COMMA: A Communications Methodology for Dynamic Module-based Reconfiguration of FPGAs

Shannon Koh and Oliver Diessel

School of Computer Science and Engineering
The University of New South Wales

Embedded, Real-Time & Operating Systems Program (ERTOS)
National ICT Australia
Kensington Laboratory
Sydney, New South Wales 2052, Australia
shannonk@cse.unsw.edu.au
odiessel@cse.unsw.edu.au

Abstract: On-going improvements in the scaling of FPGA device sizes and time-to-market pressures motivate the adoption of a module-oriented design flow. Economic factors encourage the reuse of smaller devices for high performance computational tasks. Like other researchers, we therefore envisage a need for dynamic reconfiguration of FPGAs at the module level. However, previous proposals have not focussed on the communications issues raised by dynamic module-based reconfiguration. This paper proposes a methodology for the rapid deployment of a communications infrastructure that supports the communications needs of dynamic modules when these are known at design time while allowing some flexibility for a range of unknown communication requirements to be met at run time.

1 Introduction

Allowing system designers to exploit dynamic reconfiguration raises many challenges. Of foremost concern is that we need tools to help a designer decompose the application into modules that are potentially reconfigured, or swapped while the system in which the FPGA is embedded is active. Such tools will, as a minimum, need to define the interfaces of modules as well as their timing characteristics. The partitioning of the system into modules cannot be done without reference to the resources available to implement the modules, including, necessarily, the communications infrastructure available to connect the modules together. The modules themselves may be sourced externally from IP vendors or developed in-house, which raises questions about consistency of definitions and descriptions of interfaces and behaviours. Simulation of the module communication needs to include timing. Place and route tools need to operate at the module level, and assuming the required communications can be provided or codesigned, should not perform global design or optimisation steps.

The designer is also likely to want some form of runtime or operating system to be generated or provided that can hide the burden of managing the device. With respect to dynamic module placement, it is desirable that modules be relocatable in order to support schedules that are not known at design time. This places a significant burden on the communications infrastructure to connect, when required, specific IO pins with module ports whose locations are not known until runtime. Inter-module connections involving dynamically placed modules may also need to be provided at runtime. A conflict thus arises between the need to provide fast routes of adequate width for the sake of performance and the constraint of finding and setting these at runtime, conceivably with little laxity in the task. With the release of new architectures [Xi05c], which give more recognition to the scalability of reconfiguration mechanisms, the time seems right to move forwards on practical and effective methods for harnessing reconfiguration at the module level.

It is our contention that effective module-based reconfiguration of FPGAs requires at its foundation a communications infrastructure that can support the varying communications needs of the runtime configured modules. Invariably, these dynamically placed modules will need to connect with other modules on the FPGA and to the IO pins of the device. Since the interfaces of the modules and their runtime placement will in general vary, the communications infrastructure needs to support runtime reconfiguration of the routing, or provide an indirect mechanism for connecting ports with pins. This paper proposes a methodology for the deployment of a communications infrastructure that efficiently supports the communications needs of a collection of dynamic modules when these are known at design time. The methodology also provides a degree of flexibility to allow a range of unknown requirements to be met at run time.

In the following section, we outline a classification of dynamically reconfigurable systems from a module orientation that provides a framework for discussing related work in Section 3 and our contribution thereafter. Our methodology and the target architecture are outlined in Section 4. We describe our progress to date and our assessment of a prototype of the methodology in Section 5 and further work is presented in Section 6. We conclude the paper with a summary in Section 7.

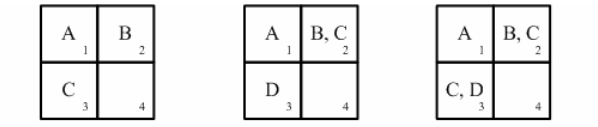
2 A Hierarchy of Dynamically Reconfigurable Systems

System designs that target FPGAs may consist of a static collection of sub-components or modules. That is, the design process results in a static circuit that is configured onto the FPGA until a system reset occurs. Design methodologies for this type of FPGA use are relatively well understood and supported by FPGA tools, although a commonly accepted module-based orientation to the design of large-scale static designs may still be lacking.

