A Partial Reconfiguration Controller for Altera Stratix V FPGAs

Zhenzhong Xiao, Dirk Koch, and Mikel Lujan
The University of Manchester, School of Computer Science, UK
Email: zhenzhong.xiao@student.manchester.ac.uk, dirk.koch@manchester.ac.uk

***Abstract*—With the introduction of the Stratix V family, the FPGA vendor Altera is now fully supporting partial reconfiguration in all their recent FPGA devices. A distinct feature in the Altera architecture is that reconfigurable regions can be arbitrarily defined which is possible by writing a configuration mask prior to writing the actual configuration data to the FPGA fabric.**

In this paper, we will present details and the flow for implementing partial reconfiguration using Altera FPGAs, as well as a study on configuration bitstream sizes and configuration speeds for various resource and bounding-box aspect ratio variants. The results are used to build a partial reconfiguration controller that is featuring a lightweight but effective bitstream decompression module for greatly improving configuration speed on a DE5-net board.

I. Introduction

With the Stratix V series, Altera as another major FPGA vendor is now fully supporting partial reconfiguration on their FPGAs. Partial reconfiguration is a core feature when using FPGAs for compute acceleration. In such scenarios, accelerators may change as part of the execution and certain cores of the design have to be kept active during reconfiguration. This can include memory controllers, on-FPGA CPUs, and/or PCIe cores.

In this paper, we present the partial reconfiguration flow for Altera FPGAs (Section II). We will provide more background information on the resulting partial configuration bitstreams in Section III. In Section IV, we present a decompression engine that is lightweight but still capable of achieving decent compression ratios. Section V reveals our configuration controller that we have developed for Altera Stratix V FPGAs. In Section VI, we conclude this paper.

II. The Altera Partial Reconfiguration Flow

The overall concept of the Altera partial reconfiguration flow is pretty much identical to the partial reconfiguration flow that the FPGA vendor Xilinx is currently using. The concept is illustrated in Figure 1. When implementing the static part of the system, one or more partial reconfiguration regions will be defined. Connection LUTs in route through mode that are placed inside the reconfigurable region provide an interface with the reconfigurable modules. A PR control block ensures that input and output wires are in a well defined state during configuration.

The reconfigurable modules are built incrementally on top of the static implementation. This uses the incremental compilation flow in Quartus for maintaining the exact place and route information of the static system when implementing partial modules. One implementation step is needed per combination of partial modules and reconfigurable region.

During our experiments on using partial reconfiguration on Altera Stratix V devices, we found that bitstream sizes can be relatively large. This issue will be investigated and tackled in the following sections.

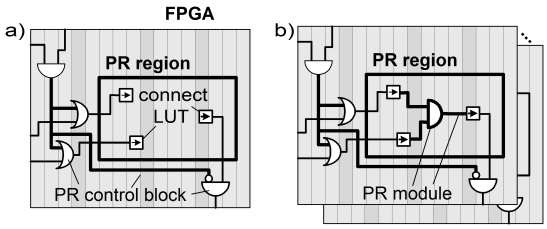

Figure 1. Partial reconfiguration flow. a) a partial reconfiguration region will be defined when implementing the static part of the system. LUTs in route through mode are used to interface with the reconfigurable modules. b) Reconfigurable modules are created incrementally on top of the static implementation while maintaining the place and route information of the static system.

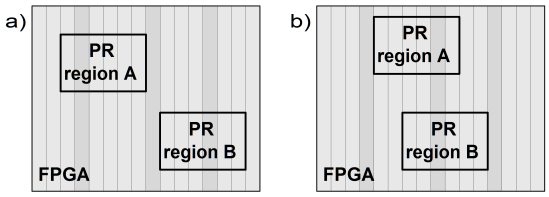

Figure 2. a) non overlapping PR regions, b) overlapping PR regions.

III. Configuration Bitstream Details

Altera Stratix V devices allow the definition of arbitrary bounding boxes that can be freely placed on the FPGA fabric. These devices do not have the limitation of only being able to be configured in clock region heights, as known from Xilinx FPGAs. This can potentially help in saving resources because bounding boxes can be better adjusted to the needs of a given design. However, in order to allow this, Altera uses a mask configuration that defines the resources that are written during the actual reconfiguration (which defines the actual logic functions and the routing of the fabric). This concept was proposed in research papers before. A hardware friendly and high performance approach was presented in [1] where a single mask defines the height of the resources to be configured. Another related approach was introduced as hyperreconfigurable FPGAs in [2].

