Altera’s Partial Reconfiguration Flow

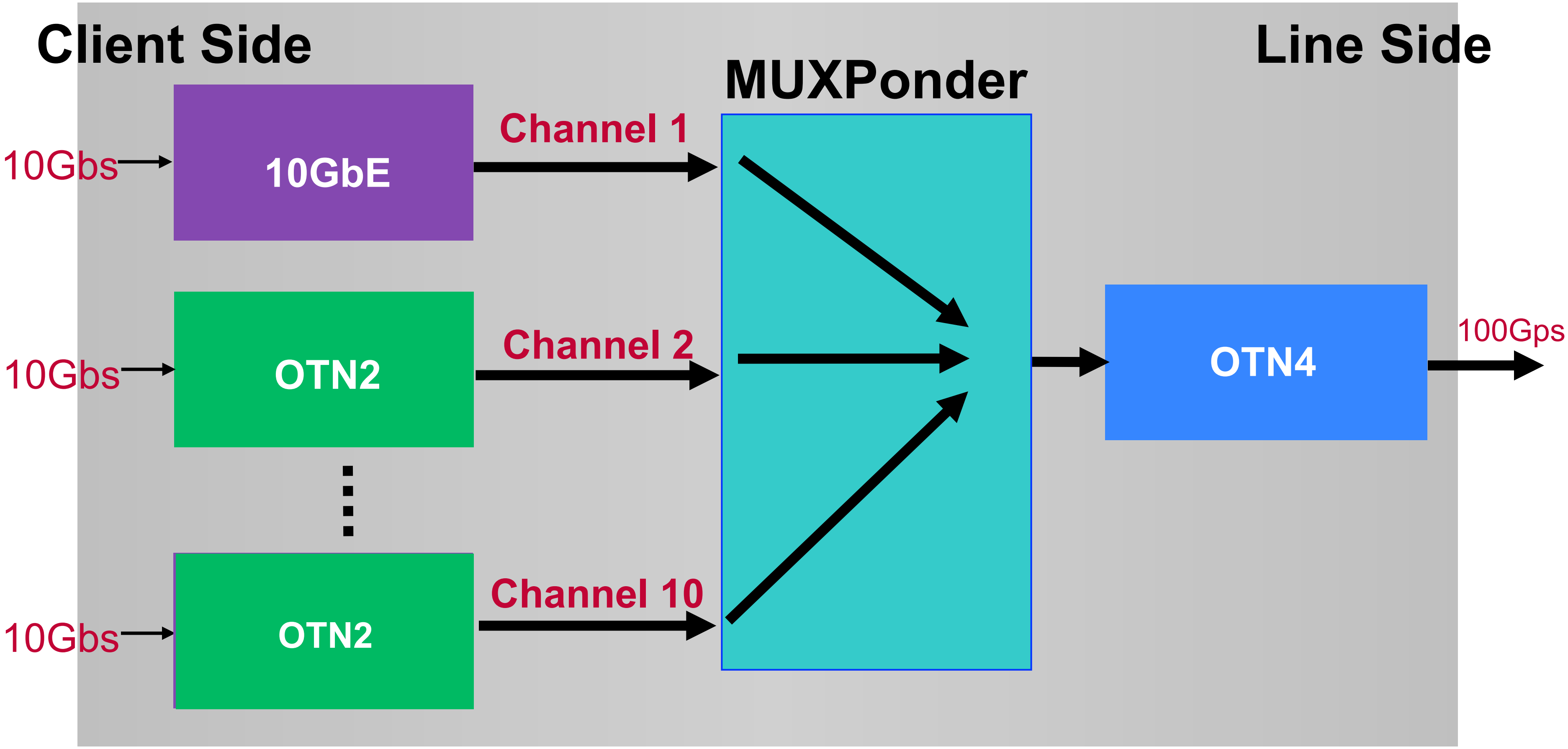
Mark Bourgeault

June 23, 2011

Outline

- High Level Overview
- Hardware Support
- Reconfigurable Projects
- Software Implementation Details
- Upcoming Features & Summary

Example System: 10*10Gbps→OTN4 Muxponder

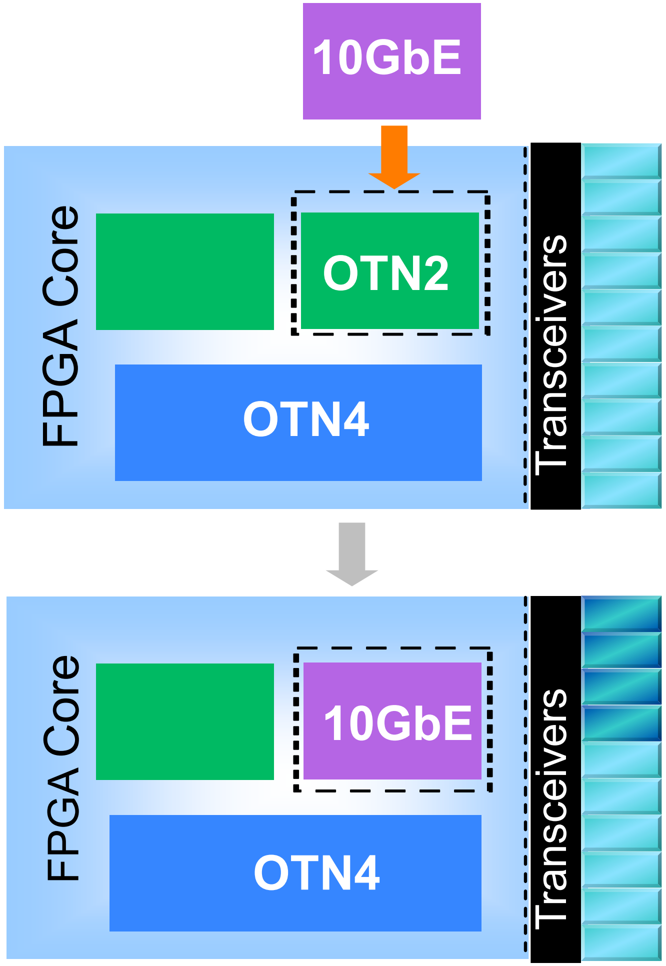


Altera Partial Reconfiguration Strategy

- Software flow is key
  - Enter *design intent*, automate low-level details
  - Build on existing incremental design & floorplanning tools
  - Intuitive simulation flow, *including effect of reconfiguration*
- Partial reconfiguration can be controlled by soft logic, or an external device
  - Load partial programming files while device operating
  - Support Configuration Via Protocol (CvP, e.g. PCIe)
- Targets multi-modal applications
  - Coarse-grained reconfiguration

Partial Reconfiguration Sequence

- Bring-up the chip with a known configuration
- Make decision to reprogram reconfigurable components
- Issue “PR request”
- Send “PR data”
  - Partial bitstream generated by Quartus
- Receive “PR ready”
- Rest of logic can now interact with reconfigured component



Sample Design File Content

```
module reconfig_channel (clk, in, out);
input clk, in;
output [7:0] out;

parameter VER = 2;  // 1 to select 10GbE, 2 to select OTN2

generate
        case (VER)
        1: gige m_gige (.clk(clk), .in(in), .out(out));
        2: otn2 m_otn2 (.clk(clk), .in(in), .out(out));
        default: gige m_gige(.clk(clk), .in(in), .out(out));
        endcase
endgenerate

endmodule
```

