Des Autom Embed Syst (2010) 14: 265–284
DOI 10.1007/s10617-010-9057-z

# Squashing code size in microcoded IPs while delivering high decompression speed

**Chengmo Yang · Mingjing Chen · Alex Orailoglu**

Received: 3 February 2010 / Accepted: 23 June 2010 / Published online: 16 July 2010

**Abstract** Microcoded customized IPs offer superior performance and direct programmability of micro-architectural structures compared to instruction-based processors, yet at the cost of drastically enlarged code sizes. Code compression can deliver size reductions but necessitates attention to performance issues, so that the performance benefits of microcoded IPs are not squandered in the process. To attain this goal, we propose in this paper a fast code compression technique through exploiting the fact that the microcodes contain a sizable amount of unspecified bits. Although the values and the positions of the specified bits are highly irregular, the proposed technique can still flexibly and precisely fill in these fully specified bits through utilizing a linear network. The linear property inherent in the compression strategy in turn enables the development of an extremely low-overhead decompression engine. At runtime, the decompressed code can be generated in such a way that all the specified bits can be filled as required by a fixed-bandwidth XOR network. The combination of the proposed flexible XOR-based network with a minimum two-level storage for highly specified fields, such as immediate values, offers utmost code compression, attained within a negligible amount of performance and hardware overhead.

**Keywords** Microcode compression · Linear compression network · Microcoded processors

C. Yang (✉)
Electrical and Computer Engineering Department, University of Delaware, 140 Evans Hall, Newark, DE 19716-3130, USA
e-mail: chengmo@udel.edu

M. Chen · A. Orailoglu
Computer Science and Engineering Department, University of California, 9500 Gilman Drive, La Jolla, San Diego, CA 92093, USA

M. Chen
e-mail: mjchen@cs.ucsd.edu

A. Orailoglu
e-mail: alex@cs.ucsd.edu

## 1 Introduction

Shrinking time-to-market and high demand for productivity has driven the use of microcoded customized IPs in embedded system design. In such design platforms, such as PICO [1], ARM OptimoDE [2], TIPI [3], NISC [4], and FlexCore [5], the complex and time-consuming task of instruction set development is completely eliminated. Instead, these microcoded IPs directly expose the entire control of microarchitectural components to the compiler. Applications thus can be directly compiled to microcodes, and the compiler can directly exploit parallelism across functional units. Such architectures, typically referred to as *Horizontal Microcoded Architectures*, can be designed without costly decoders, controllers and hardware schedulers, thus offering superior performance, lower power, and lower area than conventional RISC processors.

Despite the aforementioned benefits, the problematic issue of the enlargement of the code size of microcoded IPs remains. Storing microcodes directly on-chip requires large memory blocks that impose significant area and power overhead. As on-chip memory is one of the most limiting resources in cost-sensitive embedded systems, microcoded IPs necessitate aggressive code compression techniques. Meanwhile, to ensure that the performance benefits of microcoded IPs are not squandered as a result of code compression, the on-line decompression engine should provide high throughput instruction flow within highly constrained latency. Compression techniques based on variable-length encoding [6, 7] thus are not suitable for such systems, as these approaches require more complex decoders, incurring costs in performance, area and power in addition to long design and verification times. Dictionary-based compression techniques fail to meet the bar as well, as these techniques incur significant overhead either in storing or in constantly reloading the large dictionary.

An important property of horizontal microcodes is that each microcode typically contains a sizable number of *don't care* bits (denoted as 'X'). These X bits correspond to the control signals of the functional units that are idle at a given cycle. For example, if at a given cycle there is no write operation to the register file, the corresponding *write address* signals become *don't cares*. Clearly, these bits can be mapped to either 0's or 1's in the final executable binary without affecting program behavior. This flexibility can be exploited to attain a *dictionary-free fixed-length encoding* technique. Specifically, in this paper we propose an XOR network-based compression technique that sets the unspecified bits in such a way that the values form a linear relationship with the specified bits. This linear relationship ensures that the fully specified bits, although their values and positions may be highly irregular, can be precisely filled. Meanwhile, this linear relationship also enables the development of an extremely low-cost decompression engine, composed of only a fixed-bandwidth XOR network, thus minimizing the associated performance and power overhead.

The efficacy of the proposed compression technique is determined by the quality of the linear network as well as the amount of unspecified bits in microcodes. To improve the compression ratio, we furthermore propose a set of optimization techniques to maximally diminish the correlation of the linear network and to balance the number of unspecified bits across various microinstructions. The combination of the proposed flexible XOR network with a minimum on-chip storage for highly specified fields, such as immediate values, offers utmost code compression, attained within a negligible level of performance and hardware overhead.

A preliminary version of the proposed linear network-based compression framework was published in [8]. The work herein incorporates these technical ideas, evaluates them more thoroughly and most importantly, extends the work by theoretically analyzing its efficacy in **two** aspects. First of all, to illustrate the advantage of using a linear network for microcode compression, we perform a finer-grained comparison of this technique against traditional compression schemes, particularly the dictionary-based microcode compression schemes. Moreover, to illustrate the advantage of the proposed network construction approach, we perform a mathematical analysis to illustrate the benefit of imposing a *1-degree overlap* constraint during network construction.

The rest of the paper is organized as follows. Section 2 briefly reviews code-size reduction techniques for embedded processors and microcoded IPs. Section 3 outlines the technical motivation for this work. The proposed linear network-based compression technique and the techniques for improving the compression ratio are discussed in Sects. 4 and 5, respectively. The efficacy of the proposed compression technique is experimentally verified in Sect. 6, and Sect. 7 offers a brief set of conclusions.

## 2 Technical background

As embedded systems typically impose a strict cost and power budget, a number of code-size-reduction techniques have been proposed for embedded processors. These techniques can be categorized into three groups.

– **Compiler optimizations:** Techniques such as register renaming, interprocedural optimization, and procedural abstraction of repeated code fragments [9, 10] can be embedded in a compiler to reduce code size. These techniques impose neither runtime decompression overhead nor hardware cost, yet necessitate significant amount of modifications of compilers and/or linkers.
– **ISA modification:** This approach modifies or customizes the instruction set architecture to reduce code size. For example, the Thumb instruction set [11] consists of 16-bit versions of the 36 most frequently used 32-bit ARM instructions, while MIPS16 [12] contains a subset of 32-bit MIPS-III instructions. Clearly, the redesign of the instruction set, together with a new hardware decoder and a new set of software development tools, such as specialized compilers, assemblers, and linkers, constitutes the cost of this approach.
– **VLIW no-op elimination:** In VLIW processors, due to the limited parallelism in applications, instruction slots are frequently filled with *no-ops*. To reduce code sizes, modern VLIW architectures (such as TMS320C6x) utilize the idea of *Various Length Execution Set (VLES)*, wherein *no-ops* are removed as much as possible and a few shortened instructions are packed into one wide fetch packet.
– **Code compression:** In these techniques, the executable program is compressed offline and decompressed on-the-fly during execution [6]. As the compression and decompression are typically performed in a manner transparent to the processor and the compiler, these techniques are more desirable for embedded cores and microcoded IPs as compared to the compiler optimization and ISA modification approaches. The efficiency of a code compression scheme can be measured by the *compression ratio*, defined as follows:

$$\textit{Compression Ratio} = \frac{\textit{Compressed code size}}{\textit{Original code size}} \tag{1}$$

Compared to general purpose data compression approaches, code compression exhibits extra challenges in improving the speed of decompression, as it is to be performed at runtime. As some instructions (such as branch, jump, and return, for example) may alter the control flow, it is essential to ensure that each instruction can be decompressed *individually* with no reliance on any preceding instructions. Moreover, it is also necessary to establish a mapping between the original address space and the compressed address space so that upon a control altering instruction, the address of the target instruction can be easily recalculated. This requirement is more challenging for *variable-length* encoding techniques, wherein the mapping between the two address spaces is irregular.

