Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration

Tobias Becker[1], Wayne Luk[1] and Peter Y.K. Cheung[2]
[1] Department of Computing, Imperial College London
[2] Department of Electrical and Electronic Engineering, Imperial College London
{tobias.becker, w.luk, p.cheung}@imperial.ac.uk

Abstract

This paper introduces a method that enhances the relocatability of partial bitstreams for FPGA run-time reconfiguration. Reconfigurable applications usually employ partial bitstreams which are specific to one target region on the FPGA. Previously, techniques have been proposed that allow relocation between identical regions on the FPGA. However, as FPGAs are becoming increasingly heterogeneous, this approach is often too restrictive. We introduce a method that circumvents the problem of having to find fully identical regions based on compatible subsets of resources, enabling flexible placement of relocatable modules. In a software defined radio prototype with two reconfigurable regions, the number of partial bitstreams is reduced by 50% and the compile time is shortened by 43%.

1 Introduction

Advanced Field Programmable Gate Arrays (FPGAs) have emerged into platforms that allow the development and implementation of complex systems with short development time, high flexibility and high performance. The introduction of processors, RAMs and DSPs in platform FPGAs have made these devices capable of full system-on-chip implementations for complex applications. The continuing trend of moving dedicated hardened IP-cores such as Ethernet MACs and PCI Express [20] into the FPGA fabric is consistent with their evolution from glue logic to become the centre of the system.

Most of today's commercially available FPGAs are based on SRAM technology, which can be exploited to reconfigure the device fully or partially at run time in order to adapt the system to new computational requirements. Run-time reconfiguration can improve performance [12], increase functional density [14] and reduce power dissipation [7], [13]. Idle parts of a system can be unloaded to save power or to free up space for a new incoming module. A static part of the system can continue its operation during reconfiguration without disruption. It has been proposed to integrate the control of the reconfiguration process into the FPGA itself [1]. Embedded processors, such as the Xilinx MicroBlaze softcore [15] or PowerPC hardcore processor [16], provide a capable means of embedded processing for FPGAs. In combination with the Internal Configuration Access Port (ICAP) [1], they can form systems that quickly modify themselves.

Partial bitstreams used in reconfigurable applications are usually location specific, i.e. a bitstream contains configuration information which is specific to one location on the device. However, a significant number of reconfigurable applications could benefit from relocatable bitstreams. Relocatable bitstreams contain configuration data which can be reused in multiple reconfigurable regions on the FPGA. Early work on run-time reconfiguration considers the possibility of an operating system that runs tasks in hardware on a partially reconfigurable FPGA [2]. Brebner suggests swappable logic units that can execute a certain function and can be placed anywhere on the FPGA. This concept is supported by the regular structure of the Xilinx 6200 architecture. Hübner et. al. propose a reconfigurable platform for automotive control functions on a Xilinx Virtex-II FPGA [4]. The system contains four reconfigurable regions of identical size which are connected by a common bus system. However, the system does not employ bitstream relocation because of the irregular location of combined BlockRAM (BRAM) and multiplier columns in the reconfigurable regions. Other work considers a reconfigurable system-on-chip for a portable media platform that can move tasks from software into reconfigurable hardware [9]. This system cannot relocate tasks on the FPGA and requires a bitstream for every possible task location. Sedcole et. al. propose a reconfigurable platform for video processing [10]. The design exploits symmetries of the BRAM location in the Virtex-II FPGA to relocate image processing elements. Another interesting application for bitstream relocation are reconfigurable software defined radio systems. One research project involves a software defined radio demonstrator with two reconfigurable waveforms on a Virtex-II Pro FPGA [11].

However, the circuits processing the waveforms are not relocatable. Especially more complex systems with a larger number of waveforms would benefit from the relocatability of bitstreams.

In this paper we analyse the problem of bitstream relocation in modern heterogeneous FPGAs, and propose a method that enhances the relocatability. We illustrate this method on Virtex-4 FPGAs and point out improvements over previous approaches. Our method can also be applied to Virtex-5 and Spartan-3 FPGAs and to older architectures such as Virtex-E, Virtex-II and Virtex-II Pro.

The key contributions of this paper are:

- A model for relocation of configurations to non-identical regions in FPGAs based on compatible subsets of resources (Section 3),
- An adaptation of our method to Xilinx Virtex-4 FPGAs (Section 4 and 5),
- An implementation of our method as software driver for ICAP (Section 6),
- Demonstration of relocatable bitstreams in a reconfigurable software defined radio prototype (Section 7.1),
- A detailed analysis of module relocatability with our method compared to previous approaches (Section 7.2).

2 Background and Related Work

Traditional FPGA design involves floorplanning where modules or blocks of the design are constrained to specific areas of the device. Applications with run-time reconfigurable modules usually require stricter floorplanning to isolate dynamic modules from the static part of the system [8]. Conventionally, reconfigurable modules are assigned to a dedicated area inside the FPGA. If multiple instances of one module are used in different regions an individual bitstream is created for each region. However, it is not necessary to restrict a physically implemented instance of a module to one location. FPGAs provide a regular fabric, and one circuit could potentially reside in multiple locations.

