Communications Infrastructure Generation for Modular FPGA Reconfiguration

Shannon Koh and Oliver Diessel

School of Computer Science and Engineering
University of New South Wales, Sydney, Australia

Embedded, Real-Time, and Operating Systems (ERTOS) Program
*National ICT Australia**

{shannonk,odiessel}@cse.unsw.edu.au

***Abstract*— Modules that are swapped dynamically at run-time on an FPGA have varying communication needs over time. In order to support this, we aim to generate a wiring infrastructure that caters for the dynamically-changing module interfaces. This, however, imposes a regular structure for laying out modules on a device, which may result in longer inter-module wiring paths as compared to traditional methods where the netlists are flattened. This paper studies placing modules within a structured layout to compare resulting circuit speeds with those obtained by traditional methods. Our results indicate that the difference in critical path delay is high at very low utilisation, but that the overhead is absorbed as the number of modules and interconnection density increases to realistic levels. We conclude that implementing such a wiring infrastructure has manageable overheads while having the added advantage of being amenable to dynamic reconfiguration.**

I. Introduction

A. Overview

Module-based dynamic reconfiguration of FPGAs exploits the on-going scaling of FPGA devices and allows the reuse of smaller devices for economic reasons. However, dynamic reconfiguration of modules is hindered by traditional logic-placement methods that encourage the compaction of module logic. This is because more data may have to be loaded during reconfiguration than is necessary due to the nature of reconfiguration mechanisms in FPGAs. In addition, as module interfaces change over time, it is difficult to ensure that these dynamically-placed modules can be connected effectively at run-time. In order to overcome these barriers, we proposed a regular module placement structure and a wiring harness supporting the communications needs of dynamically-placed modules in [1] (see Section I-C).

Implementing such a regular structure and wiring harness may result in long routing paths being placed as compared to traditional methods where logic is placed in optimal locations freely about the device. In this paper, we outline an approach to laying out module-based circuits to facilitate rapid reconfiguration and assess the communication overheads that our structured approach incurs.

B. Contributions

Our contributions in this paper are:

- We present our solution for placing modules into fixed pages (amenable to dynamic reconfiguration) on a device such that the wiring path lengths and channel width are minimised. At present, this solution is for a single configuration.
- We determine the communications overheads of implementing a wiring infrastructure which supports the communications needs of modules placed in these fixed pages.
- We assess these results and show that the overheads can be absorbed as the amount of connectivity increases to a realistic level. This is also true as device sizes scale.
- We show how the solution for the case of a single configuration forms the basis for the dynamic case where there are multiple configurations.

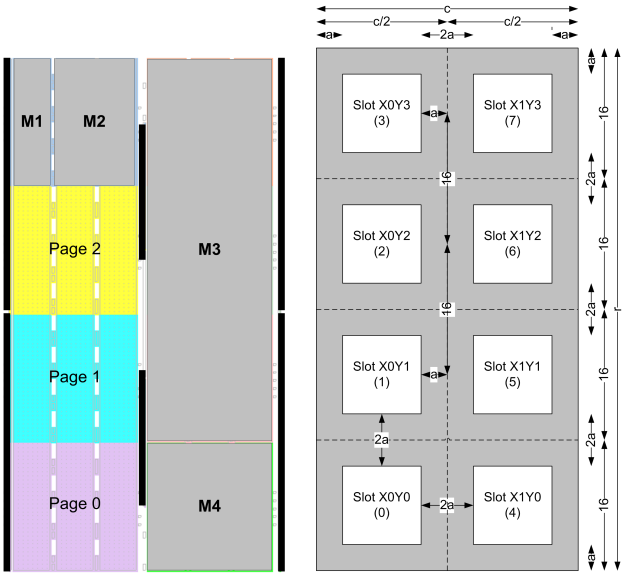
C. The COMMA Methodology

In the following, we present an overview of the COMMA methodology as previously described in [1]. We also state how this relates to the experimental work assessed in this paper.