The configuration mask can be defined very fine grained and it is automatically generated by the Quartus design tool. If only one reconfigurable region is used, this mask is fixed and included in the initial configuration. In the case that multiple reconfigurable regions are used on the same FPGA, there are two cases to distinguish: 1) if reconfigurable modules do not overlap in vertical direction (i.e. if reconfigurable regions do not use the same vertical column of resources) and 2) if modules overlap, as shown in Figure 2.

In the first case, the mask is static and it is sufficient to only write the partial module bitstreams to the FPGA for swapping modules on the FPGA as it is sufficient to

Table I
CONFIGURATION BITSTREAM SIZE FOR DIFFERENT SIZES, ASPECT RATIOS, AND LOGIC UTILIZATION LEVELS.

ALMs utilization	Mode	size [ALMs] 80 x 80	120 x 54	54 x 120
30%	AND/OR	15 794 KB	20 508 KB	11 343 KB
	SCRUB	8 720 KB	12 633 KB	6 442 KB
60%	AND/OR	15 829 KB	22 274 KB	11 375 KB
	SCRUB	8 723 KB	12 632 KB	6 444 KB
90%	AND/OR	15 766 KB	23 289 KB	11 436 KB
	SCRUB	8 720 KB	12 632 KB	6 442 KB
utilization	Mode	60 x 60	120 x 30	30 x 120
30%	AND/OR	12 007 KB	20 529 KB	6 720 KB
	SCRUB	6 707 KB	12 633 KB	4 046 KB
60%	AND/OR	12 015 KB	20 924 KB	6 920 KB
	SCRUB	7 707 KB	12 632 KB	4 046 KB
90%	AND/OR	12 068 KB	23 284 KB	6 958 KB
	SCRUB	6 707 KB	12 634 KB	4 046 KB

write the mask with the initial reconfiguration. For the second case, the mask has to be written whenever the region changes in vertical direction. However, for easing the configuration process, the mask can be included in each partial configuration bitstream. In this case, the user or system does not have to keep track about the present mask configuration at run-time.

A. Configuration Bitstream Size

Because the configuration bitstream size determines directly the configuration time, we will examine now the configuration bitstream size in more detail. We investigate the impact of 1) the module size, 2) the aspect ratio, 3) the logic utilization level, and 4) the configuration mask on the bitstream size. For this reason, we created a synthetic module that allows us to define different resources (i.e., logic, memory, and DSPs) independently to each other through generics.

We parameterized this netlist to occupy 3 different fill levels (30%, 60%, and 90%) for the resources provided in a reconfigurable region. We run experiments for two different sizes of reconfigurable regions (in terms of ALM primitives). We selected 3600 ALMs for a small and about 6400 ALMs for a larger reconfigurable region. The small (large) region represents enough ALMs to implement about 5 times (9 times) a Nios II/f softcore CPU [3]. In addition to the amount of logic (i.e. the size of the partial region), we examined if the aspect ratio has impact on the configuration bitstream size. To illustrate the experiment, we provide screenshots of the first three experiments, as shown in Figure 3.

It is not obvious how these parameters impact the bitstream size and Altera is not providing information about this in their documentation. We created in total 36 bitstreams and the results are listed in Table I.

1) Impact on the Fill Level: We could not see any significant correlation that indicated an influence of the relative amount of resources used inside a reconfigurable region on the resulting bitstream size. This indicates that no compression is used for speeding up the configuration process by the FPGA. This may, however, be possible if parts of a bitstream may not have to be written in rather empty modules (i.e. modules taking only a relatively small number of resources).

2) Impact of the Module Size and Aspect Ratio: We took the results from Table I and computed the quotients for bitstream size per row and bitstream size per column for the

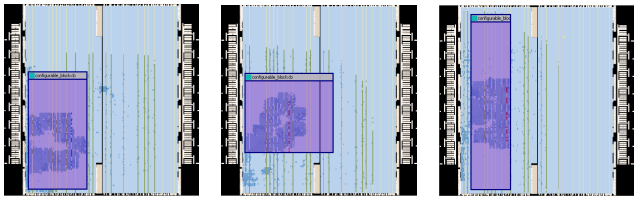

Figure 3. Three implementation variants with different aspect ratio using 30% of the ALMs inside the partially reconfigurable region.

