Configuration Bitstream Compression for Dynamically Reconfigurable FPGAs

Ju Hwa Pan Tulika Mitra Weng-Fai Wong
School of Computing
National University of Singapore
Republic of Singapore 117543
{panjuhwa, tulika, wongwf}@comp.nus.edu.sg

Abstract

Field Programmable Gate Arrays (FPGAs) holds the possibility of dynamic reconfiguration. The key advantages of dynamic reconfiguration are the ability to rapidly adapt to dynamic changes and better utilization of the programmable hardware resources for multiple applications. However, with the advent of multi-million gate equivalent FPGAs, configuration time is increasingly becoming a concern. High reconfiguration cost can potentially wipe out any gains from dynamic reconfiguration. One solution to alleviate this problem is to exploit the high levels of redundancy in the configuration bitstream by compression. In this paper, we propose a novel configuration compression technique that exploits redundancies both within a configuration's bitstream as well as between bitstreams of multiple configurations. By maximizing reuse, our results show that the proposed technique performs 26.5–75.8% better than the previously proposed techniques. To the best of our knowledge, ours is the first work that performs inter-configuration compression.

INTRODUCTION

Field Programmable Gate Arrays (FPGAs) offer a generic platform for hardware realization of application specific algorithms. There is an increasing interest in using them as alternative computing platforms. FPGAs are particularly suited for accelerating compute intensive algorithms that can take advantage of massive hardware parallelism.

Today's FPGAs are dynamically reconfigurable. At any time after an FPGA has been powered up, it is possible to suspend its operations, load in a completely new configuration, and restart its operation using the newly loaded configuration. This has the effect of virtualizing the hardware by allowing one to context switch between hardware implementations realized on the same set of reconfigurable resources. By virtualizing the hardware, a better utilization can be achieved, especially if the computation involves some form of multitasking. An application that exhibits major phase transitions in its execution may also be able to use dynamic reconfiguration to accelerate its operations over its entire execution. Dynamic reconfiguration also opens up the possibility of having a system that can actively adapt its hardware to errors, failures, or changing operating environments.

Computing with FPGAs by means of dynamic reconfiguration is not without its challenges. Multi-million gate-equivalent FPGAs are now available off the shelf. However, such FPGAs take a considerable amount of time to configure. This can potentially wipe out any advantages of dynamic reconfiguration. Compression of the configuration bitstream has been suggested as a means to alleviate configuration latency. A major component of configuration time is the time it takes to transfer a configuration bitstream to the FPGA. Reducing the size of the configuration bitstream will reduce the configuration transfer time.

All techniques of data compression seek to minimize the amount of data redundancies present. For a given application, a subset of operations may be found consistently across successive configurations. This translates into configuration data that may be repeated across multiple bitstreams. The reconfiguration process of an FPGA provides opportunities to exploit these redundancies through data reuse between successive configuration bitstreams. Existing work in configuration compression has demonstrated effective exploitation of the high degree of redundancies present within a bitstream. However, the exploitation of redundancies between successive configurations remains largely unexplored.

In this paper, we first propose a new intra-bitstream compression technique. Our results show that this technique competes favorably with the best known previous technique[5]. We then extend the technique to take advantage of inter-bitstream redundancies as well as *partial reconfigurability* supported by modern FPGAs such as Xilinx Virtex family. To the best of our knowledge, ours is the first work to do so.

CONFIGURATION ARCHITECTURE

Virtex is a family of partially reconfigurable SRAM-based FPGAs from Xilinx Inc. Each Virtex device comprises of an array of configurable logic blocks (CLBs). Also present on each device are blocks of random access memory known as SelectRAMs, input-output blocks (IOBs) clock resources, programmable routing, and configuration circuitry. The device is configured by loading a bitstream into configuration memory comprising of the SRAM cells, whose outputs control the FPGA logic and interconnect resources.

The Virtex configuration memory is organized into a series of one-bit wide *frames*, each spanning from the top of the array to the bottom [9]. A frame is the smallest unit of memory that can be written to or read from the device. Frames are aggregated to form *columns*, which are of several types. A center column configures the four global clock pins, while two IOB columns configure the IOBs present on the left and

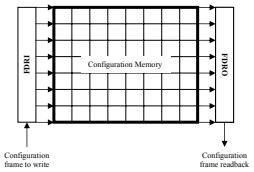

Figure 1. Virtex Configuration and Readback

right sides of the device. CLBs and SelectRAM are represented by multiple CLB and RAM columns respectively. Each CLB column is configured by 48 frames. The first and last 18 bits of each of these frames configure the IOBs located at the top and bottom of the device, respectively. Each 18-bit group found between the first and last 18 bits in turn configures part of a CLB row found in that column.