1) Implementation Target: We target the Xilinx Virtex-4 FPGAs, which have fixed-length, 41 quad-byte word configuration frames spanning 16 CLB rows (20 CLB rows for Virtex-5). These are tiled about the device into independently configurable pages that span half the device width, allowing us to readily place modules as shown in Figure 1(a). Pages can also be aggregated (e.g. M3) or sub-divided (e.g. M1 and M2). External I/O banks (depicted as dark vertical bars) are located towards the left and right edges of the device and in the centre columns.

We have used the slot layout shown in Figure 1(b) for our experiments. The dark area surrounding each slot is reserved for the wiring harness which is customised to support the commmunications needs of the modules. To take care of different module interfaces, each module is connected to the harness through thin wrappers (see "Reconfigurable Data Ports" (RDPs) in [1]) that map logical module ports to selected wiring paths in the wiring harness at reconfiguration time.

2) Design Flow: The COMMA methodology is embedded within a module-oriented design flow as illustrated in Figure 2. The first step partitions an application into its natural



(a) XC4VLX15 pages (b) XC4VLX15 slot layout

Fig. 1. Device page and infrastructure layout diagrams.

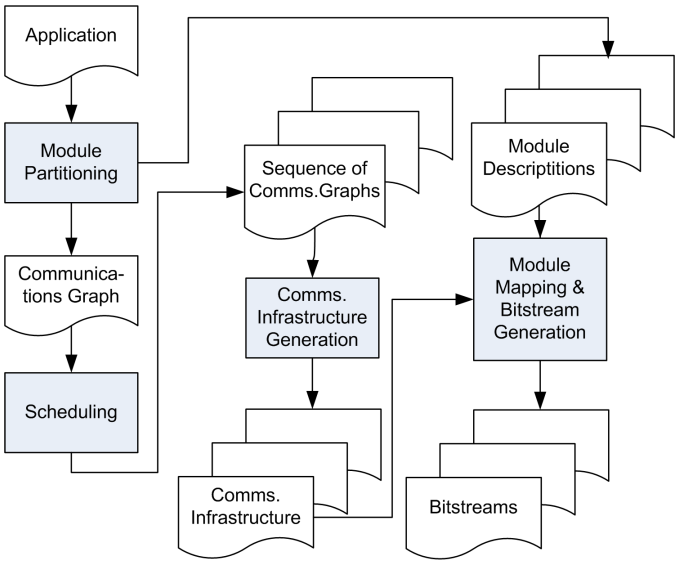


Fig. 2. COMMA Design Flow

functional units of design, resulting in a communications graph. In the next step, this graph is then scheduled into a sequence of smaller graphs, each representing a single configuration. A communications infrastructure consisting of module placements, one or more wiring harnesses, RDPs and slice macros is then generated in the "Communications Infrastructure Generation" step, while taking care to minimise reconfiguration delays by judiciously placing modules and maximising wire reuse. This infrastructure is then merged with the module logic as per the Xilinx Partial Reconfiguration design flow [2], thereby generating the infrastructure (base) and module bitstreams.

II. Implementing a Communications Graph

A. Problem Definition

The main problem tackled in this paper is to map a single communications graph to a Virtex-4 FPGA organised according to COMMA principles. The **inputs** to the problem are a communications graph and the device layout (the device size and channel width available for the wiring harness as in Figure 1(b)). The **output** is a placement of the modules into slots such that the wire availability given by the device layout and wire resource model is not exceeded, and that all propagation delays of the wires used to implement the communications link are less than some maximum delay D, the reciprocal of the desired operating frequency of the application. These are represented as a mapping from modules to slots, and a mapping from each arc in the communications graph to a set of wire resources.

B. Implementation Flow

The implementation flow is a two-step process. The first step assigns modules to slots using an Integer Linear Program (ILP) applied twice as suggested by Fekete *et al.* in [3]. The goal of the first ILP is to minimise the maximum number T of wires interconnecting nonadjacent slots crossing each horizontal dashed line in Figure 1(b). In the second step, a placement is sought that minimises the maximum wire distance subject to the constraint that no more than T wires in total occupy the vertical channels between adjacent rows of module slots. There are several differences between our formulation and that of [3] — we model wires as leaving from or arriving at the midpoints along the edges of slot boundaries; we also model external I/O using actual IOB pin locations, and do not include communications between adjacent modules as these do not need to use vertically-aligned wires in the vertical wiring channels.

