Modular dynamic reconfiguration in Virtex FPGAs

P. Sedcole, B. Blodget, T. Becker, J. Anderson and P. Lysaght

Abstract: Modular systems implemented on field-programmable gate arrays (FPGAs) can benefit from being able to load and unload modules at run-time, a concept that is of much interest in the research community. Although dynamic partial reconfiguration is possible in Virtex and Spartan series FPGAs, the configuration architecture of these devices is not amenable to modular reconfiguration, a limitation which has relegated research to theoretical or compromised resource allocation models. Two methods for implementing modular reconfiguration in Virtex FPGAs are compared and contrasted. The first method offers simplicity and fast reconfiguration times, but limits the geometry and connectivity of the system. The second method, developed recently, enables modules to be allocated arbitrary areas of the FPGA, bridging the gap between theory and reality and unlocking the latent potential of dynamic reconfiguration. The cost of this advancement is increased reconfiguration time. The second method has been demonstrated in three applications, including the first reported implementation of modular reconfiguration in a Virtex-4 device.

1 Introduction

The transistor density of field-programmable gate arrays (FPGAs) has reached a level where an entire system may be implemented within a single device. A complex system is generally composed from many functionally discrete modules that are connected to form a coherent whole. In some cases, where the requirements on the system are time-variant, not all modules need to operate concurrently. An unused module resident in the FPGA will waste power, area and cost, and therefore it would be advantageous if modules are able to be loaded only when an application is invoked and removed again once the application has terminated.

There has been a large amount of research in the area of dynamic modular systems in FPGAs [1–5]. These are predicated on the property of dynamic reconfiguration, where parts of the user logic inside the FPGA are replaced while other active circuits operate uninterrupted. This technique becomes more useful as transistor densities increase such that complete systems are implemented within a single FPGA. Without dynamic reconfiguration, the entire system would need to be disrupted every time a change is needed. Moreover, with configuration bitstreams for the largest devices now exceeding 45 Mbits [6] (Fig. 1), there is increasing impetus for composing systems as needed from a few small partial bitstreams rather than storing many large complete bitstreams.

However, module-based reconfiguration, where the logic undergoing replacement occupies an arbitrary region of the FPGA, has not been intrinsically supported in FPGAs since the demise of the Xilinx 6200 series. Although the Virtex and Spartan series of FPGAs are dynamically reconfigurable, the essentially linear organisation of the configuration memory is not amenable to the implementation of module-based systems with 2D floor-plans. As a result, research has tended to be either theoretical, or severely circumscribed, typically by reducing the resource model to 1D.

In this paper, two methods for implementing modular partial reconfiguration on Virtex FPGAs are compared. In the first method, applicable to Virtex, Virtex-II and Virtex-II Pro devices, modules must occupy the full height of the device and the topology and connectivity are limited to 1D. This we term ‘direct dynamic reconfiguration’: it is fast and simple, and has been previously documented by Lim and Peattie [7]. The second method, recently developed by the authors, demonstrates how 2D modular systems can be made tractable through the use of an innovative bitstream merging process and reserved routing. This enables modules to be assigned arbitrary rectangular regions of the FPGA and relocated at run-time, bridging the gap between theory and reality. Moreover, it is possible to achieve much greater flexibility in the connectivity of the system. The costs of these advancements are increased complexity and reconfiguration time.

The novel second method, termed ‘merge dynamic reconfiguration’, has similarities to the PARBIT tool and design methodology developed by Horta et al. [8] for the FPX platform. PARBIT operates on bitstreams, inserting modules into a target area of a device, and even re-targeting the bitstream for a different sized device [9]. The work presented in this paper differs most significantly in the following ways: (1) static routing is possible in reconfigurable

IEE Proceedings online no. 20050176

doi:10.1049/ip-cdt:20050176

Paper first received 1st November 2005 and in revised form 22nd February 2006

P. Sedcole is with the Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, UK

B. Blodget, J. Anderson and P. Lysaght are with the Xilinx Research Labs, Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124, USA