The device is configured by loading in a configuration bitstream, consisting of a series of commands and frame data. To configure a frame, the frame address must be provided, followed by the frame data. The configuration mechanism consists of a serial-to-parallel register, the *Frame Data Input Register* (FDRI) (see Figure 1). Every frame is first shifted into the FDRI before it is written to the configuration memory. Data is shifted into the bottom of the FDRI. Once the FDRI is filled up with a new frame, it is written into memory. In addition, a frame can be read back by loading it from configuration memory into a parallel-to-serial register called the *Frame Data Output Register* (FDRO).

RELATED WORK

A number of compression techniques have been proposed for FPGA architectures. The wildcard compression scheme [6] targets the Xilinx XC6200 series FPGA architecture which supported this feature. By utilizing the wildcard address registers present in the architecture, logic cells sharing the same configuration can be addressed and configured with a single write operation, speeding up configuration. Despite demonstrating a compression ratio as low as 25%, this architecture-specific algorithm exhibits poor extensibility. Furthermore, in the absence of repetitive structures that give rise to logic cells sharing common configurations, the regularities upon which the algorithm leverages upon may not be present. Run-length encoding techniques have also been proposed for this type of architecture [7]. On the whole, however, this architecture is clearly not appropriate for large FPGAs, bringing into question this entire class of methods.

Configuration cloning [11] exploits regularity and locality in bitstreams by copying configuration data from one region of the FPGA to several other regions. Without loading the entire bitstream, the whole FPGA can be configured, reducing configuration latency. However, for this method to work, a large amount of complex circuitry is needed.

It was found that bitstream regularities can be effectively encoded by dictionary-based methods. Dictionary-based algorithms depend less on specific features of the FPGA configuration architecture, providing for greater flexibility. Configuration compression [5] based on the Lempel-Ziv-Storer-Szymanski (LZSS) [1] compression scheme has been shown to be effective for the Xilinx Virtex family of FPGAs. These algorithms require that an extended version of the FDRI be used as a sliding window, such that a frame present in the upper half of the FDRI will act as the dictionary for the next frame to be configured. By deriving a suitable sequence of frame configuration, inter- and intra-frame regularities can be leveraged. This is the best configuration compression method known to date. Besides LZSS, other dictionary compression schemes such as the Lempel-Ziv-Welch (LZW) method [4], and LZ77 [10] were also reported in the literature. However, they tend to fair less well.

DIFFERENCE VECTOR (DV) COMPRESSION

An analysis of configuration bitstreams reveals a high degree of regularity among the frames configuring the CLB array. Recall that each CLB column is configured by a series of frames. Frames configuring common structures among different CLBs may share high regularity. Between such frames, one frame may be converted into another simply by flipping a few bits. LZSS compression produces good results in highly regular bitstreams [5]. However, when the lengths of the matches are small, LZSS compression proves to be less effective as the savings is offset by the encoding overhead. An analysis of our benchmark bitstreams reveals that bitstream regularities may be too fine-grained to be effectively captured by the LZSS method. We shall now describe an algorithm that taps into these regularities thereby achieving better compression ratio than the LZSS method.

Compression

By encoding the bit differences between frames, we can reduce redundancies in loading identical information repeatedly for similar frames. We then assign a suitable **reference frame** to the frame to be compressed which we call the **beneficiary frame**. We then construct a *difference vector*:

$$DV(F1, F2) = F1 \oplus F2$$

where F1, F2 are the reference and beneficiary frames respectively and $\oplus$ is the bitwise exclusive OR operator.

Given that the bit-flips obtained between two frames tend to be either few in number and scattered or clustered in bands, we observe long sequences of 0s in the difference vector, with 1s occurring in shorter sequences. Therefore, runlength encoding (RLE) can be used to effectively compress running sequences in the difference vectors. In our scheme, we first determine the runlengths of 0s and 1s before employing Huffman [8] encoding for the runlengths. Given that the runlengths of sequences of 1s exhibit vastly different entropies from that of sequences of 0s (with 1s occurring in short sequences and 0s in very long sequences), we use separate set of encodings for sequences of 1s and 0s.

Decompression

Decompression occurs at runtime. Decoding the runlengths reveals the bit locations where flips are to be performed to transform the reference frame into the beneficiary one. The FDRI register in Xilinx Virtex devices can be effectively used for decompression. Initially, it contains the reference frame. A decoder circuit is added before the FDRI register (see Figure 1). As runlength decoding proceeds, the relevant bits in FDRI are flipped. When all the code words have been decoded, the contents of the FDRI represents the beneficiary frame and it is written to the configuration array.