The advantage of being able to relocate a bitstream is reduced design time as well as reduced memory requirements. Without relocation, for M modules usable in N regions, $N \cdot M$ partial bitstreams have to be produced and stored for run-time reconfiguration. With bitstream relocation, one version of the module can be instantiated in all locations, therefore reducing the total number of required bitstreams to M. This significantly reduces the size of memory required to store the partial bitstreams. With full bitstreams containing up to 50 Mbits in Virtex-4 [19] and up to 80 Mbits in Virtex-5 [18] partial bitstreams could have up to tens of MBits. It is therefore important to keep the number of partial bitstreams as low as possible to reduce cost of the storage, especially since costly fast memories should be used to minimise impact on performance. One aspect of run-time reconfiguration is more efficient utilisation of the FPGA which should not be contradicted by requiring an expensive configuration storage. Another aspect is reduced design time. The generation of bitstreams requires physical implementation including time-consuming placement and routing. Reducing the number of partial bitstreams by a factor of N can considerably shorten the design cycle.

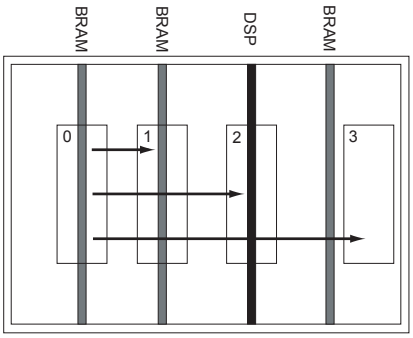

Figure 1. Relocation from region 0 to an identical (1) and non-identical regions (2, 3).

Previous work has reported methods and tools to perform bitstream relocation between identical regions as illustrated by relocating from region 0 to region 1 in figure 1. In contrast, this paper presents a method which allows relocation to non-identical regions, such as regions 2 and 3.

Previous tools include PARBIT, which can relocate partial bitstreams on Virtex and Virtex-E FPGAs [3]. It can also be used to move partial bitstream between different devices. The tool runs under UNIX or Windows and is not suitable for run-time relocation. REPLICA and REPLICA2Pro are tools for run-time relocation on Virtex, Virtex-E, Virtex-II and Virtex-II Pro [5], [6]. Both tools are implemented as hardware blocks which reside inside the FPGA and modify addresses within the bitstream as the configuration data is streamed in. Blodget et. al. propose an API for ICAP that allows run-time modification of CLB configuration data [1]. The API also includes a copy function that can be used to relocate CLB content. The API is called from application code and requires a MicroBlaze or PowerPC system inside the FPGA with ICAP peripheral.

3 Model of Flexible Configuration Relocation

Early FPGA architectures provide a regular fabric which is beneficial for the relocation of configurations. Memory

blocks are the first specialised resource to be introduced to the FPGA fabric because memory is expensive to realise in generic logic resources. Modern FPGA fabrics show a continuing trend of growing heterogeneity. Specialised resources can provide certain functions more efficiently and with higher performance than an equivalent function implemented in generic logic resources.

However, heterogeneity is problematic for relocation of configurations. In prior work identical regions are required. However for large modules, it can be difficult to find an identical target region. Even if identical regions are found, they can be impractical because of other floorplanning constraints.

3.1 Model

We propose a technique that does not require fully identical regions. Instead we utilise a compatible subset of resources in non-identical regions. We model the FPGA as an architecture with two layers as illustrated in figure 2:

- The functional layer: it contains logic resources and routing resources.
- The configuration memory layer: an underlying layer of memory cells that control the configuration of logic and routing resources of the functional layer.

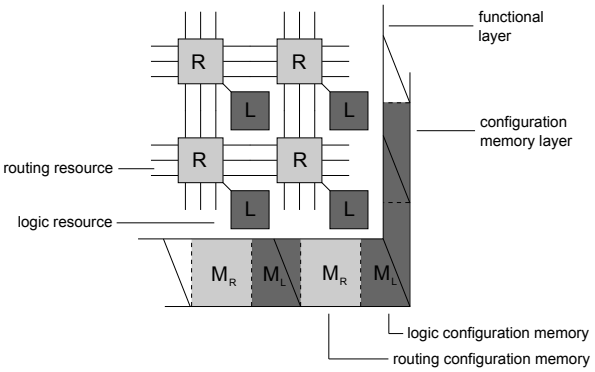

Figure 2. FPGA model with functional layer and configuration memory layer.

The configuration memory is characterised by an allocation to functional units. Logic resources L and routing resources R are mapped to memory pages M_L and M_R. In a regular FPGA structure, this mapping of resources to memory is uniform for all tiles i.e. configuration data from one specific memory location M_L can be used to configure any resource L on the device. This is the basic precondition for bitstream relocation.

First we focus on the functional layer. We consider a fabric of logic resource A that which is interrupted by heterogeneous resources B and C. Adjacent to these logic resources are routing blocks. In FPGAs routing resources are usually uniform i.e. they provide identical connectivity to neighbouring cells.

