

# Code Compressor and Decompressor for Ultra Large Instruction Width Coarse-Grain Reconfigurable Systems

Nazish Aslam<sup>1</sup>, Mark Milward<sup>2</sup>, Ioannis Nouisias<sup>2</sup>, Tughrul Arslan<sup>1,2</sup>, Ahmet Erdogan<sup>1,2</sup>  
<sup>1</sup>*Institute for System Level Integration,*  
*Alba Campus, Livingston,*  
*UK EH54 7EG*  
<sup>2</sup>*University of Edinburgh,*  
*School of Engineering and Electronics,*  
*Edinburgh, UK EH9 3JL*

## Abstract

*This paper presents a code compression and on-the-fly decompression scheme suitable for coarse-grain reconfigurable technologies. A novel unit-grouping dictionary based compression technique utilizing special control bits to increase the effective storage capacity of the dictionaries is implemented and compared against an existing suitable technique for an example reconfigurable system. Compressions ratios in the range of 40%-59% are recorded with new scheme.*

## 1. Introduction

Reconfigurable computing is a technology that aims to combine the performance of FPGAs with the programmability/flexibility found in General Purpose Processors in a unified and easy programming environment. Reconfigurable computing cores tend to be developed in terms of flexibility and performance, which corresponds to a high amount of parallel processing units with its associated interconnect. This causes the reconfigurable architectures to suffer similar code density issues as VLIW/TTA processors, and code compression can be applied to address this [1].

This paper presents a code compression and decompression scheme targeted to an available coarse grain reconfigurable architecture [2], however the techniques mentioned can be partially or wholly applied to other multiple-issue processors with similar properties. The reconfigurable architecture introduced in [2] offers a very high number of parallel processing units and thus has an ultra wide instruction width. It is dynamically reconfigurable; hence it requires the ability to store many configuration codes contexts in memory which program the processing units for a particular moment in time.

An existing suitable code compression scheme is implemented for our targeted architecture and the results of it are presented in [3]. Our proposed technique presented in this paper builds up on our previous work [3], and offers fast decode while the

decompressor remains general-purpose. A single dictionary is applied to a group of functional units and special control bits are used to mask parts of the read dictionary contents on or off. The effective dictionary sizes are increased without actually increasing the silicon area, while also improving the compressions.

## 2. Target architecture

The compression scheme proposed is targeting a coarse grain reconfigurable architecture [3]. The hardware is dynamically reconfigurable and has a large array of independent heterogeneous functional units running in parallel. Each unit may perform simple or complex computations. The wide instructions have a rigid format, where the position of each sub-instruction directly corresponds to the targeted functional unit. The units expect varying width of instructions each. The proposed scheme may be applied to other architectures which possess similar characteristics.

## 3. Implementation

The proposed compression technique is predominantly based on our previous work [3], however offers further improvement. Temporal redundancy is removed by assigning a dictionary each to a *group of functional units*, resulting in  $X$  groups. Therefore a group location tag of  $Y$  bits is sufficient to uniquely identify all the grouped dictionaries, where  $Y$  is  $\log_2(X)$  rounded to the next integer, if needed. Each compressed payload, where a payload represents a complete unit of data which can be decoded independently from other payloads, requires the addition of status bits showing which units within a given group are active in a given wide instruction. As a group can contain from 1 up to  $Z$  units,  $Z$  bits are needed for units' status information. All dictionaries are the same fixed depth in our design to allow for parallel decode of several payloads. Thus, a single payload now becomes  $(Y + Z + D)$ bits in length, where  $D$  is the dictionary index bits. By using the status bits,

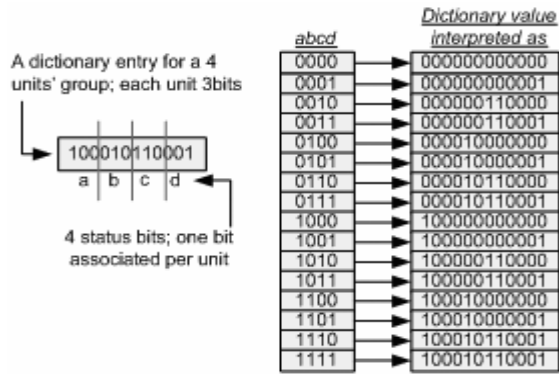


Figure 1. Status bits usage

this scheme effectively allows for more than  $2^D$  unique values to be stored in the dictionaries associated per group, without actually increasing the number of words in the dictionary. This is made possible by utilizing the status bits to mask off sections of the values read out from the dictionary. Fig. 1 gives an example of this. It shows how one dictionary location can effectively hold up to 16 unique values by utilizing 4 status bits.