Stratix V Device Floorplan

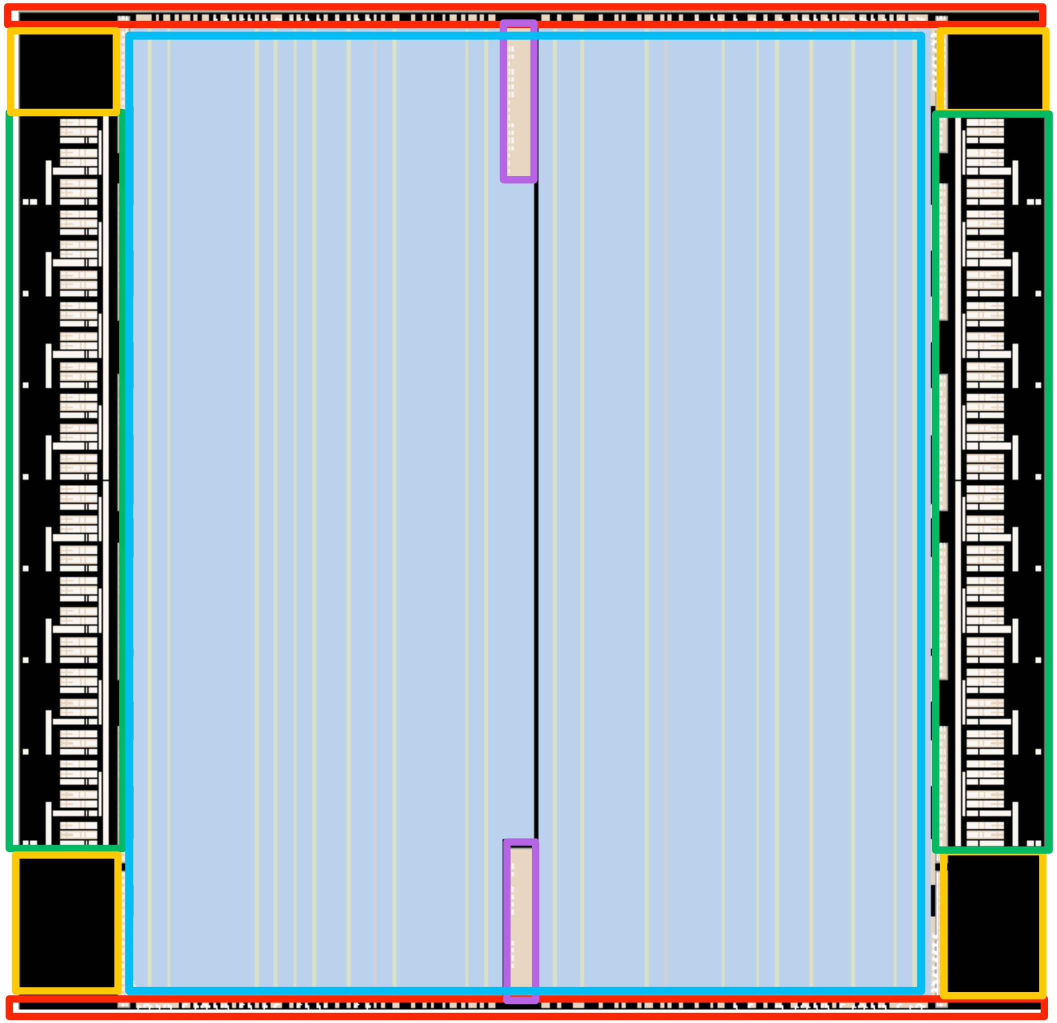
🟥 I/O, Memory Interfaces, Clocks, etc…

🟨 PLL, JTAG, CRC, control circuitry, etc…

🟩 Transceivers, Hard IP Blocks

🟦 LAB, RAM, DSP

🟪 PLLs, Clocks



Reconfiguration Modes Per Resources

Resources	Reconfiguration Type
Logic Block (LAB)	**Partial Reconfiguration**
Digital Signal Processing (DSP)	**Partial Reconfiguration**
Memory Block (RAM)	**Partial Reconfiguration**
Transceivers	**Dynamic Reconfiguration (ALTGX_Reconfig)**
PLL	**Dynamic Reconfiguration (ALTPLL_Reconfig)**
IO Blocks & Other Periphery	**Altera Memory Interface IP**

Configuration RAM Bits

- FPGA is fixed HW with millions of 1-bit memory cells (CRAM)
  - Core bits are directly addressable by Frame (column) & Index (row)
  - Large devices have >10^7 CRAMs
- By changing CRAM values, can reconfigure individual LABs, RAMs, DSPs, and *routing muxes*
  - Not all CRAM sequences are legal
  - Quartus will not provide details about CRAM → logical function

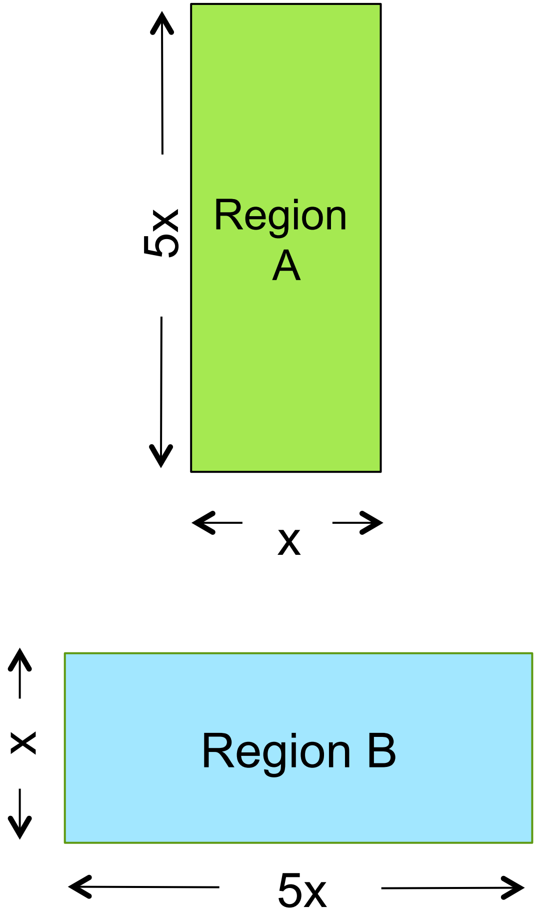
CRAM address space	Frame 1	Frame 2	• • • •		Frame m	Frame m+1	Frame m+2	• • • •			Frame n	Frame n+1	Frame n+2	• • • •		Last Frame
Bit 1					▨	▨	▨				▨	▨	▨			
Bit 2					▨	▨	▨				▨	▨	▨			
• • • •					▨	▨	▨				🟨	🟨	🟨			
					▨	▨	▨				🟨	🟨	🟨			
					▨	▨	▨				🟨	🟨	🟨			
Bit i					🟨	🟨	🟨				🟨	🟨	🟨			
Bit i+1					🟨	🟨	🟨				🟨	🟨	🟨			
• • • •					🟨	🟨	🟨				🟨	🟨	🟨			
					🟨	🟨	🟨				▨	▨	▨			
Bit i+j-1					🟨	🟨	🟨				▨	▨	▨			
Bit i+j					🟨	🟨	🟨				▨	▨	▨			
					▨	▨	▨				▨	▨	▨			
Last Bit					▨	▨	▨				▨	▨	▨			

