Reconfigurable Computing

Partial reconfiguration design

Chapter 8

Prof. Dr.-Ing. Jürgen Teich
Lehrstuhl für Hardware-Software-Co-Design
Dynamic Partial Reconfiguration Design - Introduction

- Advantages of Reconfigurable Hardware Designs
  - Speedup in comparison to GPP
  - Higher flexibility in comparison to ASIC

- Advantages of Partial Dynamic Reconfiguration
  - Area (cost) savings
  - Power savings

The Altera video surveillance camera reference design
Source: www.terasic.com.tw
A partially reconfigurable design consists of:
- A set of full reconfigurable designs
- A set of partial designs which can be separately downloaded

The full designs as well as the partial modules are available as full (partial) bitstream used to configure the device.

The partially reconfigurable modules are used to move the system from one configuration to the next one.
The purpose of this section is to learn how to design a partially reconfigurable system using the current CAD tools and devices.

The Xilinx FPGAs (Virtex and Spartan) are some of the few devices on the market allowing partial reconfiguration.

This section will focus on three Xilinx FPGA-based methodologies for designing a partially reconfigurable system:

- The Xilinx partial design flow
- The ReCoBus-Builder flow
- The Xilinx PlanAhead flow
A frame is the smallest unit of configuration. Spans the height of the FPGA (Virtex-II family) or the height of one or more clock regions (newer Virtex families)

Configuration is glitchless. If you reconfigure a frame with the same data, no glitches appear on the signal lines.

Reconfiguration “reinitializes” SRL16s and LutRAM, not BlockRam
Xilinx Partial Flow
Partial reconfiguration design - Approach

- Traditional design flow
  - Full circuit
  - Constraints define:
    - Placement Constraints
    - Relative Location Constraints
    - Timing Constraints
- Only one full circuit is generated
Partial reconfiguration design - Approach

- Partial reconfiguration flow:
  - Placement constraints must be provided
  - Modules are compiled separately
  - The result is a set of full and partial implementations (EDIF and bitstream).
- The partial reconfigurable bitstreams are used to move the device from one configuration to another.

Diagram:
- VHDL + Basic Constraints (pins, timing, ...)
- Placement Constraints (block positions and area)
- Technology Mapping
- Place and route
- Full Netlist 1, 2, 3
- Partial Netlist 1, 2, 3
- Full Bitstream 1, 2, 3
- Partial Bitstream 1, 2, 3
Partial reconfiguration design - Approach

- Delaying placement constraints increases degree of freedom

VHDL  
SystemC  
HandelC  

Basic constraints

Netlist 1  Netlist 2  Netlist 3
Full  Partial  Full

Placement Constraints
(block positions and area)

Run-time Relocation

Full Bitstream 1  Full Bitstream 2  Full Bitstream 3

Partial Bitstream 1  Partial Bitstream 2  Partial Bitstream 3

Area constraints provided after a first evaluation

Reconfigurable Computing
Partial reconfiguration on Xilinx Virtex FPGAs

- Create a bitstream database for full and partial modules to be used at run-time for device reconfiguration

Source: Images generated with Xilinx ISE design tool
The partial design flow

- Modular implementation of a large project
- The team manager defines the structure of the overall project (top-level)
- Each designer or team of designers implement and test each module separately
- The implemented modules are integrated in the final design
- A top-level consists of
  - A set of independent modules
  - Interfaces between the modules
  - Interfaces with the pins
- Each module is assigned a given position and area on the device by means of area constraints
For partial reconfiguration, the goal is to generate:
- A set of full designs
- A set of partial designs
- The partial designs are used to move from one full design to another

The input is structured as follows:
- Top_level
  - Module_1
  - Module_2
  - ...
  - Module_N

The input language can be any HDL.
The partial design flow – Example

- Modular design
  - Static module is fixed for all times
  - Only partial module can be reconfigured

- Insertion of communication macros at fixed positions
The partial design flow – Use of Slice Macros

- The routing of two designs creates unpredictable paths
- Signals connecting two reconfigurable modules in two different designs can be routed in different ways
- This can produce malfunction of the design after reconfiguration
- This can be avoided by providing fixed communication channels (slice macros) among reconfigurable modules
- Bus macros are tri-state lines running over 4 CLBs in FPGA
  Must be placed only at the top level!
The partial design flow – Four Steps

1) Build the top level context
   - Slice Macros at fixed positions are used to communicate between static design logic and reconfigurable logic.
   - Note that there is NO logic except IOs and clocks in the top level design. All logic is contained in one or more 'modules' (e.g., AREA_GROUP).
   - Required files: top.ngc and top.ucf (for constraints)
   - Output file: top.ngo.