### 2.1 Variable-length encoding

Compression techniques based on variable-length encoding exploit the uneven occurrence ratio of instructions to attain code size reduction. For example, in [6], the Compressed Code RISC Processor (CCRP) based on Huffman encoding, is proposed. Unique instructions in the program are stored in a dictionary, wherein the indices of the instructions are determined by Huffman coding. In IBM Codepack [7], each 32-bit instruction is evenly split into two parts, while two dictionaries are used to record the unique patterns of each part. This two-dictionary architecture is furthermore extended in [13], wherein the instruction bits are rearranged so as to balance the dictionary sizes. This concept of dictionary-based compression has been extended to sequence of instructions [14, 15], wherein repetitive sequences of instructions, instead of individual instructions, are identified and stored in a dictionary.

While the aforementioned techniques deliver low compression ratio, the decompression speed is typically quite slow due to the variable sizes of compressed instructions. To minimize the decompression overhead, these variable-length encoding techniques usually rely on the existence of a cache, so that decompression can be performed only upon a cache miss. These architectures are denoted as *pre-cache* decompression. Compression thus does not improve the utilization of the cache. Moreover, as missed instructions do not reside at the same address in the cache as in memory, a line address table (LAT) [6] is needed to record the mapping between the compressed and the original address spaces. This extra overhead, together with the necessity of caches, makes these variable-length compression techniques less desirable for cost-sensitive embedded systems, especially microcoded IPs.

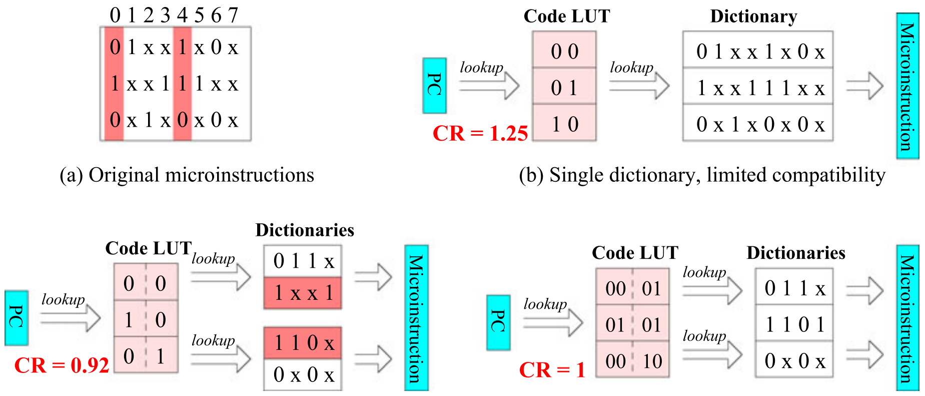
### 2.2 Fixed-length encoding

To attain high-speed decompression in a cache-free embedded system, fixed-length encoding approaches are usually employed.

Essentially, a program can be viewed as an array of $W$ words (microinstructions) of $B$ bits each. One set of compression techniques, typically denoted as *ROM encoding* [16], exploit the *bit-wise compatibility* in microcoded programs. The bit dimension is partitioned into groups so that each group of signals can be driven by the same decoder. The distinct bits contained in each group should be *compatible* such that their values are never set to 1 at the same cycle. Clearly, this bit-wise compatibility decreases as the number of microcoded instructions increases, implying that the technique is only effective for small programs.

Another set of fixed-length encoding techniques exploit the *word-wise compatibility* in microcoded programs. The basic idea is to store all the unique microinstructions in a *lookup table* (LUT). Each microinstruction is then replaced with an index to the dictionary which contains the original instruction sequence. The compression ratio thus is determined by the number of entries in the LUT, which is in turn determined by the number of unique codewords within a program. Obviously, the attainment of a low compression ratio hinges on the existence of high repetition of codewords in the program.

As two distinct codewords may partially share a sequence of values in common, researchers have also proposed to vertically partition each codeword of a program into multiple groups [17, 18] to maximally exploit the potential repetition. Although the compression



(a) Original microinstructions (b) Single dictionary, limited compatibility

(c) Multiple dictionaries, redundant entries (d) Merged dictionary, larger size of indices

**Fig. 1** Inability of LUT-based compression to fully utilize the flexibility offered by 'X' bits

ratio is improved, the required LUT size is enlarged as a result, which in turn increases the LUT index size. It has been reported in [19] that for the EEMBC benchmarks, the technique proposed in [17] requires a LUT with 300–400 entries. Such a large LUT thus imposes significant hardware overhead and access latency onto the target embedded processor or microcoded IP. To reduce the LUT size, researchers have also proposed to dynamically reload the content of the LUT using dedicated table manipulating instructions [19]. However, such instructions will not only increase the static code size but also incur performance overhead during program execution.

## 3 Technical motivation

An important property of microcoded IPs is that compared to instruction-based processors, they display an extra degree of freedom in specifying the microcoded instructions. Each microcode typically exhibits a sizable number of unspecified bits, corresponding to the control signals of the functional units that are idle at a given cycle. Examples include register-file read and write addresses, MUX selection, and ALU operation signals.

Although the unspecified bits in microinstructions can be flexibly filled as either 0's or 1's in the final executable binary, this flexibility has not been fully exploited by the traditional LUT-based code compression approaches. Although the program may contain a significant number of unspecified bits, the attainable compression ratio is still constrained by the number of *vertical bit conflicts*, particularly the conflicts in fully specified columns of instructions. To concretely illustrate this constraint, Fig. 1 presents three microinstructions, compressed using three LUT-based compression schemes that respectively deliver compression ratios of 1.25, 0.92, and 1. Although each microinstruction contains 4 'X' bits, Fig. 1a shows that the conflicts in the two fully specified columns, namely the 0th and the 4th bits, fully distinguish the three instructions. As a result, if a single dictionary is used to capture entire microinstructions, such as the one in Fig. 1b, all three instructions need to be stored in the dictionary. This scheme therefore delivers no benefit in terms of code size reduction. To enhance word-wise compatibility, each microinstruction can be vertically divided into two partitions that are mapped to distinct dictionaries, as shown in Fig. 1c. However, this scheme falls short of exploiting the compatibility among distinct partitions, resulting in redundant entries (highlighted in Fig. 1c) in each dictionary. To eliminate such redundancy, the two dictionaries can be merged, as shown in Fig. 1d. However, as the merged dictionary contains more entries, the corresponding index size needs to be enlarged as well, thereby hurting the compression ratio.

The examination of traditional LUT-based compression indicates that techniques relying on the exploitation of inter-codeword compatibility can not fully utilize the flexibility provided by the unspecified bits in codewords. To attain low compression ratio while minimizing the associated on-chip storage, it is essential to directly exploit the flexibility in filling these unspecified bits. The challenge, however, is to compactly capture the specified bits regardless of their irregularity.

In a microcoded program, not only the values, but the positions of the specified bits are also irregular. To compactly capture these bits, a set of linear transformations need to be applied, thus motivating the development of a *linear network-based* compression technique. Specifically, we propose to directly utilize the flexibility offered by the unspecified bits, by setting them in such a way that their values form a linear relationship with the specified bits. Such a linear relationship enables not only the values but furthermore the bit positions of the specified bits to be captured in a compact form. In this way, these specified bits can be precisely filled, even though the distribution of them throughout the codeword may display high levels of irregularity.