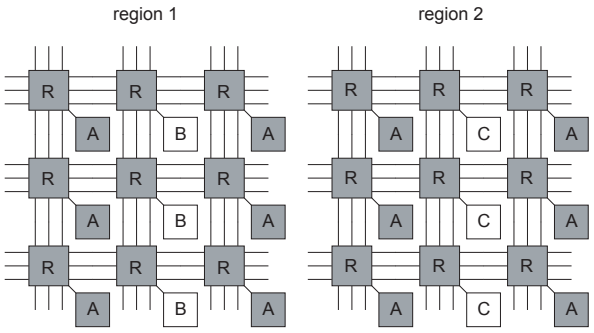

Figure 3. Heterogeneous FPGA fabric with compatible subset of resources.

Figure 3 illustrates an example of two regions that are not fully identical. Region 1 contains heterogeneous resources of type B whereas region 2 provides resources of type C. To enable relocation between these two regions, we identify an identical subset of resources. Identical in both regions are the two outer columns of resource A as well as all routing resources R. Even though logic resources B and C cannot be used, their associated routing resources are still available. The compatible subset of resources is highlighted grey.

Configuration data corresponding to this subset can be considered as relocatable between the two non-identical regions as illustrated in figure 4. This configuration can create a functioning circuit in both region 1 and region 2.

3.2 Design Flow

In order to create relocatable configuration data the designer has to ensure that only the compatible subset of resources is used by the implementation tools. We assume that the design is already partitioned into a static part and a number of reconfigurable modules. The design flow consists of the following steps:

1. Floorplanning: the reconfigurable regions are placed on the FPGA. If fully identical regions cannot be found the designer tries to minimise the number of mismatching resources. Other floorplanning considerations can include the connectivity of regions, device fragmentation and the location of static logic.

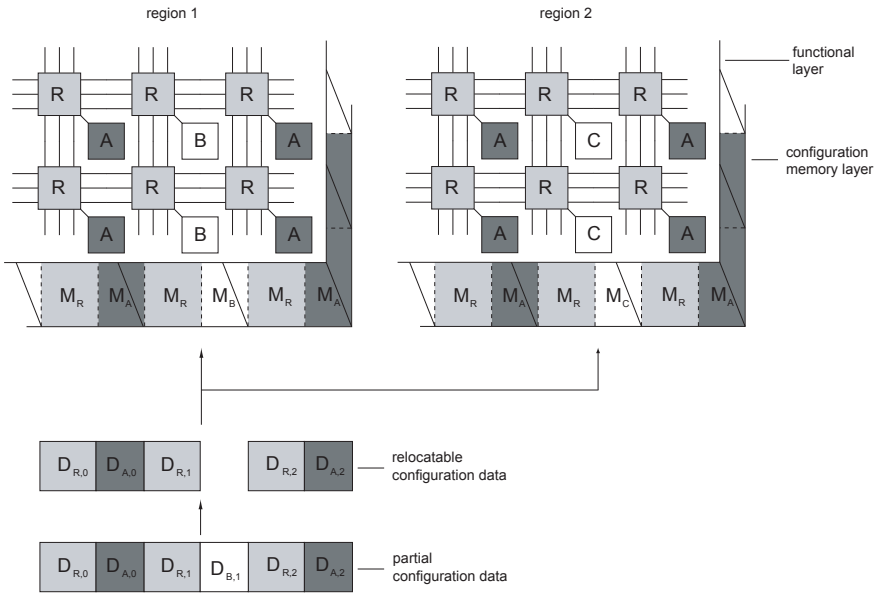

Figure 4. Model of a relocatable bitstream: configuration data corresponding to the compatible subset can be loaded in both locations.

2. Selection of the module implementation region: all modules are only instantiated in one region. This region is used for physical implementation.

3. Constraint setup: area constraints are created for all reconfigurable regions to keep them free from static logic. Additionally the designer has to set up implementation constraints that instruct the placement program not to place any logic instances into mismatching logic resources. If for instance in figure 3 region 1 is chosen as module implementation region, the placement program is not allowed to place logic into resources B. However, the router is allowed to use all routing resources. All other constraints such as timing requirements and IO locations are added.

4. Physical implementation: the design tools map, place and route to implement the static design and all reconfigurable modules on the FPGA according to the design constraints.

5. Configuration generation: a full configuration file is generated for initial configuration of the static system. Additionally one partial configuration file is generated for each module.

As shown in figure 4, relocatable configuration data can be produced by simply removing configuration data $D_{B,1}$. When loaded into region 1 or 2 it will leave the mismatching logic resources B or C in their unconfigured default state. Alternatively, zero-configuration data for resource B or C can be dynamically inserted into the configuration data when loading it into the device.

4 Virtex-4 Configuration Architecture

All Xilinx Virtex families support partial run-time reconfiguration. We demonstrate the technique for Virtex-4 FPGAs [19] which support run-time reconfiguration of two-dimensional blocks. We briefly describe the Virtex-4 configuration architecture to outline the implementation of our method. Figure 5 shows the structure of the Virtex-4 LX25 FPGA. The device consists of a regular CLB grid which is interrupted by columns of specialised resources. These specialised columns can contain BRAMs, DSPs and IOs. A notable difference to Virtex-II and Virtex-II Pro devices, which have combined columns of BRAMs and multipliers, is the separation of BRAMs and DSPs in Virtex-4. Another change in the architecture is the presence of one IO column in the middle of the device. Overall the heterogeneity has increased compared to previous Virtex families.

