Rapid Overlay Builder for Xilinx FPGAs

Michael Xi Yue
University of British Columbia
Vancouver, Canada
xiy@ece.ubc.ca

Dirk Koch
University of Manchester
Manchester, UK
dkoch@cs.man.ac.uk

Guy G.F. Lemieux
University of British Columbia
Vancouver, Canada
lemieux@ece.ubc.ca

***Abstract*—Overlays are emerging as useful design patterns for solving reconfigurable computing problems. Overlays consist of compiler-like tools and an architecture written in RTL, making it easier for users to quickly compile high-level languages into FPGAs. Despite a high degree of regularity and repetition present in most overlays, it takes a long time for FPGA tools to generate the configuration bitstream. This paper proposes a methodology called Rapid Overlay Builder, or ROB, that combines module relocation, module variants and an efficient form of "routerless" module stitching that we call zipping. Our case study demonstrates up to 22 times speedup in compile-time over a regular Xilinx ISE compilation, while achieving higher clock speeds. By applying ROB, we anticipate that overlays can be implemented more quickly and with more consistent clock rates.[1]**

Keywords—overlays, CGRA, component-based design, module variants, module relocation, module stitching

I. Introduction

Modern FPGA devices contain over 1 million LUTs and over 1000 dedicated memory and multiplier blocks. As circuits continue to scale up, the long place-and-route process required by the CAD tools forms a growing concern. To address this issue, vendor tools accelerate the compilation process using different techniques, including parallel compilation, design partitions, netlist preservation and trading circuit performance for faster compilation. However, the speedup provided by these approaches is still limited.

Recently, overlays have emerged as a way to solve reconfigurable computing problems. They provide promise of portability across devices and rapid development tools for compiling algorithms into the overlay architecture. However, implementing an overlay architecture in an FPGA device still suffers from long place-and-route times [1]. With long place-and-route times, overlays cannot be as nimble and dynamic, e.g. through specialization, where the overlay architecture is highly customized to the needs of the application.

To overcome these limitations, we propose the Rapid Overlay Builder (ROB) methodology for Xilinx FPGAs. ROB can be used to place-and-route any logic design that can be floorplanned into a set of adjacent modules. Most overlays fall into this category and also contain a high degree of repetition and regularity. For example, array-based coarse-grained reconfigurable architectures (CGRAs) are ideal in that they have a regular layout and they replicate a similar (but not necessarily identical) processing element (PE) at each site. Although the ROB methodology can build both homogeneous and heterogeneous overlay architectures, its effectiveness rises with repetitions found in the floorplan.

ROB is a methodology, including a set of scripts and tools, that interacts with the Xilinx ISE toolchain. To obtain fast place and route speeds, it takes advantage of three key underlying techniques: (1) module relocation, (2) module variants, and (3) stitching modules by zipping. *Module relocation* compiles a module into a (placed and routed) hard macro; due to the column-wise resource layout of Xilinx FPGAs, modules can usually be relocated almost anywhere vertically with little or no additional CPU effort. *Module variants* (aka design alternatives) are modules with the exact same functionality but use a different resource mixture (e.g. the ratio of LUTs to DSP blocks). This assists horizontal relocation in the presence of heterogeneous columns found in modern FPGAs. *Zipping* is a routerless method of stitching adjacent modules with zero logic overhead, such that their interconnect aligns perfectly without any extra logic. Zipping allows very tight packing of adjacent modules. Although these three techniques have been previously reported [2-6], they have not been demonstrated together within a single methodology to reduce place-and-route effort. In particular, module relocation and module variants have never been used together with zipping to place-and-route an entire design.

A number of previous research efforts accelerate place-and-route, most notably based on the module relocation in Xilinx FPGAs. All prior hard macro-based approaches, however, rely upon an extra routing step for module stitching. In contrast, ROB is able to pack modules more tightly, employ module variants to reduce the amount of effort even further, and skip the routing process entirely. Moreover, by implementing modules in predefined bounding boxes, we achieve a better placement than the vendor tools. This results in consistently higher clock rates.