## 4 LUT-free fixed-length microcode compression

This section presents in detail the proposed linear-network based microcode compression scheme. Given a set of linear equations that define the relationship among the various bit positions, each microinstruction can be compressed into a seed. The seed is selected in such a way that the specified bits can be precisely generated, while the unspecified bits can be flexibly filled in. At runtime, a decompressor, composed of a number of XOR gates, takes these seeds as inputs and generates microinstructions. This compression/decompression process is concretely shown in Fig. 2. In Fig. 2a, the three microinstructions shown in Fig. 1 are compressed into 4-bit seeds. The corresponding linear equations and the decompression network are shown in Figs. 2b and 2c.

Clearly, the attainable compression ratio of this proposed scheme is determined by the structure of the linear network. More formally, an $N$-to-$M$ XOR network receives as inputs an $N$-bit seed, and generates an $M$-bit fully specified code through performing $M$ groups of XOR operations on the seed bits. Such a network delivers a compression ratio of $N/M$.

To develop a high-quality linear network-based compression technique, two questions need to be resolved: (1) Given a linear network, how do the seeds get generated? (2) How is the quality of the linear network to be improved so as to attain lower compression ratios? These questions are addressed in detail in the remaining parts of this section.

### 4.1 XOR network-based code compression

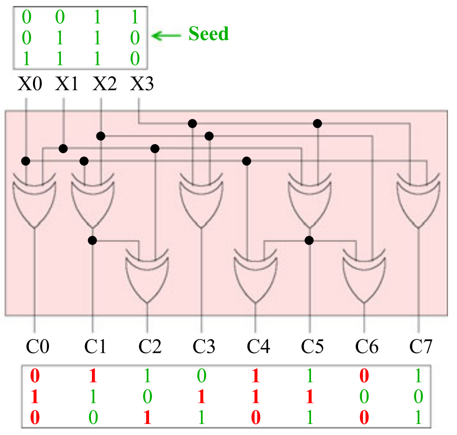
As each individual XOR operation performed in the network is a linear operation, the transformation from the $N$-bit seed to the $M$-bit codeword constitutes a linear transform, with each of the $M$-bit codewords being a linear combination of the multiple bits in the seed. Moreover, such a linear transformation can be characterized by the structure of the network, which can be furthermore represented by an $M \times N$ coefficient matrix, wherein a ‘1’ in

| $C_0C_1C_2C_3C_4C_5C_6C_7 \rightarrow X_0X_1X_2X_3$ |
|---|
| 0 1 x x 1 x 0 x → 0 0 1 1 |
| 1 x x 1 1 1 x x → 0 1 1 0 |
| 0 x 1 x 0 x 0 x → 1 1 1 0 |

**(a)** 8-bit microcodes to 4-bit seeds

$$\begin{aligned}
C_0 &= X_0 \oplus X_1\\
C_1 &= X_0 \oplus X_2\\
C_2 &= X_0 \oplus X_1 \oplus X_2\\
C_3 &= X_2 \oplus X_3\\
C_4 &= X_0 \oplus X_1 \oplus X_3\\
C_5 &= X_1 \oplus X_3\\
C_6 &= X_1 \oplus X_2 \oplus X_3\\
C_7 &= X_0 \oplus X_3
\end{aligned}$$

**(b)** Linear equations



**(c)** XOR gate-based hardware decompressor

**Fig. 2** Linear network-based compression/decompression

the matrix represents the existence of a connection between the output and input bits of the XOR network. As an example, the matrix representation of the XOR network in Fig. 2c is shown in (2).

$$\begin{bmatrix} 1&1&0&0\\ 1&0&1&0\\ 1&1&1&0\\ 0&0&1&1\\ 1&1&0&1\\ 0&1&0&1\\ 0&1&1&1\\ 1&0&0&1 \end{bmatrix} \begin{bmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{bmatrix} = \begin{bmatrix} C_0\\ C_1\\ C_2\\ C_3\\ C_4\\ C_5\\ C_6\\ C_7 \end{bmatrix} \tag{2}$$

To generate the original microinstruction, the set of linear equations, which describe the linear combination relationship among these bits, needs to be solved. Clearly, the unspecified bits, as they can be flexibly filled in, impose no constraint on seed generation. Only the linear equations corresponding to specified bits need to be satisfied. Therefore, depending on the positions of the specified bits, a subset of the rows of (2) is selected. For example, in the first microinstruction in Fig. 2a, only $C_0$, $C_1$, $C_4$ and $C_6$ are specified, with the corresponding values being [0 1 1 0]. The linear equation system for this code thus can be specified as follows:

$$\begin{bmatrix} 1&1&0&0\\ 1&0&1&0\\ 1&1&0&1\\ 0&1&1&1 \end{bmatrix} \begin{bmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{bmatrix} = \begin{bmatrix} C_0\\ C_1\\ C_4\\ C_6 \end{bmatrix} = \begin{bmatrix} 0\\ 1\\ 1\\ 0 \end{bmatrix} \tag{3}$$

These equations can be solved through the utilization of any Gauss-Jordan elimination [20] methodology. If such a linear equation system has at least one solution, the corresponding microinstruction thus can be successfully compressed.

The compression approach outlined above clearly confirms that the proposed technique is able to compactly capture the specified bits regardless of their irregularity. Given two codewords with distinct distribution of the specified bits, different sets of linear equations are selected. The likelihood of finding a seed, which determines the attainable compression ratio, is independent of the particularity of the specified bits. Instead, this likelihood is determined by the number of unspecified bits. Clearly, the more unspecified bits in a codeword, the fewer equations need to be satisfied, and hence the higher the likelihood that a valid seed can be found. In fact, a detailed examination indicates that compression is guaranteed to be successful if the number of specified bits in a codeword does not exceed the *rank* of the coefficient matrix, since in this case there exists at least one solution in such a linear equation system. On the other hand, if the number of specified bits exceeds the rank of the matrix, the existence of a solution is determined by the values of the specified bits.

To maximally preclude the generation of an unsolvable linear system, the *root cause* of linear equation unsolvability needs to be identified. If we incorporate the column vector of the specified bits into the coefficient matrix, an *augmented* matrix with a size of $M \times (N+1)$ can be constructed. A comparison between the *augmented* matrix and the *coefficient* matrix indicates that no solution exists for this linear system if and only if the rank of the augmented matrix exceeds the one of the coefficient matrix. An illustrative example is shown in Fig. 3, wherein the three rows in the coefficient matrix are *linearly dependent* such that any row can be generated through XORing the other two rows. In contrast, in the augmented matrix this linear dependency disappears, thus forcing the augmented matrix to display a larger rank than the coefficient matrix.

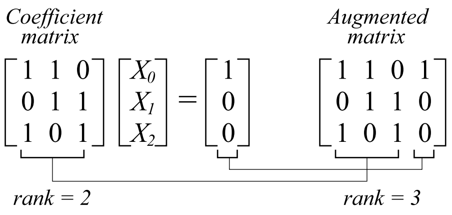
The occurrence frequency of the unsolvable cases strongly depends on the linear dependencies among the various XOR equations. A high correlation would greatly reduce the rank of the coefficient matrix, thus increasing the likelihood of the unsolvability condition outlined above. Therefore, the construction of a low correlation XOR network is essential for improving the effectiveness of the proposed compression scheme.