Table II
CONFIGURATION BITSTREAM SIZE PER NUMBER OF ALM ROWS (HEIGHT) AND ALM COLUMNS (WIDTH).

	height / width	80 / 80	54 / 120	120 / 54
bitstream size (average)	AND/OR	15 794 KB	22 024 KB	11 385 KB
	SCRUB	8 721 KB	12 632 KB	6 443 KB
$\frac{\text{size}}{\text{height}}$	AND/OR	197.4 KB	407.8 KB	94.9 KB
	SCRUB	109.0 KB	233.9 KB	53.6 KB
$\frac{\text{size}}{\text{width}}$	AND/OR	197.4 KB	183.5 KB	210.8 KB
	SCRUB	109.0 KB	105.3 KB	119.3 KB

average bitstream size, as listed in Table II. Interestingly, the configuration bitstream size depends preliminary on the width of a reconfigurable module, but not its height. Consequently, the mask approach used by Altera is not using the mentioned approach in [1] that would allow bitstream sizes that correlate directly with the size and regardless to the actual aspect ratio.

Our results reveal that the configuration bitstream size is on average 111.2 KB/ALM-column and 197.3 KB/ALM-column when considering the additional configuration mask. We labelled our tables in the same way Altera labels configurations containing a configuration mask (AND/OR) and configurations that do not contain the mask (SCRUB). Please note that differences in the quotient arise from additional memory and multiplier columns (that we have not explicitly listed for the sake of brevity. However, the here reported values are good rule of thumb values for arbitrary systems implemented on the DE5-Net FPGA development board and allow a good estimate of partial bitstream sizes (and corresponding configuration times) just by looking on the number of ALM columns allocated for the reconfigurable region. In a nutshell, the partial configuration bitstream size has a similar behaviour than what is know from Xilinx Virtex-II FPGAs with the difference that the configuration mask allows reconfiguring arbitrary parts of a column without any savings on bitstream size.

If we put the partial bitstream size in relation to the full (initial) configuration bitstream size (which is 31 010 KB without a mask), we see that a module that takes only 25% of the resources can easily come with a partial bitstream size that is half the full configuration memory requirements or even more than 2/3 when taking a mask into consideration. Consequently, whenever possible, a narrow, but tall aspect ratio for reconfigurable regions (and consequently the reconfigurable modules) will improve partial bitstream storage and reconfiguration speed. Moreover, this fits better the Altera FPGA fabric that provides some features that are available in vertical direction of the fabric but not in horizontal, such as carry chains and wires for cascading memories and multiplier blocks [4]. Finally this may allow building reconfigurable systems that do not need including the configuration mask information into the bitstream.

3) Configuration Mask Size: We also measured the size of the mask, which is the difference between the rows AND/OR and SCRUB in Table II:

$\frac{\text{size}}{\text{width}}$	AND/OR	197.4 KB	183.5 KB	210.8 KB
	SCRUB	109.0 KB	105.3 KB	119.3 KB
	difference	88.4 KB	78.3 KB	91.5 KB

As we can see, the mask is also proportional to the width of a reconfigurable module and that the mask is making a bitstream about 86 KB larger per used ALM column, which is a 77% overhead over the configuration mode that does not need writing a configuration mask. Consequently, the freedom in defining bounding boxes on Altera devices comes at a penalty in configuration time, when multiple reconfigurable modules share the same column of resources. Considering the default configuration process that is used for OpenCL acceleration, this is not a concern because only one accelerator is configured exclusively to the FPGA at a given point in time, and hence no reconfiguration mask is needed.

IV. Configuration Bitstream Compression

Altera provides a compression option that is build into their Altera Enhanced Configuration (EPC) devices [5]. However, we have not found any support for configuration decompression by the FPGA itself. Because the configuration bitstream size is large, partial reconfiguration 1) may take too much time, 2) may come with extra monetary cost for configuration storage, and 3) may impact system performance if configuration data transfer is congesting the memory subsystem of a design. To overcome these issues, we examined bitstream compression as a viable option to reduce bitstream sizes and to fight the afore mentioned drawbacks of large configuration bitstreams.

Configuration compression has been proposed several times before as an effective solution to reduce bitsteam sizes (e.g., [6], [7], and [8]) and we used LZSS compression [9] (which was originally proposed by Hauck in [10] for configuration compression) because it is well suited for FPGA implementations. It achieves in general good compression ratios at low implementation cost while still providing fast operation.

The algorithm can be parameterized in various ways and combined with other algorithms such as Huffman encoding. We examined different parameters for the LZSS algorithm and we considered an 8-bit datapath as well as an 16-bit datapath. For our decompression module, we aimed for a resource efficient implementation as we do not want to offset the benefits of partial reconfiguration by an expensive configuration controller. Therefore, we decided for a relatively short shift register with just 32 entries (depending on the mode, 8 or 16 bits wide). Consequently, five bits are sufficient to encode the LZ offset. We used three bits to encode the length field and we experimented with different length values to improve the compression ratio. The selection for 5 bits for the offset and 3 bits for the length fields makes interfacing with an 8-bit or 16-bit datapath very resource efficient. Finally, the short shift register fits well the distributed memory primitives in the ALMs which means we need no dedicated memory blocks for the implementation of the shift register.