INTRA-BITSTREAM COMPRESSION

The similarity between the reference frame and the beneficiary frame is the key to achieving good compression ratio. Both LZSS and DV compression require a suitable configuration sequence of frames such that similar frames are next to each other thereby improving the compression ratio. We use the algorithms presented in [5] to generate the configuration sequence for LZSS compression and modify it suitably to adapt it to DV compression.

Inter-frame Regularity Graph (IRG)

An *inter-frame regularity graph* (IRG) [5] is a directed graph where each frame is represented by a node. A directed edge $u \rightarrow v$ between two nodes u and v represents the size of the compressed encoding of v by treating u as the reference frame. To discover inter-frame regularities, each frame is used as a reference frame and every other frame is treated as a beneficiary frame, i.e., IRG is a complete graph. In case of LZSS compression, it is easy to find the size of the compressed encoding for the beneficiary frame. However, for DV compression, the Huffman codes for runlengths are designed for the entire bitstream instead of individual frames. That is, it depends on the configuration sequence. Therefore, we use the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions in the difference vector between frame u and frame v as the edge weight of $u \rightarrow v$. We expect that minimizing the number of transitions will minimize the length of the encoding. As an added advantage, minimizing the number of runlengths reduces decompression time by reducing the number of Huffman table lookups.

Given the IRG, we present two schemes for generating configuration sequence: one that does not require any readback of frames, and one that does.

Configuration WithOut Readback (CWOR)

The optimal configuration sequence that does not have readback corresponds to the *shortest* Hamiltonian path (i.e., a path that visits each vertex exactly once) in the IRG. A greedy algorithm is presented in [5] that starts with the minimum cost edge and expands its both ends to cover all the vertices. This algorithm generates close to optimal results. For LZSS compression, this scheme requires extension of the FDRI register. However, DV compression requires minimum modification

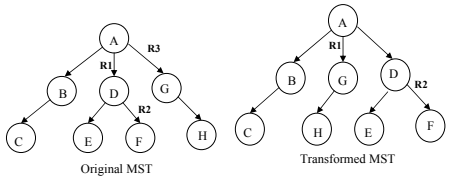

Figure 2. Original and Transformed MST

to the existing configuration architecture. Only a decompression unit is added before the FDRI register. Let $\langle f_1, \ldots, f_n \rangle$ be the configuration sequence of frames. f_{i-1} acts the reference frame for f_i. Before f_i is decoded, its reference frame f_{i-1} is present in the FDRI register as it was the last configured frame. Therefore, FDRI register can be used as reference frame and its bits are flipped according to f_i's encoding. The modified FDRI register represents the beneficiary frame f_i and is written to the configuration array. FDRI will now be used as reference frame to decompress the frame f_{i+1}.

Configuration with Readback (CWR)

The configuration sequence without readback is too restrictive in taking full advantage of the similarity among frames. Therefore, Li and Hauck [5] proposed another scheme where it is allowed to readback already configured frames from the FPGA and use them as reference frames. This corresponds to finding the directed minimum spanning tree (MST) [3] in the IRG. An edge $u \rightarrow v$ in the tree indicates that u should be used as the reference frame for v. The configuration sequence is generated by a pre-order traversal of the MST. Li and Hauck [5] propose to readback the frame in FDRI register itself. Instead, we use the existing readback infrastructure to read frames from configuration array into FDRO register (see Figure 1). The contents of the FDRO register is transferred to the FDRI register and then used as the reference frame as before.

Once we readback a frame in the FDRO register, it remains there till another readback command is executed. Therefore, it is possible to reuse the FDRO register. Let v be a node in the MST with more than one children $c_1, \ldots, c_n$ and suppose that none of the subtrees rooted at each child node is a chain (simple path). Then the configuration sequence generated by pre-order traversal requires $n-1$ readbacks of v to configure $c_2, \ldots, c_n$. This is because frame v in FDRO is overwritten due to readbacks during the subtree traversal of a child. However, if the subtree at child c_i is a chain, then the FDRO register will still contain frame v after the frames in subtree of c_i have been configured. c_{i+1} can simply use the FDRO register thereby avoiding the readback of v. We exploit this fact by transforming the MST such that at any node, the left children are those with "chain" subtrees as shown in Figure 2. The original MST requires three readbacks (R1, R2, R3) while the the transformed one (obtained by moving the subtree rooted at G to the left) requires only two readbacks.

INTER-BITSTREAM COMPRESSION

Utilizing intra-bitstream regularities, the algorithms presented so far are effective for single bitstream compression. However, they do not take into account inter-bitstream regularities inherent across multiple bitstreams in reconfigurable systems. Regularities between configuration bitstreams may arise due to the following reasons, and subsequently can be exploited.