The configuration memory is organised in frames which are usually loaded in a sequential order. Frames are random addressable and represent the smallest unit of configuration. All Virtex-4 FPGAs have a fixed frame length of 1312 bits.

This differs from previous Virtex families which are configured by frames that span the entire height of the device. Therefore reconfigurable modules also have to cover the full height in these devices. Two-dimensional reconfiguration is also possible but requires more complex techniques [11]. The fixed frame length in Virtex-4 corresponds to rows with a height of 16 CLBs. Rows can be configured without disturbing any logic above or below, thus enabling much more flexible floorplans. Figure 5 shows a possible floorplan with four reconfigurable modules. Multiple reconfigurable modules as well as static logic can coexist in the same column.

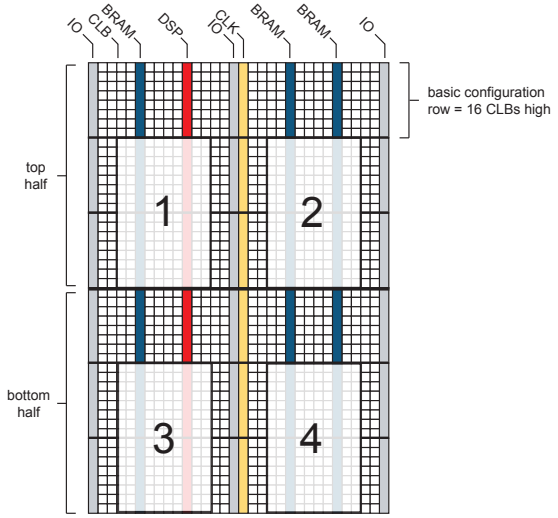

Figure 5. Structure of the Virtex4 LX25 FPGA with an exemplary floorplan of four reconfigurable modules.

Configuration frames have a 32-bit address which is composed of five values. The memory map of the first four values is indicated in figure 6. The block type address field can have three different values: Block type 0 contains all frames for CLB, DSP, IO and clock configuration. Block type 1 contains BRAM interconnect configuration frames. Block type 2 specifies frames with BRAM data content. The top/bottom bit indicates if a frame is located in the top-half or the bottom-half of the device. The row address specifies the row within one half of the FPGA and is incremented from the centre of the device to the outside. Within each row resources are organised in columns. A column has a height of 16 CLBs, 4 DSPs or 4 BRAMs. The column address is incremented from the left side of the FPGA to the right. Since BRAM configuration frames have a different block type, they are addressed independently from all other resources.

Each column in the configuration memory is configured

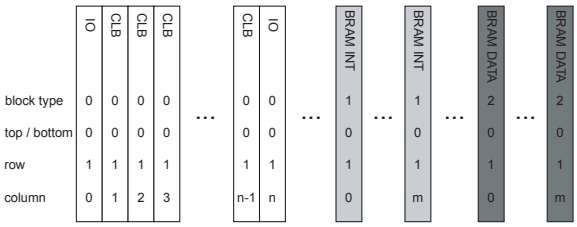

Figure 6. Configuration memory map in Virtex-4 FPGAs.

by multiple frames as illustrated in figure 7. The frame number is the fifth value in the address field.

CLB columns are configured by 22 frames where the first 20 frames control routing resources and the last two frames configure the logic resources of the CLB column. A DSP column is configured by 21 frames. Again the routing resources are configured by the first 20 frame. The last frame contains the configuration of the DSP. BRAM configuration data consist of two separate block types. The first block contains 20 frames of routing information. A second block contains 64 frames of BRAM data. CLB, DSP and BRAM columns contain equal memory sections for routing configuration memory. The first 20 frames of each column correspond to routing configuration memory M_R.

All frames have a fixed length of 1312 bits or 41 32-bit words. Frames in the bottom half of the device are bit reversed to frames in the top haft with exception of the middle word of the frame.

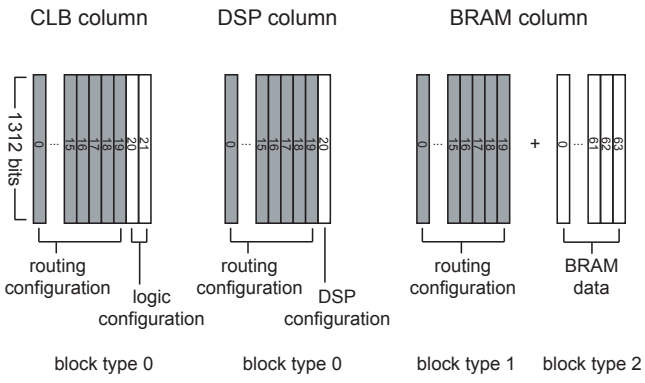

Figure 7. Configuration memory of CLB, DSP and BRAM resources.

During the reconfiguration process, frame addresses can be auto-incremented. A bitstream usually contains a start

address and frames are sequentially written throughout all frames, columns, rows, top/bottom and finally block types.