By virtue of their reconfigurability, systems including FPGAs may exhibit a degree of flexibility and dynamism. This may take a number of forms and we propose a hierarchy based on the constraints imposed on the reconfigurable system in order to provide a framework for subsequent discussion. In our work, we are concerned with supporting

partial dynamic reconfiguration, in which part of a FPGA device is reconfigured to support new functionality while the system in which it is embedded, or indeed, the parts of the FPGA device that are not being reconfigured, remain active. The thrust of this work is to support module-based or core-based reconfiguration, such as the replacement of one video codec with another, rather than fine-grained reconfiguration, such as the specialisation of a constant coefficient multiplier.

Apart from static designs, three types of dynamically reconfigurable systems, herein denoted DR1, DR2 and DR3 respectively, can be distinguished. DR1 is the most constrained of these and applies to systems in which we know at design time which sub-component of a design may be swapped with a given (set of) alternative(s) and, crucially, which region of the placed and routed circuit is to be reconfigured with a given dynamic module (see Figure 1 for a diagrammatic comparison with the static case).



a) A ***static*** assignment of modules A, B & C to pages 1, 2 & 3

b) ***DR1***: page 2 may have modules B or C assigned to it

c) ***DR2***: module C may be placed at pages 2 or 3 at runtime

Figure 1: Classification of Module-Based Reconfigurable Systems

The class DR2 includes those systems for which the static components and the set of possible dynamic modules is known at design time, but their placement onto the FPGA fabric at runtime is not known. It is possible for this situation to occur when we do not know the order in which dynamic modules are called for at runtime. For the sake of a complete classification, we anticipate scenarios in which the set of modules for which communications needs to be supported is not known at design time. Such systems are classified as DR3 in our scheme.

3 Related Work

Xilinx Application Note 290 (now in [Xi05a]) is the only commercial modular partial reconfiguration methodology known to the authors. Modules are placed in fixed slots and all inter-module communication goes through tri-state bus macros straddling module boundaries and is thus limited to modules adjacent to one another. XAPP290 is essentially a module version-swapping methodology that can support DR1, but is likely to be too unwieldy and limiting for implementing DR2 or DR3 systems. Newer Virtex-4 devices are not supported as they do not have tri-state lines. In [KPR04] Kalte et al. describe a reconfigurable system implementation in which all inter-module communication is performed through fixed tri-state lines implementing an on-chip bus system. Modules may have arbitrary width but must implement the AMBA bus protocol, which may limit the types of modules that can be used. The overheads of managing this bus system can be relatively high and Virtex-4 devices are not supported. Bobda et al. have described a dynamic network-on-chip that allows for modules to be plugged into a

network of routers [Bo05]. However, how the modules communicate through this network, what messages are sent, and how addressing is performed is not described. The module-router interface is unspecified and each module has to be tailored to the network, again limiting the use of currently available IP. Marescaux et al. [Ma02] proposed a network-on-chip utilising the XAPP290 methodology and wormhole routing. This approach not only inherits the limitations in XAPP290 but also requires that some form of network interface be provided in order for IP to be used. Horta et al. described a dynamic hardware plugin platform [HLP02], using a separate FPGA to implement the communications network. All communication includes the overheads of communicating off-chip to and from this FPGA, which may be quite unacceptable or unnecessary.

4 The COMMA Approach

The approach we follow is founded on our desire to provide tools to assist designers in effectively exploiting dynamic reconfiguration. We aim to provide practical, efficient methods for supporting the communications needs of dynamic modules and hope to be able to support future devices with this approach.

We propose developing a tool-suite to automatically generate an on-chip communications infrastructure for dynamic modules. Our approach aims to support a range of devices and platforms and at the same time is optimised for the hardware, the application requirements and available design-time knowledge.

4.1 Reference Target Device

The reference device family we use in order to describe and demonstrate our approach is the Xilinx Virtex-4 family of FPGAs. The frames in these devices are unique in that they are of a fixed-length of 41 quad-byte words, each spanning 16 CLB rows. The frames are tiled about the device into “pages” that span half the width of the device, as shown in Figure 2 which depicts an XC4VFX12 with a 64x24 CLB array and one PowerPC core. The external I/O banks are located on the left, in the middle and on the right (shown in Figure 2 as black bars), rather than around the periphery as in predecessor families. The left and right banks in some device/package combinations (e.g. XC4VFX100-FF1517-10) are located 6 CLBs away from the edges of the device.