T. Becker is with the Department of Computing, Imperial College London, SW7 2AZ, UK

E-mail: pete.sedcole@imperial.ac.uk

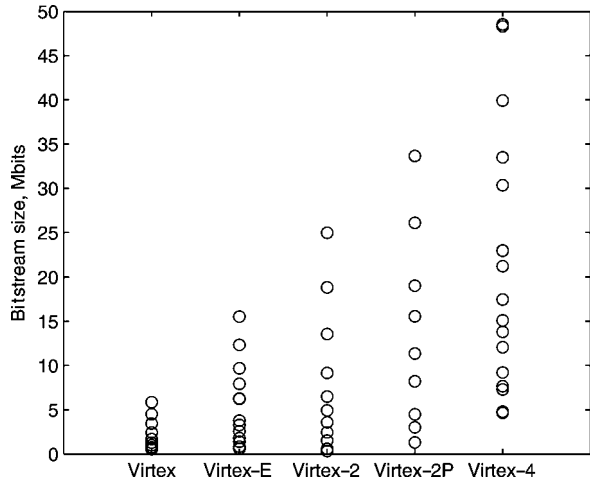

Fig. 1 *Bitstream sizes for Virtex FPGAs*

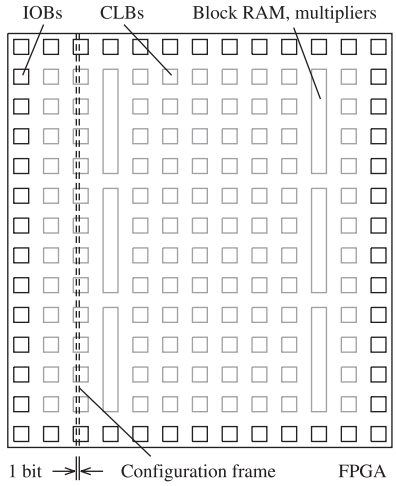

Fig. 2 *Virtex-II configuration architecture*

regions, (2) bitstreams are integrated at run-time, (3) the target bitstream is read from configuration memory before the integration operation, which enables (4) more sophisticated integration operations to be used.

Another tool widely used in dynamic reconfiguration research is JBits [10]. As with merge reconfiguration, JBits uses bitstream information read from the device at run-time, and has been used for logic relocation [11]. Significantly, the JBits interface is low-level and architecture-dependent, and it does not integrate easily with high-level design flows. This makes JBits more suitable for fine bitstream manipulations than module-level reconfiguration and relocation. Addressing this problem by combining JBits with a high-level HDL is the subject of recent unfinished research [12].

The work reported here was originally published in the work of Sedcole *et al.* [13]. This expanded paper provides further details, and describes the application of the merge dynamic reconfiguration method to the latest generation of Xilinx FPGAs, the Virtex-4.

2 Virtex configuration architecture

The configuration architecture of the Virtex family of FPGAs is described in a Xilinx Application Note [14], and is essentially the same for Virtex-II [15] and Virtex-II Pro [16] devices. The configuration is stored in SRAM memory that can be read from or written to without halting the device. The smallest unit of configuration memory that can be read or written is a 'frame', which spans the entire height of the device (including I/O blocks) and a fraction of one column (Fig. 2).

It should be noted that Virtex-II/Pro FPGAs have the characteristic of 'glitchless dynamic reconfiguration': if a configuration bit holds the same value before and after configuration, the resource controlled by that bit will not experience any discontinuity in operation [17], with the exception of LUT RAM and SRL16 primitives. It is therefore possible for a reconfigurable module to occupy an arbitrary area, provided that (1) the areas above and below the module area do not contain LUT RAM or SRL16 logic and (2) the configuration data written to these areas when the module is replaced overwrites the existing configuration with exactly the same values. Similarly, static, system-level routing may pass through a reconfigurable region if its configuration data are persistent when the module is reconfigured.

The latest generation of Virtex FPGAs, the Virtex-4 family, marks a significant change in layout over previous devices. As shown in Fig. 3, the configuration architecture is still frame-based, but a frame spans 16 rows of configurable logic blocks (CLBs) rather than the full device height [6]. Clock distribution regions are also aligned in blocks of 16 CLB rows, unlike earlier Virtex devices, where clock regions were defined to be quadrants. Note that I/O blocks are arranged in columns (like all other resources) rather than a ring. The Virtex-4 shares the glitchless dynamic reconfiguration property of earlier devices, but this now applies to all primitives including LUT RAM and SRL16 logic.

3 Direct dynamic reconfiguration

In the direct dynamic reconfiguration process, reconfigurable modules are composed from complete frames of configuration memory. This implies that a module occupies the full height of the device, including the I/O at the top and bottom of the reconfiguration region (Fig. 4). The module may be a variable number of CLB columns in width, and all logic and routing within the reconfiguration region are dedicated to the module. Using this scheme, a module may be replaced very simply by writing over the

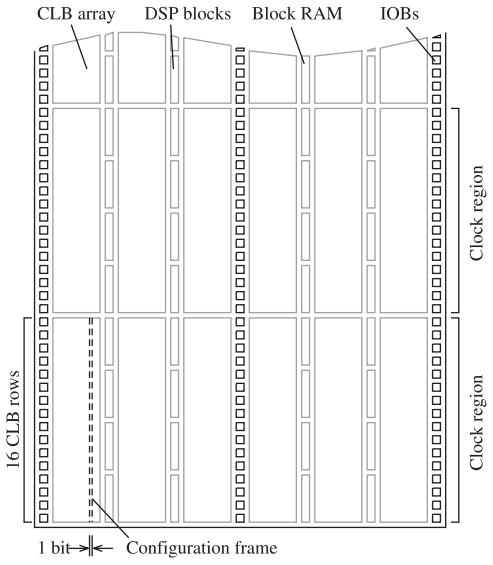

Fig. 3 *Virtex-4 configuration architecture*

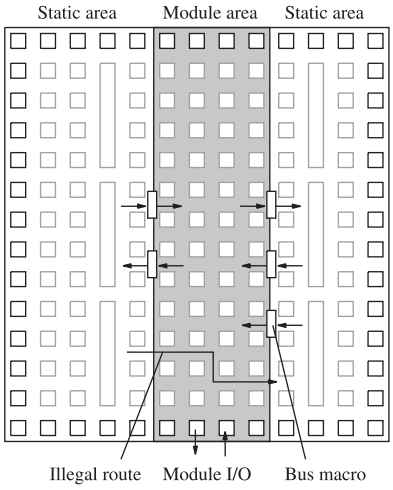

Fig. 4 *Direct reconfiguration*

existing configuration for the frames that coincide with the module area, using a partial bitstream. 'Bus macros' are predefined units of logic and wiring that ensure the locations at which signals pass between the module and the rest of the system are preserved from module to module. More information on this method can be found in the work of Lim and Peattie [7].