▨ Non-PR Region

🟨 PR Region

Basic PR Operation

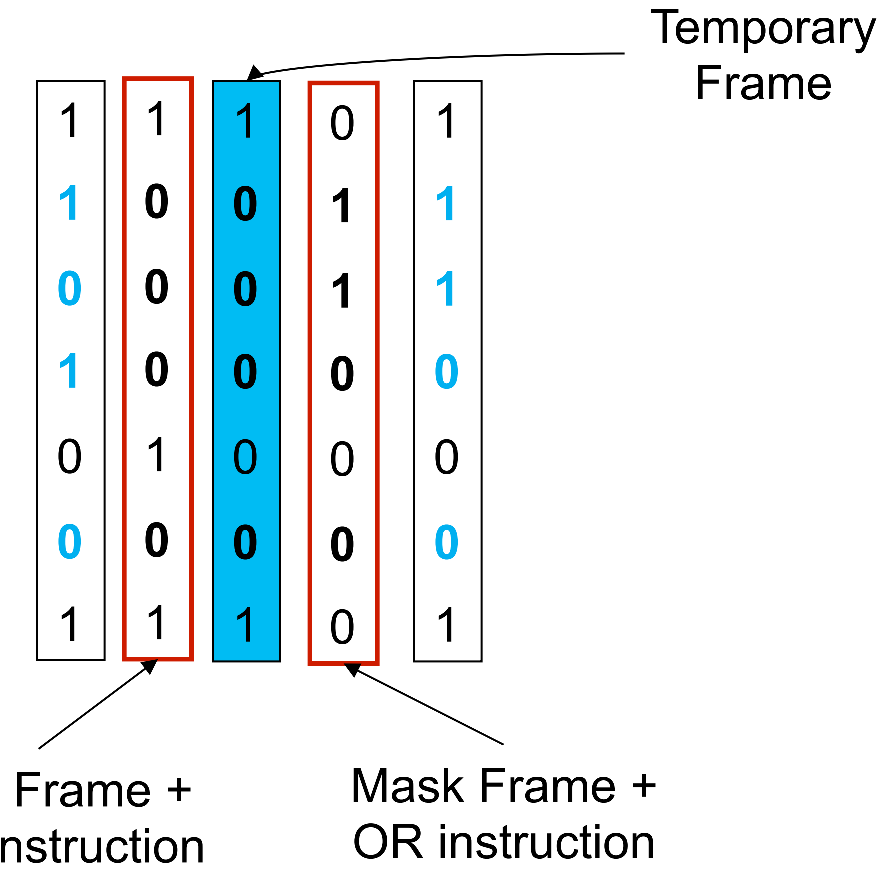
- Use handshaking commands
  - From user I/O or from core logic
- Speed of operation
  - Small overhead to engage PR
  - Depends on size of region
  - Depends on orientation of region
- Sample configuration times
  - ~5kLEs
  - Region A ~ 2ms
  - Region B ~ 10ms
  - 2ms @ 400 MHz = 800,000 cycles



© 2011 Altera Corporation—**Public**

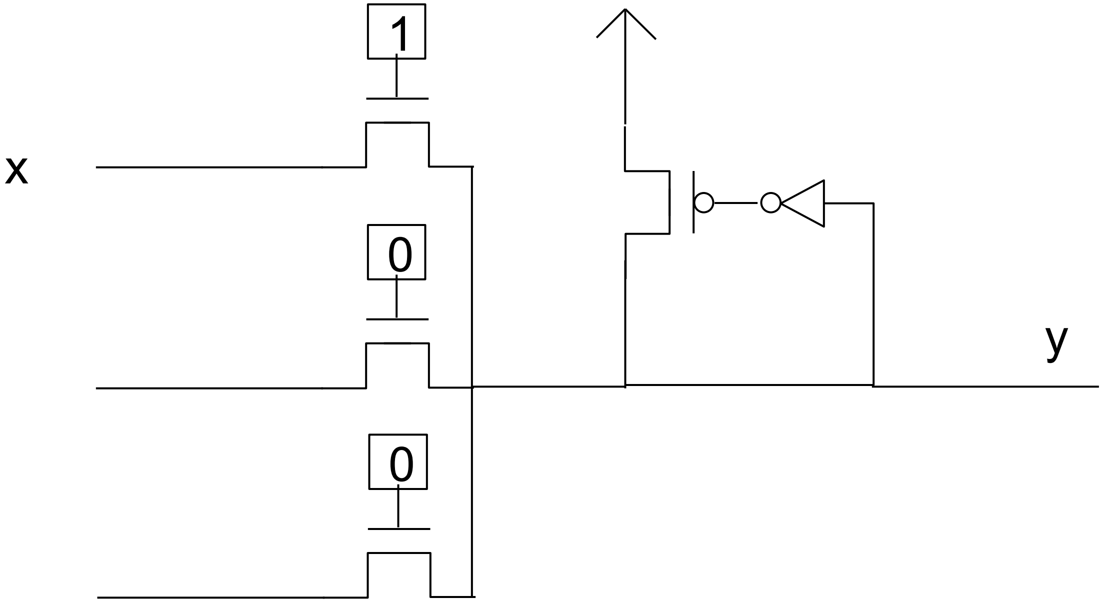
Changing CRAM Bit State

- For each affected frame, control block generates
  - An AND instruction to set reconfigured bits to zero
  - An OR instruction to set them to one



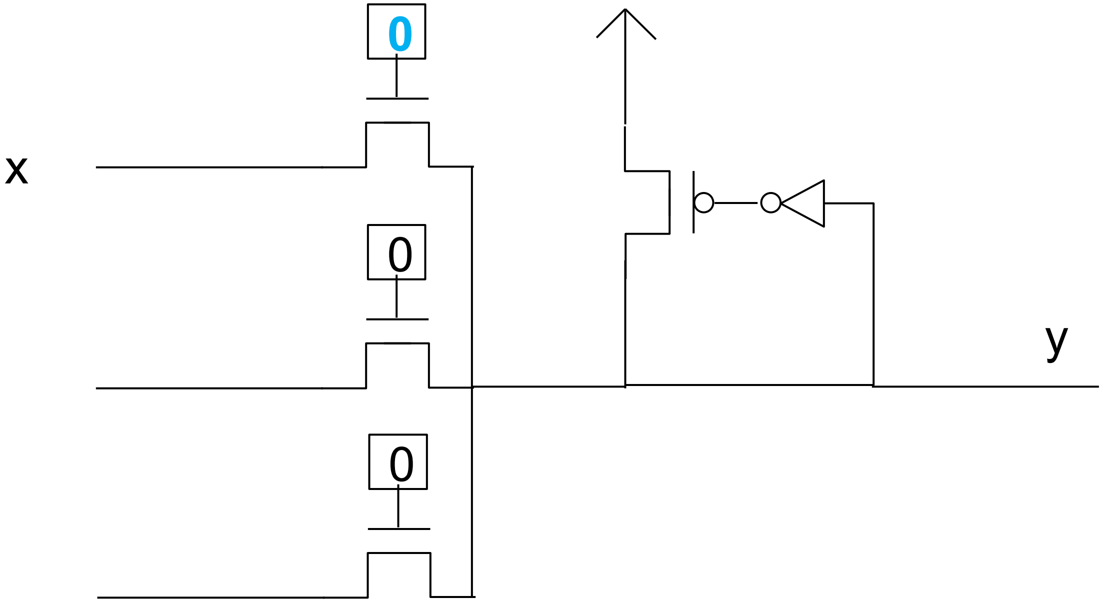
Implications of Temporary Frame

- Must carefully consider electrical effects
  - NMOS-based logic is very common
  - Requires weak pull-up transistor to strengthen internal node when input signal is VDD
  - Global “freeze” signal can’t be used during PR



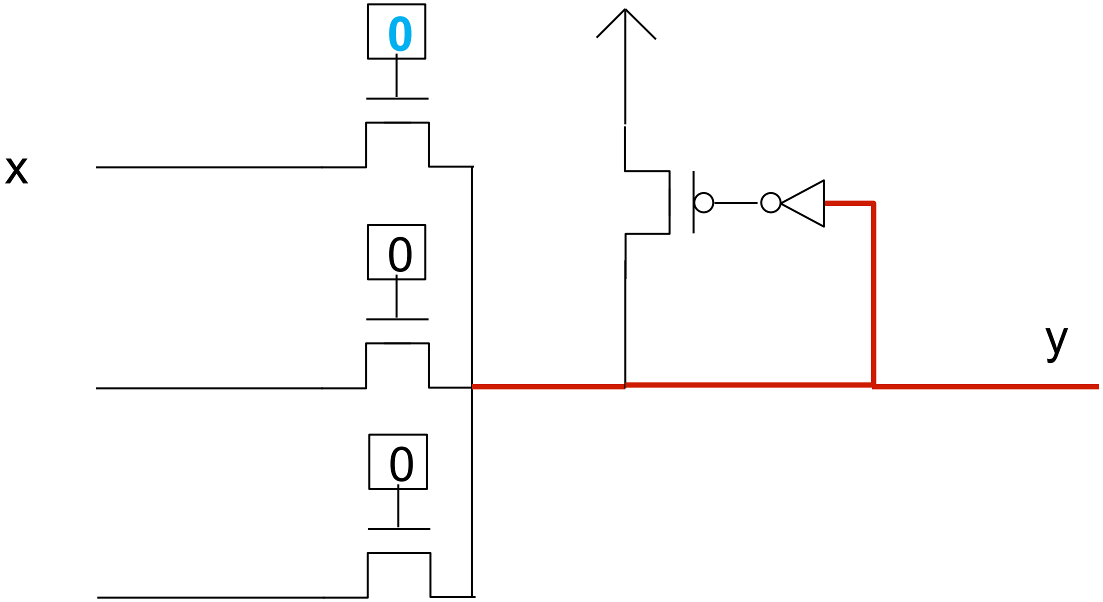
Implications of Temporary Frame

- ‘AND’ instruction may cause all CRAM bits of a routing driver to be zero
  - No electrical issues if ‘x’ is VDD at the time when the CRAM bit flips from a ‘1’ to a ‘0’