In the second step (routing), slice macros are placed on the slot boundaries and RDPs are generated to connect the wires to the modules. The wires to be used to implement the wiring harness are then finally selected. This could be done using a custom router, or using some sort of FPGA Editor script, or using ISE's PAR tool. This follows on to the "Module Mapping and Bitstream Generation" stage as specified in Section I-C.2 and depicted in Figure 2.

C. Related Work

There are some other noteworthy approaches to supporting communications for dynamic modules discussed in detail in [1]. These invariably propose solutions for earlier device families, or are viewed as impractical at present. To our knowledge this is the first study of the runtime overheads of a commmunications infrastructure for dynamic reconfiguration of modules.

III. Experimental Evaluation

A. Purpose of Evaluation

Placing modules according to COMMA principles may result in long wiring paths that are unavoidable (e.g. if there are 8

modules to be placed onto the device in Figure 1(b), there may be an unavoidable routing path between, say, slot X0Y0 and slot X1Y3). In contrast, if a general ISE flow were to be used, the module logic would typically be flattened, distributed, and placed in optimal locations on the device. To assess this, we have created an automated testbench (targeted to the Virtex-4 LX-15 as in Figure 1(b)) to place-and-route synthetic task graphs. Since we are assessing inter-module routing paths, the logic within the modules is not important. Using synthetic task graphs allows many different communications scenarios to be tested at this early stage of the work.

B. Experimental Setup

The following parameters were used: communication graphs with 2, 4, 6 or 8 modules; graphs with 10%, 20%, 40% and 80% of the full module interconnection pattern ($n \times (n-1)$ edges); graphs in which 25%, 50%, 75% and 100% of the modules have inputs from or outputs to IOBs. We define the total external connectivity as the sum of the percentages of modules having external inputs and external outputs. Arc weights were chosen uniformly at random from the set $\{$ 2, 4, 8, 16 $\}$. A co-located set of IOBs was chosen for each external connection, but the sets were randomly distributed about the IOB space. 10 graphs were generated per parameter combination and the results averaged. Allowing for some parameter combinations that did not make sense, over 2500 graph implementations were assessed.

As per Section II-B, each task graph was processed by the ILP with an exhaustive in-house solver. The graph and placement were then input to a VHDL generator to produce thin modules (comprised of XOR gates), slice macros, and slot placements. The code was then synthesised and placed-and-routed. The maximum pin-to-pin delay, (i.e. the maximum delay of the wiring harness) was obtained for both the general ISE flow and the COMMA flow using the constraints described in Section II.

C. Results

ISE usually obtained a lower delay, but sometimes COMMA did better. This is expected since both flows use the heuristic PAR tool. In analysing the results we found that the best indicator of relative performance is the total number of wires (interconnection density). For the range of module numbers explored, Figure 3 illustrates the average increase in the critical path delay of COMMA relative to ISE with respect to percentage internal connectivity between the modules. The plots suggest that the overheads are relatively high for low interconnectivity and decrease as interconnectivity increases.

Figure 4 plots contour maps of the average critical path delay for both ISE and COMMA against the number of wires in the graph and the average wire length. The white region in the upper right half of each map corresponds to a region for which no data was obtained.

The average peak delay in both cases occurs at an average wire length of 30 ± 5 CLBs and 490 ± 35 wires (5.134 ns average critical path delay for ISE vs 5.174 ns for COMMA ($<$ 1% overhead)). This is consistent with Figure 3 and indicates that the overhead is absorbed at high interconnection densities.