We examined manually several length value sets for the LZSS compression engine and Figure 4 lists the corresponding results. Table III gives a summery taking the average sizes of the different compression methods into account. As can be seen in the table, we achieved compression ratios well below 50% which in some cases may double the configuration speed, when configuring from a slow configuration source. In addition to the original uncompressed values, we added the compressed file sizes for Altera Enhanced Configuration (EPC) devices [5] (which include a decompression module) and zip compression results (Zip 2.31 running in default mode). The figure reports our results for 8-bit wide shift registers and 16-bit wide shift registers used in the LZSS engine. As can be seen, the 8-bit wide datapath delivers better compression ratios because there is a higher probability for finding matches on shorter words in the buffer window. However, an 8-bit datapath delivers substantially less throughput than the 16-bit variant.

Table III
Average compression ratios (compressed/uncompressed) for the bitstreams from Table I.

mode	EPC	zip	LZSS-8	LZSS-16
AND/OR	49.5%	11.0%	27.4%	32.2%
SCRUB	36.1%	17.1%	36.0%	43.5%

While our decompression engines cannot compete with zip, our 8-bit LZSS engine is substantially better in compression ratio as well as in throughput than the EPC device from Altera. Interestingly, our compression engine achieves better compression ratios for configuration bitstreams containing a mask, while the EPC device compresses better plain bitstreams that do not contain bitstream masks.

From Figure 4 it can be observed that the utilization level of reconfigurable modules has no relevant impact on the achieved compression ratio. This is an indicator that the partial configuration bitstream may contain the configuration information of the surrounding static system (i.e., the bitstream data that defines resources above and below a reconfigurable region). However, as the configuration mask suggests that this data is not actually written into the fabric, we could eventually substitute this static fraction of the partial bitstream to substantially further improve compression ratios. This will be addressed in further research.

V. Configuration Controller

This section presents our configuration controller that we developed for Altera devices. The original motivation for developing our own configuration controller came from the fact that a reference design (provided by Altera) for partial reconfiguration used two large on-FPGA memories for storing the configurations of two tiny partially reconfigurable modules. While this is sufficient to give a glimpse on how the partial reconfiguration flow works, this is obviously not suited to implement real-world applications. Furthermore, we found that partial bitstream sizes are relatively large (as investigated intensively in Section III-A). Consequently, we developed a configuration controller that incorporated compression as well as providing an easy adaptable interface such that various configuration sources (e.g., PCIe, DMA engines, or serial communication controllers) can be used with our configuration controller.

Figure 5 gives an overview of our controller. A partial configuration bitstream that was generated and compressed at system design time is send through an input FIFO to our LZSS decompression engine and the decompressed stream is passed through another FIFO to the configuration state machine. The state-machine keeps track about the

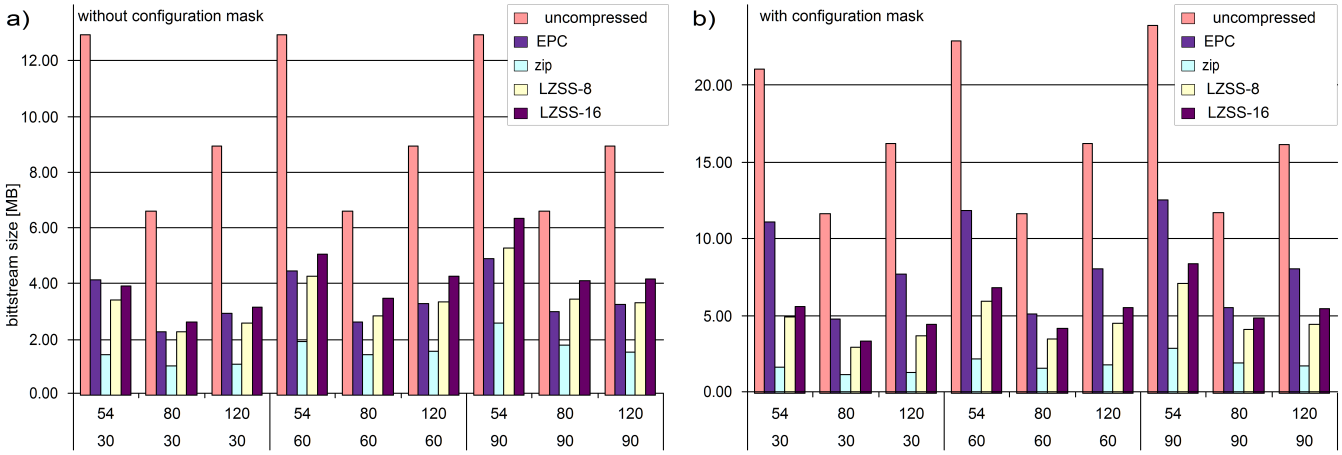

Figure 4. Compression results for the bitstreams from Table I.