One limitation that needs to be addressed is that the ROB methodology currently requires users to manually floorplan their designs on the device. Although this step can ultimately be automated, it is quite common to floorplan overlay designs in order to improve timing performance.

II. Background and Related Work

A. Overlays and CGRAs

Overlay architectures can be thought of as pre-compiled circuits that are reconfigurable themselves. A number of overlay architectures have been previously proposed,

[1] Complete details for using the ROB methodology can be downloaded from http://www.ece.ubc.ca/~lemieux/downloads

including IF [7], MXP [8] and Mesh-of-FUs [9], revealing the potential to boost design productivity. However, these overlays still suffer from very long place-and-route times. To overcome this issue and to demonstrate the use of the ROB methodology, this paper adopts a 2D array of processing elements (PEs) as may be found in a typical CGRA.

In the CGRA under study, each PE contains an integer ALU that communicates with its four nearest neighbours. To demonstrate use of heterogeneous circuits, a predesigned IP block called FPGA Driver [10] is instantiated to connect the CGRA with DDR3, Ethernet, and a PC host over PCIe. The FPGA Driver occupies an area equivalent to 30 PEs. To meet I/O timing, it is constrained in the right side of the device. It should be apparent how to adapt ROB to other overlays.

B. Module Variants

A placed and routed module is called a *pre-built* module in this paper. For a pre-built module, we define its *footprint mask* as the set of underlying column resources covered by the module. For example, a placed module spans columns of type {D, M, M, B}, we say this region has a *footprint mask width* of 4. The letters indicate the type of column resource used: DSPs (D), slice-L logic only LUTs (L), slice-M logic-or-memory LUTs (M), and BRAMs (B).

A pre-built module is relocatable to locations with an identical resource footprint. However, if the destination footprint mask differs, the module needs to be re-placed and re-routed within an appropriate bounding box [5]; it may also need to be re-synthesized to better match available resources (eg, to use LUT-based multipliers). After this, we call the new module instance a *variant* of the original module. The ROB methodology creates a set of *module variants* to increase relocation flexibility. The targeted device in our case study is a Xilinx XC6VLX240T-FF1156 from an ML605 board. Its footprint mask width is 101 columns. The left and right sides of the device have similar sub-footprint masks, reducing the number of required variants by half.

C. Module Relocation

Module relocation with component-based design is an efficient technique to accelerate placement and routing. Although Xilinx tools do not support module relocation well, various other works support this feature. Recent work on module relocation is based on hard macros at the netlist level [2][3][4]. These utilize custom CAD tools to find valid placements that match the footprint mask requirements. In ROB, module relocation has greater flexibility due to *module variants*. The variants result in a set of equivalent pre-built hard macros that can be placed in more candidate locations.

D. Routerless Stitching

In addition to module relocation, this work stitches modules without invoking the router. This saves run-time and area as gaps are not required to route between modules.

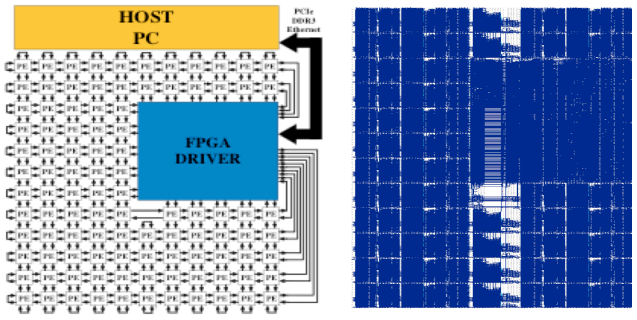
One way to achieve this routerless objective is to use bus macros [11]. However, this option was not considered here for three reasons: (1) bus macros need considerable extra logic, (2) they add extra latency, and (3) bus macros would have impacted the placement flexibility (e.g. they cannot be placed on BRAM columns).

