Tile-based Bottom-up Compilation of Custom Mesh-of-Functional-Units FPGA Overlays

Davor Capalija and Tarek Abdelrahman

FPL, Munich
Sept. 2, 2014
Why aren’t FPGAs dominant accelerators?

• Peak performance – great strides in last 10 years
  - Stratix  ~15 GFLOPs (2004)
  - Stratix V ~600 GFLOPs (2012)
  - Arria 10 ~1 TFLOPs (2014)
  - Stratix 10 ~10 TFLOPs (2015)

• Commodity accelerator boards available
  - Nallatech
  - BittWare
  - Terasic
“FPGA Programmability Wall” - HDL Era

- How do software programmers see FPGAs?
- 1985 - 2010
  - The only option was HDL

- Bricks in the “programmability wall”:
  - Difficulty of hardware design
  - Very low level of abstraction
  - Extremely long compile times
  - Can’t fit
“FPGA Programmability Wall” - HLS Era

• How do software programmers see FPGAs?
• 2010 - 2014
  – HLS is available and competitive with HDL

• Bricks in the “programmability wall”:
  – Difficulty of hardware design
  – Very low level of abstraction
  – Extremely long compile times
  – Can’t fit
What's missing?

Application in high-level language
10 hour compile time

Application in high-level language
< 1 min application compile time
Overlays

- Programmable pre-compiled FPGA circuits
  - Overlay / virtual fabric / intermediate fabric
  - Example: a soft vector processor
Overlays

Application in high-level language
10 hour compile time

Application in high-level language
< 1 min application compile time

- Short compile (<1min)
- Fast development cycles
- Short reconfiguration time (1us)
Overlay research challenges – previous work

- What is “FPGA natural” overlay architecture?
  - Exposes massively parallel FPGA architecture
  - Leverage FPGA customizability
  - “Sea of FUs” with mesh interconnect

- Programming model for the overlay?
  - Data flow graphs (DFGs)
  - Pipelined execution of DFGs

- Mapping algorithm to the overlay
  - Place, route, and latency balance
Overlay research challenges – this paper

- Efficient compilation of mesh-of-FUs to FPGA fabric
- 1) Maintain high Fmax as mesh is scaled up
- 2) Quickly compile custom overlays
Mesh-of-FUs overlay architecture

- “Sea” of wide **Functional Units** with mesh interconnect
- Expose massive parallelism of FPGA fabric!

![Diagram of mesh-of-FUs overlay architecture with overlay I/O, word-wide FU, elastic pipeline units, and overlay cell (ocell) with programmable routing.](image)
Compiling overlays to FPGA fabric: flat flow shortcomings

- **Flat Flow**
  - The default push-button flow of the CAD tool
  - The entire overlay is compiled at once and from scratch
- **Does not maintain high $F_{\text{max}}$ as mesh size is increased**
  - Despite regular topology and distributed control of the architecture

![Graph showing $F_{\text{max}}$ vs overlay mesh size](image)

- $F_{\text{max}}$ drops as mesh is scaled up
Overlays need to leverage FPGA customizability

- Create a library of many custom overlays
  - Customize functional layout → selection & arrangement of FUs
- Match functional layout to application's compute needs
  - Map to more FUs – higher performance

![Different functional layouts](image)
Compiling overlays to FPGA fabric: flat flow shortcomings

- From scratch compile
  - Very slow, takes hours
- Always compiles the entire overlay
  - Modifying a single FU requires a full recompile
- Can't reuse overlay components to build new overlays
  - Compile time is spent redoing the work already done for existing overlays

- We want to quickly compile a custom overlay
- We need a faster flow with ability to reuse overlay components
New approach: Tile-based bottom-up flow

• Hierarchical physical design
  – Partitioning and floorplanning
  – Overlay is divided into regular *tiles*
  – Tiles are isolated with elastic buffers

• Benefits
  – Maintains high $F_{\text{max}}$ as mesh size grows
  – Build overlays by quickly stitching tiles
  – Reuse tiles from tile library
  – Compile tiles in parallel
In a nutshell: Tile-based bottom-up flow

1) independent compilation of tiles
2) stitching of tiles
Bottom-up flow – independent tile compile