## 4.2 XOR network construction

In general, the construction of an XOR network for compression/decompression purposes should take into consideration three factors, namely, the *latency*, the *cost*, and the *linear dependencies* among outputs. Clearly, latency reduction and cost reduction can be easily attained through reducing the levels of XOR gates and the total number of XOR gates in the network, respectively. In comparison, the reduction of *linear dependencies* among outputs is more crucial, as this factor directly constrains the attainable compression ratio.

In the proposed linear compression network, *input bit-overlap* is the root cause of the linear dependencies among outputs, since in this case one input bit can concurrently impact the results of multiple XOR functions. However, as the network, used as a decompressor, should produce more outputs than the number of inputs, the sharing of common inputs among distinct outputs cannot be fully eliminated. Therefore, our aim in network construction is not

**Fig. 3** No solution case



$$\begin{bmatrix}1&1&0\\0&1&1\\1&0&1\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1\\0\\0\end{bmatrix}\qquad\begin{bmatrix}1&1&0&1\\0&1&1&0\\1&0&1&0\end{bmatrix}$$

the full *elimination* of linear dependencies among XOR functions, but rather the *enlargement* of the minimum size of linearly dependent output groups. In the proposed linear compression network, a linear dependency can be eliminated if there exists one unspecified bit in the corresponding set of dependent bit positions. Accordingly, by enlarging the minimum size of the dependent groups, the occurrence frequency of the aforementioned unsolvable cases can be minimized even given a relatively low density of unspecified bits, thus enhancing the compression ratio.

### *4.2.1 Output correlation minimization*

In this subsection, we first introduce the conditions for a set of output equations to be *linearly dependent*, and then analyze the impact of *input bit-overlap* on the difficulty in forming dependent output groups. We show that by constraining the amount of input bit-overlap, the *signal correlation intensity* within the linear network can be largely reduced, thus significantly increasing the difficulty of forming dependent output groups.

Essentially, a set of output equations are *linearly dependent*, if and only if all the related inputs are covered an even number of times. In the example shown in Fig. 3, the three equations captured in rows are linearly dependent, as each column corresponding to an input is covered exactly twice.

The condition outlined above confirms that to form a dependent group, all the input bits that are covered for an odd number of times (denoted as *odd-cover inputs*) need to be eliminated. The size of a dependent group is therefore determined by how fast the odd-cover inputs are eliminated, which is in turn determined by the maximum number of common input bits between any two XOR functions (denoted as *overlap degree*). The higher the overlap degree is, the faster the odd-cover inputs can be eliminated, and hence the smaller the size of dependent output group is.

To illustrate the impact of *overlap degree* on the difficulty of forming dependent groups, let us examine the change in the number of *odd-cover inputs* bits during the construction of a dependent group. To simplify the analysis, we assume without loss of generality, that all the XOR functions are generated by a fixed number of inputs, denoted as $S$.

Initially, the first XOR function in the group introduces $S$ odd-cover bits that are only covered once by itself. Subsequently, whenever a function $C_i$ is added into the group, it can overlap with each of the $(i-1)$ functions already existing in the group at $p$ input positions, with $p$ being the *overlap degree*. If all the overlaps happen to be at the position of odd-cover bits, $C_i$ can end up eliminating a total number of $(i-1)p$ odd-cover bits. Meanwhile, as $C_i$ needs to cover $S$ inputs, it also needs to introduce $S-(i-1)p$ new inputs that obviously constitute odd-cover bits in the group. The combination of these two aspects implies that the incorporation of the function $C_i$ introduces a minimum number of $S-2(i-1)p$ odd-cover bits into the group. As a concrete example, Fig. 4 shows a dependent group composed of four 6-input XOR functions with an *overlap degree* of 2 (i.e., $S=6$ and $p=2$). Clearly, the first function $C_1$ introduces 6 new inputs, the second function $C_2$ eliminates 2 odd-cover

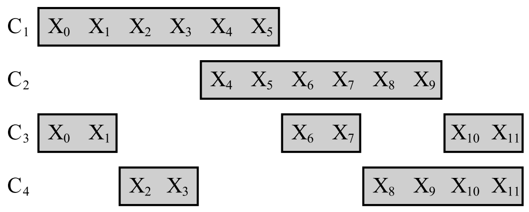


**Fig. 4** Minimum dependent group with an overlap degree of 2

bits while introducing 4 new inputs, and the third function $C_3$ eliminates 4 odd-cover bits while introducing 2 new inputs.

In sum, the number of odd-cover bits after incorporating the $n$th function can be computed by summing up the number of odd-cover bits introduced by each of these $n$ functions, as shown below:

$$\sum_{i=1}^{n}(S-2(i-1)p) = n*S - \sum_{i=1}^{n}2(i-1)*p \tag{4a}$$

$$= n*S-(n-1)n*p \tag{4b}$$

Equation (4b) indicates that the total number of odd-cover bits might increase when the group size, $n$, is small, but will eventually drop rapidly when more functions are added to the group. When the asymptotic curve specified by (4b) drops to the value of 0, all the odd-cover bits will have been eliminated and hence the functions $(C_1, C_2, \dots, C_n)$ form a dependent group. Therefore, the size of the minimum dependent group, denoted as $DGsize_{min}$, satisfies the following relationship:

$$DGsize_{min}(S,p) \geq \lceil S/p \rceil + 1 \tag{5}$$

Equation (5) clearly shows that the minimum size of dependent output groups is inversely proportional to the overlap degree $p$. A high overlapping level tends to accelerate the elimination of odd-cover inputs, thus engendering dependent groups of small sizes. In an extreme case, if no constraint is imposed on the overlap degree (i.e., $p = S$), a dependent group can be composed of two *identical* XOR functions. On the other extreme, by imposing a *1-degree overlap* constraint (i.e., $p = 1$) during network construction, the size of the minimum dependent group can be enlarged to $S + 1$.

### *4.2.2 1-Degree overlap network construction*

The analysis outlined in the last subsection clearly confirms that the 1-degree overlap constraint can effectively increase the difficulty in forming dependent groups, thus in turn delivering an appreciable compression ratio. Yet this constraint also sharply reduces the number of attainable XOR functions (output bits), thus necessitating the development of an algorithm capable of maximally identifying a set of input combinations that satisfy the strict overlap constraint.

Assuming that each output bit in the linear network is generated by an $S$-input XOR function, the set of $N$ input bits can be partitioned into $\lceil N/S \rceil$ disjoint sets. These disjoint sets constitute a *partition group*, denoted as *PG*. Clearly, if the inputs in each set are used to generate a single output, the disjointness ensures that the overlap degree of any two sets (i.e., two output functions) within a single *PG*, denoted as *intra-PG overlap degree*, is consistently 0. The problem of forcing the *overall overlap degree* to be 1 thus translates to the identification of a maximum number of *PG*s such that for any two sets in distinct *PG*s, the *inter-PG overlap degree* is 1.

To ensure 1-degree overlap among partition groups, we employ the deterministic partitioning strategy proposed in [21]. Specifically, the $k$th partition group is constructed through a linear shuffle of elements across the multiple sets in the 0th, that is, the original partition group, using the following shuffling strategy:

$$P(k,b,i) = P(0, b \oplus (k \otimes i), i) \tag{6}$$

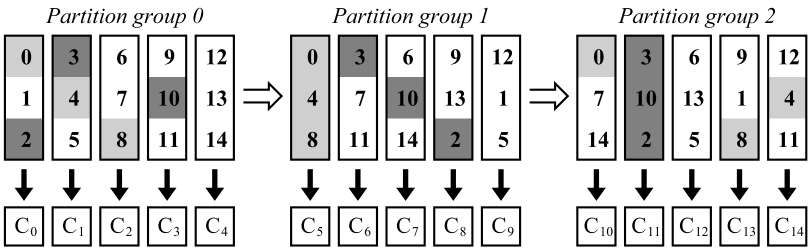


**Fig. 5** Partition group generation: 3 *PG*s, each contains 5 disjoint sets of 3 bits

with $P(k, b, i)$ denoting a specific instance of input selection for the $i$th element in the $b$th set of the $k$th partition group.