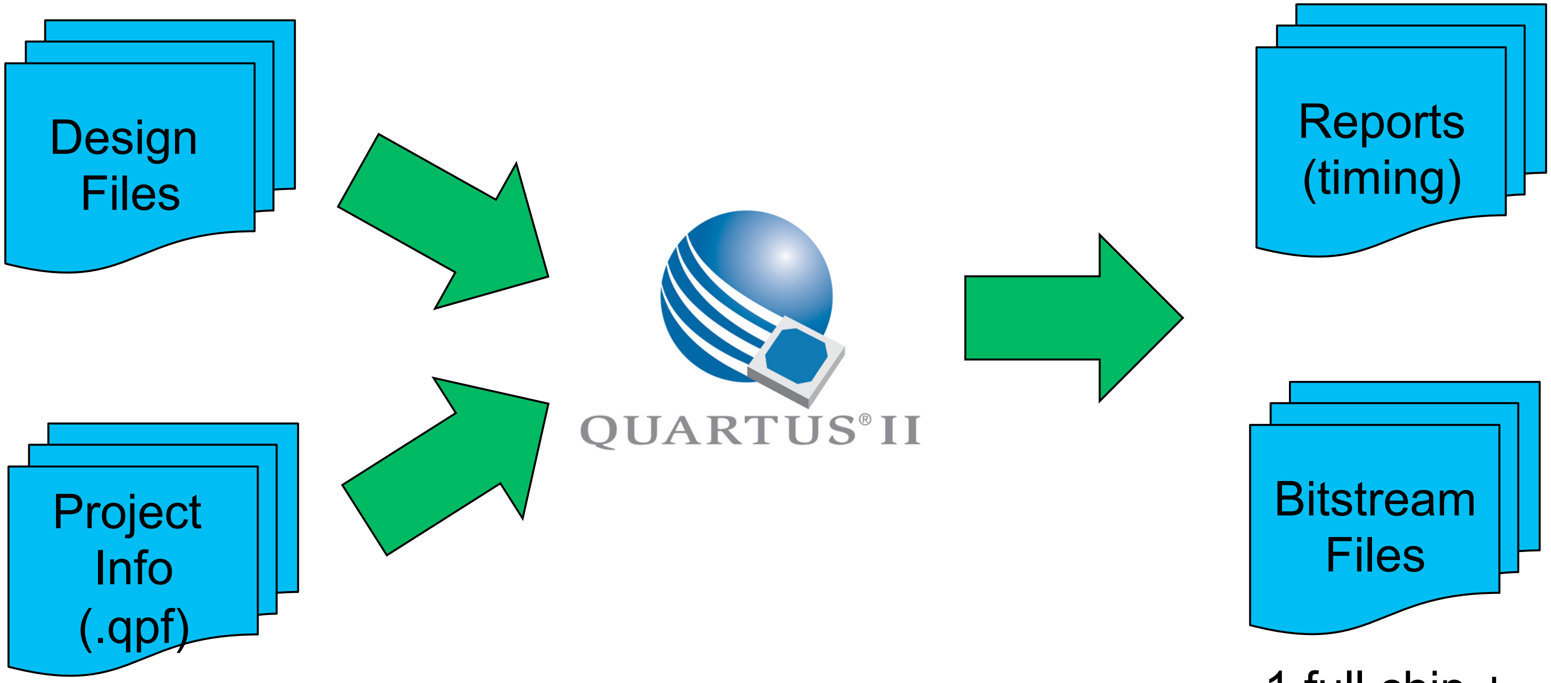


Implications of Temporary Frame

- ‘AND’ instruction may cause all CRAM bits of a routing driver to be zero
  - However, if ‘x’ is GND when the CRAM flips, voltage on ‘y’ can drift causing unwanted current



Partial Reconfiguration Inputs & Outputs



Design Files

Project Info (.qpf)

QUARTUS® II

Reports (timing)