- independent compile of each tile in a separate CAD project
- inter tile-elastic buffers
- tile I/O constraints
Bottom-up flow – independent tile compile

- independent compile of each tile in a separate CAD project
- inter tile-elastic buffers
- tile I/O constraints

Tile I/O constraints:
- fine-grained regions
- tile I/O registers

Inter-tile elastic buffers

Tile: 4x5 ocells

FPGA
Bottom-up flow – tile stitching

- top-level CAD project to stitch the tiles
- only paths between the tiles are routed
- no need to re-place any logic
Floorplanning granularity: cells

ocell's physical region

cell-based floorplan – 80 physical regions
8x10 overlay
Floorplanning granularity: tiles

tile's physical region

tile-based floorplan – 4 physical regions

8x10 overlay
4 tiles with 4x5 ocells
Bottom-up flow – tile library and tile reuse

Template floorplan
9 physical regions

3 overlays
- each 12x15 mesh
- 9 tiles, tile 4x5 ocells
Bottom-up flow – tile library and reuse

*Tile library:* 3 overlays = 27 tiles
Bottom-up flow – tile library and reuse

Tile library: 3 overlays = 27 tiles

Tile reuse
Bottom-up flow – 2 ways to improve compile time

Flat flow: entire overlay compiled from scratch
Bottom-up flow – 2 ways to improve compile time

(1) Reuse library tiles + tile stitching
Bottom-up flow – 2 ways to improve compile time

(2) Parallel compile of tiles + tile stitching
Experimental Evaluation

- High-end Stratix IV (200K ALMs)
- 5 custom overlays with 12x15 mesh size
  - different FU selection and different functional layouts

- Fmax
  - best-of-5 seeds
  - from 1x1 up to 12x15 mesh size

- Overlay compile time

- Flow resource usage
  - overhead of inter-tile elastic buffers
Comparison of flows – $F_{\text{max}}$ vs. mesh size

Mesh size in overlay cells/FUs

- 37% avg. $F_{\text{max}}$ improvement for 12x15 mesh
- Stitching of library tiles also delivers high $F_{\text{max}}$
  - 330-350 MHz range
Tile-based bottom-up flow: compile time

- **12x15 mesh**
  - 9 tiles, each tile has 4x5 ocells

- **Flat flow:** from-scratch single-machine compile
  - 4 hours

- **Bottom up (1):** reuse and stitch library tiles
  - 35 minutes

- **Bottom up (2):** compile overlay tiles in parallel
  - 9 cluster nodes, one per tiles
  - 1 hour
    - Each tile takes 23 minutes
    - Tile stitching in 35 min
Tile-based bottom-up flow: resource overhead

• 12x15 mesh
  – 9 tiles, each tile has 4x5 ocells

• Overhead of using inter-tile elastic buffers
  – 8% increase in logic usage over flat flow

• Bottom-up vs. flat flow
  – 8% resource cost
  – 37% avg. better $F_{\text{max}}$
  – → 27% better $F_{\text{max}}$/area
Related Work

- Partitioning and floorplanning
  - Exist in commercial tools for a while – little published on benefits
- Koch et al. [FPL’08]
  - Template floorplan for dynamic/partial reconfiguration
- Lavin et al. [FCCM’11, FPL’13]
  - Stitching of hard-macros for rapid prototyping
- LaForest and Steffan [FPT’13]
  - Show benefit of partitioning for homogeneous mesh of soft processors
Future work

- Tile stitching in a minute
- Different tile sizes
- Library with 1000+ tiles
- Tile relocation
- Automatic design of custom functional layouts
Conclusions – tile-based bottom-up flow

• Key benefits
  – Maintain high Fmax as overlay mesh size is increased
  – Enable tile library, tile reuse and parallel compile

• Key contributions
  – Demonstrate benefits of partitioning and floorplanning for modular regular circuits
  – Rectangular coarse-grain tiles
    • Resource efficiency
    • Template floorplans
  – Independent compilation of tiles
    • Inter-tile elastic buffers and tile I/O constraints
Conclusions – tile-based bottom-up flow

- The paper also contains details on:
  - Top-down flow
  - Comparison: flat vs. top-down vs. bottom-up flow