To satisfy the 1-degree overlap constraint between any two sets in any two *PG*s, the operation $b \oplus (c \otimes i)$ in (6) needs to be a *bijective function*, i.e., one-to-one and onto. For a partition group with a size of $B$ (i.e. it contains $B$ disjoint sets), the addition and multiplication operations defined in the Galois field $GF(B)$ can be used to construct the bijective shuffling function. If $B$ is a prime number, a straightforward implementation can be attained through modulo-$B$ addition and multiplication operations.

To concretely illustrate the shuffle function defined in (6) Fig. 5 presents an example for a 15-input XOR network construction. As each output bit is generated by XORing 3 input bits, the 15-bit inputs are evenly partitioned into 5 disjoint sets of 3 bits, yielding $B = 5$ and $S = 3$ for this particular example. In other words, each partition group contains 5 sets and each set contains 3 inputs. As the size of the partition group, 5, is a prime number, the modulo-5 addition and multiplication operations are utilized for constructing the various partition groups. Accordingly, each codeword can be generated using the following equation:

$$C_{k*B+b} = P(k, b, 0) \oplus P(k, b, 1) \oplus P(k, b, 2) \tag{7a}$$

$$P(k, b, i) = P(0, (k + i * b)\%B, i), \quad i \in 0, 1, 2 \tag{7b}$$

In Fig. 5, two sets of elements highlighted in $P_0$, namely, $(0, 4, 8)$ and $(3, 10, 2)$, are selected to form the first set in $P_1$ (corresponding to Codeword $C_5$) and the second set in $P_2$ (corresponding to Codeword $C_{11}$), respectively. As can be seen, to generate each set, the shuffling function selects only one and exactly one element from three distinct sets in $P_0$. This partition generation approach also guarantees the 1-degree overlap property between $P_1$ and $P_2$. In sum, a maximum number of 5 partition groups can be generated using the proposed approach. These groups are listed in Table 1, wherein each row corresponds to a partition group $PG_i$ and each box represents the three inputs that are used to generate one output bit. It can be easily verified from the table that the *intra-PG overlap degree* is 0, while the *inter-PG overlap degree* is 1.

The aforementioned XOR network construction approach can generate a total number of $B$ partition groups, each of which contains $B$ partitions. As a result, the proposed technique can generate a maximum number of $B^2$ outputs that satisfy the 1-degree overlap constraint. Clearly, the values of $S$ and $B$ can be determined according to the values of $N$ and $M$, that is, the input and output bandwidth of the XOR network. This relationship is formally specified in the following equation:

$$S = \lceil N/B \rceil \tag{8a}$$

$$B^2 \geq M \tag{8b}$$

**Table 1** 15-input XOR network construction ($S = 3$, $B = 5$)

| | $P_0$ | | | $P_1$ | | | $P_2$ | | | $P_3$ | | | $P_4$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $PG_0$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| $PG_1$ | 0 | 4 | 8 | 3 | 7 | 11 | 6 | 10 | 14 | 9 | 13 | 2 | 12 | 1 | 5 |
| $PG_2$ | 0 | 7 | 14 | 3 | 10 | 2 | 6 | 13 | 5 | 9 | 1 | 8 | 12 | 4 | 11 |
| $PG_3$ | 0 | 10 | 5 | 3 | 13 | 8 | 6 | 1 | 11 | 9 | 4 | 14 | 12 | 7 | 2 |
| $PG_4$ | 0 | 13 | 11 | 3 | 1 | 14 | 6 | 4 | 2 | 9 | 7 | 5 | 12 | 10 | 8 |

The network shown in Table 1 can generate up to 25 output bits using 15 inputs, implying that the best attainable compression ratio is 0.6. More formally, the ratio of (8a) over (8b) indicates that the **compression ratio** of this XOR network, defined as $N/M$, is constrained by the value $S/B$. The attainment of a lower compression ratio therefore requires a smaller value of $S$.

A smaller value of $S$ also implies that the network needs fewer number of XOR gates, which in turn reduces the hardware and performance cost of the network. More precisely, the entire network can be constructed using $S - 1$ levels of 2-input XOR gates, implying that the **total hardware cost** is $M * (S - 1)$ XOR gates.

However, a reduction in the value of $S$ degrades the randomness of the network, as the network construction approach delivers a bit overlap ratio of $1/S$. A set of experiments indicates that generating each output by XORing 3 or 4 inputs provides sufficient randomness, thus resulting in the value of $S$ being set to 3 or 4 during the network construction process.

## 5 Compression ratio enhancement

The discussion in Sect. 4.1 confirms that in the XOR network-based compression technique, each specified bit corresponds to one equation that needs to be satisfied during seed generation. Accordingly, the attainable compression ratio is determined by the ratio of the specified bits, rather than the particularity of these bits. However, in a program, the specified bits typically display a highly unbalanced distribution across microinstructions, implying that the overall compression ratio is constrained by the instruction with the maximum number of specified bits. To enhance the overall compressibility, it is therefore preferable to reduce the number of specified bits in the *hard-to-compress* codewords while retaining the unspecified bits. We propose **two** techniques to attain this goal, namely, a *column merging* technique and a *hybrid compression* approach.

Another important aspect of the proposed compression technique is that the attainable compression ratio of a codeword is furthermore determined by the positions of the unspecified bits. A successful compression needs to ensure the existence of at least a single unspecified bit within each set of *linearly dependent* bit positions. This goal can be accomplished through manipulating the positions of the unspecified bits, thus motivating the proposal of a *column reordering* technique.

In the following subsections, the aforementioned compression ratio enhancement techniques as well as the overall code compression flow is in detail discussed.

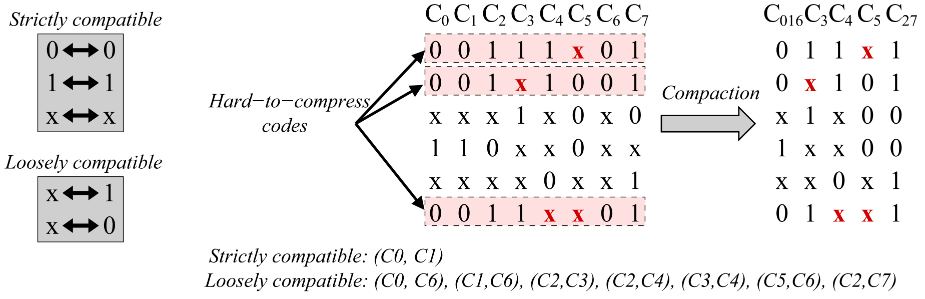


**Fig. 6** Reducing specified bits through column compaction

## 5.1 Column merging

Column merging opportunities, which can be exploited to reduce the number of specified bits in the *hard-to-compress* codewords, are derived from the redundancy in codewords. As customized microcoded IPs are typically developed to hold a set of applications, a particular program may not make full utilization of the provided control signals. For example, the IP may provide a divider that may not be used by a particular application. Similarly, the IP may provide 16-bit immediate values, among which only 10 bits are utilized by a particular application. As a result of this resource underutilization, the values in certain columns of the horizontal microcodes, such as the divider control signals and the most significant bits of the immediate field, will be largely identical. These columns therefore do not need to be driven by individual output bits of the decompressor. Instead, they can be concurrently driven in a broadcasting manner by a single output bit of the decompressor.

