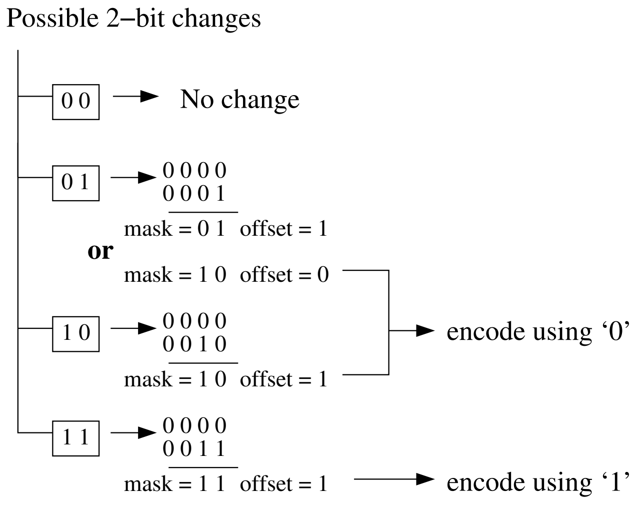

Figure 7. Four possible values of a 2-bit bitmask

Similarly, Figure 8 shows how a 3-bit bitmask can be encoded using 2-bits with different offset values. Theorem 1 proves that it is always possible to encode a n-bit bitmask using only n-1 bits.

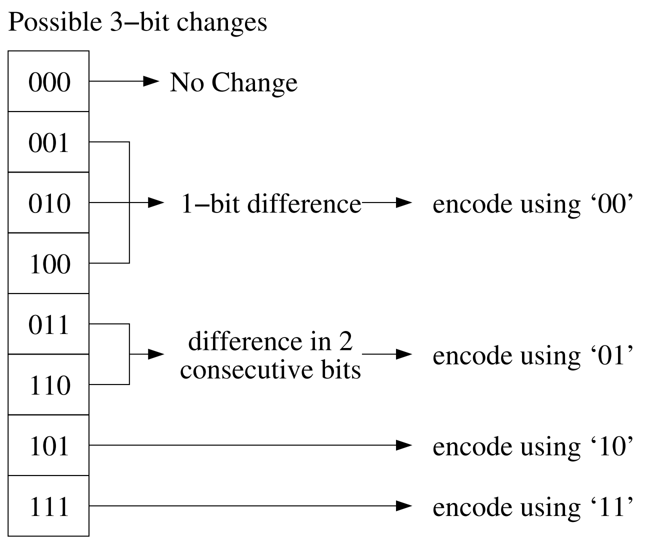

Figure 8. Encoding 3-bit changes using 2 bits

Theorem 1: Let n be the number of consecutive bit changes to encode between two words w_1 and w_2. Then $n-1$ bits are sufficient to encode the n bit changes.

Proof: We prove this theorem using induction. It is clear that we do not need any bits to indicate the difference for $n = 1$, since the location (offset) information is sufficient. Similarly as described earlier (Figure 7), we need only one bit to encode two bit changes ($n = 2$).

Let us assume that a $n-1$ bit bitmask can be encoded using $n-2$ bits. Now we show that, to encode n bits we require only $n-1$ bits. A n-bit bitmask can encode 2^n differences out of which $f_{lower} = \frac{2^n}{2} = 2^{n-1}$ differences have MSB set to 0 and $f_{upper} = \frac{2^n}{2} = 2^{n-1}$ differences have MSB set to 1. To encode f_{lower} differences without the MSB we need only $n-2$ bits (based on our assumption). Similarly, we need $n-2$ bits to encode f_{upper} differences without the MSB. Now to encode the differences with MSB we need an extra bit along with $n-2$ bits. Thus we need a total of $n-2+1 = n-1$ bits to encode n-bit bitmasks. This completes the proof.

5 Experiments

The application of our approach improves the compression efficiency in scenarios where input patterns are compressed using small dictionaries and most of the words are encoded using one or more bitmasks. To evaluate the effectiveness of our approach, we have applied it on three different domains – code compression, FPGA configuration bitstream compression, and NISC control word compression.

5.1 Code Compression

Embedded systems are constrained by the available memory. Code compression techniques address this issue by reducing the code size of application programs. The bitmask based code compression discussed in Section 3.2 reduces the instruction size significantly by storing the most frequently occurring instructions in a restricted dictionary and replacing them with shorter dictionary index. The instructions which are different from the dictionary entries in some bit positions are encoded using dictionary index along with bitmasks.

