*IEEE Symposium on FPGAs for Custom Computing Machines*, 1999.

# Abstract: Runlength Compression Techniques for FPGA Configurations[1]

Scott Hauck, William D. Wilson
Dept. of ECE, Northwestern University
Evanston, IL 60208-3118 USA
{hauck, wdw510}@ece.nwu.edu

## Abstract

The time it takes to reconfigure FPGAs can be a significant overhead for reconfigurable computing. In this paper we develop new compression algorithms for FPGA configurations that can significantly reduce this overhead. By using runlength and other compression techniques, files can be compressed by a factor of 3.6 times. Bus transfer mechanisms and decompression hardware are also discussed. This results in a single compression methodology which achieves higher compression ratios than existing algorithms in an off-line version, as well as a somewhat lower quality compression approach which is suitable for on-line use in dynamic circuit generation and other mapping-time critical situations.

## Target Architecture

These compression techniques were developed for the Xilinx XC6200 [Xilinx97]. This FPGA is an SRAM based, high performance Sea-Of-Gates FPGA optimized for reconfigurable computing. All configuration resources are accessed by addressing the SRAM through a standard memory interface. The Xilinx XC6200 series are partially reconfigurable devices. The configuration file consists of a set of address/data pairs. Since the device is partially reconfigurable, the target addresses written to may not be contiguous. Therefore, if the data is compressed the addresses must be compressed as well.

## Compression Considerations

The data compression for FPGA configurations must normally be lossless. The chosen compression strategy must be able to completely recover the exact data that was compressed. The compression technique chosen must allow for online decompression. Although compression will normally occur offline, where the entire configuration sequence is available, the entire compressed configuration sequence will not be available upon decompression. Finally, the compression technique may reorder the data, with some restrictions.

### Run-Length Compression

A variation of Run-Length encoding perfectly meets the requirements for the address compression. A series of addresses with a common offset can be compressed into a codeword of the form: base, offset, length. Base is the base address, offset is the offset between addresses, and length is the number of addresses beyond the base with the given offset. The configuration data sometimes repeats data values many times, allowing the compression of data streams with Run-Length encoding as well, although the compression may not be as great.

### Lempel-Ziv Compression

Lempel-Ziv takes advantage of repetitive series of data, replacing them with a single codeword. This codeword will tell when the series previously occurred, how long it is, and will give a new piece of data that is not part of the series. The codeword will consist of the form: pointer, length, and lastSymbol. The pointer will represent where the start of the series previously occurred within a window of time. This will meet the requirements for the data compression. This cannot be used for the address compression since the address stream has almost no repetition.

## Compression Strategies for Address/Data Pairs

1) **Basic Run-Length:** Compresses the address and data stream using runlength compression.
2) **Lempel-Ziv:** Compresses the address stream using runlength and the data stream using Lempel-Ziv.
3) **Run-Length with Reordering:** Reorder the address/data pairs in a more optimal manner.
4) **Adaptive Run-Length Reordering:** This strategy is similar to strategy #3 except that the optimal codeword and parameter sizes can be chosen on a configuration by configuration basis. The particular combination used is programmed into the hardware prior to configuration.

## Bus Transactions

### Multiplexing of the Address Bus

Since the addresses must be compressed and sent in addition to the data, our compression strategies use both the address and data buses to send the compressed codewords.

### Variation in Codeword Parameter Sizes

Through experimentation, we will determine how to divide up the available bits within a codeword into each field. In addition, we will vary the size of the codewords, in attempt to find the optimal combination.