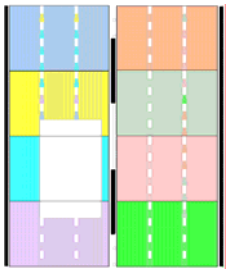


Figure 2: Pages and I/O Banks

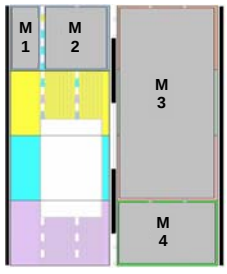


Figure 3: Page Aggregation and Division

4.2 Module Placement Strategy

Since each frame spans 16 CLBs vertically, and the minimum unit of reconfiguration is one frame, it can be seen that we should be able to reconfigure any of the 8 pages shown in Figure 2 independently of every other. Were modules mapped to these pages, they could be dynamically swapped while other modules are left running. Another advantage to this approach is that each clock region on the device corresponds to each of the pages; thus each module can be clocked independently, and with the new support for dynamic reconfiguration of functional blocks [Xi05b], DCMs may be dynamically reconfigured to adjust the clock frequency in each page independently.

This placement strategy opens up possibilities for placing a module in two or more adjacent pages such that larger modules can be accommodated. Since the centre column effectively divides the FPGA resources into two halves it is preferable to aggregate pages vertically instead of horizontally and since the ratio of rows to columns in Virtex-4 devices is always large it is natural to do so. This arrangement also allows carry chains to be expanded. Since the minimum unit of reconfiguration is one frame, it is also possible to divide each page into sub-pages at the granularity of a single CLB column. This may be desired if it is known that there will be very small modules available and that some space savings are necessary. Note that aggregation and division (see Figure 3) need not be permanent. Choosing the granularity presents a trade-off between reconfiguration overheads (increases as granularity decreases) and reconfiguration delay (decreases as granularity decreases). These above concepts can also be implemented on predecessor device families if the entire device is viewed as a single page. The modules are then placed into horizontally divided subpages.

4.3 Pin Virtualisation

One of the unresolved issues with respect to current solutions such as those in Section 3 is the limited connectivity between module and external I/O pins. This limits bandwidth, placement flexibility, and restricts implementation to particular devices and/or platforms.

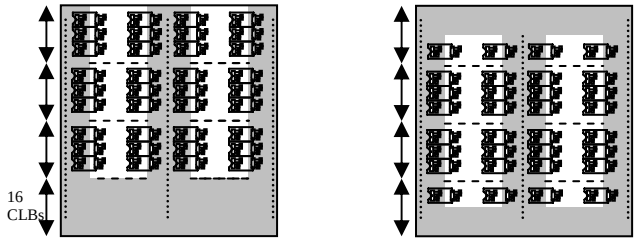


Figure 4: “Trident” and “Double-Ring” Layouts

Our approach allows virtualisation of any module and external I/O pin in the fabric to allow any module to access any other pin. This can be implemented with a communications infrastructure that envelops all the required IOBs. Stylised layouts of such infrastructures are shown in Figure 4 (the grey areas indicate where the infrastructures reside). Each module area occupies a little less than one page and is

connected to the infrastructure via 8-bit LUT-based slice macros [Se05] (the number of and size of macros depicted is not illustrative of actual implementation scenarios).

The "trident" layout on the left has homogeneous module areas (with respect to CLBs), thus allowing for simple means of relocation utilising major-address modification techniques such as REPLICA [Ka05]. However, the critical path from the top-left corner to the top-right corner is long; the "double-ring" layout on the right overcomes this critical delay at the cost of slightly greater relocation complexity as there are actually three different types of slots (the two at the top, the four in the middle and the two at the bottom). As reconfiguration should be glitchless in Virtex-4 devices reconfiguring modules in the top and bottom module areas should not be an issue, but only modules of the same slot type can be relocated.