There are a number of limitations with this approach. The design flow, which creates the bitstreams for the static portion of the design and the modules, leverages the Xilinx's Modular Design™ methodology, which restricts the position of the modules in the logical hierarchy of the design to the top level. Driver contentions can occur if one module configuration is written over immediately with another, which must be avoided by replacing the module with the default empty configuration before loading in the next module. For large devices, a full height module is not the most efficient use of resources, and timing closure can become problematic for modules with large aspect ratios. Finally, although it is possible to pass signals across the reconfigurable area via bus macros on either side, it is highly probable that the signal path will be re-routed during reconfiguration, making the signal non-valid at this time. This means that while a module is undergoing reconfiguration, the parts of the system on either side of the module are isolated from each other.

Although a port has not yet been performed, there is no practical obstacle to the use of direct reconfiguration on Virtex-4 devices. The reduced configuration frame size in the Virtex-4 should ameliorate a few of the issues described earlier, as modules would be 16 CLB rows in height (or a multiple of 16) rather than the full device size. This means resources can be allocated more effectively, and static areas are not necessarily isolated from each other.

4 Merge dynamic reconfiguration

The merge dynamic reconfiguration method was created in order to circumvent the limitations of direct reconfiguration, and exploits the glitchless reconfiguration property of Virtex FPGAs. As noted in Section 2, a statically routed signal can pass through a reconfigured region unperturbed provided the configuration bits associated with the route persist in the new configuration. However, as the module designs are placed and routed independently from the static part of the design, the resources allocated to a static route could also be used in one or more module implementations. This is avoided through the use of reserved routing: within a module region, certain routing resources are always reserved for static routing and modules must avoid using any of these resources, even if unused by the static design. For example, in the Virtex routing architecture, each horizontal and vertical channel has 24 long lines and 120 hex lines as well as other more local routing resources. Routing within a module uses short wires, whereas static signals that pass through the module are best routed on long wires. Therefore we chose to allocate 100% of the long lines and 20% of the hex lines within module regions to statically routed signals. This choice is arbitrary and could be adjusted from application to application.

Reserving routing provides a high degree of separability between static and module designs; re-implementing placement and routing for the static design does not mean module designs must also be re-implemented. The cost incurred is a reduction in the freedom of the router, and therefore potentially lower quality routing. As yet, this has not been quantified.

The production router in the ISE `par` tool has the ability to follow very specific constraints, but unfortunately there is no way to provide `par` with these constraints in the current tool flow. Therefore a post-`par` re-routing step is performed on both the static and module designs. It is possible but laborious to do this by hand. This step was automated with the use of a custom tool that can generate routing constraints on a tile-by-tile basis and give this information to the production router.

Statically routed signals in module regions are important in Virtex-II/Pro devices to avoid the isolation of static regions. In the Virtex-4, isolation can be avoided by routing static signals around module regions; nevertheless, the reserved static routing technique is still useful. Routing congestion and delay are reduced by routing through module regions, and module regions can be contiguous.

The second major innovation in merge reconfiguration is in the way the partial bitstream is loaded. Rather than writing the bitstream directly to the configuration memory, the current configuration is read back from the device and modified with information from the partial bitstream before being written back. This is performed on a frame-by-frame basis, which minimises the amount of memory required to store the bitstreams. A module may occupy less than the full height of a frame, by only modifying tiles which fall within a given boundary region. As a result, it is possible to have two or more module regions vertically aligned within the same frame-space. Clearly, this is of particular use in Virtex-II/Pro devices. However, in all devices in the Virtex family, this technique allows module regions to be shaped and positioned arbitrarily.

Within the module region, static routing is preserved by using an exclusive OR (XOR) operator to merge the partial bitstream with the current configuration. The XOR merge technique has a number of advantages, applicable in all Virtex devices:

- Although it is still necessary to remove the module before loading a new one, the same operation and bitstream can be used for loading and unloading, as repeating the XOR operation returns the value to the original state ($a \oplus b \oplus b = a$). This reduces the amount of storage required, as a default empty bitstream is no longer needed.
- As the module includes no information on static routing, it is position-independent. This is significant, because it means

modules can be relocatable. To verify this, relocation has been demonstrated in an example application (Section 5.2).
• A module can be loaded in several stages by separating information into several bitstreams that are then effectively overlayed on one another. An example where this ability is useful is given in Section 5.3.