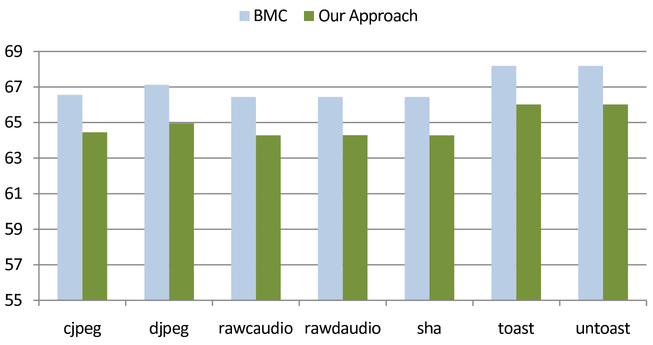

Figure 9. BMC versus our approach

Figure 9 shows the comparison of our approach with bitmask-based compression (BMC [20]). These experiments were conducted using 32-bit binaries with dictionary size of 512 entries and two sliding bitmasks (2- and 3-bits wide). We have used benchmarks from various application domains including TI and Mediabench [2] benchmark suites. It is observed that on an average there is an improvement of 2 to 3% on overall compression efficiency. An advantage of this technique is that the improvement is achieved without adding any extra logic or overhead on decompression.

5.2 FPGA Bitstream Compression

Field programmable gate arrays (FPGA) store configuration bitstream in memories which are usually limited in capacity and bandwidth. As FPGAs are commonly used in reconfigurable systems and application specific integrated circuits (ASIC), configuration memory becomes a key factor in determining the number of IP cores that a reconfigurable system can support and the configuration delay. Compression of configuration bitstream solves memory constraint issue by reducing the size of the bitstreams, whereas fast decompression increases the communication bandwidth reducing the configuration delay.

Figure 10 shows the comparison with bitmask-based compression (BMC). We have used benchmarks from various application domains including image processing and encryption benchmarks [10]. These experiments were conducted using 16-bit FPGA bitstreams with dictionary size of 512 entries and one 2-bit sliding bitmask. It is found that on an average there is an improvement of 2 to 3% on overall compression efficiency.

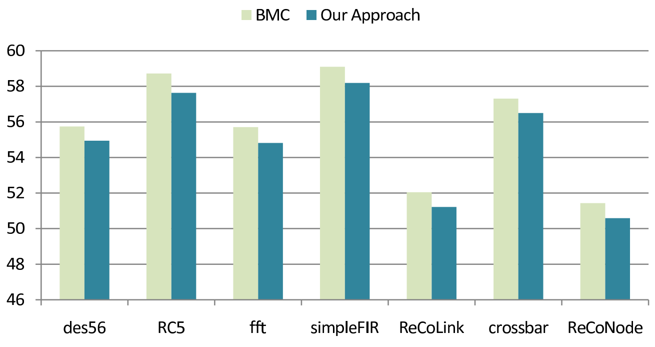

Figure 10. FPGA bitstream compression

5.3 NISC Control Word Compression

It is not always efficient to run an application on a generic processor, whereas implementing a custom hardware is not always feasible due to cost and time considerations. One of the promising direction is to design a custom datapath for each application using its execution characteristics. The abstraction of instruction set in generic processors limits from choosing such custom data path. No instruction set architecture (NISC [16]) promises faster performance guarantees by analyzing the datapath behavior and eliminating abstraction of instruction set to choose a custom datapath, thus controlling the selection of optimal datapath to meet applications performance requirements. The datapath or control word (CW) tend to be at least 4 to 5 times wider than regular instructions, thus increasing the code size of applications. One promising approach is to reduce these control words by compressing them.

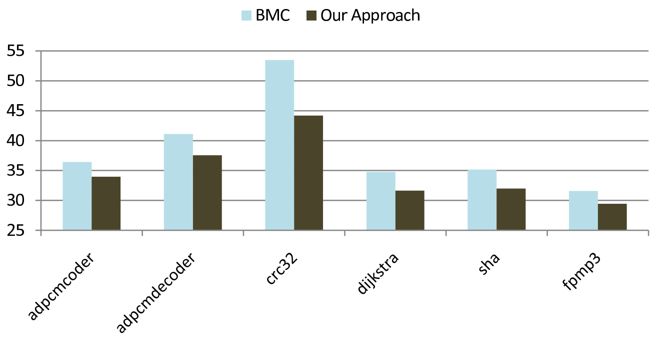

Figure 11. NISC control word compression

Figure 11 shows the comparison with bitmask-based compression (BMC). These experiments were conducted using 32-bit control words with dictionary size of 1024 entries and two sliding bitmasks (2- and 3-bits wide). The control word benchmarks are generated by NISC compiler [5] and is based on MiBench [6] benchmarks. It is found that on an average there is an improvement of 5 to 10% on overall compression efficiency.