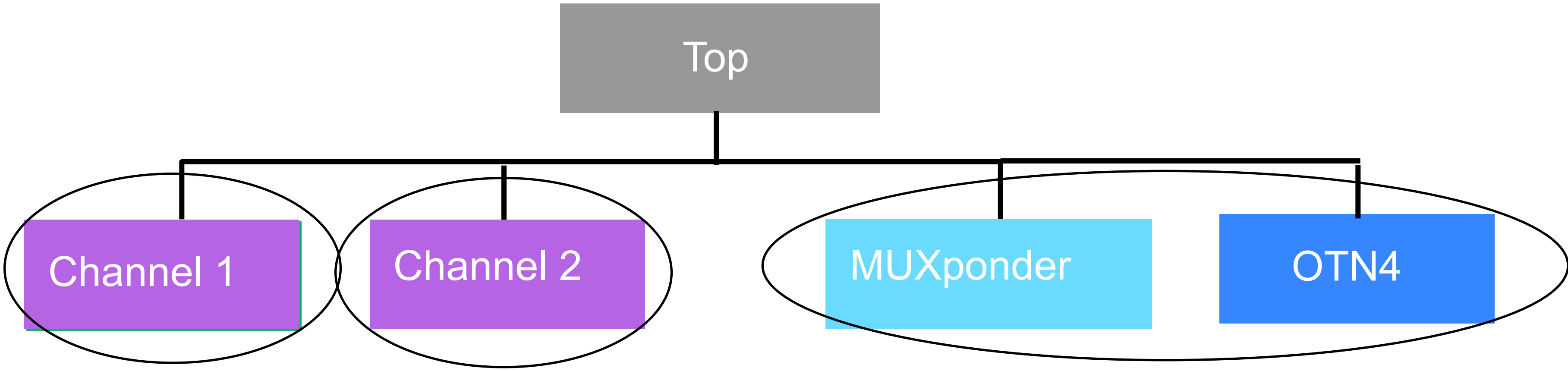
Bitstream Files

1 full chip +
1 per persona

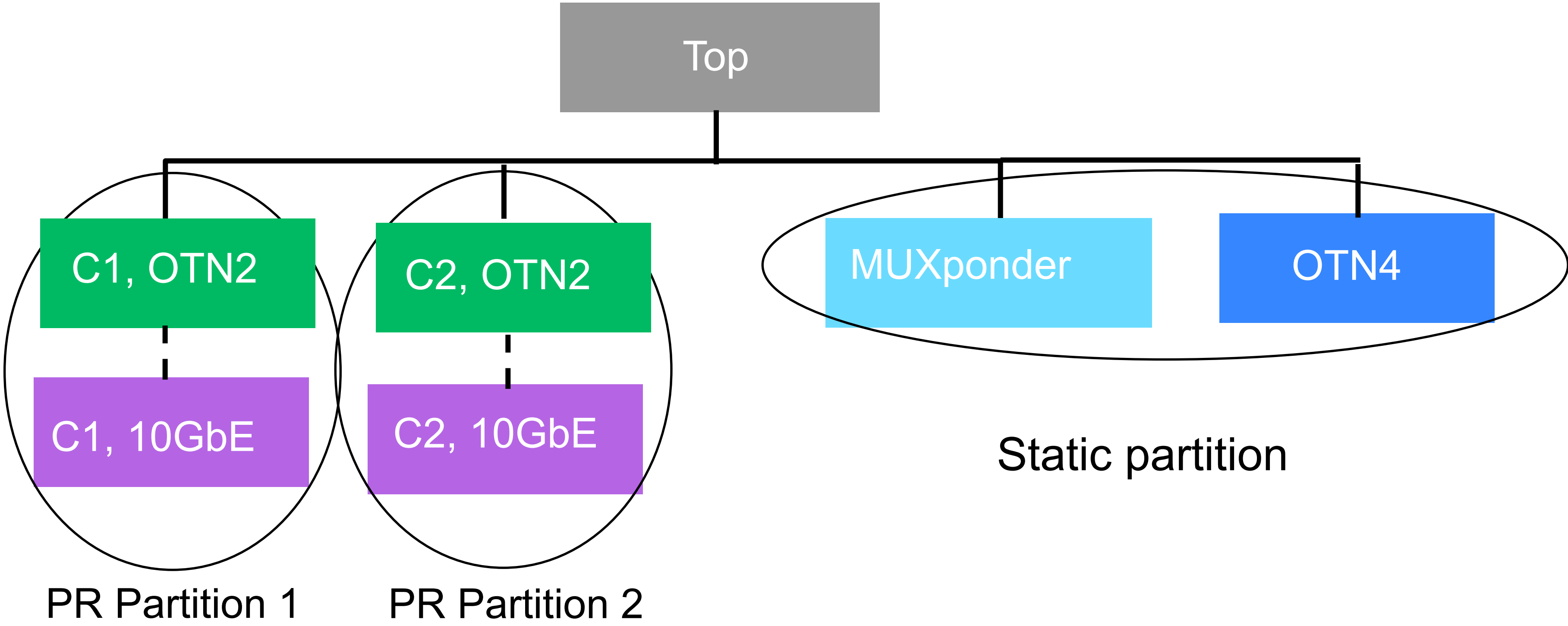
Multiple Compiles Needed!

Incremental Design Background

- Specify *partitions* in your design hierarchy
- Can independently recompile any partition
  - CAD optimizations across partitions prevented
  - Can preserve synthesis, placement and routing of unchanged partitions
  - Supports multiple processors for faster compile

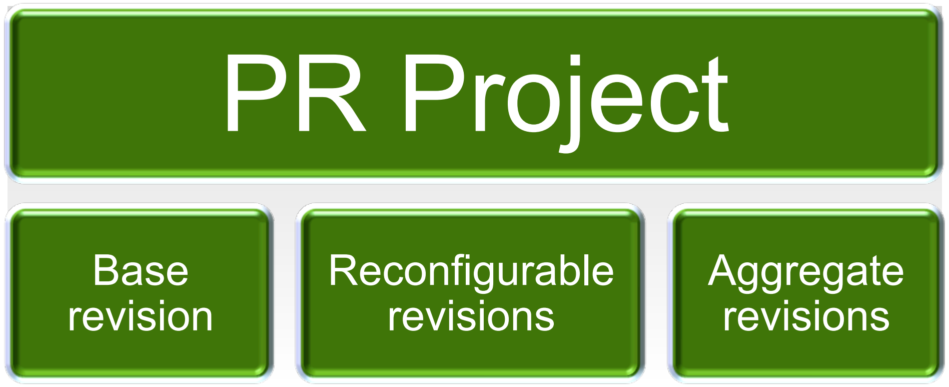


Reconfigurable Instances



- Each design instance in a reconfigurable partition is called a “persona”
  - Above design hierarchy has 4 personae (C1 x 2, C2 x 2)
  - PR partitions must be assigned to fixed regions on the chip

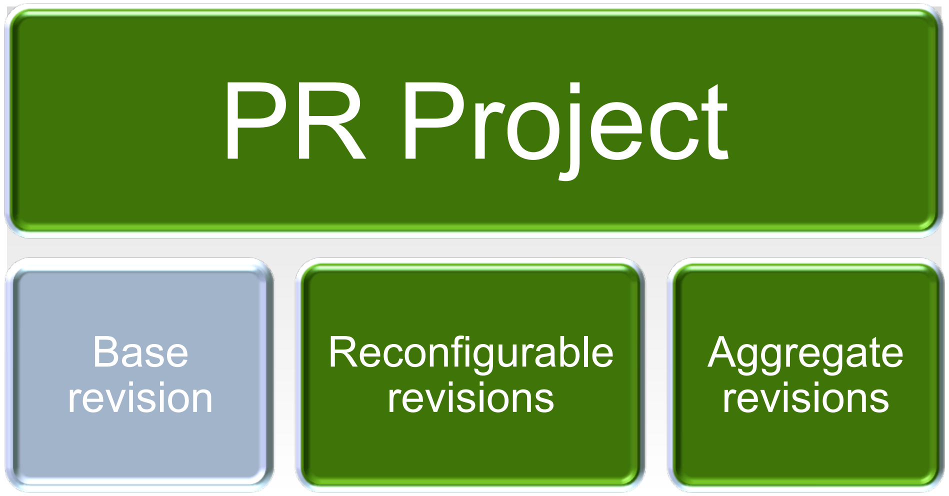
Partial Reconfiguration Project

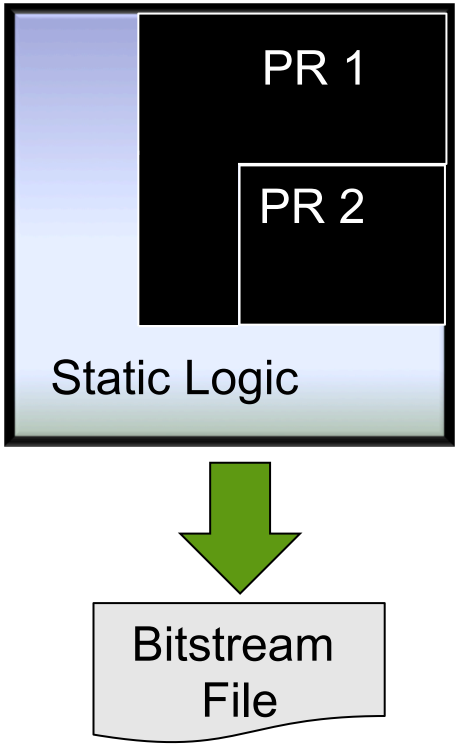


- Revision = set of input files for a single compile
  - Can change parameters, timing constraints, fitter settings, etc…
- Revision types in a PR project
  - Base: Content that always exists on the chip (a.k.a. static compile)
  - Reconfigurable: Content that can be programmed via PR commands
  - Aggregate: User-defined groupings of reconfigurable revisions

    (consider 10 PR partitions with 6 personae each, 10^6 possible combos)
- No limit to the # of regions or # of personae / PR region