Static Kernels. The implementation of an application based on a reconfigurable computing model may require several runtime reconfigurations of its FPGA component. Each configuration may consist of multiple independent computational kernels, each implementing a different part of the application. Configurations may share static kernels (modular circuits maintained in the same position across multiple configurations). This occurs when the application requires some kernels persistently, whereas it dynamically swaps in and swaps out non-persistent kernels from the remaining fabric. Without partial reconfiguration, the whole FPGA has to be reconfigured every time any part of the fabric changes. Instead of loading in an entire bitstream, the partial reconfigurability of the Virtex FPGA allows us to load in a partial bitstream only for the part of the device that requires changes.

Chip Utilization. Due to the highly flexible nature of the FPGA interconnect, only a small proportion of the configuration bits in the bitstream for a given circuit may be important. Hence, only these bits in the configuration present on-chip may differ from those of the incoming bitstream.

In this section, we will first explore the feasibility of using partial reconfiguration to speed up dynamic reconfiguration, before proposing algorithms to apply Difference Vector and LZSS compression in an inter-bitstream paradigm.

Partial Reconfiguration

With the partial reconfiguration capability present in Virtex FPGAs, configuration time can be greatly reduced by transferring only those portions of the next bitstream that differ from the current bitstream already in the FPGA, with no additional hardware overhead. We have implemented a tool targeting the Xilinx Virtex XCV1000 FPGA on the Celoxica RC1000 board, that performs partial reconfiguration by detecting the differences between two bitstreams and loading only these differences onto the FPGA during reconfiguration. Our experiments showed that reconfiguration time increases nearly proportionally with the amount of data transferred, with negligible latency in addressing the changed portions of the bitstream.

One major drawback of partial reconfiguration is its reliance on the appropriate placement of kernels. As mentioned previously, the smallest configuration unit is one frame, which spans the configuration array vertically. The kernel layout for Benchmark5 in Figure 7 allows for partial reconfiguration of the static *idct* kernel. Whereas the static *smalldes* kernel for Benchmark2 in Figure 7 will not offer any opportunity for partial reconfiguration as all the frames are modified.

The physical constraints and IO requirements for a kernel may render a layout facilitating partial reconfiguration difficult. On the other hand, by detecting inter-bitstream regularities that lie within a section of a frame, the regularity brought about by static circuitry spanning all the frames can be captured by DV and LZSS compression.

Compression

DV and LZSS compression have been shown to be effective on stand alone bitstreams. Given a reconfigurable computing model, these techniques may be extended to exploit regularities spanning across bitstreams.

The configuration memory on the FPGA acts as a natural cache for storing frames in the previous configuration, which we shall call the **old frames**. These frames may be very similar to the **new frames** - the incoming configuration to be compressed. Just as intra-bitstream compression exploits similarities among new frames, inter-bitstream compression can readback old frames and use them to compress the new frames. The complication here is that the reference frames from the old bitstream cannot be overwritten prior to their being readback for reuse. We have extended both intra-bitstream CWOR and CWR techniques to approximate optimal configuration sequences that maximize the use of reference frames within the same bitstream and across successive bitstreams.

Inter-frame Regularity Graph (IRG)

We define a variant of the IRG graph previously utilized in LZSS and DV schemes for stand alone bitstreams. The IRG in this case is a multi-digraph, i.e., a directed graph with multiple edges between vertices as well as self-loops. Each node u has a pair of directed edges $Intra(u \rightarrow v)$ and $Inter(u \rightarrow v)$ to every other node v. The weight of the edge $Intra(u \rightarrow v)$ is the cost of using u_{new} as a reference frame for v_{new}. Similarly, the weight of $Inter(u \rightarrow v)$ is the cost of using u_{old} as the reference frame for v_{new}. In addition, each node u has a self-loop $Inter(u \rightarrow u)$ annotated with the cost of using u_{old} as the reference frame for u_{new}. We now define some special nodes in the IRG.

Retained Nodes A node u is a retained node if the content of u_{new} is the same as the content of u_{old}. These frames do not need to be reconfigured.

Self-Referenced Nodes A node u is a self-referenced node if it is *not* a retained node, but u_{old} is used as the reference frame for u_{new}.