Instead, it is possible to directly connect wires in adjacent modules if the wiring requirements line up on both sides of a shared edge. This was previously demonstrated to produce zero logic and delay overhead [6] using the GoAhead tool to generate appropriate constraints for the Xilinx place and route tools [6]. We follow the same approach using GoAhead, but generate all of the constraints automatically as part of the ROB flow. We name this process zipping.

III. Rapid Overlay Builder Methodology

This section presents the Rapid Overlay Builder methodology, or ROB for short. We will demonstrate the use of this methodology to build the CGRA described in Section II while targeting a Xilinx XC6VLX240T-FF1156 device.

There are seven tasks needed to build an overlay in ROB. Out of the seven tasks, only the first two tasks presently require manual engagement from the users, while scripts automate the other five tasks. The first two tasks are also ultimately scriptable, but this is currently not implemented. Below, we cover these seven tasks in greater detail.



(a) (b)

1. *Resource budgeting for a PE tile.* Before floorplanning the CGRA, an initial synthesis and placement of the PE tile is required to obtain the set of required resources as a reference. This is a standard Xilinx ISE compilation run and does not require any constrained floorplan of the PE tile. However, a PE design may require heterogeneous types of resources. Later on, we may find this resource mixture does not match the available footprint of the required placement, such as providing enough hard multiplier blocks. In such a case, the PE tile can be re-synthesized with some soft multipliers, for example. Therefore, the PE tile sometimes needs to be built with different synthesis options. This provides a comprehensive set of reference designs with different resource demands and that add flexibility when creating placed PE variants later. Designers need a preliminary understanding of the size of the PE tile and the resource footprint of the device.

2. *Floorplanning the CGRA design.* On the targeted Virtex 6 device, the height of one multiplier block is

equivalent to the height of five CLBs. To simplify vertical relocation, the height of the PE tile needs to be a multiple of five CLBs as well. After reserving room for the FPGA Driver, Table I shows floorplan alternatives consisting of PE tiles with different aspect ratios that accommodate about 230 logic slices (115 CLBs), which is the amount needed for one PE. It also gives the external fragmentation (leftover CLBs) after instantiating the maximum number of PE tiles on the device. In some of the floorplan alternatives, logic-only PE tiles (of the same height as the other PE tiles) are used when DSP blocks are not available at the target placement position.

Table I shows that floorplan options #3 and #4 yield the lowest external fragmentation. We will use floorplan option #4 because the resulting PE tiles will be half the height of a clock region. Hence, PE tiles will not span across the clock region boundary. This also allows future work to use the floorplan as part of a dynamically reconfigurable system.

As shown in Figure 1(a), the chosen floorplan consists of 11 PE tiles in the horizontal direction and 12 PE tiles in the vertical direction. Since the PE tiles are relocatable in the vertical direction, only one PE variant is needed for every column of PE tiles. Furthermore, with the feature similarity found on the left and right side of the device, only 6 variants of each PE are required across the 11 PE columns. Using these 6 PE variants, we decided to build CGRAs of 8 different sizes to test scalability of the ROB methodology.

3. *Placing and routing initial PE variants using Xilinx ISE.* The next step is to manually define the PE-to-PE connection wires with the help of connection anchors to be located around a *PE tile*. A PE tile is a bounding-box constrained region (of size PE Width x PE Height) that contains all of the PE logic and routing. One PE tile will be produced for each PE variant. The connection anchors are temporary appendages that will be discarded.

For each PE tile, a set of connection anchors is required on each of the four sides of the rectangular PE tile. A connection anchor is a pre-built hard macro that will connect the signals for communication between adjacent PE tiles, each one forming one half of the interface for zipping later. When PE tiles are abutted, it is important for interconnect between adjacent PE tiles to lie in a straight line. Otherwise, a wasteful "dogleg" shape connection would be required. To avoid doglegs, the connection anchors on the opposite sides of a PE tile must physically correspond to each other.