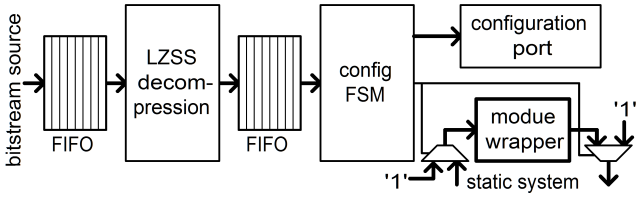

Figure 5. Configuration controller for Altera Stratix V FPGAs featuring bitstream decompression.

reconfiguration process and manages flow control and back pressure, if needed. The FIFOs are implemented using distributed memory and enhance throughput because the output and input data rate of the decompression engine is data dependent.

We implemented the whole configuration controller for a 16-bit datapath supporting 100 MHz (i.e. the maximum configuration speed of Stratix V FPGAs). Depending on the input source, this allows configuring at 200 MB/s. which we achieved when configuring from small on-FPGA buffers. The whole configuration controller including the decompression engine and the two FIFOs requires 1,701 ALMs out of the 234,720 available on the DE5 board (0.7%).

VI. Conclusions

In this paper we presented the new Altera partial reconfiguration flow and identified and addressed the problem of extensive sizes for partial reconfiguration bistreams. We investigated the impact of different design parameters on the bitstream size and found that reconfigurable regions should ideally by rather narrow and tall instead of the other way round. Moreover, reconfigurable regions should not overlap in vertical direction which allows omitting writing a configuration mask prior to the actual fabric configuration. This allows saving about 77% in configuration time.

Based on these observations, we developed and evaluated a configuration controller that can be used in a large variety of systems. The controller will be made available on request. Future work will target building more complex reconfigurable systems on Altera Stratix V FPGAs which will in particular include designing reconfigurable modules using OpenCL.

Acknowledgment

This work is kindly supported by the project *Reconfigurable Tera Stream Computing* funded by the Defence Science and Technology Laboratory, UK under grant DSTLX10000092266. Mikel Lujan is supported by Royal Society University Research Fellowship.

References

[1] Koch et al., "FPGA Architecture Extensions for Preemptive Multitasking and Hardware Defragmentation," in *Proc. of Int. Conf. on Field-Programmable Technology 2007 (ICFPT '07)*. IEEE, Dec. 2007, pp. 433–436.

[2] S. Lange and M. Middendorf, "Models and Algorithms for Hyperreconfigurable Hardware," in *Dynamically Reconfigurable Systems*, M. Platzner, J. Teich, and N. Wehn, Eds. Springer, Heidelberg, Feb. 2010, pp. 75–94.

[3] Altera Inc., "Datasheet: Nios II Performance Benchmark," Dec. 2015, available online: `https://www.altera.com/en_US/pdfs/literature/-ds/ds_nios2_perf.pdf` (accessed 28.03.2016).

[4] Altera Inc., "Stratix V Device Handbook," Dec. 2015, available online: `https://www.altera.com/-content/dam/altera-www/global/en_US/pdfs/-literature/hb/stratix-v/stx5_core.pdf` (accessed 28.03.2016).

[5] Altera Inc., "Enhanced Configuration (EPC) Device Datasheet," Jan 2012, available online: `https://www.altera.com/en_US/pdfs/literature/-hb/cfg/cfg_cf52001.pdf` (accessed 28.03.2016).

[6] Z. Li and S. Hauck, "Configuration Compression for Virtex FPGAs," in *Proc. of the 9th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, 2001, pp. 147–159.

[7] Koch et al., "Hardware Decompression Techniques for FPGA-based Embedded Systems," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 2, pp. 9:1–9:23, June 2009.

[8] Pan et al., "Configuration Bitstream Compression for Dynamically Reconfigurable FPGAs," in *Proc. of the IEEE/ACM Int. Conf. on Computer-aided design (ICCAD)*, 2004, pp. 766.

[9] J. A. Storer and T. G. Szymanski, "Data Compression via Textual Substitution," *Journal of the ACM*, vol. 29, no. 4, pp. 928–951, 1982.

[10] S. Hauck and W. D. Wilson, "Runlength Compression Techniques for FPGA Configurations," in *Proc. of the 7th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM99)*. IEEE Computer Society, 1999, pp. 286.