As Fig. 5 shows, the static and module bitstreams are created in separate parallel design flows. Some information, such as clock trees and bus macros, is common between module designs and the static design. Within the bitstreams, common information can be easily identified as a data bit that is set to one in both the module and static bitstreams. If information is present in both the original configuration and the partial bitstream, it will be removed by the XOR operation (as $1 \oplus 1 = 0$). In order to prevent this, redundancies are removed from the module bitstream in a simple post-processing step: for each frame in the module bitstream, a bitwise AND NOT operation is performed on the module configuration data, using data from the corresponding frame in the static bitstream. This processing is computationally simple and has been fully automated.

5 Experiments and applications

The novel merge reconfiguration method has been applied in three applications, one of which (Sonic-on-a-Chip [18]) has been implemented on both Virtex-II Pro and Virtex-4 platforms. The applications are described in this section. All scenarios employ the self-reconfiguring platform reported by Blodget *et al.* [19] (Fig. 6). It should be noted that although this was a convenient framework for development, particularly as bitstream manipulation functions can be easily added by extension of the existing driver, self-reconfiguration is not a necessity and the functionality could be provided by an external embedded processor or even by a PC.

5.1 Software defined radio

In a collaboration between Xilinx, Inc. and ISR Technologies, a demonstrator of a software defined radio

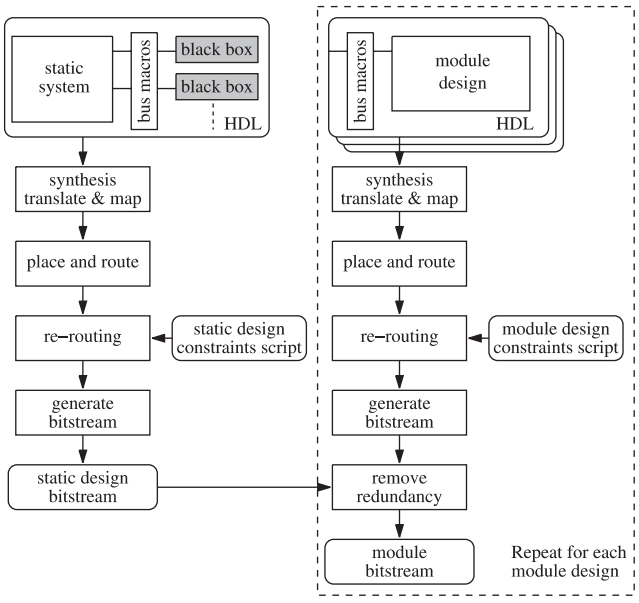

Fig. 5 *Design flow for merge dynamic reconfiguration*

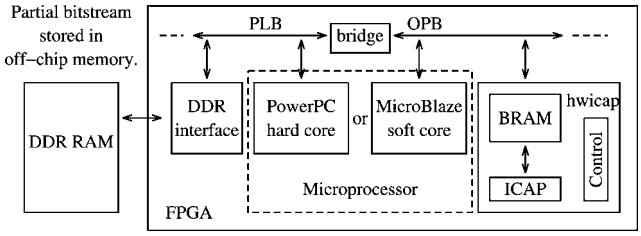

Fig. 6 *Self-reconfiguring platform*

(SDR) has been developed. Although part of the radio is software-based, the modulation and demodulation is performed by hardware modules (peripherals on the processor bus) that are loaded using dynamic reconfiguration.

Intended as a proof-of-concept, the demonstrator was developed with a predecessor of the merge reconfiguration method which allows for a single module design only per reconfiguration region. In addition, static configuration information is incorporated into the module bitstreams at design-time, rather than at run-time. Nevertheless, the SDR demonstrates the advantages obtained by exploiting the property of glitchless reconfiguration: the modules are less than the full height of the device (Fig. 7) and there are hundreds of statically routed signals that pass through the reconfigurable regions. Note that slice-based macros (similar to those in the work of Huebner *et al.* [20]) were developed to enable greater signal densities at module interface points.