Although not supported by our CGRA, communication between non-adjacent PE tiles can also be achieved. If the communication is a one-time case, the user can add a path to route through the intermediate PE tiles; those tiles may end up being new variants. If the array itself must support long wires, then the CGRA tiles must be designed to accommodate wire twisting in exactly the same way an FPGA array with long wires is built using a single layout tile.

Floorplan Option #	PE Width (CLBs)	PE Height (CLBs)	Number of PEs	External Fragmentation (CLBs)
1	24	5	93	1920
2	12	10	95	1680
3	8	15	102	720
4	6	20	101	840
5	5	25	78	3330
6	4	30	81	3360

TABLE I. FLOORPLAN ALTERNATIVES

In our early experiments, we found was that the routing is more congested at the boundary of the PE tile. This congestion manifests itself as longer route times in ISE and lower clock frequency. This is because logic resources at the border have access to fewer wires for routing than the logic resources that are located in the center of the PE. To address this issue, we prohibited the logic from being placed in the top or bottom tile rows for faster routing times.

Once the physical constraints of the PE variants are set, every PE variant needs to be placed and routed using ISE. Since the compilation process is independent for every variant, this process can be trivially parallelized across workstations. Depending upon the precise overlay design and usage, it may be possible to precompute this initial PE build process so it is not observed by users.

4. *Extracting the PE tiles from the initial PE variants.* Once the PE variants are placed and routed, the NCD netlists of the PE variants are automatically converted to XDL netlists by scripts. With the XDL netlists, the PE variants can be cut out along the boundary of the PE tile, leaving the data bus wires as floating antennas; these cut interconnect wires will be zipped to adjacent tiles. These XDL representations are stored in a pre-built PE tile library. The library of pre-built PE tiles will be used for assembling the final CGRA.

5. *Relocating PE tiles on the device.* In the case study, every pre-built PE tile spans 20 rows of CLBs, which is half the height of a clock region. Five PE tiles have a footprint mask width of 8; one PE tile contains only logic resources and has a footprint mask width of 10. The PE tiles from the library are relocated and instantiated according to a floorplan.

6. *Interconnecting adjacent PE tiles.* In this case study, adjacent PE tiles are placed next to each other. By design, the floating interconnect located along the zipping boundary of each tile perfectly aligns with each adjacent tile, so no additional routing is needed. Not having to use the router saves time and also avoids the potential for allocating additional logic or routing resources to form connections between mismatched components. This also helps achieve a size-independent clock rate for the overlay.

7. *Interconnecting the CGRA design and the FPGA Driver.* After all the PE tiles are stitched together, the CGRA and the FPGA Driver are also automatically stitched together using similar netlist manipulations. As shown in Figure 1(b), the FPGA Driver was floorplanned and built in such a way that all of the IO ports of the FPGA driver correspond with their adjacent PE tiles. Since the FPGA Driver is a common part of the design and can be fit in the compilation process as a hard macro partition, the corresponding compile time is not included in the standard ISE flow nor in the ROB flow below. Therefore, the CAD time for building the FPGA Driver is excluded from all experimental run-time results.

In the present system, the FPGA Driver is the only interface enabled in the ROB methodology to control data flow inside the CGRA, to carry out the personalization process, and to write CGRA bitstreams to the PEs.

IV. Experimental Results

All the experiments used a Dell Workstation T5500, which features an Intel Xeon 3.33GHz processor and 8GB RAM. Xilinx ISE 14.7 was used under Windows 7 (64-bit). The FPGA device was specified as a -1 speed grade.

In the case of building the array of integer-only PEs, two user scenarios were considered: (1) user builds CGRAs from scratch; (2) user builds CGRAs with pre-built PE tiles. Table II shows the elapsed CAD time for both the Xilinx ISE and the new ROB methodology. The process of building initial PE tiles takes 18 minutes to complete using 4 CPU cores and the associated CAD time is included in Table II. While the ROB methodology obtained considerable speedup in building CGRA designs, the bottleneck of obtaining further speedups lies in the final XDL conversion process, which converts back to NCD netlists for bitstream generation. We anticipate that Xilinx can greatly accelerate this conversion process. The ROB flow is able to obtain speedups up to 92x with the final XDL netlist conversion time excluded.