Configuration WithOut Readback (CWOR)

The CWOR scheme for intra-bitstream compression does not require any readback. However, for inter-bitstream compression, we allow a *restricted form of readback*. We allow a frame u_{old} to be readback only for the configuration of u_{new}. This relaxation allows us to exploit scenarios depicted in the Benchmark2 in Figure 7 involving the static kernel

Input: Inter-Bitstream IRG G
Output: Configuration Sequence CS
$CS := \phi$;
Delete Retained Nodes from G;
Delete edges $Inter(u \rightarrow v)$ if $u \neq v$;
Mark all nodes in G as unvisited;
for *all nodes* $v \in G$ **do**
  if $Inter(v \rightarrow v) < \min_{u \in G, u \neq v} (Intra(u \rightarrow v))$ **then**
    /* v is a Self-Referenced Node */
    $CS := CS \cup \{\langle v \rangle\}$;
    Mark v as visited;
  end
end
if $CS = \phi$ **then**
  /* There is no self-referenced node */
  Let $(u \rightarrow v)$ be minimum cost edge in G;
  $CS := \{\langle u, v \rangle\}$;
  Mark u and v as visited;
end
while $\exists$ *unvisited nodes in* G **do**
  Let $u \rightarrow v$ be the shortest edge in G s.t. v is unvisited and u is the tail node of a sequence $S \in CS$;
  if $Inter(v \rightarrow v) < Intra(u \rightarrow v)$ **then**
    /* v is a Self-Referenced Node */
    $CS := CS \cup \{\langle v \rangle\}$;
  else
    Append node v to S;
  end
  Mark v as visited;
end

Algorithm 1: **Inter-Bitstream CWOR**

Input: Rooted Tree T
Output: Configuration Sequence
mark all nodes as unvisited;
for *all nodes* $v \in T$ *in reverse level traversal order* **do**
  $current := v$;
  if *current is visited* **then**
    continue;
  end
  while *(*$current$ *has incident "Inter" edge or is self-referenced)* **do**
    /* configure current */
    mark $current$ as visited;
    if $current$ *is self-referenced* **then**
      $parent := \phi$;
      `compress` $current$ `w.r.t.` $current_{old}$;
    else
      $parent$:= parent of $current$;
      `compress` $current$ `w.r.t.` $parent_{old}$;
    end
    /* Traverse current's subtree */;
    `PREORDER(`*sub-tree rooted at* $current$`)`;
    if *(*$parent \neq \phi$ *and* $parent$ *does not have unvisited child with "Inter" edge)* **then**
      $current := parent$;
      continue;
    else
      break;
    end
  end
end
`PREORDER` (T);

Algorithm 2: **Inter-Bitstream CWR**

smalldes. Algorithm 1 is a greedy algorithm for constructing the CWOR configuration sequence.
The algorithm finds a configuration sequence (CS) for the non-retained frames. Note that the retained frames need not be configured. The only "Inter" edges considered are of the form $Inter(v \rightarrow v)$. At any point of time, CS is of a set of sequences (paths) and the algorithm tries to extend them along the tails with shortest edges. The final configuration sequence consists of a set of directed paths. If a sequence contains a self-referenced node, it will be at the head of the sequence and its configuration will require a readback.

Configuration With Readback (CWR)

The aim of the inter-bitstream CWR algorithm is to find a configuration sequence that pairs up each new frame with the best possible reference frame. The reference frames can be both old and new frames. However, we need to make sure that a old reference frame is not overwritten before it is used by the corresponding beneficiary frame. The algorithm consists of two phases. The first phase pairs up each non-retained new frame with a reference frame. This phase generates a set of directed trees. The second phase computes an efficient traversal order of these trees to minimize the number of readbacks.

Generation of Trees. First, we mark the retained nodes and remove all their incoming edges from the IRG. All the outgoing "Intra" edges of the retained nodes are also removed. Similarly, we find all the self-referenced nodes v in IRG G s.t. $Inter(v \rightarrow v) < \min_{u \in G, u \neq v} (Intra(u \rightarrow v))$. Again, we remove all the incoming edges to the self-referenced nodes. For the remaining set of nodes V in G, we derive a set of trees such that each node in V appears exactly once in the trees with in-degree equal to 1. The goal is to minimize the total cost of all the edges in the tree. This is achieved through a modification of the directed minimum spanning tree (MST) algorithm [3]. The basic idea is to start with the set of best pairing of nodes and then eliminate cycles, if any, one by one.

1. For each node other than the retained and self-referenced ones, select the incoming edge with the minimum cost. Let S be the set of selected edges. If no retained or self-referenced

Benchmark	Source	Device	Utilization
rsa	Opencores	XCV100	56%
idct	Xilinx	XCV150	93%
dctidct	Xilinx	XCV300	86%
des	Xlinx	XCV400	76%
tripdes	Xilinx	XCV800	82%
jpeg	OpenCores	XCV1000	96%

Table 1. Benchmarks for Intra-Bitstream Compression

frames exists, then select an arbitrary node and remove its incoming edge from S.
2. If the set S does not contain any cycle, then S is the set of trees. Otherwise, continue.
3. For each cycle formed, the nodes in this cycle are collapsed into a pseudo-node (k). For each node j in the cycle, modify the cost all its incident edges except for the one that belongs to the cycle as follows.