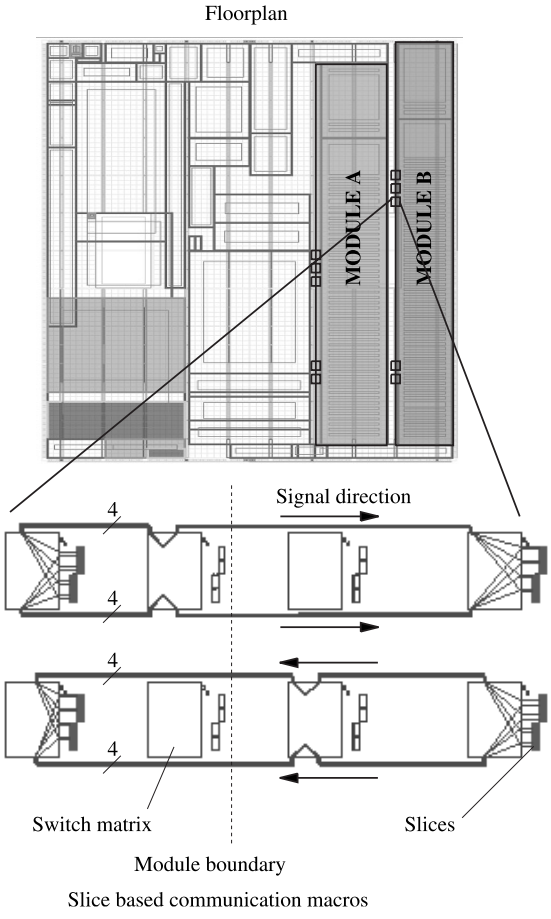

Fig. 7 *Floorplan for the SDR demonstrator and the new communication macros*

5.2 Microprocessor peripheral

The second experiment was a test framework created to assist the development of the merge dynamic reconfiguration method, and was used in particular to demonstrate the re-targeting of a module bitstream. The test setup used the Xilinx ML300 development board based on a XC2VP7 part. The FPGA has 34 CLB columns, with one PowerPC processor embedded towards the right-hand-side of the device in columns 20–27. Owing to the layout of the board, the external DDR-RAM memory is connected to I/O blocks on the left-hand-side of the FPGA. Space was allocated in columns 3–10 for two reconfigurable modules placed one above the other. The signals from the processor subsystem to the DDR-RAM are necessarily routed through the area for the reconfigurable modules, and must persist during reconfiguration, as the program code and module partial bitstreams are stored in the external memory. Two very simple bus peripheral modules (a single register, and a one's complimenting register) were designed, which attached directly to the on-chip processor local bus (PLB). The slice-based bus macros from the SDR demonstrator were reused for this design.

Following the procedure for merge dynamic reconfiguration, bitstreams for the static microprocessor subsystem and the two modules were created in three separate implementation phases. The two module designs targeted the lower of the two reconfiguration regions. The modules were successfully loaded into and unloaded from the lower target location.

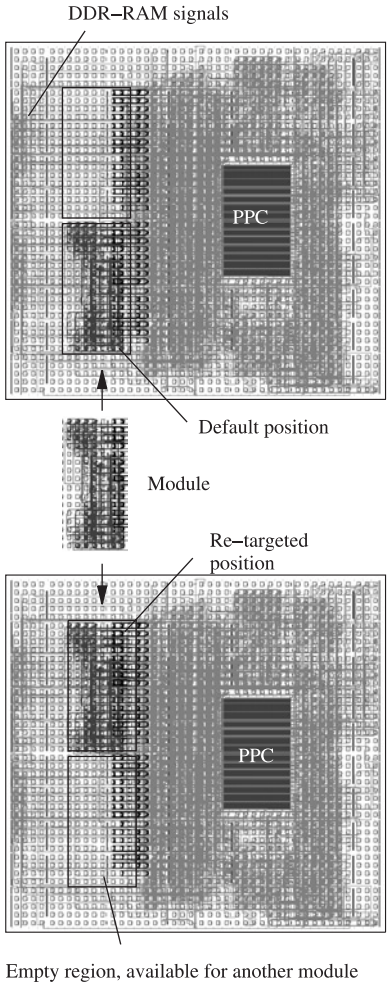

Fig. 8 *Illustration of modules loaded in default and re-targeted positions*

As an extension to this application, module bitstreams were re-targeted to the top location at run-time, as illustrated in Fig. 8. This was achieved by shifting all the frame bits of the module partial bitstreams by 16 CLB rows (corresponding to 1280 bits or 160 bytes). Relocation is possible because the bus macro placements are identical between the upper and lower target locations, and the arrangement of resources in the new location is identical to the original target area. In general, it may be noted that the reconfigurable fabric in Virtex devices is translationally symmetric; the fabric is formed by a repeated periodic spatial pattern. In the vertical direction, the periodicity of this pattern is one row for CLB resources and four rows for Block RAM and embedded multiplier or digital signal processing (DSP) blocks. Therefore using a vertical shift to re-targeted modules requires the modules to be aligned on four-row boundaries.

The exception to the translational symmetry is found in the clock tree; configuration bits controlling clock-tree branching are not associated with logic tiles but grouped at the ends of each frame. Therefore these bits are treated specially; they are excluded from the shift process, and within each frame, all branches of a clock tree are enabled if one branch has a bit set.

5.3 Sonic-on-a-Chip

In the previous two examples, modules are connected directly to the PLB bus; this is not the only, nor necessarily the most effective, connectivity model. The final study involved the application of merge reconfiguration to two Sonic-on-a-Chip prototypes. Sonic-on-a-Chip, an architecture for reconfigurable video image processing systems, uses a custom bus structure and protocol, which are designed to be an effective and efficient solution specifically for dataflow applications. The first of the prototypes was implemented on a Virtex-II Pro part, using the ML300 development board, whereas the second has been developed for the Virtex-4-based ML401 board.

The Virtex-II Pro implementation used specially designed tristate buffer macros to lock bus routing to specific wires. Newer FPGAs, such as the Virtex-4, do not incorporate tristate buffers, therefore a logic-OR bus structure was developed to replace the tristate buffers. In both cases, it was required that the bus operation continue uninterrupted during reconfiguration, implying the tristate buffers (or equivalent) be disabled. This created an interesting problem, as signals sourced from inside the module are indeterminate during reconfiguration, including the disable signals.