5 Bitstream Relocation in Virtex-4

In this section we describe how our method of configuration relocation can be applied to Virtex-4 devices. Because of its columnar structure, we have to consider vertical and horizontal relocation separately. A vertical relocation from region 1 to 3 in figure 5 provides identical resources on the functional layer. Thus, a circuit can be placed in both regions without restrictions. However, in our model we assume that the mapping of resources to configuration memory is uniform for all tiles on the device. This condition is violated for relocation between top-half and bottom-half because of the reversed order of bits in each frame. The solution to this problem is simply to bit-reverse the entire frame. The centre word is not reversed because it has the same order of bits in both halves. Vertical relocation has the advantage of providing identical resources. On the other hand relocatability is limited to steps of 16 CLBs.

For horizontal relocation we have to consider the heterogeneity of the device. Depending on the size of the module and other floorplanning constraints it can be impossible to find completely identical regions. Regions 1 and 2 in figure 5 exhibit a mismatch between a DSP column in region 1 and a BRAM column in region 2. However, both columns provide the same routing resources to their neighbouring CLB columns. These routing resources are configured by the same 20 frames of configuration data. Therefore the configuration can be relocated by copying the first 20 frames of the DSP column into the memory location of the according BRAM interconnect columns. The last frame of DSP configuration data is discarded. All other columns can be copied by address offsetting. For the reverse transformation from region 2 to 1, 20 frames of BRAM interconnect data are copied into the according memory location of the DSP column. The last frame number 21 can be omitted or filled with a pad frame if address auto-increment is used. The same operations apply to transformations between CLB and DSP and between CLB and BRAM. In any case the first 20 frames of routing configuration are copied; DSP or CLB logic configuration frames can be filled with one or two pad frames. In comparison to vertical relocation, horizontal relocation allows much finer placement steps of 1 CLB. However it is preferable to choose a placement that reduces the number of non-matching columns.

In order to enable the horizontal relocation between two or more not fully identical regions, all mismatching resources have to be prohibited from being used by the implementation tools. The Xilinx placement and rooting tool PAR can be directed by PROHIBIT constrains to not place logic instances into certain resources. However, the router is still allowed to use the according routing resources. If in our current example a module is physically implemented in region 1 then PROHIBIT constrains have to be applied to all DSPs in this region. The resulting module can be used in all four regions. For relocation to region 4 both frame reversal and the DSP to BRAM frame transformation are applied. It should be noted that we do not need to prohibit all heterogeneous resources in relocatable modules. In our example a module can make use of the right BRAM column if required. Only non-matching resources are prohibited.

6 Software Implementation

This section presents a software implementation of our method for FPGA self-reconfiguration with an internal processor. However, self-reconfiguration is not strictly necessary and our software solution could be adapted to run on an external processor. Instead of generating pre-parsed bitstreams which lack configuration data for mismatching resources as indicated in figure 4 we chose to transform regular partial bitstreams produced by the implementation tools. A bitstream is parsed and modified at run time according to the requirements of the target region.

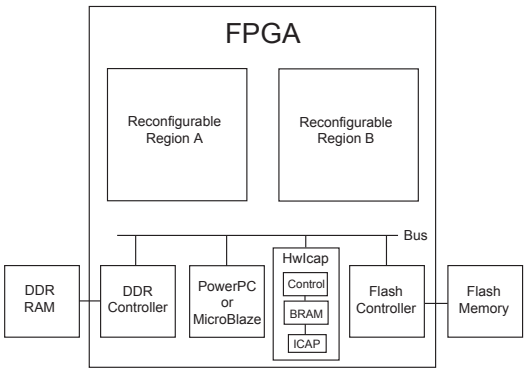

Figure 8. Example of a self-reconfigurable system with HwIcap.

We have implemented our technique as a software driver for the OPB HwIcap core [17]. HwIcap is an IP core provided by Xilinx as part of the Embedded Development Kit (EDK). The core contains the ICAP primitive and is designed as an OPB peripheral. This supports the fast and easy creation of a processor based system for FPGA self-reconfiguration. Figure 8 illustrates the structure of such a system. The HwIcap core contains one BlockRAM and additional control logic. During a reconfiguration process the BRAM is filled with configuration data which is subsequently streamed into the configuration port. A low-level

device driver enables basic data transfers to the core and a high-level driver provides a function setConfiguration() to load a partial bitstream. Our device driver builds on the low-level driver and provides a function loadModule() which allows to load any partial bitstream with an X,Y CLB offset from its original location. The vertical offset has to be a multiple of 16 CLBs since this is the basic vertical unit of configuration. The horizontal offset can be any whole number of CLBs.

The function performs three tasks to relocate the bitstream:

1. Address transformation: the memory addresses for configuration data have to be calculated based on a location offset applied to the bitstream.

2. Bit reversal: if configuration frames are relocated into the other half of the FPGA the order of bit has to be reversed.

3. Column conversion: if a column is relocated to a non-identical location the routing frames are copied and pad frames are added if necessary.

Our driver function first calculates the original location based on the address given in the bitstream. The bitstream is then parsed into blocks of configuration data which correspond to one column of resources, e.g. one column of CLB, DSP or BRAM configuration data. As a next step the new target address is determined. The new column address can not be calculated based on simple equations because configuration data can be exchanged between block type 0 and block type 1 in case of a resource mismatch involving BRAM. BRAMs follow a different address scheme than CLBs, DSPs, IOs and clocks and it is not possible to correlate both address spaces without further device information. Therefore we read the bitstream IDCODE to determine the device type. Our function contains a look-up table with locations of heterogeneous resources in Virtex-4 devices. We use this information to transform both address spaces into a unified linear address space which enables calculation of the target address and checking for mismatching resources.