It is important to note that these observations simply imply that the communications infrastructure should be freeform and optimised to the needs of the application. For example, the infrastructure only needs to envelop those external I/O pins that are required for communication.

Each module area is capable of connecting a large number of bits to the communications infrastructure (a maximum possible 8x2x16=256 bits of communication for a module using all CLBs in its leftmost and rightmost columns for implementing slice macros). We introduce Reconfigurable Data Ports (RDPs) as a means of mapping module ports to slice macros. In Figure 5 the two output ports destined for module M2 are mapped to the one 8-bit RDP (because they can be routed together) whilst the 3-bit port has its own 3-bit RDP. The purpose of the RDPs is to map the module's interface to the slice macros.

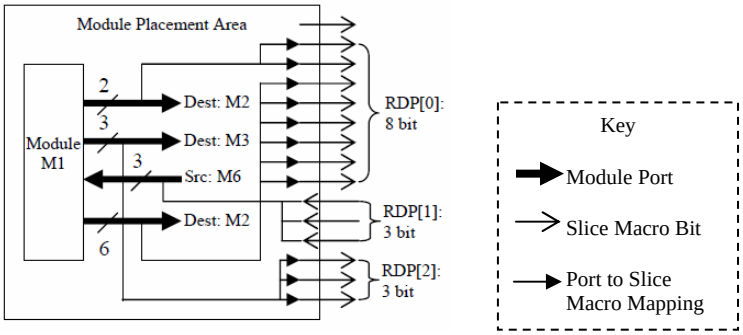


Figure 5: RDP Mapping

The definition of RDPs allows communication channels to be set up in the infrastructure. Channels of varying bitwidths can be defined to ease routing complexity (it is less complex to route a group of bits because there are fewer possible destinations than if each bit were individually routed).

In terms of implementation, the communications infrastructure can be optimised depending on how much is known *a priori* about the modules, and any user-imposed constraints. The following implementation details refer to the hierarchy of Section 2:

- DR3: The user specifies a fixed number of channels and their bitwidths, the necessary external I/O pins, and appropriate number of slice macros to provide per module so that the infrastructure can be optimised.
- DR2: Modules are available a priori and tools can take over some tasks of specifying the communications requirements, including tailoring of module areas (i.e. size, location and slice macros) for optimal use of the module set.
- DR1: Further optimisations are possible in addition to those specified in DR2 as the routing is now entirely static since the placement is fixed.
- Static: This will be similar to a standard modular design flow and is a more restricted case of DR1 where there is only one possible module in each area.

The management of modules arriving and leaving and the setup of channels and routes may be performed by on- or off-chip agents. The essential requirements to be fulfilled by such agents (with respect to communication) are module registration (the registration of module IDs in order to locate peer modules when modules request communication); RDP registration (the registration of module interfaces for channel setups); and channel registration (informing the infrastructure as to what routes should be set up).

4.4 Design Automation and Tool Support

In order for this infrastructure to be readily implemented we propose the development of tools to perform two tasks – the automated generation of the communications infrastructure, and the preparation of modules (e.g. RDP definition) for placement. Figure 6 depicts the complete design and tool flow for communications synthesis. The flow includes three tools that we intend to create, and an incarnation of a partial reconfiguration tool flow (see Section 5).

This pre-processing tool uses Xilinx-supplied device information with user-supplied parameters to create a configuration file that serves as the input for the next tool. The user specifies the infrastructure layout, I/O pad, channel, slice macro and slot preferences to assist the infrastructure generation tool in optimisation.

The infrastructure generator analyses the configuration file and generates an optimised communications infrastructure consisting of HDL, constraints, slice macros and accessory macros. Once the infrastructure has been generated, modules can be wrapped to enable placement into the infrastructure at any time thereafter (even after deployment) by using the wrapper tool to generate RDP interfaces for the modules.

When the infrastructure and an initial set of modules (which may be blank fillers) are available, they should be synthesised using an available partial reconfiguration flow (e.g. the XAPP290 modular flow, a difference-based flow, or one we are proposing in section 5), and the bitstreams can be generated. The system can then be deployed. Additional modules that may be added in the future can be wrapped for use as long as the CCC and infrastructure files are present.