The proposed column merging technique exploits **two** types of compatibility among various bit positions (i.e., columns) in a microprogram. First of all, two columns are considered to be *strictly compatible* if their values in each codeword are identical. As an example. columns $C_0$ and $C_1$ in Fig. 6 are strictly compatible as they display identical values in each codeword. Clearly, this strict compatibility relationship is *transitive* and hence can be utilized in a greedy manner.

As strict compatibility requires complete identity, the number of strictly compatible columns is usually quite limited in programs with a large number of microinstructions. In this situation, further merge opportunities can only be identified through exploiting the flexibility of merging a specified bit with an unspecified bit. Two columns are therefore considered to be *loosely compatible* if they do not display specified yet distinct values in any codeword. According to this criterion, it can be easily confirmed that in Fig. 6 there exist 7 pairs of loosely compatible columns. However, unlike the *strictly compatible* relationship, this *loosely compatible* relationship is *intransitive*. A column (such as $C_2$ in Fig. 6) may be individually compatible with two columns (such as $C_3$ and $C_7$ in 6) which by themselves are not compatible. To maximally exploit this compatibility relationship while minimizing the consumption of unspecified bits, we first identify all the loosely compatible choices for a column, and then select the one that consumes the minimum number of unspecified bits.

Clearly, the outlined column merging approach is able to reduce the number of specified bits. However, the technique also consumes unspecified bits, more precisely, in the third *strictly compatible* case and the two *loosely compatible* cases shown in the leftmost gray boxes in Fig. 6.

As the goal of column merging is to balance the ratio of unspecified bits in each codeword, the merging process should be prioritized so as to maximally reduce the number of specified bits in the *hard-to-compress* codewords while retaining the unspecified bits. Specifically, a column should be precluded from being merged with other columns, if this merge process consumes an unspecified bit in a hard-to-compress codeword. In Fig. 6, columns $C_3$ and $C_4$ respectively contain 'X' bits in the second and the sixth codewords that are considered to be hard-to-compress. These columns, although they are loosely compatible, are precluded from being merged. In contrast, columns $C_2$ and $C_7$ can be merged. Although this merge process consumes 'X' bits in the third, the fourth, and the fifth codewords, this consumption does not impact the overall compression ratio that is limited by the hard-to-compress codewords. As can be expected, this compaction strategy thus results in a much more balanced unspecified bit distribution among the codewords (e.g., the compacted microcode shown in Fig. 6), which in turn delivers an enhanced overall compression ratio.

In the proposed work, a codeword is considered as *hard-to-compress* if its number of 'X' bits falls below a predetermined threshold. For an XOR-network with $N$ inputs and $M$ outputs (thus offering a compression ratio of $N/M$), this threshold can be set to the value $(M - N)$.

## 5.2 Column reordering

As discussed in Sect. 4.1, if a set of linearly dependent bits are all specified in a codeword, the codeword might be incompressible as this linear dependency reduces the rank of the coefficient matrix and possibly leads to a rank mismatch between the coefficient matrix and the augmented matrix. To maximally preclude such an unpalatable situation, at least one unspecified bit needs to be inserted in each set of *linearly dependent* bit positions so as to break the linear dependency. This insertion is able to match the ranks of the coefficient and the augmented matrices by reducing the rank of the latter, thus resulting in a solvable equation system.

In the proposed XOR-network construction technique, each partition group, composed of $B$ sets, covers every input of the XOR network exactly once. Accordingly, the output functions corresponding to any two partition groups will be linearly dependent. Maximal preclusion of the unsolvability condition necessitates the inclusion of at least one unspecified bit in each partition group. As each partition group maps to $B$ consecutive output bits of the XOR network, at least one unspecified bit thus needs to be included in these $B$ consecutive bit positions. To attain this goal, we exploit the flexibility in reordering the columns of microcodes. This flexibility is provided by custom IPs that only hold a highly limited number of applications. The design can be customized in such a way that the original control sequence is attained through rerouting the outputs of the decompressor.

The proposed column reordering process is presented in Algorithm 1. The fundamental goal is to simultaneously fulfill the requirement of including at least one unspecified bit in every $B$ consecutive columns for **all** the codewords. This is attained through ensuring that in each codeword, the maximum length of the specified bit sequence is less than $B$. Using the variable *max_length* to denote the largest length of the specified bit sequence in any codeword. the goal of the algorithm is the satisfaction of the requirement of $max\_length \leq B - 1$.

It needs to be noted that in Algorithm 1, column reordering is performed only when necessary so that the partial order of the columns can be maximally preserved. At the beginning of each iteration, the algorithm checks the value of *max_length*. If the value is less than $B - 1$, any of the remaining columns in *Col_list* can be placed as the next column, and thus no reordering is performed. On the other hand, if $max\_length = B - 1$, all the codewords that display a consecutive sequence of $B - 1$ specified bits are examined. Here, the current

**Algorithm 1** Column reordering algorithm

```
1: $Col\_list \Leftarrow \{$all columns$\}$;
2: $max\_length \Leftarrow 0$;
3: while $Col\_list \neq \phi$ do
4:    if $max\_length < B - 1$ then
5:       Select the first column $k$ in $Col\_list$;
6:    else
7:       Identify all the columns in $Col\_list$ that contains a 'X' in codeword $j$, if
         length$(j) = B - 1$;
8:       Among this set, select column $k$ with the minimum number of X's;
9:    end if
10:   $Col\_list \Leftarrow Col\_list - \{k\}$;
11:   for codeword $j = 0$ to $C_{max}$ do
12:      if code$(j, k) =$ 'X' then
13:         length$(j) \Leftarrow 0$;
14:      else
15:         length$(j) \Leftarrow$ length$(j) + 1$;
16:      end if
17:   end for
18:   $max\_length \Leftarrow max_j($length$(j))$;
19: end while
```

length of the specified bit sequence of the $j$th codeword is denoted as *length*$(j)$. A column $k$ is marked as *suitable* if it contains X's in all the codewords with *length*$(j) = B - 1$. Subsequently, among all the *suitable* columns, the one with the minimum number of X's is selected. This column selection process is concretely shown in lines 4–9. Finally, upon the selection of a new column $k$, the length of the specified bit sequence in each codeword is updated, as shown in lines 11–17.

## 5.3 Hybrid compression approach

It is clear that given a microcode program, the proposed XOR network-based compression technique can effectively compress the columns with a balanced number of unspecified bits. In comparison, the standard LUT-based compression technique, as it exploits inter-codeword compatibility, is more effective for columns with highly clustered values. In light of this observation, we therefore propose a *hybrid* compression approach that combines the advantages of both compression schemes to improve the overall compression ratio. Based on a functional decomposition of the microwords, a microprogram is vertically partitioned into two sets, one to be compressed using an XOR network and one using a LUT.

A functional level examination indicates that certain fields in a microcode exhibit an extremely biased unspecified-bit distribution. The large immediate field, as a representative example, is either fully specified or fully unspecified. This field creates large variations in the number of unspecified bits within a codeword, thus limiting the attainable compression ratio of an XOR network. On the other hand, the immediate values can be effectively captured in a small LUT, as a program typically does not utilize all the $2^k$ distinct immediate values provided by a $k$-bit immediate field.

