PRFloor: An Automatic Floorplanner for Partially Reconfigurable FPGA Systems

Tuan D. A. Nguyen (1) & Akash Kumar (2)

(1)  ECE Department, National University of Singapore, Singapore
(2)  Chair of Processor Design, Center for Advancing Electronics Dresden, TU Dresden, Germany
Partial Reconfiguration (PR)
Xilinx ISE PR Design Flow

1. Design using Xilinx XPS or ISE
2. Generate then Import netlists into PlanAhead
3. Determine sizes and locations (placements) for PR Regions
4. Run Place and Route
5. Generate bitstreams
Xilinx ISE PR Design Flow

1. Design using Xilinx XPS or ISE
2. Generate then Import netlists into PlanAhead
3. Determine sizes and locations (placements) for PR Regions
4. Run Place and Route
5. Generate bitstreams
Floorplanning

- DSP Half Column
- PRR 2
- Violate Overlapping Constraint
- PRR 1
- BRAM Full Column
- CLB Half Column
- PRR 3
- Reserved for regional clock spine.
Problem?
Problem?

8 PRRs

15 PRRs
So?
Design using Xilinx XPS or ISE

Generate then Import netlists into PlanAhead

Determine sizes and locations (placements) for PR Regions

Run Place and Route

Generate bitstreams

Execute PRFloor
Common Issue of Previous Works

Only consider PR regions (PRRs)
[Rabozzi14, Duhem13, Vipin12, Bolchini11, Montone11, Montone08]
Common Issue

1. PRR 1 and 2 are too far away

2. There is not enough DSP left for static module
Another issue

There are so many (static + PR) modules in MPSoC, up to hundreds in total
Recursive Cut-size Driven Netlist Bi-partitioning


Bipartitioning in FPGA

Smallest possible placement of Module A in the left partition

Cut Line

Smallest possible placement of Module A in the right partition
PRFloor - Overview

Find all possible placements for modules on FPGA

Use NLP-based bipartitioner to scatter the modules across the FPGA
    Each module is assigned a preferred location called “anchor point”

The modules and their placements are heuristically filtered and sorted

Find the feasible combination of the placements
PRFloor - Overview

1. Find all possible placements for modules on FPGA

2. Use NLP-based bipartitioner to scatter the modules across the FPGA. Each module is assigned a preferred location called “anchor point”.

3. The modules and their placements are heuristically filtered and sorted.

4. Find the feasible combination of the placements.
PRFloor - Overview

Find all possible placements for modules on FPGA

Use NLP-based bipartitioner to scatter the modules across the FPGA
Each module is assigned a preferred location called “anchor point”

The modules and their placements are heuristically filtered and sorted

Find the feasible combination of the placements
Recursive Pseudo-bipartitioning Heuristic

Anchor point
Non-linear Integer Program (NLP)

\[ \text{nets}_{ij} = n_{ij} \times (1 - m_i) \times m_j + n_{ij} \times m_i \times (1 - m_j) \]
\[ = n_{ij} \times (m_i + m_j - 2 \times m_i \times m_j) \]  

**Objective:**

\[ \sum_i m_i \times \sum_{j \neq i} n_{ij} - 2 \times \sum_i m_i \times \sum_{j \neq i} (n_{ij} \times m_j) \]  

**Subject to:**

\[ \text{Total}_{0CLB} = \sum_i ((1 - m_i) \times \text{CLB}_{0i}) \leq \text{MAX}_{0CLB} \]  
\[ \text{Total}_{1CLB} = \sum_i (m_i \times \text{CLB}_{1i}) \leq \text{MAX}_{1CLB} \]  

Balance the number of CLBs occupied in two partitions.

The number of nets between 2 modules that cross 2 partitions.

The total number of crossing-nets between all modules.

The total number of CLBs occupied by the modules in each partition should not exceed the available CLB in that partition.