The next step is to relocate the configuration data. For a horizontal relocation the function checks if the target location provides an identical resource. If the target column provides the same resource the configuration data block is written out without modification. In case of a resource mismatch the function copies the first 20 frames. CLB target columns are filled with two pad frames and DSP target columns with one pad frame. In the case of vertical relocation, the function checks if the target region is located in the opposite half of the FPGA. Is this case the frame is bit reversed. For both horizontal and vertical relocation both steps are executed sequentially.

Each block of configuration data is combined with the new address and directly written to the configuration memory before the next block is parsed. This keeps the memory requirements during the bitstream parsing low, because data do not have to be copied between block type 0 and block type 1 within the bitstream.

Figure 9 illustrates an example for a horizontal relocation that contains one non-identical column. The original partial bitstream generated by the implementation tools contains a start sequence, start address, configuration data and end sequence. For relocation the bitstream is parsed into columns which are written out individually. The function DSP2BRAM copies the first 20 frames and discards the last one. Each column is transferred with a start and end sequence to avoid synchronisation loss with ICAP between transfers. The average increase of overall data volume is only 4% compared to the original bitstream. However, an improved transfer process might not require start and end sequences for each individual column. In this case the average data volume would be equal to the original bitstream.

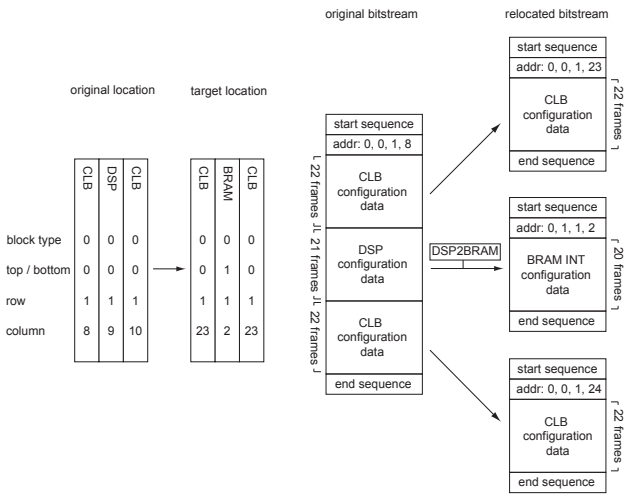

Figure 9. Example of a bitstream relocation from DSP to BRAM.

7 Results

In this section we provide one implementation of a reconfigurable application that employs bitstream relocation. We also analyse how our method enhances the relocatability of bitstreams.

7.1 Software defined radio prototype

To prototype a relocatable software defined radio application, we developed a proof-of-concept design with relocatable audio filters. Software defined radio is an interesting application for run-time reconfiguration because the software waveforms can be accelerated by reconfigurable hardware modules, thus providing the same flexibility as a pure software implementation with hardware performance. Reconfigurable waveform modules are relatively large and therefore difficult to relocate.

The design was implemented on a Virtex-4 LX25 FPGA using the ML401 board. The control system consists of a MicroBlaze processor running at 100 MHz with 64 kB instruction memory, HwIcap and UART core as well as external Flash and DDR memory. The system is controlled via UART by a terminal program. The design has two reconfigurable regions which can host filters for the left and right audio channel. In this example relocatability reduces the number of partial bitstreams by 50% since one implementation of a filter can be used in both regions. Figure 10 shows the placed and routed design with the high pass filter present in the left region. The right region is empty. Four different filter types are available: low pass, band pass, high pass and all pass. Our method enables the relocation to the non-identical right region therefore reducing the number of partial bitstreams from 8 to 4. The compile time for the entire design was reduced from 54 min to 31 min on a PC with a 3.2 GHz Pentium 4 CPU and 1 GB of RAM. This corresponds to a reduction of 43%.

If a filter is configured into the left region the unmodified bitstream is loaded with the setConfiguration() function of the original device driver. In order to load a filter to the right region the bitstream is relocated with the loadModule() function of our device driver. The function transforms the bitstream accordingly. This design could prove that a bitstream designed for the left region can be successfully loaded into a non-identical region using bitstream transformations. The performance is measured using a bus timer module. The partial bitstreams have a size of 127 kB and can be loaded to the original location in 25.4 ms. This is equivalent to a transfer rate of 5 MB/s. Loading the bitstream to the right location with our device driver takes 27 ms which represents a transfer rate of 4.7 MB/s. This represents performance penalty of 6% which can be explained with the increased data volume. Overall the computational requirements are very low. These results could vary with system frequency, system load or when using the PowerPC. The overall relatively low transfer speed can be explained by the fact that the OPB bus does not support burst transfers.