It is also more desirable to capture a group of control signals in a LUT when they are always fully specified and exhibit a highly limited set of value combinations. Examples include *control-altering* signals, *interrupt* signals, as well as *write-enable* signals for register files and the memory. Not only are these three groups of signals always fully specified, but they are also *mutually exclusive* such that if any signal in one group is high, none of the signals in the other two groups can be high. This strict constraint in value combinations therefore enables them to be effectively captured by a small LUT.

Except for the aforementioned two cases, the remaining set of control signals in a customized IP typically displays an appreciable amount of randomness in terms of both the unspecified bit distributions and the possible value combinations of specified bits. Signals such as register names, due to the small width, would not create a sizable variation in the number of unspecified bits within a codeword. Meanwhile, as a program typically makes maximum utilization of the available registers, a large number of value combinations of register names will occur. Accordingly, it is more desirable to compress these signals using an XOR network instead of through a LUT.

According to the examination outlined above, the decomposition procedure maximally identifies the columns that are either fully specified or fully unspecified, as well as the columns that are always fully specified yet exhibit limited value combinations. By capturing these two sets of control signals in a LUT, the number of specified bits in the hard-to-compress codes can be sizably reduced, yet the number of unspecified bits is retained intact. The attainable compression ratio of the XOR network thus can be significantly improved. Meanwhile, as the LUT only captures a small set of control signals with highly repetitive patterns, both the bitwidth and the number of entries in the LUT are quite small. At runtime, decompression is performed by accessing the small LUT and the XOR network in parallel using distinct fields of the compressed codes. In this way, the hybrid compression is able to attain utmost code compression within minimum LUT size.

## 5.4 Overall code compression flow

The overall code compression flow with all the three compression ratio enhancement techniques integrated is presented in Fig. 7. The *column merging* procedure is first evoked to maximally reduce the width of the microcodes. Both the LUT width and the number of outputs of the XOR network can be reduced as a result. The remaining columns are subsequently decomposed into two sets according to the criteria outlined in Sect. 5.3. The goal of this process is the maximal reduction of the number of specified bits in the *hard-to-compress* codewords without significantly enlarging the LUT size. These two sets of columns are then compressed individually. For the groups of signals to be compressed using the XOR network, the *column reordering* procedure is first evoked to attain a more random distribution of unspecified bits.

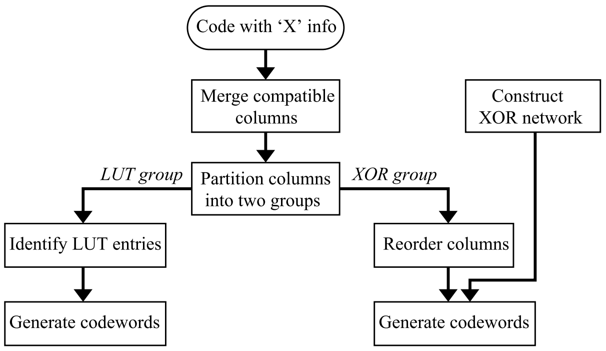


**Fig. 7** Hybrid compression flow, both an XOR network and a LUT are used

While the column *reordering* and the column *merging* procedures end up changing the original bit sequence of the control signals, the original sequence can still be restored at runtime. For embedded systems dedicated to a single application, this can be easily implemented through a customized routing of the interconnects without inducing any hardware overhead. On the other hand, for systems with more general applications, a fixed interconnect routing might not deliver the optimal compression for varying programs. In this case, a high-performance reconfigurable interconnect architecture [22, 23] can be incorporated in the decompression hardware to provide the routing flexibility required by different applications, at the cost of slightly increased area and performance overhead.

## 6 Experimental evaluation

To evaluate the proposed compression framework, **three** techniques have been implemented in our experimental studies: the standard LUT-based compression technique proposed in [17], the pure XOR network-based method, and the hybrid compression method. It has been reported in [17] that the LUT-based compression technique generally attains the best compression ratio when the columns are divided into three sets with distinct dictionaries. Accordingly, this technique has been evaluated for both 1-dictionary and 3-dictionary configurations in our experimental studies. On the other hand, the number of dictionaries used in the hybrid compression approach is set to 2. To attain a fair comparison, the *column merging* technique proposed in Sect. 5.1 is consistently applied to all three compression techniques to maximally reduce the width of the microcodes.

The microcoded programs with unspecified bits are generated using the NISC toolset [4]. The toolset provides a custom datapath, and compiles a program described in a high-level language to an executable binary that directly drives the control signals of components in the datapath. In our experiments, the width of the microcodes is 86 bits.

The most significant advantage of the proposed fixed-length compression technique is the extremely low hardware cost. The XOR compression network is implemented using approximately 200 XOR gates. As for on-chip storage, the pure XOR network-based approach requires no LUT, while the hybrid compression scheme only requires a small LUT. Accordingly, the three compression techniques are evaluated in terms of both the compression ratio and the LUT size.

Figure 8 presents the compression ratio attained by each technique. As can be seen, the pure XOR network-based technique delivers a compression ratio comparable to the

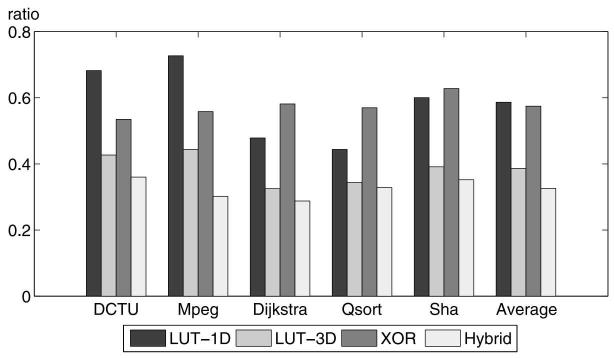


**Fig. 8** Compression ratio of various techniques

**Table 2** The LUT + XOR cost (Kbits) required by each technique

| Benchmark | Code size | LUT + XOR cost | | | |
|---|---|---|---|---|---|
| | | LUT-1D | LUT-3D | XOR | Hybrid |
| DCTU | 21.164 | 12.466 | 4.604 | 0.2 | 0.685 |
| Mpeg | 21.248 | 13.467 | 4.739 | 0.2 | 0.936 |
| Dijkstra | 34.770 | 13.403 | 4.028 | 0.2 | 1.308 |
| Qsort | 77.182 | 26.163 | 7.675 | 0.2 | 1.309 |
| Sha | 49.719 | 24.640 | 7.319 | 0.2 | 1.525 |
| Average | 40.816 | 18.028 | 5.673 | 0.2 | 1.153 |

LUT-based technique with the 1-dictionary configuration. The *hybrid* compression technique consistently delivers the lowest (the best) compression ratio among the four techniques for all the benchmarks. The average compression ratio attained by the hybrid technique is 0.326, a 16% improvement as compared to the average compression ratio of the 3-dictionary LUT-based technique. These results clearly confirm the efficacy of the proposed hybrid compression technique in combining the advantages of both the LUT-based and the XOR network-based techniques.

Table 2 present the size of the original microcoded program, as well as the total hardware cost (LUT and XOR) of each technique, under the assumption that one XOR gate and one SRAM storage bit have approximately the same cost. As can be seen, the LUT-based compression technique needs to capture 44% and 14% of the original code size in the on-chip LUT, respectively for the 1-dictionary and 3-dictionary configurations. This significant storage thus drastically degrades decompression speed. In contrast, the pure XOR network-based technique requires no on-chip storage at all, while the hybrid compression technique only needs to use 2.3% of the original code size in the on-chip LUT. This extremely low storage requirement thus reduces both the hardware cost and the leakage power consumption, while enabling the development of an extremely high speed decompressor as well.