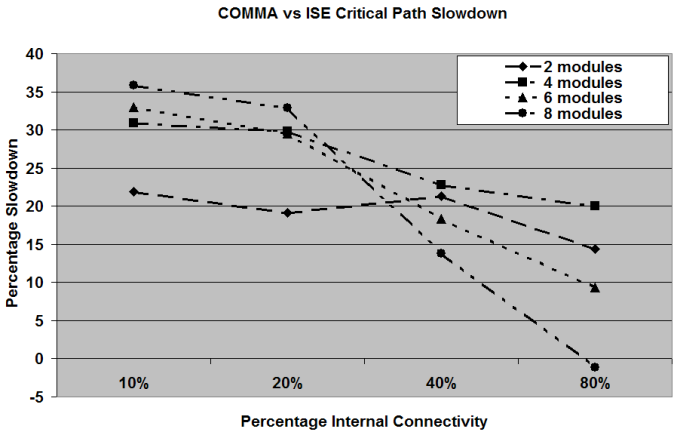
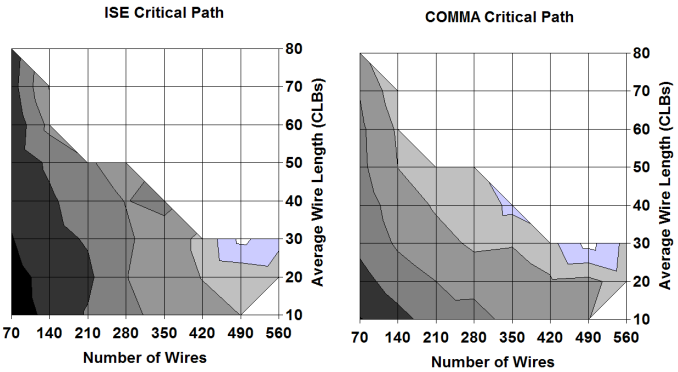


Fig. 3. COMMA vs ISE Critical Path



(a) ISE delays (ns) (b) COMMA delays (ns)

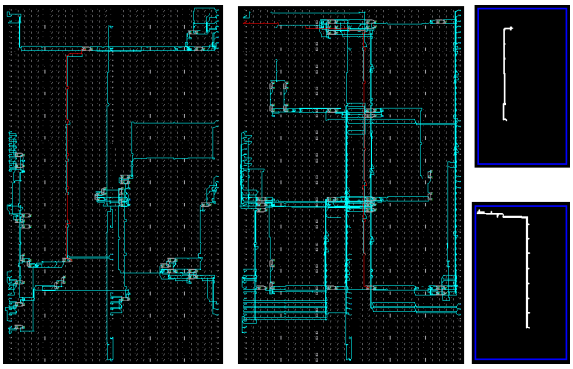
Fig. 4. Critical path delay contours for ISE & COMMA.

The worst case average overhead for COMMA occurs at an average wire length of 40 ± 5 CLBs and a population of 350 ± 35 wires (ISE:3.344 ns, COMMA:4.651 ns (39% overhead)). Although this relative overhead is high, the actual overhead of 1.307 ns can be considered to be small in realistic FPGA applications.

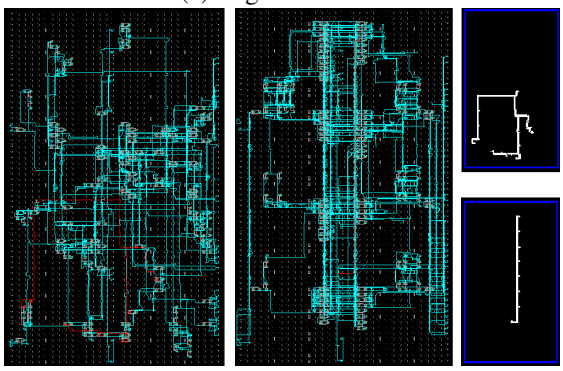
COMMA had an average overhead of 12.5% across all the regions. This is a remarkably good result considering the benefit of the COMMA approach is that modules are easily replaced, whereas in the flattened ISE circuits, the modules are difficult to efficiently reconfigure. The maps also indicate that the COMMA wire delays increase more gradually than ISE with increasing interconnection density. This is likely due to the imposed regularity and explains the relative improvement with higher interconnectivity observed in Figure 3.