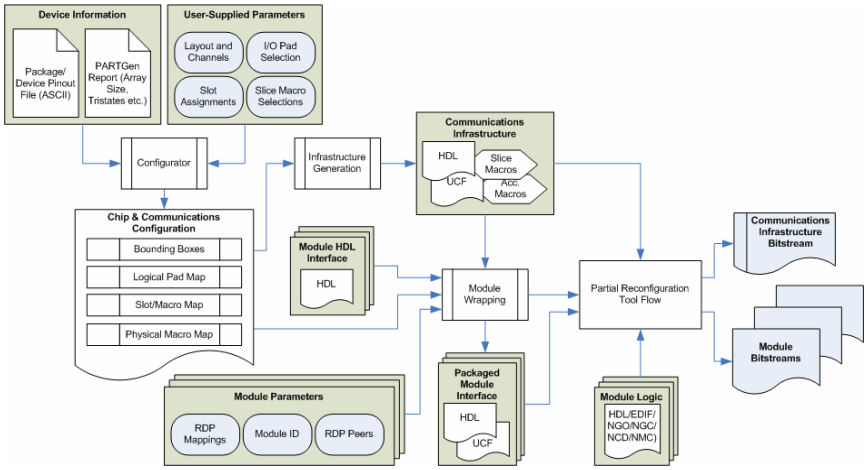


Figure 6: Design and Tool Flow for Communication Infrastructure Synthesis

5 The development so far, and its assessment

To date we have synthesised and placed a trident-shaped communications infrastructure on a Virtex-4 FX12 chip with 4 module areas having 8 inputs and 8 outputs each, and 16 external I/O pins (8 inputs and 8 outputs).

We have implemented a circuit-switched prototype using LUT-based multiplexers, to route each bit coming into the infrastructure onto every outgoing bit. The Virtex-4 CLB architecture allows the implementation of a 2:1 multiplexer with one LUT, a 4:1 with one slice, an 8:1 with two slices and a 16:1 with four slices (or one CLB). The delay of a 4:1 MUX is 0.35ns, an 8:1 is 0.55ns and a 16:1 is 0.75ns.

The control of the communications infrastructure is implemented with a simple microcontroller. Before placing a module's bitstream onto the device, the host assigns communication lines by selecting a particular bit of communication and specifying the source of communication (another module pin or external I/O).

The host then places the bitstream onto the device and enables the slot to start the module running. To remove a module, the host disables the slot, and places a blank module bitstream onto the device to clear the configuration memory.

Due to the lack of support for partial reconfiguration of Virtex-4 in ISE 7.1i, the internal routings of modules ignore the area constraints specified. Special care must therefore be taken to prevent internal module routing from using the infrastructure routes. In order to do this we have adopted a special design flow to implement the modules. This flow, depicted in Figure 7, implements the "Partial Reconfiguration Tool Flow" process of the complete design flow outlined in Section 4.4 and depicted in Figure 6.

This special design flow executes multiple iterations of the Xilinx Modular Design Flow [Xi05a]. The first part of the flow uses the generated communications infrastructure as a module and executes the initial budgeting and active module phases of the modular design flow, resulting in a placed and routed circuit (NCD) of the communications infrastructure.

At this point the NCD design is converted to a hard macro (NMC) and is reinstantiated into the top-level. After the second initial budgeting phase, the Implementable Top Level NGC will contain the infrastructure as well. Blank modules can then be inserted to generate an initial top-level bitstream with final assembly. Multiple iterations of the active module phase are executed to generate individual partial module bitstreams that are ready to be loaded onto the device.