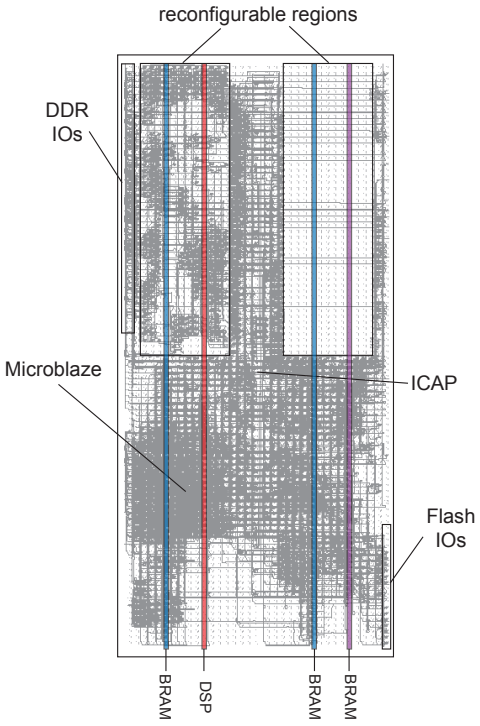

Figure 10. Reconfigurable and relocatable software defined radio prototype: filter for two audio channnels on a Virtex-4 LX25 FPGA. A filter is implemented only in the left region and one bitstream is stored. The configuration can be relocated dynamically.

7.2 Analysis of relocatability

In this section we analyse how the relocatability is enhanced by our method. The analysis is performed on a Virtex-4 FX140 FPGA. We chose this device because it is the largest in the Virtex-4 FX series and therefore a good potential candidate for the implementation of a complex reconfigurable software defined radio system. The FPGA has height of 192 CLBs and a width of 84 CLBs. A total of 12 BRAM, 2 DSP and 3 IO columns interrupt the CLB fabric. These resources are arranged in a pattern where four CLB columns are interleaved with one heterogeneous column. The device also contains two PowerPC processors which are excluded from our method. Although it might be possible to identify a compatible subset between regions with PowerPCs we consider this as impractical. However, rows above or below the PowerPC core can be used without restriction.

To put module sizes in perspective, we try to obtain a size estimate of potential reconfigurable modules. The au-

dio filters from our previous example require about 1000 logic slices which is equivalent to 250 CLBs. Thus, a filter could be constrained to 16x16 CLBs. The radio waveforms in [11] are approximately four times larger and could cover an area of 32x32 or 16x64 CLBs. We therefore consider a width of 16 CLBs as slim reference module, and a width of 32 CLBs as wide reference module. These numbers correspond to a relative width of 19% and 38% of the device.

In the following analysis we consider the horizontal relocation of modules with variable width, and measure the number of alternative placement options. The width of modules measured relative to the width of the device. Figure 11 illustrates the relocatability for three different scenarios. Traces in the diagrams correspond to regions with a fixed left boundary and a growing width. The upper diagram illustrates alternative placements without our method; the target region has to be fully identical. Small modules with a width up to 10% have plenty of placement options. Modules with a width between 10% and 15% can have up to five further placement options and all other modules with a width up 30% have at most one further placement option. All modules wider than 30% cannot be relocated. For the slim reference module, there is only one fully identical relocation target. However, regions identified as valid options do not necessarily have to be feasible in a real floorplan. The placement can be too close to a PowerPC, or can cause undesirable fragmentation of the device. The wide reference module cannot be relocated.

With our method it is possible in principle to relocate a module to any other region. Since all non-identical columns can only be used for routing, it is preferable to maximise the percentage of matching resources. The middle diagram illustrates alternative placement options if we require at least 90% identical resources in the target region. Overall, the relocatability is significantly increased and there are 11 alternative placements for the slim module and two for the wide module. Modules with a width up to 46% can be relocated. If we expect at least 80% matching resources modules with a width up to 99% can be relocated all illustrated in the lower diagram. The slim module can have 34 alternative placements and the wide module 11. For our reference modules a parameter of 90% match seems to be appropriate. It has to be noted that non-matching resources might not be needed by the application. For example a module solely implemented in CLB logic uses BRAM columns only for routing. If this module is relocated over a DSP column no additional resources are wasted by our method.

Future FPGA architectures can be optimised for bitstream relocation without having to relinquish heterogeneity. If specialised columns are arranged in a regular interleaved pattern similar to Virtex-4 FX devices, relocation is significantly simplified. An FPGA designer can try to create a high level of symmetry in the device. Potentially columns can be rearranged without affecting the overall performance. This could provide enough identical regions from smaller modules. Larger modules might be hindered by one or two non-identical columns. Our technique will also enable the relocation of those modules.