Where there is only one large unit in a group, the status bits can be associated to various bit-sets of the single value, consequently allowing more values to be stored in limited space. This also results in fewer dictionary initialization bits needing stored in the main program memory, since instead of writing 16 new unique values to the dictionary, only one unique value has to be written. As a result an improved compression is observed. Another advantage of unit groupings is that fewer demultiplexers are now required in the design, which can lower the logic area overhead. The decompressor (see Fig. 2) assumes it is able to decode 8 payloads in parallel; therefore results in a reduced instruction fetch bitwidth from the original  $W$  bits wide instruction. Spatial redundancy is tackled by removing all grouped instructions in which all the units

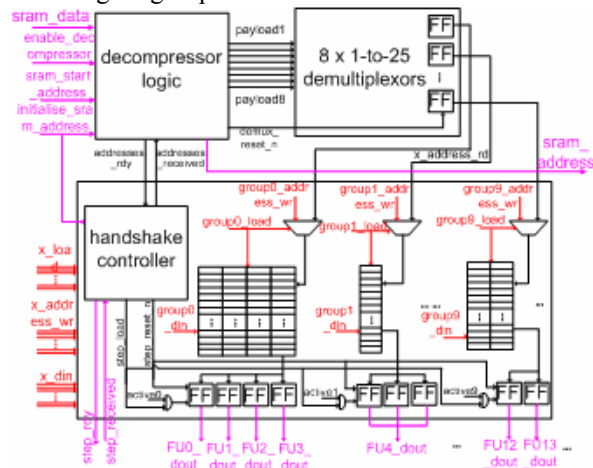


Figure 2. Decompressor design

are inactive. The dictionaries are also usable as a local memory to directly store short ( $< 2^D$  wide instructions) uncompressed programs, bypassing the decompressor.

## 4. Results

The decompressor design is synthesized onto UMC  $0.13\mu\text{m}$  technology using Synplicity ASIC. For our targeted reconfigurable architecture,  $X = 25$ ,  $Y = 5$ ,  $Z = 4$ ,  $D = 10$ ,  $W = 474$ . With the proposed scheme, the best compression achievable for a single wide instruction is 474bits reduced down to 19bits. This happens if only one functional unit is active. If all the functional units within a wide instruction are active then 474bits expand to only 475bits and this represents the worst case scenario. The performance of this scheme is compared with the results from [3].

Table I. Performance comparison results

	Existing scheme [3]	Proposed scheme	
<b>Decompressor logic</b>	0.16mm <sup>2</sup>	0.11mm <sup>2</sup>	
<b>Dictionaries area (485376bits)</b>	3.79mm <sup>2</sup>	2.59mm <sup>2</sup>	
<b>Total decompressor area</b>	<u>3.95mm<sup>2</sup></u>	<u>2.7mm<sup>2</sup></u>	
<b>Clock</b>	500MHz	500MHz	
<b>Instruction fetch bit-width</b>	128bits	152bits	
<b>Best case instruction decode</b>	2 cycles	2 cycles	
<b>Worst case instruction decode</b>	10 cycles	5 cycles	
<b>Power consumption, <math>\mu\text{W}/\text{MHz}</math></b>	1253	764	
<b>Program size after compression</b>	<b>2D DCT</b>	73.75%	59.1%
	<b>Min error</b>	40.2%	40.3%
	<b>Wimax</b>	41.51%	40.1%
	<b>H.264</b>	46.1%	40.6%

## 5. Summary

A dictionary based code compression scheme is implemented for a multiple-issue coarse-grain reconfigurable architecture. A comparison shows the new scheme outperforms the existing techniques in all aspects. Our novel technique of units-grouping combined with the use of special control bits to increase the effective dictionary sizes, achieved significant compression ratios in the range of 40-59%.

## 6. References

- [1] Xie, Y.; Wolf, W.; Lekatsas, H.; *Code compression for embedded VLIW processors using variable-to-fixed coding*, IEEE Trans. On Very Large Scale Integration Systems, Vol. 14, No. 5, pp.525-536, May 2006.
- [2] Reconfigurable Instruction Cell Array, *U.K. Patent Application Number 0508589.9*.
- [3] Aslam, N.; Milward, M.; Nousias, I.; Arslan, T.; Erdogan, A.; *Code compression and decompression for instruction cell based reconfigurable systems*, International Parallel and Distributed Processing Symposium, Reconfigurable Architectures Workshop, Mar. 2007.