The chosen solution to this involved a multiple-phase module reconfiguration process. The static and module designs were implemented as per Fig. 5. For each module implementation, a second design was created by copying the original implementation and isolating the module from the bus wires by disconnecting the bus drivers. Two partial bitstreams were generated, the first containing the configuration for the disconnected module, and the second comprising the difference between the two designs. By removing all redundancies from the second bitstream, it carries just the information required to connect the module to the bus lines. The module could thus be loaded with two successive merge operations, as depicted in Fig. 9. Removing the module is done by repeating these operations in reverse.

Note that a reset signal is generated locally to hold the module in a known state once the reconfiguration is complete. A third phase dynamic reconfiguration step was

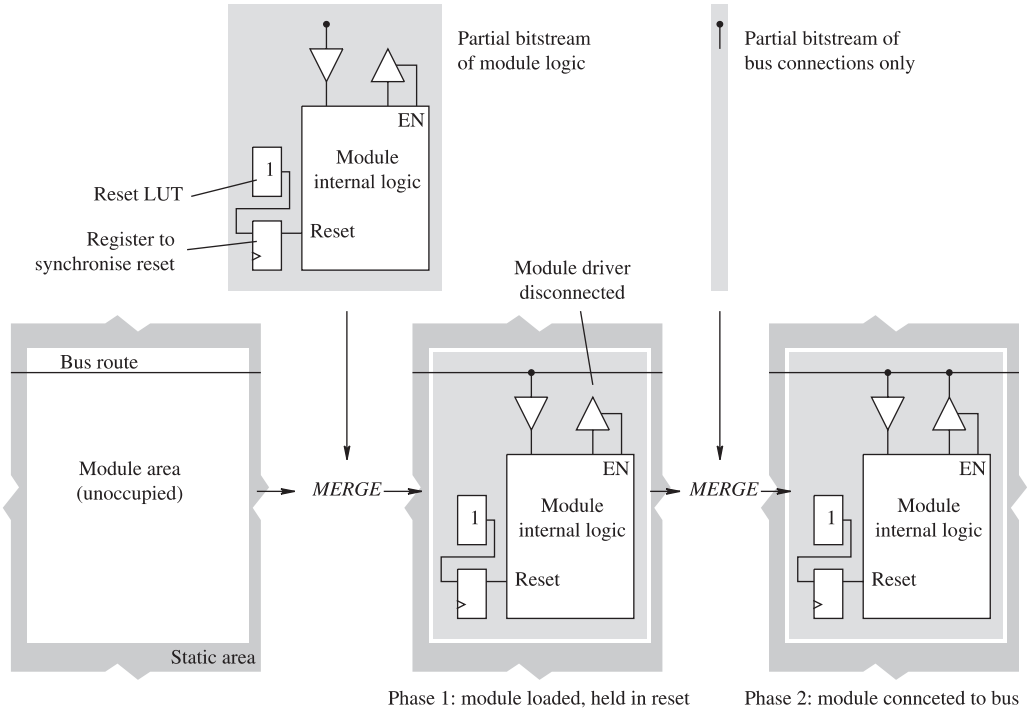

Fig. 9 *Module loaded in multiple phases*

used to deassert the reset signal when the module state has stabilised, although this is not shown in the figure.

Fig. 10 illustrates the placed and routed static design in the Virtex-4 XC4VLX25 device. Two module regions are defined, which are aligned with the clock regions in order to use the regional clock buffers [21] for locally generated clocks. As the figure shows, some bus macros used in this implementation are located at the module boundaries. However, using reserved routing enables bus macros to be embedded within the module region. This is achieved by designating all routing as static in the CLB tile, where static signals connect to the macro, as shown in Fig. 11. Embedded macros increase connectivity and floorplanning choices.

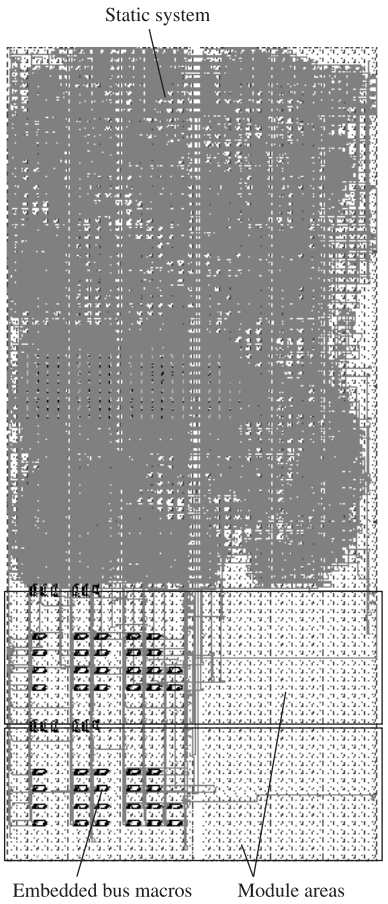

Fig. 10 *Sonic-on-a-Chip in an Virtex-4*

5.4 Configuration overhead

The use of a read–modify–write operation to configure partial bitstream comes at a cost of increased configuration time. In the applications examined, the operational time of an instantiated module is in the order of seconds to minutes or even hours; thus the reconfiguration time overhead in both direct and merge configurations is orders of magnitude