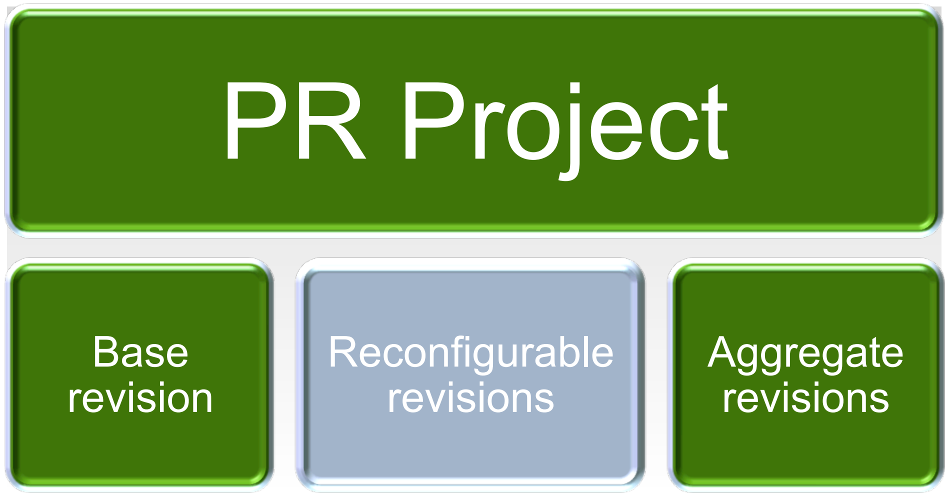
Partial Reconfiguration Project





- #1: Compile static logic
  - User can specify PR1 & PR2 for base revision
  - Can be empty, initial content, or “hardest-to-route” personae

Partial Reconfiguration Project

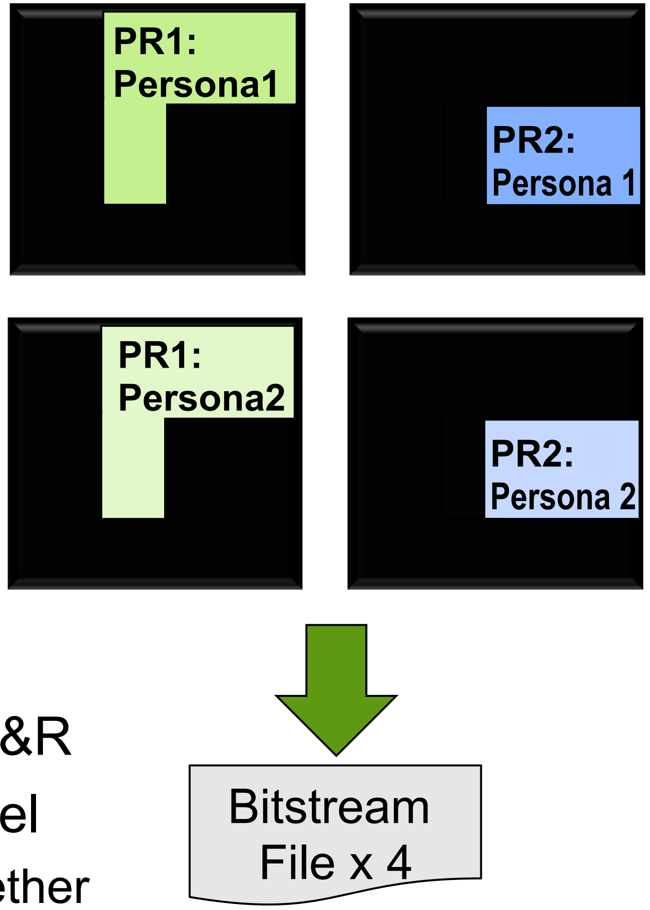


PR Project

Base revision

Reconfigurable revisions

Aggregate revisions



PR1: Persona1

PR2: Persona 1

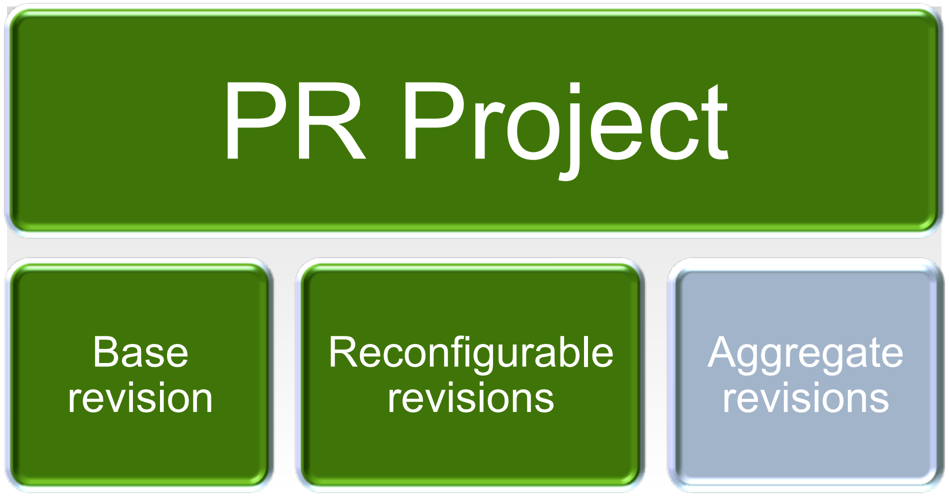
PR1: Persona2

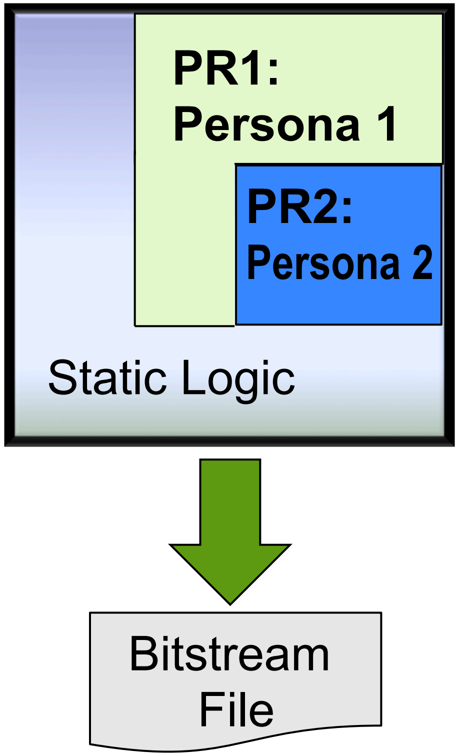
PR2: Persona 2

Bitstream File x 4

- #2: Compile reconfigurable logic
  - Personae compile imports base revision P&R
  - Quartus can compile all personae in parallel
    - Combine personae in different regions together
    - # compiles = max # personae across all regions