$$c(i \rightarrow k) = c(i \rightarrow j) - \left(c(x(j) \rightarrow j) - \min_{l \in G}(c(x(l) \rightarrow l) \right)$$

where $c(i \rightarrow j)$ is the cost of the incident edge, $c(x(j) \rightarrow j)$ is the cost of the edge in the cycle incident to j. Note that we also consider self loops ($i = j$) as incident edges.
4. Among the modified edges, let $i \rightarrow k$ be the edge with minimum modified cost and let $i \rightarrow k$ enter the cycle at node j. If $i = j$, then mark j as a self-referenced node and remove $x(j) \rightarrow j$ from the set S. Otherwise, select $i \rightarrow k$ as the incoming edge of the pseudo-node k and replace $x(j) \rightarrow j$ with $i \rightarrow j$ in the set S.
5. Delete all the incident self-loop edges, i.e., delete $i \rightarrow k$ if i is a node in the original cycle.
6. Go to step 2 with the contracted graph.
Figure 3 shows an illustration of tree generation algorithm. As there is no self-referenced node, node 1 is chosen as the root. The original set of edges generates a cycle which is eliminated in the figure on the right hand side.
The result of the first phase is a set of disjoint trees. Each tree can have at most one retained or self-referenced node. This is because these special nodes have in-degree = 0 and all the other nodes have in-degree = 1. Similarly, by construction, there can be at most one node with in-degree = 0 that is not a retained or self-referenced node (see Step 1). The number of trees in S corresponds to the number of nodes with in-degree = 0 in S. Each such node is made the root of the corresponding tree.

Traversal of Trees. The second phase of the algorithm traverses each rooted tree so as to minimize the number of readbacks and to ensure that a reference frame is not overwritten before it is used. Each node v is assigned a $depth$ value equal to the length of the path from root to node v. Algorithm 2 details the traversal. The basic idea of the algorithm is to visit the nodes with incoming "Inter" edges first so as to make sure that their reference frames are not overwritten. The algorithms traverse the nodes in reverse level-order, i.e., its starts with the nodes of highest depth and works its way towards the root. However, it also attempts to exploit a frame that has just been configured (present in FDRI) or has just been readback (present in FDRO). The first case is taken care of through a PREORDER traversal of the subtree rooted at a configured node (*current* in Algorithm 2 which is present in FDRI). The PREORDER traversal is similar to the pre-order traversal described in intra-bitstream CWR scheme (see Section). However, it stops exploring a node further if it has already been visited. The second case (a frame present in FDRO) is exploited by the fact that for a *parent* node (possibly present in FDRO), we visit all the children with "Inter" edges before configuring *parent* (last condition in Algorithm 2). After all the "Inter" edges have been visited, the algorithm performs another pre-order traversal starting from root node to configure the unvisited nodes. Figure 4 shows an example of a traversal of two trees. The figure on the left represents the configuration sequence of nodes before the final PREORDER traversal of the whole tree for unvisited nodes is performed. The lightly shaded nodes represent the ones that are covered by PREORDER traversal of subtrees rooted at configured nodes. The figure on the right shows the final configuration sequence.
Algorithm 2 generates the order in which the frames should be configured and the reference frames to be used for their compression. Given this order, it is easy to see that a readback will be necessary if the reference frame is not present in FDRI or FDRO register. Given the traversal order, a configuration bitstream is generated that includes readback commands as well commands to use FDRI or FDRO register as reference frames.

EXPERIMENTAL RESULTS

In this section, we present the experimental evaluation of our compression. We perform compression using a selection of circuits typically implemented on FPGAs in the encryption and image processing application domain (see Table 1).

Intra-Bitstream Compression

Figure 5 shows the intra-bitstream compression ratios for the DV and LZSS compression. The compression ratio is defined as the compressed configuration size as a percentage of the uncompressed configuration size. We included the size of the Huffman tables in compressed configuration size for DV. Figure 5 shows that DV compression performs better than LZSS irrespective of readbacks. Note that configuration with readback (CWR) performs better than configuration without readback (CWOR) for DV compression. Reading back similar data allows full exploitation of inter-frame regularities. However, LZSS fails to exploit fine-grained bitstream regularities as effectively as DV compression. Therefore, the slight reduction in frame encoding size is insufficient to offset the frame addressing overheads involved in readbacks. This led to the CWOR scheme performing better than the CWR scheme under the LZSS compression.