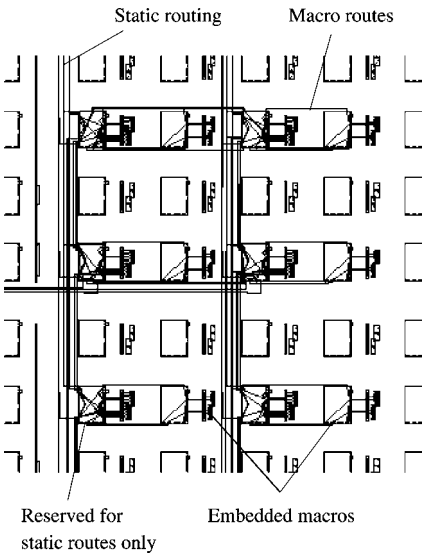

Fig. 11 *Embedded bus macros*

smaller than the operational time. Nevertheless, the responsiveness of the application to a user-initiated request is directly related to the absolute reconfiguration time, and is therefore of interest.

Measurements ascertained that the configuration time for a direct reconfiguration operation can be approximated by

$$T_d \simeq f_d \times \left(\frac{1}{t} + \frac{1}{w_d}\right)$$

where f_d is the equivalent number of frames in the bitstream (including pad frames and other information), t is the rate at which a frame is transferred from memory to the on-chip peripheral bus (OPB) peripheral and w_d is the rate at which a frame is written into the internal configuration access port (ICAP). Note that f_d is found by dividing the total partial bitstream length by the frame size.

Using the read–modify–write operation requires the partial bitstream to be processed in software to identify which frames need to be read from the device; the frames must be read, parts of the frame be modified and the frames written back. The configuration time is approximately

$$T_m \simeq f_m \times \left(\frac{1}{p} + \frac{1}{r} + \frac{c}{m} + \frac{1}{w_m}\right)$$

where f_m is the number of actual configuration frames to merge (which excludes pad frames and other information), p is the rate at which the bitstream is processed, r is the rate at which frames are read from the device, m is the modification rate per row of CLBs, c is the number of CLB rows the module occupies and w_m is the rate at which frames are written back to the ICAP. It should be noted that when reading configuration information, the data are prepended by a 'pad' frame. Similarly, when writing a configuration, an extra pad frame is required after the final frame of real data. This means operating on a single frame at a time is much slower than operating on several contiguous frames in one go, and is one reason $w_m \ll w_d$.

In an earlier work, we calculated estimates of the parameters based on non-optimised driver code, which had significant inefficiencies in data movement [13]. Improvements to the software have since been made by removing unnecessary, extraneous code, function calls and memory synchronisations. Using the Virtex-4-based Sonic-on-a-Chip platform (MicroBlaze processor, system CPU/bus clock speed 100 MHz, 8 kB instruction cache), we obtained the following values for the parameters (all in frames/ms unless noted): $t = 52.6$, $w_d = 580$, $p = 333$, $r = 27.2$, $m = 147$ CLB rows/ms, $w_m = 11.4$. From these values it can be calculated that the time for the read–modify–write configuration in this case is between 6.2× and 11.4× slower than that for the write-only configuration, depending on the height of the module. For example, the modules in the Virtex-4 Sonic-on-a-Chip implementation are 36 columns wide and 16 CLB rows high; the partial bitstream has 795 equivalent frames (f_d) or 790 real frames (f_m). The reconfiguration times are $T_d = 17$ ms for the direct case, and $T_m = 187$ ms for the merge method. It may be observed that although improvements have been made in the driver, in both reconfiguration techniques configuration times are still limited by data movement and software complexities.

6 Conclusion and future work

This paper presented two dynamic reconfiguration methods for modular systems in Virtex FPGAs. The first method uses partial bitstreams directly to reconfigure the FPGA. However, owing to the organisation of the configuration memory, in Virtex-II/Pro devices, modules exclusively occupy complete vertical sections of the device, severely restricting resource allocation and connectivity. The problems would be diminished but not eliminated if the method were applied to the Virtex-4. These restrictions are avoided with the second, novel, merge dynamic reconfiguration method. In this method, information in the partial bitstream is merged with the current configuration as read back from the device. By using an exclusive-OR function to combine the two bitstreams, existing configuration information is preserved and the module can be removed by repeating the XOR operation. Modules can be allocated any rectangular region in the device, and static routes can pass through reconfiguration regions. To avoid conflicts, some of the routing resources are reserved for the static routes.

The merge dynamic reconfiguration method has been employed in three applications that have successfully demonstrated run-time re-targeting of module bitstreams and multi-phase reconfiguration. Unsurprisingly, merge reconfiguration was found to be slower than the direct method. Measurements made on a self-reconfiguring platform quantified the reconfiguration time increase to between 6.2× and 11.4×. The majority of this increase is not caused by the additional time required to read the configuration, as may be expected intuitively, but is due to software and data movement overheads.