We believe the reason for these observations is that ISE is free to place module logic and module pins as close as possible to communicating peers and IOBs. COMMA, however, is constrained to placing module logic within the slots and slice macros on the slot boundaries. This, and the fact that the IOB sets were randomly distributed, forced COMMA to implement

long wiring paths irrespective of its slot location. The ISE (left) and COMMA (right) circuits in Figure 5(a) illustrate such an example. The overhead here was significant (2.235 ns critical path delay for ISE vs 4.992 ns for COMMA). The smaller figures on the right of the two circuits show the



(a) High Overhead



(b) COMMA Surpassing ISE

Fig. 5. Circuit Examples

longest paths for ISE (top) and COMMA (bottom). We can see that for COMMA the longest path connects a module at slot X1Y0 to an IOB on the top left of the device. This is a graph with 8 modules, low internal connectivity (10%) and high total external connectivity (150%). All the slots were used and COMMA had no choice but to place the module at slot X1Y0, thereby yielding this expected result.

In contrast, the circuits in Figure 5(b) depict an example where the COMMA layout (figure right) was significantly better (8 modules, 80% internal connectivity, 125% total external connectivity; results: ISE:5.243 ns, COMMA:3.339 ns). The longest path in the COMMA circuit is again an external I/O connection and that for the ISE circuit is also again a module-to-module connection. In this case, the COMMA structure allows longer wires to be used to good effect, whereas ISE needs to employ several shorter wires to route around the heuristically-placed module logic.

In conclusion, we deduce the longest path for COMMA, given sufficient channel width, should be the sum of the full width and height of the FPGA. This delay may be lower than a local minimum found in ISE, and is usually due to the fixed locations of IOBs or relative module placements. As the number of nets increase, we believe that this overhead becomes insignificant as routing algorithms struggle to find shorter paths and opportunities for logic replication are diminished.

D. Implications on Device Scaling

With the organisation of Figure 1(b), the slot sizes grow in width as we consider larger devices. This means we could pack more logic into a single module and thereby reduce intermodule communication. Larger devices also offer a greater number of configuration "pages" and thus more slots, accommodating larger applications with more modules and thus more intermodule wires. Our results suggest COMMA will perform well under such conditions. There may be longer paths to external IOBs, but these may be mitigated by co-locating a module's external connections and trying to pack modules with closely located IOBs into the same slot. Alternative slot/channel organisations may also help. The additional resource availability provides more opportunities for ISE to replicate and place logic in good locations, while longer wiring paths in COMMA will lead to greater wire delays, but this effect could be mitigated with suitable buffering of COMMA interconnections.

IV. Conclusion and Further Work

In this paper we presented our method for automatically generating a communications infrastructure to suit a page-oriented layout of modularised circuits. The method employs an ILP to place the modules and ISE to route the interconnections between module ports and external IOBs. We have determined the runtime overheads of using the generated infrastructure, and have assessed our results with synthetic communications graphs. We observed an additional critical path delay that is significant at low utilisation, but which decreases as module and interconnection densities rise. As the number of modules, module sizes and interconnections densities reach realistic levels, we believe these overheads become insignificant and are more than offset by the reconfiguration benefits of separating module logic from intermodule wiring paths.

The goal of dynamically reconfigurable applications is to generate a wiring infrastructure that supports a sequence of graphs representing the temporally active module configurations with minimum reconfiguration overheads. Techniques for achieving this goal are currently being developed.

*Acknowledgements

National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

[1] S. Koh and O. Diessel, "COMMA: A communications methodology for dynamic module-based reconfiguration of FPGAs," in *19th International Conference on Architecture of Computing Systems (ARCS), Workshop Proceedings*, 2006, pp. 173–182.

[2] Xilinx, "Early access partial reconfiguration user guide," Xilinx Inc., Xilinx User Guide UG208, 2006.

[3] S. Fekete, J. van der Veen, M. Majer, and J. Teich, "Minimizing communication cost for reconfigurable slot modules," in *FPL*, 2006.