PRFloor - Overview

Find all possible placements for modules on FPGA

Scatter the modules across the FPGA surface as uniformly as possible. Each module is assigned a preferred location called “anchor point”

The modules and their placements are heuristically filtered and sorted

Find the feasible combination of the placements
PRFloor - Overview

Find all possible placements for modules on FPGA

Scatter the modules across the FPGA surface as uniformly as possible. Each module is assigned a preferred location called “anchor point”

The modules and their placements are heuristically filtered and sorted

Find the feasible combination of the placements
# Experiments: Synthetic Systems

<table>
<thead>
<tr>
<th>System</th>
<th>No. Mod</th>
<th>%CLB</th>
<th>%BRAM</th>
<th>%DSP</th>
</tr>
</thead>
</table>
| 3 PRRs | 99      | 65% $\rightarrow$ 85%  
(41% $\rightarrow$ 60%) | 42% $\rightarrow$ 60%  
(9% $\rightarrow$ 26%) | 6% $\rightarrow$ 13%  
(4% $\rightarrow$ 11%) |
| 8 PRRs | 116     | 65% $\rightarrow$ 85%  
(36% $\rightarrow$ 56%) | 28% $\rightarrow$ 31%  
(16% $\rightarrow$ 19%) | 14.5% $\rightarrow$ 15.1%  
(11.1% $\rightarrow$ 11.7%) |
| 15 PRRs| 130     | 65% $\rightarrow$ 87.8%  
(34% $\rightarrow$ 57%) | 45% $\rightarrow$ 53%  
(27% $\rightarrow$ 34%) | 25% $\rightarrow$ 28%  
(22% $\rightarrow$ 25%) |
| 24 PRRs| 126     | 65% $\rightarrow$ 85%  
(33% $\rightarrow$ 52%) | 45% $\rightarrow$ 60%  
(21% $\rightarrow$ 36%) | 23% $\rightarrow$ 32%  
(22% $\rightarrow$ 31%) |
Execution Time

Increases almost linearly with the number of modules
Experiments: Real systems

- Instantiate PR-HMPSoC [Nguyen14] with varying number of PRRs (3 to 8)
- Compare the maximum achievable clock frequency with the comparable static system

The maximum clock frequency results obtained from PR systems are not worse than the static ones.
Compare with [Rabozzi14]

PRR 1 and 2 are too far away

Wastage is 19% lower
Total Manhattan distances
is 35% smaller

Compare with [Rabozzi14]

There is not enough DSP left for static module

There is sufficient DSP resources for static module

Conclusion

• The automatic floorplanner, PRFloor, is presented with the NLP-based bipartitioner
• PRFloor can provide high quality result in couple of minutes
Future Work

- **Improve the quality and performance**
- Control the designer choices over wire-length or wastage better
- Accelerate the first step of finding placements for modules
- **Support bitstream relocation [Oomen15]**

if "pr_module" is 0, this tile will be considered as normal module however, setting "pr_module" to 0 has not been tested yet

```
<processor name="tile_mb_hw" type="tile_mb_hw" pr_module="1">
  <!- properties of PR module -->
  <pr_properties>
    <!--
    The "module_name" and "inst_name" must be identical to the name of PR module and its instance name in Tile
    -->
    <pr_instance module_name="pr_wrapper" inst_name="pr_wrapper_0" />
    
    <!--
    "pr_variant" means different implementation of one PR module
    PRGen will look for variant's netlist in: "./pr_netlist/<type>/<module_name>).ngc
    the "netlist" field is ignored in the current version of PRGen but it must be there
    You can add as many variants as you want for a Tile
    -->
    <pr_variant name="microblaze" type="microblaze" netlist="pr_netlist" />
    <pr_variant name="adder_xor" type="hw_adder_xor" netlist="pr_netlist" />
  </pr_properties>
</processor>
```

```xml
archgraph.xml [xml] [54,1] &ff
```
Thank you!
Appendix
Large PR MPSoC

[Nguyen14]


FPGA Model
Why half-column granularity?
Pareto Ranking
Sort the placements

\[ \text{OBJ } \downarrow \text{placement} = \alpha \times \text{wastage} + \beta \times \text{dist_to_anchor} \]
PRFloor - Overview

Build FPGA model

Create ROOT partition

Find all possible placements for all modules

Do recursive pseudo-vertical-cut for ROOT

Do recursive pseudo-horizontal-cut for ROOT

Calculate the normalized wastages and distances

Select placement candidates

Sort the placements of each module

Sort the modules in decreasing order of resource

Find possible combination

Success?