The degradation in reconfiguration speed using the merge technique suggests this is an area that should be addressed in future work. Improvements could be made in the data transfer rate across the peripheral bus, and some driver functionality could be migrated to the peripheral hardware. Further investigation is also required into the allocation of reserved routing resources, which was based on an educated guess in this paper and would benefit from an empirical examination. Moreover, the new Virtex-4 devices deserve more study to determine what opportunities are presented by the latest modifications to the configuration architecture. In particular, work is ongoing in the development of module relocation for Virtex-4 devices.

7 Acknowledgments

The authors wish to thank Jean Belzile, Normand Leclerc, Pierre-André Meunier and David Roberge from ISR Technologies for their invaluable contributions, and also Peter Cheung of Imperial College London for his particularly insightful suggestions. The first author gratefully acknowledges the financial support given by the Commonwealth Scholarship Commission and the New Zealand Vice-Chancellors' Committee.

8 References

1 Brebner, G., and Diessel, O.: 'Chip-based reconfigurable task management'. Field-programmable logic and application, August 2001, (Springer-Verlag), pp. 182–191

2 Burns, J., Donlin, A., Hogg, J., Singh, S., and de Wit, M.: 'A dynamic reconfiguration run-time system'. IEEE Symp. on FPGAs for Custom Computing Machines, April 1997, (IEEE Computer Society), pp. 66–75

3 Mignolet, J-Y., Nollet, V., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R.: 'Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable System-on-Chip'. Design, Automation and Test in Europe, March 2003, (IEEE Computer Society), pp. 986–991

4 Steiger, C., Walder, H., and Platzner, M.: 'Heuristics for online scheduling real-time tasks to partially reconfigurable devices'. Field-Programmable Logic and Applications, September 2003, (Springer-Verlag), pp. 575–584

5 Wigley, G.B., Kearny, D.A., and Warren, D.: 'Introducing ReConfigMe: An operating system for reconfigurable computing'. Field-Programmable Logic and Applications, September 2002, (Springer-Verlag), pp. 687–697
6 Xilinx Inc. 'Virtex-4 configuration guide', UG 071, v1.1, 2004
7 Lim, D., and Peattie, M.: 'Two flows for partial reconfiguration: module based or small bit manipulation', Application Note 290, Xilinx, 2002
8 Horta, E.L., Lockwood, J.W., and Kofuji, S.: 'Using PARBIT to implement partial run-time reconfigurable systems'. Field-Programmable Logic and Applications, September 2002, (Springer-Verlag), pp. 182–191
9 Horta, E.L., and Lockwood, J.W.: 'Automated method to generate bitstream intellectual property cores for Virtex FPGAs'. Field-Programmable Logic and Applications, August 2004, (Springer-Verlag), pp. 975–979
10 Guccione, S., Levi, D., and Sundararajan, P.: 'JBits: Java based interface for reconfigurable computing'. Military and Aerospace Applications of Programmable Devices and Technologies Int. Conf., 1999
11 Gericota, M.G., Alves, G.R., Silva, M.L., and Ferreira, J.M.: 'Run-time management of logic resources on reconfigurable systems'. Design, Automation and Test in Europe, March 2003, (IEEE Computer Society), pp. 974–979
12 Poetter, A., Hunter, J., Patterson, C., Athanas, P., Nelson, B., and Steiner, N.: 'JHDLBits: The merging of two worlds'. Field-Programmable Logic and Applications, August 2004, (Springer-Verlag), pp. 414–423
13 Sedcole, P., Blodget, B., Becker, T., Anderson, J., and Lysaght, P.: 'Modular reconfiguration in Virtex FPGAs'. Field-Programmable Logic and Applications, August 2005, (IEEE), pp. 211–216
14 Xilinx Inc. 'Virtex series configuration architecture user guide', Application Note 151, 2004
15 Xilinx Inc. 'Virtex II platform FPGA handbook', UG 002, v1.0, 2000
16 Xilinx Inc. 'Virtex II Pro™ platform FPGA user guide', UG 012, v2.0., 2002
17 Blodget, B., Bobda, C., Huebner, M., and Niyonkuru, A.: 'Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs'. Field-Programmable Logic and Applications, August 2004, (Springer-Verlag), pp. 801–810
18 Sedcole, N.P., Cheung, P.Y.K., Constantinides, G.A., and Luk, W.: 'A reconfigurable platform for real-time embedded video image processing'. Field-Programmable Logic and Applications, September 2003, (Springer-Verlag), pp. 606–615
19 Blodget, B., James-Roxby, P., Keller, E., McMillan, S., and Sundararajan, P.: 'A self-reconfiguring platform'. Field-Programmable Logic and Applications, September 2003, (Springer-Verlag), pp. 565–574
20 Huebner, M., Becker, T., and Becker, J.: 'Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration'. Symp. on Integrated Circuits and Systems Design, 2004, (ACM), pp. 28–32
21 Xilinx Inc. 'Virtex-4 user guide', UG 070, v1.2, 2005