2) Build the static modules
   - These are the modules (E.G. logic and routing) that will NOT be dynamically reconfigured.
   - Required files: .ngc for each static 'module'.
   - Output files: top_routed.ncd (without reconfigurable logic)
3) Build the dynamic modules

- Build each flavor of each dynamically reconfigurable module.
- Required files: .ngc for each dynamic 'module'.
- Output files: Routed .ncd file for each flavor of a dynamically reconfigurable module WITHOUT logic and routing from static modules.
The partial design flow – Four Steps

4) Assemble full design with each flavor of each dynamically reconfigurable module. Generate the required bitstreams.
   - Required files: .ncd for static modules and for each flavor of each reconfigurable module
   - Output files:
     - a bitstream for full design with each flavor of each reconfigurable module
     - a partial bitstream for each flavor of each reconfigurable module
     - report file
The partial design flow – Example

- Two modules
- One VGA controller
- One colour generator
- The two modules can be partially reconfigured

Source: Images generated with Xilinx ISE design tool
ReCoBus-Builder Flow
ReCoBus Tool Flow - Introduction

The Simple Formula for Building Bus-based Reconfigurable Systems

• Example system:

- Fine slot grid for reducing internal fragmentation
- Alignment-multiplexer allows free module placement
- Interface grows together with the module complexity (size)
- Very low logic overhead (no extra connection logic within the slots)
- Allows high speed / high throughput
- Hot-swap module exchange

I/O-Bars for Point-to-Point Links

The ReCoBus-Builder

- Easy usable builder for reconfigurable systems
- Available on www12.cs.fau.de/research/recobus

ReCoBus

System Specification
(Communication Architecture & Floorplan)

ReCoBus-Builder – Design flow

Design Entry, Static/Dynamic Partitioning, and Verification

ReCoBus & connection bar protocol specification [ReCoBus-Builder]

bus & bar
RTL model

static system

module1

functional simulation [Modelsim]

n
OK?

budgeting [Xilinx XST]

floorplanning and communication synthesis [ReCoBus-Builder]

budgeting [Xilinx XST]

static netlist

dynamic constraints

ReCoBus I/O bars

module1 constraints

module, netlist

place & route static [PAR]

place & route module1, [PAR]

build static bitstream [bitgen]

build module, bitstream [bitgen]

static system bitfile

bitstream linking [bitscan]

initial system bitfile

partial bitstream extraction [bitscan]

full module, bitfile

partial module, bitfile

 physical implementation

bitstream assembly

repository for the run-time system

ReCoBus-Builder – Design flow

**Physical Implementation**

- **Physical Implementation**
  - OK?
  - **budgeting** [Xilinx XST]
  - **floorplanning and communication synthesis** [ReCoBus-Builder]
  - **bitstream linking** [bitscan]
  - **bitstream assembly**
  - **initial system bitfile**
  - **static system bitfile**
  - **static netlist**
  - **module1 bitfile**
  - **partial module1 bitfile**

ReCoBus-Builder – Design flow

ReCoBus-Builder – Design flow

Tested Design

Creation of Partial Modules

- User modules need to be restricted in its place
- 2 Aspects of Restriction:
  - Mapping of the functions
  - Routing

- Mapping restrictions:
  - Use prohibit statements in the ucf-file of the Xilinx flow

- Routing restrictions:
  - Block all not allowed routing resources by dummy signals
Design Flow: Blocking

Start window of the ReCoBus-Builder

Blocking with the ReCoBus-Builder

Building an I/O bar

Building a ReCoBus


Reconfigurable Computing
In the lab, you will build a small demo system:
Xilinx PlanAhead Flow
Xilinx PlanAhead