Partial Reconfiguration Project

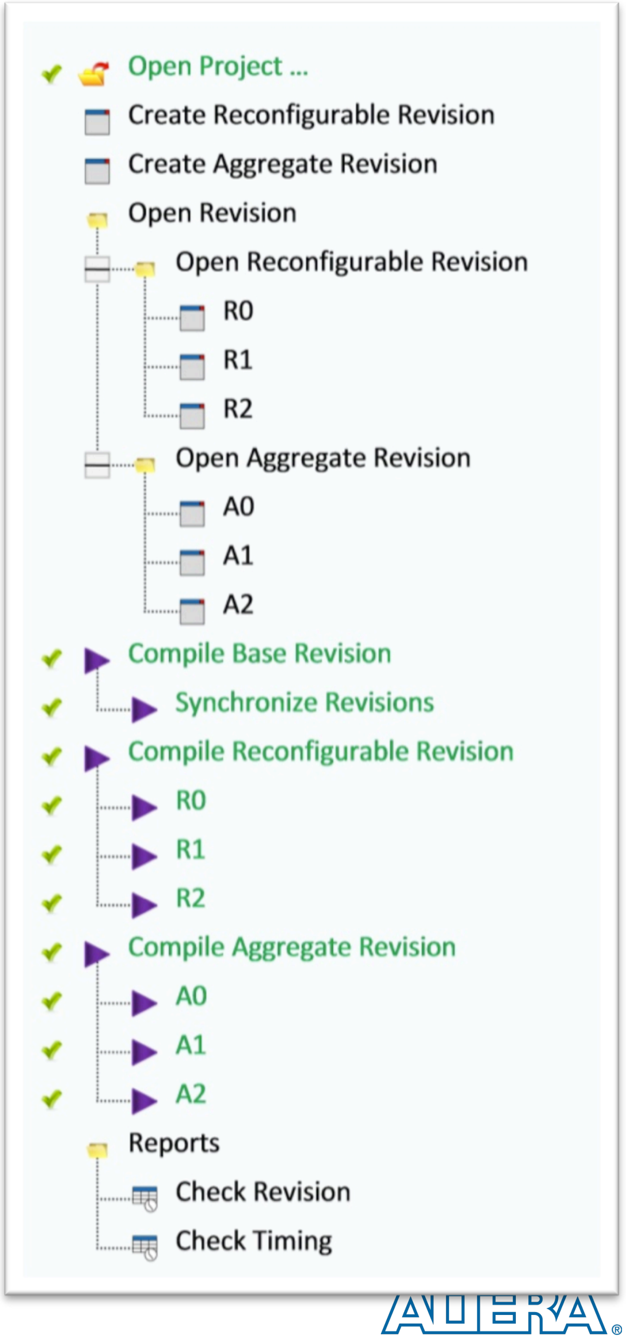




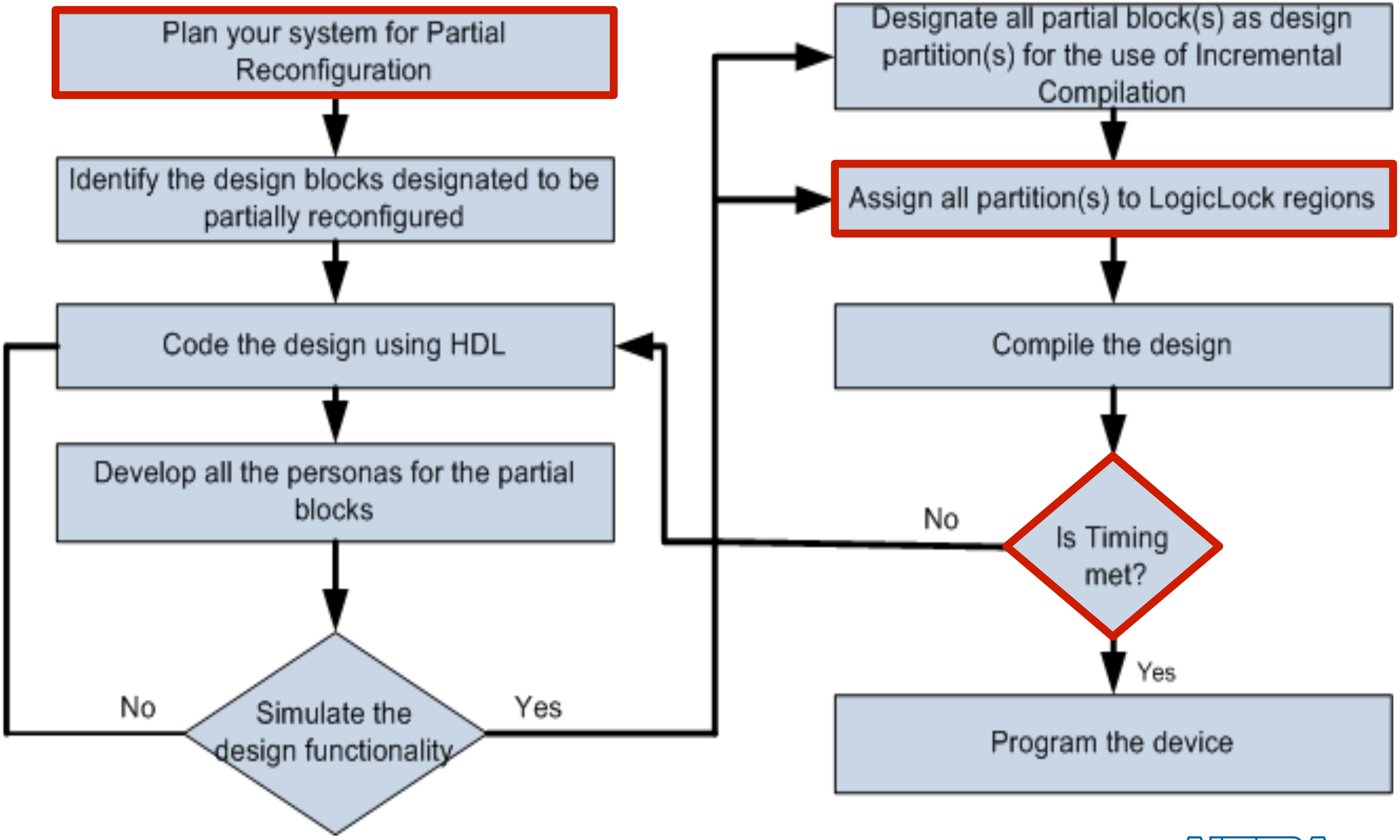
- #3: Compile user-defined groups of reconfigurable logic
  - This step is completely optional
  - Generates complete bitstream, without requiring PR commands
  - Useful for timing analysis / simulation

Managing Revisions

- Quartus contains dedicated GUI to navigate PR project
  - Can directly launch “regular” Quartus GUI for a specific revision
  - Can easily create new revisions
  - Can “compile all” via single click
  - Can obtain summary of all compiles



Design Flow

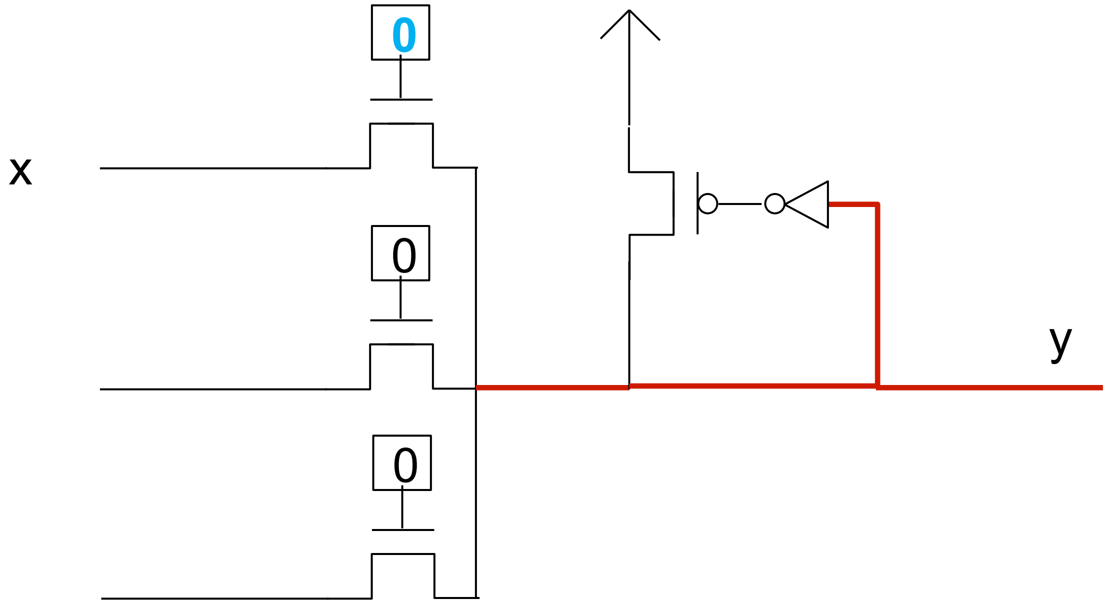

Plan your system for Partial Reconfiguration
Identify the design blocks designated to be partially reconfigured
Code the design using HDL
Develop all the personas for the partial blocks
Simulate the design functionality
No
Yes
Designate all partial block(s) as design partition(s) for the use of Incremental Compilation
Assign all partition(s) to LogicLock regions
Compile the design
Is Timing met?
No
Yes
Program the device

PR Introduces New Software Challenges

- Handling inputs/output signals to PR partitions
  - PR inputs/outputs must stay in same spot in all compiles
- Partitioning routing resources across all compiles
  - Different PR partitions cannot use same routing resource
  - Different personae in same PR region **can** use same routing wires
- Utilizing shared FPGA resources
  - E.G. pre-fabricated clock networks
- Timing closure on cross-partition paths
  - E.G. reg in PR1 → reg in PR2