No.
Move the first vertical cut-line to the right

YES!
DONE!
Recursive Pseudo-bipartitioning Heuristic

```
Algorithm 2 Recursive Pseudo-bipartitioning Heuristic
Require: parent_partition ≠ ∅ and valid cut_line and cut_type
1: create partition0 ← ∅ from cut_line and parent_partition
2: create partition1 ← ∅ from cut_line and parent_partition
3: list_mod ← ∅ {list of modules used for bipartition process}
4: area_constraint ← 90%
5: if cut_type = vertical cut and first cut then
6:   area_constraint ← 100%
7: end if
8: for all module ∈ parent_partition do
9:   list_placement0 ← ∅; list_placement1 ← ∅
10:  for all placement of module ∈ parent_partition do
11:      area_ratio0 ← area_of_placement ∩ partition0
12:      area_ratio1 ← area_of_placement ∩ partition1
13:      if area_ratio0 ≥ area_constraint then
14:         add placement to list_placement0
15:      else if area_ratio1 ≥ area_constraint then
16:         add placement to list_placement1
17:      end if
18:  end for
19:  if list_placement0 = ∅ and list_placement1 = ∅ then
20:      deduct the resource of module from the total resource of partition0 and partition1
21:    else
22:      resource0 ← estimate_resources(list_placement0)
23:      resource1 ← estimate_resources(list_placement1)
24:      add module → list_mod
25:  end if
26: end for
27: run NLP_Bipartitioner(list_mod, partition0, partition1)
28: if SUCCESS then
29:    update anchor points of modules
30:  execute Recursive Pseudo-bipartitioning for partition0
31:  execute Recursive Pseudo-bipartitioning for partition1
32: end if
```


**Estimate occupied resources**

**Algorithm 3** Estimate Occupied Resources

1: \( a = \bar{x} \)
2: if \( a > \bar{x} \) then
3: \( a = \tilde{x} \)
4: end if
5: \( \text{result} = a - 1.5 \times \sigma_x \)
6: if \( \text{result} < \text{minimum\_occupied} \) then
7: \( \text{result} = \text{minimum\_occupied} \)
8: end if
9: return \( \text{result} \)

\( \bar{x} \): arithmetic mean \hspace{1cm} \tilde{x} \): median \hspace{1cm} \( \sigma_\chi \): standard deviation
Bipartitioner

- The available resources in two partitions can be different
- The resources occupied by the possible placements of one module in two partitions can be different
- Each type of resource occupied by modules in two partitions can be balanced individually
Quality of the NLP Bipartitioner


The recursive process used to find the floorplan is very fast.
It takes only at most 1.2% of the total runtime. In most cases, almost 0.
Effect of $\alpha$ and $\beta$ to Wire-length and Wastage

$OBJ_{\text{placement}} = \alpha \cdot \text{wastage} + \beta \cdot \text{dist\_to\_anchor}$
Resource requirement PRRs compared with [Rabozzi14]

<table>
<thead>
<tr>
<th>PR Regions</th>
<th>Fig. 14a</th>
<th>Fig. 14b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLB</td>
<td>BRAM</td>
</tr>
<tr>
<td>PRR 1</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>PRR 2</td>
<td>50</td>
<td>18</td>
</tr>
<tr>
<td>PRR 3</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>PRR 4</td>
<td>200</td>
<td>0</td>
</tr>
</tbody>
</table>