Hierarchical Reconfiguration of FPGAs
What is Hierarchical Reconfiguration?

Configuring modules inside reconfigurable modules
Why Hierarchical Reconfiguration?

Fewer modules & faster reconfiguration for complex systems
Why Hierarchical Reconfiguration?

With a set of $M$ level 1 modules and a set of $N$ level 2 submodules:

- $|M| \cdot |N|$ bitstreams multiplicative for plain reconf.
- $|M| + |N|$ bitstreams additive for hierarchical reconf.

Hierarchical reconfiguration is for rather complex systems
Why Hierarchical Reconfiguration?

Possible examples:

• Reconfigurable instruction set extensions in a reconfigurable CPU

• Key updates in a crypto module

→ Consider a reconfigurable CPU (level 1) with a reconfigurable crypto core extension (level 2) where the keys are changed by a level 3 module

• Dynamic function loading:
  • Type-dependent comparators in a sorter
  • Regular expression matching

• Code updates in a softcore CPU (BRAM content)
How to implement it?  Static (Level 0)

Step 1: Placement

Connection macros replace (substitute) the partial module
How to implement it?  Static (Level 0)

Step 2: Blocker (routing constraints)

Connection macros replace (substitute) the partial module.
How to implement it? Static (Level 0)

Step 3: Routing

Connection macros replace (substitute) the partial module
How to implement it?  Static (Level 0)

Step 4: Clean (optional)

Connection macros replace (substitute) the partial module
How to implement it?  Static (Level 0)
How to implement it? Module (Level 1)

Step 1: Placement

Connection macros replace (substitute) the static system and the reconfigurable submodule
How to implement it? Module (Level 1)

Step 2: Blocker (routing constraints)

Connection macros replace (substitute) the static system and the reconfigurable submodule
Connection macros replace (substitute) the static system and the reconfigurable submodule
How to implement it? Module (Level 1)

Step 4: Clean (optional)

Connection macros replace (substitute) the static system and the reconfigurable submodule.
How to implement it? Submodule (Level 2)

Step 1: Placement

Connection macros replace (substitute) the reconfigurable submodule
How to implement it? Submodule (Level 2)

Step 2: Blocker (routing constraints)

Connection macros replace (substitute) the reconfigurable submodule
How to implement it? Submodule (Level 2)

Step 3: Routing

Connection macros replace (substitute) the reconfigurable submodule
How to implement it? Submodule (Level 2)

Step 4 : Clean (optional)

Connection macros replace (substitute) the reconfigurable submodule
How to implement it? Submodule (Level 2)

GoAhead Flow (Floorplanning)
How to implement it? Submodule (Level 2)

user submodule

sub_o → sub_i

urder module

logic synthesis, technology mapping, placement (Xilinx ISE)

VHDL wrapper

static placeholder

sub_i → sub_o

placement constraints
UCF

routing constraints
XDL

submodule library

blocker integration (GoAhead)

routing (Xilinx ISE)

module extraction (GoAhead)
How to implement it?  

Design Flow

Manual by the programmer

• Partitioning of the system into different config. levels  
  (not different to what we know from writing software)

• Decide if modules are suitable for reconfiguration  
  → in general it is not possible to decide if parts in a  
    system can be executed mutually exclusive

• Floorplanning of the hierarchical level-1 modules

All other steps can be carried out automatically
Case Study: Reconfigurable CPU

Reconfigurable CPU (level 1) with reconfigurable instruction extensions (level 2)
Case Study: Reconfigurable CPU

<table>
<thead>
<tr>
<th># configurations</th>
<th>plain single level</th>
<th>hierarchical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 configurations</td>
<td>150</td>
<td>6</td>
</tr>
<tr>
<td>Level 2 configurations</td>
<td>-</td>
<td>18</td>
</tr>
<tr>
<td># total</td>
<td>150</td>
<td>24</td>
</tr>
<tr>
<td>Bitstream storage</td>
<td>12.2 MB</td>
<td>652 KB</td>
</tr>
<tr>
<td>Reconfiguration time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1 reconfiguration</td>
<td>2.5 ms</td>
<td>2.5 ms</td>
</tr>
<tr>
<td>Level 2 reconfiguration</td>
<td>-</td>
<td>240 us / 475 us</td>
</tr>
</tbody>
</table>

~ 6 x fewer bitstreams, 20 x less memory and 5-10 times faster reconfiguration
Other Issues

- Crossing static routing through reconfigurable regions (possible by corresponding wire allocation)
- Guarantee timing for relocatable modules (we can use static timing analysis)
- Using multiple clock domains (possible, but might impact relocation)
- How to deal with heterogeneous resources? (multiple design alternatives for a module)
- Bitstream generation
  - See talk from Christian Beckhoff:
    Portable Module Relocation and Bitstream Compression for Xilinx FPGAs
    Session FH2c Thursday 13:30-14:45 CR3