The results in Table 2 confirm that the LUT size required by a LUT-based compression technique is generally proportional to the original size of the program. Accordingly, for a microcoded IP that holds a set of applications, the required LUT size is usually determined by the application of the largest code size. In contrast, the LUT size required by the proposed hybrid compression technique is far less sensitive to the original code size. As a result, even if the applications held by the microcoded IP display highly unbalanced code sizes, the on-chip LUT size can still be effectively controlled.

The decompression speed of the pure XOR network-based approach is extremely fast. As each output of the XOR network is produced in parallel, the hardware decompression only displays two levels of gate delay. Accordingly, the decompression speed of the hybrid compression technique is determined by the access latency of the on-chip LUT. We have employed Cacti [24] to evaluate the LUT access latency of the LUT-based compression and the hybrid compression techniques. The configuration of the largest LUT and the corresponding access latency values are shown in Table 3. As can be seen, the LUT required in the hybrid compression technique exhibits both a smaller width and fewer entries as compared to the dictionaries used in the LUT-based compression technique. This in turn results in a 63% reduction in the overall access latency, achieved as a result of the reduced delay in both the address decoder and the output drivers.

Given these experimental results in compression ratio, LUT size, and access latency, it can be clearly concluded that the proposed hybrid compression method delivers utmost com-

**Table 3** The configuration and access latency of LUTs for each technique

| Benchmark | LUT configuration | | | Access latency | | |
|---|---|---|---|---|---|---|
| | LUT-1D | LUT-3D | Hybrid | LUT-1D | LUT-3D | Hybrid |
| DCTU | 185 × 69 | 119 × 23 | 34 × 9 | 1.537 | 1.318 | 0.950 |
| Mpeg | 197 × 70 | 103 × 23 | 30 × 13 | 1.631 | 1.291 | 0.957 |
| Dijkstra | 183 × 75 | 80 × 25 | 47 × 16 | 1.545 | 1.242 | 1.021 |
| Qsort | 367 × 73 | 167 × 24 | 53 × 13 | 1.650 | 1.326 | 1.024 |
| Sha | 332 × 76 | 133 × 25 | 67 × 16 | 1.647 | 1.377 | 0.976 |

pression ratio as well as high speed decompression, achieved within a highly constrained amount of extra hardware.

## 7 Conclusions

We have proposed in this paper an extremely fast and cost-effective code compression technique for microcoded IPs. Through utilizing a linear network, the proposed technique can precisely fill in the fully specified bits in each microcode, despite the high irregularity of the values and positions of these bits. The linear property inherent in the compression strategy in turn enables the development of an extremely low-overhead decompression engine, composed of only a fixed-bandwidth XOR network. A set of functional level optimization approaches, including a *column reordering* and a *column merging* technique, have been proposed to further improve the compression ratio. Through combining the flexible XOR network with a minimum two-level storage for highly specified fields, such as immediate values, a *hybrid* compression technique is able to deliver utmost code compression within a negligible amount of storage overhead. Experimental results show that the proposed hybrid compression technique is able to attain an average compression ratio of 0.326, while only 2.3% of the original microcodes need to be stored in an on-chip LUT. Such high efficiency thus enables the incorporation of this compression technique into most microcoded IPs to attain utmost code size reduction within a negligible amount of performance and hardware overhead.

## References

1. Schreiber R, Aditya S, Mahlke S, Kathail V, Rau BR, Cronquist D, Sivaraman M (2002) PICO-NPA: High-level synthesis of nonprogrammable hardware accelerators. VLSI Signal Process 31(2):127–142
2. Clark N, Zhong H, Fan K, Mahlke S, Flautner K, Nieuwenhove KV (2004) OptimoDE: Programmable accelerator engines through retargetable customization. In: Hot Chips
3. Weber S, Keutzer K (2005) Using minimal minterms to represent programmability. In: CODES+ISSS, Sept 2005, pp 63–68
4. Reshadi M, Gorjiara B, Gajski D (2005) Utilizing horizontal and vertical parallelism with a no-instruction-set compiler for custom datapaths. In: ICCD, Oct 2005, pp 69–76
5. Thuresson M, Sjalander M, Bjork M, Svensson L, Larsson-Edefors P, Stenstrom P (2007) FlexCore: Utilizing exposed datapath control for efficient computing. In: IC-SAMOS, July 2007, pp 18–25

6. Wolfe A, Chanin A (1992) Executing compressed programs on an embedded RISC architecture. In: International Symposium on Microarchitecture, Dec 1992, pp 81–91
7. Kemp TM, Montoye RK, Harper JD, Palmer JD, Auerbach DJ (1998) A decompression core for PowerPC. IBM J Res Dev 42(6):807–812
8. Yang C, Chen M, Orailoglu A (2008) Squashing microcode stores to size in embedded systems while delivering rapid microcode accesses. In: CODES-ISSS, Oct 2009, pp 249–256
9. Cooper KD, McIntosh N (1999) Enhanced code compression for embedded RISC processors. In: Conference on programming language design and implementation, May 1999, pp. 139–149
10. Debray SK, Evans W, Muth R, Sutter BD (2000) Compiler techniques for code compaction. ACM Trans on Program Lang Syst 22(2)
11. Segars S, Clarke K, Goudge L (1995) Embedded control problems, thumb, and the ARM7TDMI. IEEE Micro 15(5):22–30
12. Grehan R (1999) 16-bit: The good, the bad, your options. Embed Syst Program 12(8)
13. Pechanek GG, Larin S, Conte T (2002) Any-size instruction abbreviation technique for embedded DSPs. In: ASIC/SOC Conference, Sept 2002, pp 8–12
14. Corliss ML, Lewis EC, Roth A (2003) DISE: a programmable macro engine for customizing applications. In: ISCA, June 2003, pp 362–373
15. Lau J, Schoenmackers S, Sherwood T, Calder B (2003) Reducing code size with echo instructions. In: CASES, Oct 2003, pp 84–94
16. Agerwala T (1976) Microprogram optimization: a survey. IEEE Trans Comput 25(10):962–973
17. Gorjiara B, Gajski D (2007) FPGA-friendly code compression for horizontal microcoded custom IPs. In: FPGA'07, pp 108–115
18. Borin E, Breternitz M, Wu Y, Araujo G (2007) Clustering-based microcode compression. In: ICCD'07, Oct 2007, pp 189–196
19. Thuresson M, Sjalander M, Stenstrom P (2009) A flexible code compression scheme using partitioned look-up tables. In: HiPEAC, Jan 2009, pp 95–109
20. Stewart G (1973) Introduction to matrix computations. Acadamic Press, New York
21. Bayraktaroglu I, Orailoglu A (2005) The construction of optimal deterministic partitionings in scan-based BIST fault diagnosis: Mathematical foundations and cost-effective implementations. IEEE Trans Comput 54(1):61–75
22. Kim D, Lee K, Lee S-J, Yoo H-J (2005) A reconfigurable crossbar switch with adaptive bandwidth control for networks-on-chip. In: ISCAS, Jan 2005, pp 2369–2372
23. Wan M, Zhang H, George V, Benes M, Abnous A, Prabhu V, Rabaey J (2001) Design methodology of a low-energy reconfigurable single-chip DSP system. J VLSI Signal Process Syst 28:47–61
24. Thoziyoor S, Muralimanohar N, Ahn JH, Jouppi NP (2008) CACTI 5.1, Tech report, HP Labs, April 2008