- New advanced GUI for the complete FPGA Design-Flow
- Project management
- Floor planning
- Critical path analysis
- Implementation viewer
- Integration of the Partial Flow

Source: Xilinx, Inc.
1. **Step:** Synthesis of all partial and static modules in individual netlists (Static netlist has black boxes)

2. **Step:** Creation of a new PlanAhead project

3. **Step:** Creation of Reconfigurable Partitions
   - A reconfigurable partition (RP) might consist of several reconfigurable modules (RM)
   - Assign a partial netlist to each RM
   - A RM can also be a black box (empty module)
4. **Step:** Floor planning reconfigurable partitions

- Create Area Groups
- PlanAhead automatically creates the communication ports of the reconfigurable partition
- Port proxy logic: LUT1 (old flow: Bus macro)
- PlanAhead automatically creates the constraints file (UCF)

Source: Xilinx, Inc.
Xilinx PlanAhead – Design flow

5. Step: Run design rule check (DRC) to verify the design

6. Step: Create the first reconfigurable configuration
   - Consisting of the static module and for each RP a RM
   - Implement this configuration
   - Promote this configuration
7. **Step:** Create further configurations  
- Import the static design  
- Implement the partial module

8. **Step:** Create the static and partial bitfiles

Source: Xilinx, Inc.
Xilinx PlanAhead – Under the Hood

- The old partial flow is improved and integrated into PlanAhead

- The differences are:
  - New proxy logic: Only one LUT is needed instead of two
  - Possibility to analyze the timing over the borders of partial modules
Flow Comparison
Flow Comparison: Tiling the Reconfigurable Area

- Different ways of tiling the system
  a) island style
  b) slot style
  c) grid style

Xilinx Partial Flow
Xilinx PlanAhead

ReCoBus-Builder
Flow

Future research

static part of the system
unused reconfigurable area
different modules

Reconfigurable Computing 43
## Flow Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Xilinx Partial Flow</th>
<th>Xilinx PlanAhead</th>
<th>ReCoBus-Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floor planning</td>
<td>manually</td>
<td>Script and GUI-based</td>
<td>Script and GUI-based</td>
</tr>
<tr>
<td>Routing constraint technique for PR region signal crossing</td>
<td>Explicit with dedicated hard macros</td>
<td>Implicit based on an incremental design flow and proxy logic</td>
<td>Explicit by binding signals directly to wires (blocking)</td>
</tr>
<tr>
<td>Routing constraint planning</td>
<td>Manual macro instantiation and placement</td>
<td>Automatic placement of the proxy logic</td>
<td>Semiautomatic signal to wire binding</td>
</tr>
<tr>
<td>Module Relocation</td>
<td>No, but possible with handcrafting</td>
<td>No - a module is bound to one fixed position</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi Module Instantiation (place same module multiple times on the FPGA)</td>
<td>No, but possible with handcrafting</td>
<td>No – one bitfile for each module/position permutation</td>
<td>Yes</td>
</tr>
<tr>
<td>Module Migration among different systems</td>
<td>No</td>
<td>No</td>
<td>Possible within a device family</td>
</tr>
</tbody>
</table>
## Flow Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Xilinx Partial Flow</th>
<th>Xilinx PlanAhead</th>
<th>ReCoBus-Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module encapsulation</td>
<td>No, but possible with handcrafting</td>
<td>No – changes in the static design require rerouting of all partial modules</td>
<td>Yes – the static system and all modules are implemented fully independent</td>
</tr>
<tr>
<td>Overhead</td>
<td>2 LUTs/signal wire</td>
<td>1 LUT/signal wire</td>
<td>No logic overhead</td>
</tr>
<tr>
<td>Island style</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1D slot style</td>
<td>possible with handcrafting (only coarse-grained)</td>
<td>No</td>
<td>Yes (fine and coarse-grained)</td>
</tr>
<tr>
<td>2D grid style</td>
<td>possible with handcrafting (only coarse-grained)</td>
<td>No</td>
<td>Yes, by cascading multiple 1D rows</td>
</tr>
</tbody>
</table>