---

[1] The complete paper is available at http://www.ece.nwu.edu/~hauck/publications.html

## Experiments

The algorithms described above were implemented in C++, and were run on a set of benchmarks collected from current XC6200 users. The results are given in the left portion of the table below, which lists the number of bus transactions required for address and data compression. Also listed is the overall compression ratio, which is the ratio of original file size to compressed file size. Given these results it is clear that Reordering Runlength is the preferred approach, since it achieves better compression results than the other approaches while requiring much simpler hardware decompressors than Lempel - Ziv.

| | | Basic Runlength | | | Lempel - Ziv | | | Reorder Runlength | | | Adaptive Reorder | | | Wild |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CalFile | Length | Addr | Data | Comp | Addr | Data | Comp | Addr | Data | Comp | Addr | Data | Comp | Comp |
| counter | 198 | 56 | 37 | 2.13 | 56 | 42 | 2.02 | 58 | 31 | 2.22 | 57 | 25 | 2.41 | 1.88 |
| parity | 208 | 7 | 21 | 7.43 | 7 | 18 | 8.32 | 14 | 14 | 7.43 | 8 | 14 | 9.45 | 7.43 |
| adder4 | 213 | 53 | 34 | 2.45 | 53 | 36 | 2.39 | 59 | 31 | 2.37 | 56 | 25 | 2.63 | 2.21 |
| zero32 | 238 | 22 | 27 | 4.86 | 22 | 22 | 5.41 | 28 | 22 | 4.76 | 22 | 20 | 5.67 | 4.18 |
| adder32 | 384 | 12 | 135 | 2.61 | 12 | 29 | 9.37 | 29 | 10 | 9.85 | 26 | 10 | 10.67 | 5.26 |
| smear | 695 | 135 | 183 | 2.19 | 135 | 136 | 2.56 | 210 | 68 | 2.50 | 210 | 61 | 2.56 | 2.28 |
| adder4rm | 907 | 331 | 264 | 1.52 | 331 | 233 | 1.61 | 419 | 70 | 1.85 | 417 | 67 | 1.87 | 1.61 |
| gray | 1200 | 371 | 388 | 1.58 | 371 | 353 | 1.66 | 592 | 80 | 1.79 | 587 | 79 | 1.80 | 1.85 |
| top | 1366 | 597 | 489 | 1.26 | 597 | 408 | 1.36 | 810 | 126 | 1.46 | 772 | 118 | 1.53 | 1.41 |
| demo | 2233 | 337 | 277 | 3.64 | 337 | 224 | 3.98 | 527 | 34 | 3.98 | 494 | 34 | 4.23 | 4.1 |
| ccitt | 2684 | 295 | 297 | 4.53 | 295 | 222 | 5.19 | 484 | 35 | 5.17 | 430 | 35 | 5.77 | 5.82 |
| t | 5819 | 972 | 769 | 3.34 | 972 | 602 | 3.70 | 1568 | 55 | 3.59 | 1534 | 55 | 3.66 | 5.51 |
| correlator | 11001 | 2513 | 4162 | 1.65 | 2513 | 1761 | 2.57 | 2665 | 144 | 3.92 | 2621 | 144 | 3.98 | 5.86 |
| Sum | 27146 | 5701 | 7083 | | 5701 | 4086 | | 7463 | 720 | | 7234 | 687 | | |
| Geometric Mean | | | | 2.64 | | | 3.20 | | | 3.34 | | | 3.60 | 3.28 |

Table 1. Overall comparison of complete compression algorithms. The "Wild" column represents our previous wildcard based compression algorithm [Hauck98]

## Conclusions

In this paper we have presented algorithms, communication methodologies, and hardware support for accelerating reconfiguration via compressing datastreams. The algorithms include techniques for harnessing runlength encoding and Lempel - Ziv approaches to the unique features of FPGA configurations. The bus formats and parameter settings provide efficient communications of these items, allowing for relatively simple hardware, embedded in the FPGA architecture, to perform the decompression. Our Adaptive Runlength algorithm provides significant compression results, reducing configuration size (and thus bandwidth requirements) by a factor of 3.60. Faster on-line algorithms can also use this hardware to achieve a compression ratio of 2.64. Such an on-line algorithm can be used for dynamic circuit creation and other situations where configuration compile time is a significant concern, including many applications of reconfigurable computing. Combined, this provides a complete and efficient compression suite for FPGA configuration management.

## References


[Hauck98] S. Hauck, Z. Li, E. J. Schwabe, "Configuration Compression for the Xilinx XC6200 FPGA", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 138-146, 1998.

[Xilinx97] Xilinx, Inc., "XC6200 Field Programmable Gate Arrays Product Description", April 1997.