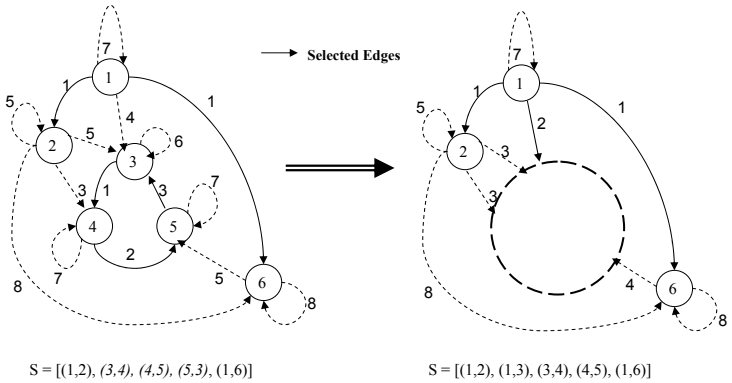

Figure 3. Tree Generation for Inter-Bitstream CWR

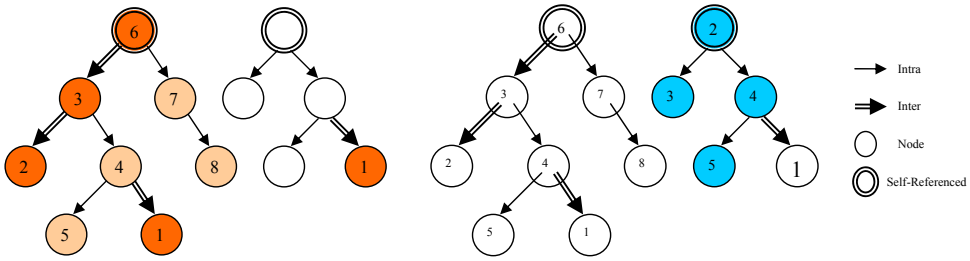

Figure 4. Tree Traversal for Inter-Bitstream CWR

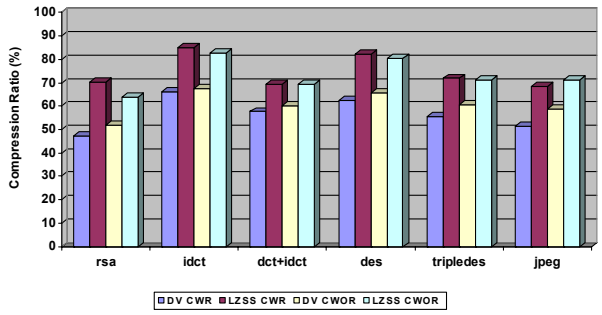

Figure 5. Intra-bitstream Compression with DV and LZSS

Despite yielding better results, some issues may arise during actual hardware implementation of DV compression. One concern is that on-chip memory may not be sufficient for Huffman look-up tables, although our experiments have shown the largest of these tables to be less than 3KB.

Second, the Huffman trees to retrieve the Huffman-encoded runlengths may be a potential bottleneck in the decompression process. Speedups can be achieved by N-Level implementation of the look-up tables, which trades off memory requirements with speed of decompression, depending on the number of levels used [12].

Inter-Bitstream Compression

To investigate the inter-bitstream DV and LZSS compression, we use benchmarks consisting of a pair of bitstreams (O, N). For each new bitstream N, we obtain the intra-bitstream compression as well as the inter-bitstream compression using O as the old bitstream. The characteristics of the benchmarks is shown in Figure 7. For each benchmark, the left one is the old configuration and the right one is the new configuration. The device used is mentioned next to the benchmark and device utilization is given at the bottom of each benchmark.

Benchmarks 2 to 5 consist of bitstreams demonstrating high correlation typical of reconfigurable systems - O and N sharing static kernels. Each kernel was synthesized independently and then hand mapped onto the assembled design to ensure that the kernels do not overlap with each other before being passed through place and route tools. Also different kernel placements were experimented with to investigate the effect of performing inter-bitstream compression with varying degrees of partial reconfiguration enabled.

Figure 6 indicates that inter-bitstream compression consistently outperforms intra-bitstream compression. The best compression ratio achieved with any inter-bitstream compression is 26.5%-75.8% better than the best compression ratio achieved with any intra-bitstream compression for our benchmarks. Moreover, DV maintains better performance than LZSS over both the inter- and intra-bitstream compression models. However, note that the advantage of DV over LZSS is reduced a little bit in inter-bitstream model compared to intra-bitstream model. Static kernels offer longer symbol matches between reference and beneficiary frames, facilitating LZSS compression.