In addition to speeding up the process of building overlays, maintaining high logic utilization levels and clock rates are also goals of the ROB methodology. It was found in the experiments that ROB was able to achieve logic utilization levels over 89%. However, utilization levels resulting from the ROB methodology are slightly lower than ISE. This is likely because the initial PE tiles were implemented with their top and bottom rows of the resource in the bounding box prohibited, as described in Section III.

The clock rates from the ROB methodology are consistent and always higher than ISE. When increasing the size of CGRAs, the F_{max} from ISE fluctuates between 88.1MHz and 115.8MHz, while the F_{max} from the ROB methodology consistently stays at 120.7MHz. While the ISE F_{max} can be stabilized and made faster using multiple placement seeds, this is not necessary with ROB.

V. Conclusion

In this work, we present the ROB methodology that efficiently builds overlays which contains a high degree of regularity and repetition on Xilinx FPGAs. By applying the techniques of module relocation, module variants, and stitching, we are able to obtain up to 22x speedup in building CGRAs. The ROB methodology could have been implemented using HMFlow [2], but only at the cost of substantial lower logic utilization levels (HMFlow does not work well above 50%) and at lower clock speeds (HMFlow invokes the router to connect modules). In contrast, ROB achieved over 89% utilization levels while outperforming ISE on clock speed.

CGRA Size	Total Time (seconds)		User Scenario 1	User Scenario 2
	ROB Methodology	Xilinx ISE		
18 PEs	1189	2397	2.0x	22.0x
41 PEs	1346	3642	2.7x	13.7x
49 PEs	1435	4405	3.1x	12.4x
57 PEs	1538	5739	3.7x	12.5x
65 PEs	1617	5583	3.5x	10.4x
77 PEs	1881	9767	5.2x	12.2x
89 PEs	2037	9698	4.8x	10.1x
101 PEs	2259	10988	4.9x	9.3x

TABLE II. CAD Time Comparison

References

[1] D. Capalija and T. Abdelrahman, “Tile-based Bottom-up Compilation of Mesh-of-FUs FPGA Overlays,” in *Proc. of FPL’12*.

[2] C. Lavin et al., "HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping," in *Proc. of FCCM’11.*

[3] C. Beckhoff, D. Koch, and J. Torresen, "GoAhead: A Partial Reconfiguration Framework", in *Proc. of FCCM’12.*

[4] J. Coole and G. Stitt, "BPR: Fast FPGA Placement and Routing Using Macroblocks," in *Proc. of CODES+ISSS’12.*

[5] A. Wold, D. Koch, and J. Torresen, "Component-based design using constraint programming for module placement on FPGAs," in *Proc. of ReCoSoC’13.*

[6] D. Koch, C. Beckhoff , J. Torresen, "Zero Logic Overhead Integration of Partially Reconfigurable Modules, " in *Proc. of SBCCI’10.*

[7] J. Coole and G. Stitt, "Intermediate Fabrics: Virtual Architectures for Circuit Portability and Fast Placement and Routing," in *Proc. of CODES+ISSS’10*.

[8] A. Severance, and G.G.F. Lemieux, "Embedded supercomputing in FPGAs with the VectorBlox MXP Matrix Processor," in *Proc. of CODES+ISSS’13.*

[9] D. Capalija and T.S. Abdelrahman, "A High-performance Overlay Architecture for Pipelined Execution of Data Flow Graphs," in *Proc. of FPL’13.*

[10] P. Lysaght et al., “Invited Paper: Enhanced Architecture, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs,” in *Proc. of FPL’06.*

[11] K. Vipin et al., "System-level FPGA Device Driver with High-level Synthesis Support," in *Proc. of FPT’13*