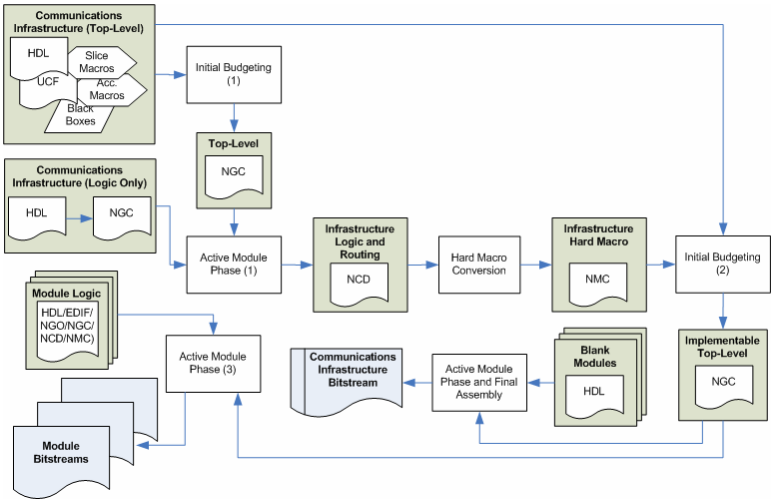


Figure 7: Partial Reconfiguration Tool Flow

The maximum clock speed of the communications infrastructure after place and route (i.e. the hard macro) as per the tool flow above is 358MHz (speed grade –10 i.e. the slowest available), which should be sufficient for most modules. Each slot is 576 slices large (9 columns by 16 rows of 4 slices per CLB) and there are 4 slots in total. The infrastructure itself takes up 572 slices (~10.5%) and the area overhead per page is 3 CLB columns. In comparison, an 8-bit divider takes up 104 slices (234 MHz), a DES core 476 slices (182 MHz) and a MicroBlaze Processor 988 slices (200 MHz). The FX12 thus accommodates a DES core with room to spare in one slot and can fit a MicroBlaze processor in two slots.

It should be noted that this implementation utilised the smallest FX-family device in the Virtex-4 range. Looking across the range of devices available, the method outlined here extends to a maximum number of 22 slots, each approximately 50 CLB columns wide each (3200 slices) for the largest LX device.

6 Further Work

Since the methodology and tool flow have been determined, the next step is to implement the tools identified in Section 4. We plan to implement the Configurator, Module Wrapper and Partial Reconfiguration Tool Flow and combine them to form a development environment. We will also examine methods for determining efficient routing infrastructures and incorporate them into the Infrastructure Generation tool to optimise the routes connecting slice macros and external I/O pins. We have introduced two suggested layouts in this paper but these are not necessarily optimal. Once the tools are complete, an analysis of alternative layouts can be performed to devise better routing strategies.

7 Conclusion

In this paper we motivated the need for a module-oriented methodology to cope with design pressures and expected device scaling. We discussed support needed for dynamic reconfiguration at the module level and argued that the provision of a flexible communications infrastructure that affords a degree of pin address indirection is desirable. We then presented the COMMA approach to supporting communications for new tiled FPGA architectures and outlined design flows to enable its rapid deployment. A prototype was implemented on a Virtex-4 FX12 device and the resulting design was assessed. Our plans for further work were outlined.

References

[Bo05] Bobda C. et. al.: DYNOC: A dynamic infrastructure for communication in dynamically reconfigurable devices. In: International Conference on Field Programmable Logic and Applications, 2005; pp. 153–158.

[HLP02] Horta, E.L; Lockwood, J.W.; Parlour D.: Dynamic Hardware Plugins in an FPGA with partial run-time reconfiguration. In: 39th DAC, 2002, pp. 343–348.

[KPR04] Kalte, H.; Porrmann, M.; Rückert, U.: System-on-programmable-chip approach enabling online fine-grained 1D-placement. In: 18th International Parallel and Distributed Processing Symposium, 2004 p. 141.

[Ka05] Kalte, H. et. al.: REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems. In: 19th IEEE IPDPS, 2005, 8 pp.

[Ma02] Marescaux, T. et. al.: Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs. In: 12th FPL, 2002, 795–805.

[Se05] Sedcole, P. et. al.: Modular partial reconfiguration in Virtex FPGAs. In: 15th FPL, 2005, pp. 211-216.

[Xi05a] Xilinx Inc. Xilinx ISE 7.1 Development System Reference Guide, 2005.

[Xi05b] Xilinx Inc. Virtex-4 Configuration Guide. User Guide UG071, Aug. 2005.

[Xi05c] Xilinx Inc. Virtex-4 Family Overview. Prelim. Product Spec. DS112, Jun. 2005.