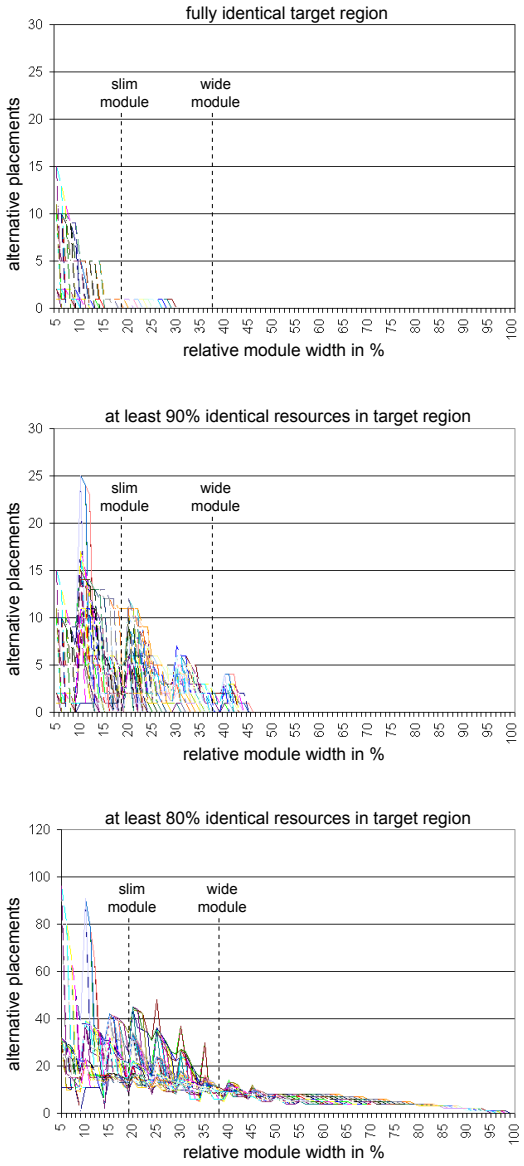

Figure 11. Alternative placement options for software defined radio application on an Virtex-4 FX140 FPGA for different relative module widths and for fully identical, at least 90% identical and at least 80% identical target regions.

8 Summary

We have introduced a model of bitstream relocation based on a compatible subset of resources. Our method solves the problem of growing heterogeneity by not requiring fully identical regions for bitstream relocation. Our method can significantly improve flexibility when floorplanning smaller relocatable modules, and serves as an enabling technique for the relocation of larger modules. Based on Virtex-4 FPGAs we have shown the feasibility of our method and have provided a concrete implementation as a software driver for the HwIcap core. In a software defined radio prototype with two reconfigurable regions the number of partial bitstream is reduced by a factor of 2 and the design time shortened by 43%.

Current and future work includes the adaptation of our driver to the latest devices such as Virtex-5. Another aspect is the inclusion of IO columns which are currently not supported. We further plan to develop various applications to illustrate the benefit of our approach, such as a software defined radio demonstrator that shows relocation of waveforms in multiple non-identical target regions.

Acknowledgements

The support of Xilinx is gratefully acknowledged. The authors would in particular like to thank Adam Donlin, Brandon Blodget and Patrick Lysaght for their contributions.

References

[1] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan. A self-reconfiguring platform. In *Field-Programmable Logic and Applications*, LNCS 2778, pages 565–574. Springer, 2003.

[2] G. Brebner. A virtual hardware operating system for the Xilinx XC6200. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers (FPL'96)*, pages 327–336. Springer-Verlag, 1996.

[3] E. Horta and J. W. Lockwood. PARBIT: A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs). Technical Report WUCS-01-13, Washington University, Department of Computer Science, 2001.

[4] M. Hübner, T. Becker, and J. Becker. Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration. In *Proceedings of the 17th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 28–32. ACM Press, 2004.

[5] H. Kalte, G. Lee, M. Porrmann, and U. Rückert. REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems. In *19th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2005.

[6] H. Kalte and M. Porrmann. REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 403–412. ACM Press, 2006.

[7] J. Liang, R. Tessier, and D. Goeckel. A dynamically-reconfigurable, power-efficient turbo decoder. In *Field-Programmable Custom Computing Machines*. IEEE Computer Society Press, 2004.

[8] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Enhanced architectures, design methodologies and CAD tools for dynamic reconfigurtion on Xilinx FPGAs. In *Field Programmable Logic and Applications*, pages 12–17. IEEE, 2006.

[9] J. Mignolet, V. Nollet, P. Coene, D. Verkest, and V. Lauwreins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 986–991. IEEE Computer Society, 2003.

[10] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk. A reconfigurable platform for real-time embedded video image processing. In *Field-Programmable Logic and Applications*, LNCS 2778, pages 606 – 615. Springer, 2003.

[11] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght. Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proceedings Computers and Digital Techniques*, 153(3):157–164, 2006.

[12] H. Styles and W. Luk. Compilation and management of phase-optimized reconfigurable systems. In *Field-Programmable Logic and Applications*, pages 311–316. IEEE, 2005.

[13] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson. A reconfigurable, power-efficient adaptive Viterbi decoder. *IEEE Transactions on VLSI Systems*, 13(4):484–488, 2005.

[14] M. Wirthlin and B. Hutchings. Improving functional density using run-time circuit reconfiguration. *IEEE Transactions on VLSI Systems*, 6(2):247–256, 1998.

[15] Xilinx. *MicroBlaze Microcontroller Reference Design User Guide v1.5*, September 2005.

[16] Xilinx. *PowerPC 405 Processor Block Reference Guide*, July 2005.

[17] Xilinx Inc. *Xilinx Logic Core: OPB HWICAP v1.3*, March 2004.

[18] Xilinx Inc. *Virtex-5 Configuration Guide v2.1*, October 2006.

[19] Xilinx Inc. *Virtex-4 Configuration Guide v1.5*, January 2007.

[20] Xilinx Inc. *Virtex-5 Family Platfrom Overview LX and LXT Platforms v2.2*, January 2007.