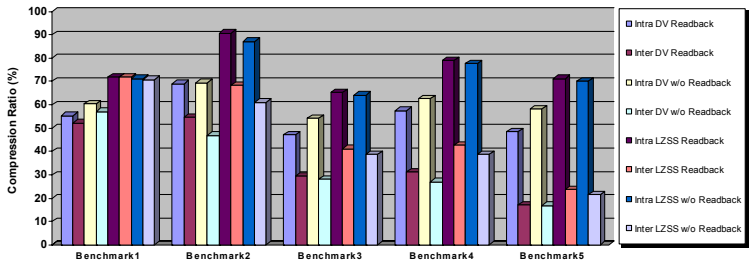

Figure 6. Compression Ratio for Inter-Bitstream DV and LZSS Schemes

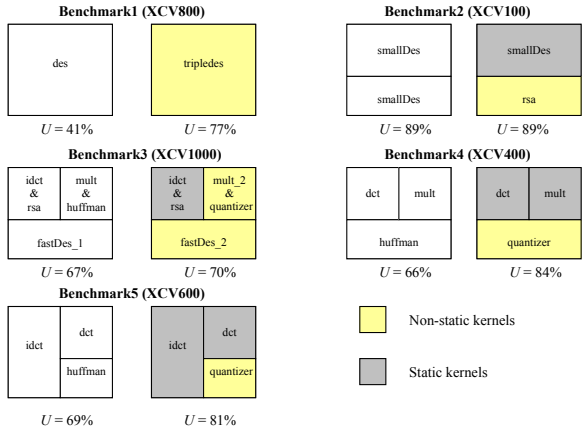

Figure 7. Benchmarks for Inter-Bitstream Compression.

Inter-bitstream CWOR performs better than the inter-bitstream CWR for both DV and LZSS compression. This is because inter-bitstream compression is enhanced with regularities offered by static kernels, giving rise to the predominance of readbacks in which each frame f_{new} uses f_{old} as the reference frame. Inter-bitstream CWOR achieves this without having to mention the address of the readback frame. Inter-bitstream CWR, on the other hand, caters for readback of any frame, and in doing so it has to specify the address of the readback frame, thereby incurring extra overhead.

The amount of inter-bitstream regularities present relies heavily on the amount of static configuration data and placement of static kernels [2]. For Benchmark1, which has no static kernel, inter-bitstream techniques performs marginally better than the corresponding intra-bitstream techniques by taking advantage only of the random regularities. Benchmarks 2 to 4 each maintains static kernels aligned horizontally across the device. Significant reduction in compression ratios is achieved with inter-bitstream compression over intra-bitstream compression. The reduction is not proportional to the static kernel size as it is dependent on the compression efficiency of the intra-bitstream compression. Using partial reconfiguration combined with DV or LZSS inter-bitstream compression, the compression efficiency can be improved significantly as demonstrated for Benchmark5.

CONCLUSION

In this paper, we proposed a class of configuration compression techniques targeted at Xilinx Virtex devices. Our novel compression technique for single configuration bitstream performs better than the best reported technique in the literature. We then extend the scheme to inter-configuration compression by exploiting the potential for data reuse between successive configurations. As far as we know, this is the first such algorithm proposed and it outperforms intra-configuration compression by a large margin.

ACKNOWLEDGMENTS

This work was partially supported by A*STAR Project 022-106-0043.

REFERENCES

[1] J. A. Storer amd T. G. Syzmanski. Data compression via textual substitution. *Journal of the ACM*, 29, 1982.

[2] K. Bazargan, R. Kastner, and M. Sarrafzadeh. 3-D floorplanning: Simulated annealing and greedy placement methods for reconfigurable computing systems. *Design Automation for Embedded Systems*, April 2000.

[3] Y. J. Chu and T. H. Liu. On the shortest arborescence of a direct graph. *Science Sinica*, 14, 1965.

[4] A. Dandalis and V. Prasanna. Configuration compression for FPGA-based embedded systems. In *FPGA*, 2001.

[5] S. Hauck and Z. Li. Configuration compression for Virtex FPGAs. In *FCCM*, 2001.

[6] S. Hauck, Z. Li, and E. Schwabe. Configuration compression for Xilinx 6200 FPGA. *IEEE TCAD*, 18(8), 1999.

[7] S. Hauck and W. Wilson. Runlength compression techniques for FPGA configuration. In *FCCM*, 1999.

[8] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers 40*, 1952.

[9] Xilinx Inc. *Virtex Series Configuration Architecture User Guide*, 2000.

[10] A. Khu. *Xilinx FPGA Configuration Data Compression and Decompression*, 2001.

[11] S. R. Park and W. Burleson. Configuration cloning: Exploiting regularity in dynamic DSP architecture. In *FPGA*, 1999.

[12] S. D. Warhade. Implementation approaches to Huffman decoding. Technical report, Wipro Technologies, 2002.