The improvement of compression efficiency by our approach is dependent on the number of bitmasks used in compressed patterns since it reduces the size (number of bits) of each bitmask by one. Clearly, this approach will perform better in the scenarios where multiple bitmasks are used for compression. It is important to note that the improvement in compression efficiency does not introduce any decompression overhead since our method changes only the offset information and reduces the bitmask size.

6 Conclusions

This paper presented a novel encoding scheme to efficiently represent a n-bit bitmask using n-1 bits, thus reducing the compressed data size. We applied this technique for compressing a wide variety of input patterns including application programs, FPGA configuration bitstreams, as well as NISC control words. Our experimental results demonstrated an improvement of 3 to 10% in compression efficiency without introducing any area or performance penalty. Our approach is applicable in all compression scenarios where mismatch (correction) information is used but it is most effective in scenarios where small dictionaries are used for compression and most of the data patterns are compressed using one or more bitmasks. Our approach can also be applied in other domains where differences are encoded as offset and value pairs to reduce the bits required to encode these differences.

Acknowledgments

This work was partially supported by NSF CAREER award 0746261. We would like to thank Xiaoke Qin for his insightful comments and suggestions.

References

[1] A. Wolfe and A. Chanin. Executing compressed programs on an embedded RISC architecture. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, pages 81–91, 1992.

[2] C. Lee, M. Potkonjak and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, pages 330–335, 1997.

[3] C. Lefurgy, P. Bird, I. Chen and T. Mudge. Improving code density using compression techniques. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, pages 194–203, 1997.

[4] C. Murthy and P. Mishra. Bitmask-based control word compression for NISC architectures. In *Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2009.

[5] B. Gorjiara and D. Gajski. FPGA-friendly code compression for horizontal microcoded custom IPs. In *Pro-*

ceedings of Field Programmable Gate Arrays (FPGA), 2007.

[6] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of International Workshop on Workload Characterization (WWC)*, 2001.

[7] H. Lekatsas, J. Henkel and V. Jakkula. Design of an one-cycle decompression hardware for performance increase in embedded systems. In *Proceedings of Design Automation Conference (DAC)*, pages 34–39, 2002.

[8] J. Prakash, C. Sandeep, P. Shankar and Y. Srikant. A simple and fast scheme for code compression for VLIW processors. In *Proceedings of Data Compression Conference DCC*, page 444, 2003.

[9] K. Basu and P. Mishra. A novel test-data compression technique using application-aware bitmask and dictionary selection methods. In *Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 83–88, 2008.

[10] D. Koch, C. Beckhoff, and J. Teich. Bitstream decompression for high speed FPGA configuration from slow memories. In *Proceedings of International Conference on Field Programmable Technology (ICFPT)*, pages 161–168, 2007.

[11] D. Koch, C. Beckhoff, and J. Teich. FPGA bitstream compression benchmark. Dept. of Computer Science 12, University of Erlangen-Nuremberg, 2007.

[12] M. Ros and P. Sutton. A hamming distance based VLIW/EPIC code compression technique. In *Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, pages 132–139, 2004.

[13] N. Ishiura and M. Yamaguchi. Instruction code compression for application specific VLIW processors based of automatic field partitioning. In *Proceedings of Synthesis And System Integration of Mixed Information Technologies SASIMI*, pages 105–109, 1997.

[14] Opencore. IP Core repository. Stockholm, Sweden, 1999. Opencore.org.

[15] J. Pan, T. Mitra, and W. Wong. Configuration bitstream compression for dynamically reconfigurable FPGAs. In *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pages 766–773, 2004.

[16] M. Reshadi. No-Instruction-Set-Computer (NISC) technology modeling and compilation. In *PhD thesis*, Irvine, CA, USA, 2007. University of California Irvine.

[17] S. Liao, S. Devadas and K. Keutzer. Code density optimization for embedded DSP processors using data compression techniques. In *Proceedings of Advanced Research in VLSI*, pages 393–399, 1995.

[18] S. Seong and P. Mishra. A bitmask-based code compression technique for embedded systems. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 251–254, 2006.

[19] S. Seong and P. Mishra. An efficient code compression technique using application-aware bitmask and dictionary selection methods. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 582–587, 2007.

[20] S. Seong and P. Mishra. Bitmask-based code compression for embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(4):673–685, April 2008.