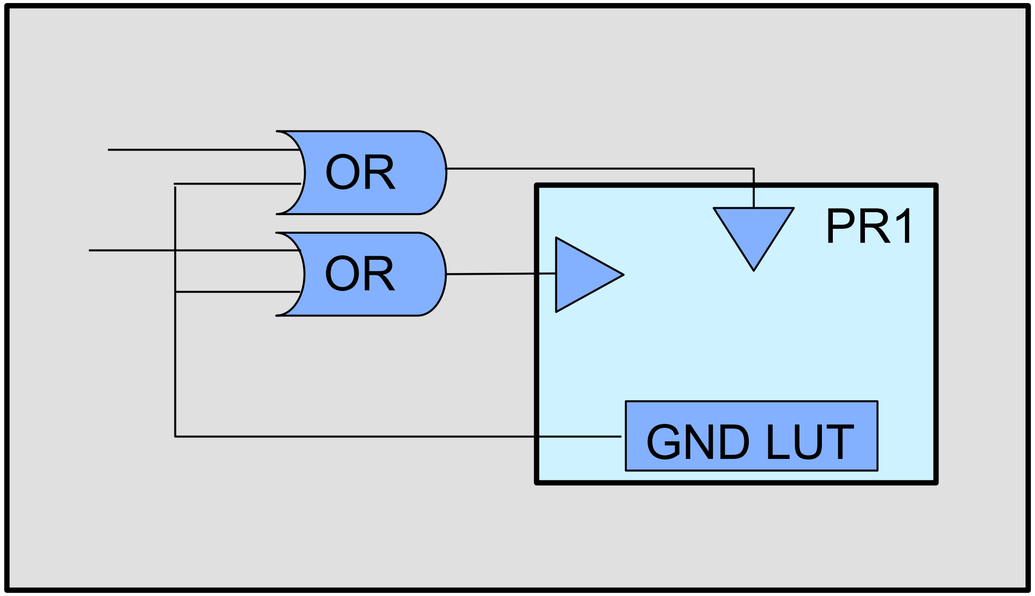
PR Partition Input Signals

- Recall electrical problem with GND inputs
  - Option 1: User “guarantees” inputs will be VDD when PR engages
  - Option 2: Let compiler solve the problem automatically



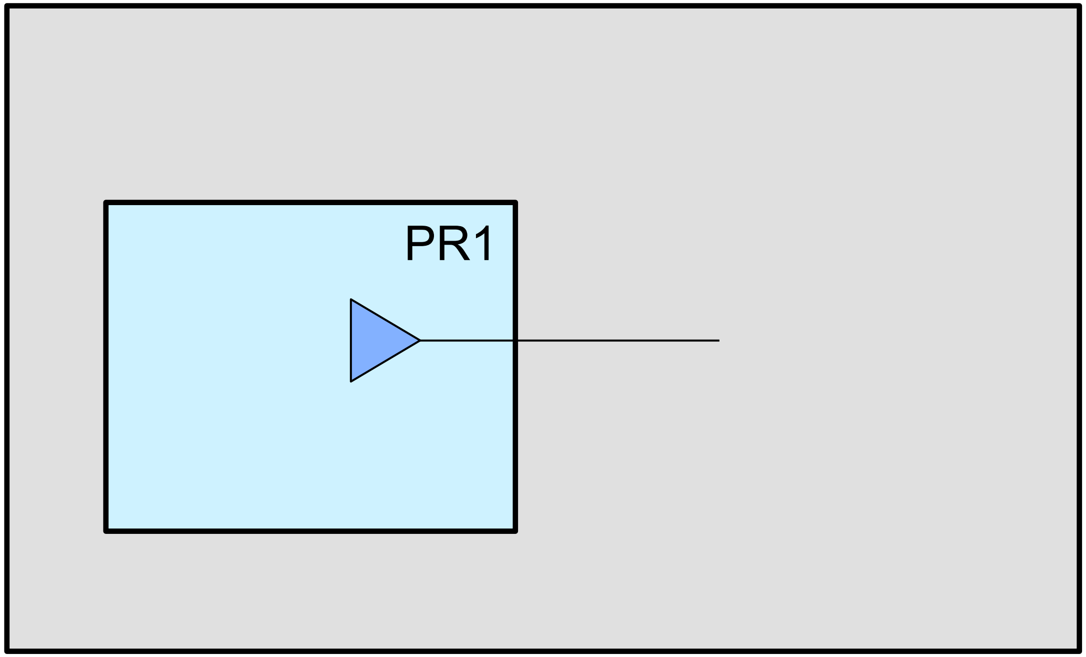
PR Partition Input Signals

- For each PR partition, reserve one LUT for GND
- For each input signal into a PR partition,
  - Add OR gate **outside PR partition** and connect GND LUT to other input
  - Connect OR gate output to a 1-input anchor LUT inside PR partition
- This connectivity is inserted during the base revision



PR Partition Output Signals

- For each output signal,
  - Create anchor 1-LUT inside PR partition
  - Connect output to logic in static region, according to design files

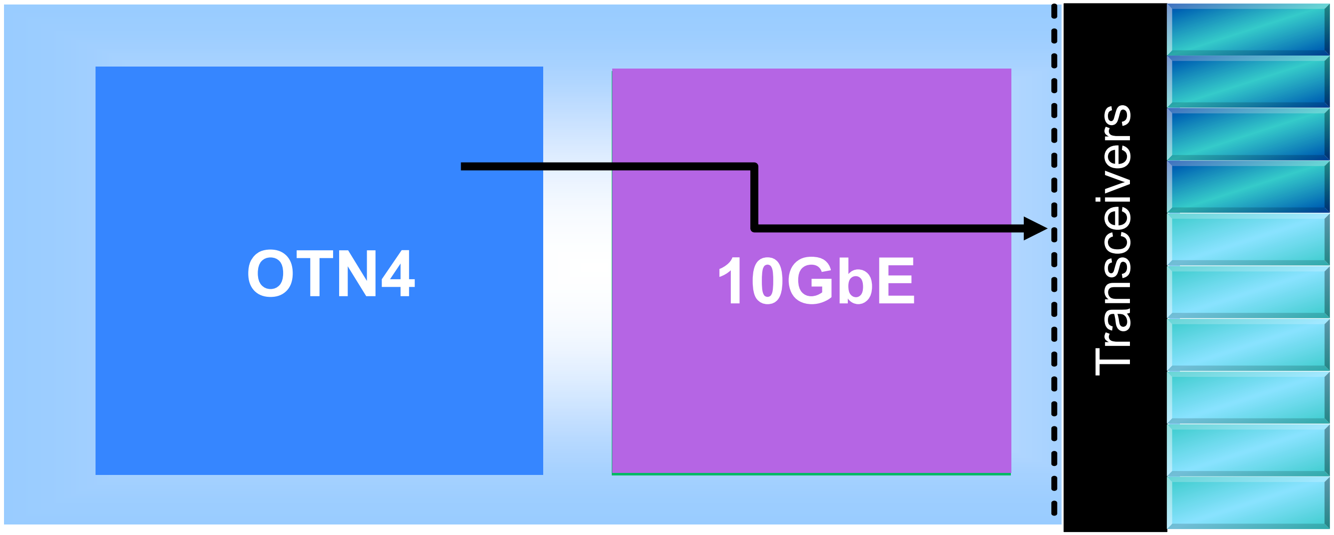


Using Anchor 1-LUTs in PR Partitions

- (+) Increases routing flexibility on inputs/outputs
  - FPGA architecture not designed to route to/from specific wires
- (+) P&R is well-defined for each compile
  - Otherwise, how should path from OR gate to logic be chosen?
  - Inputs can branch from the OR gate → 1-LUT path, if desired
- (-) Reduces # of usable LUTs in PR region
- (-) Increases latency on signals entering region

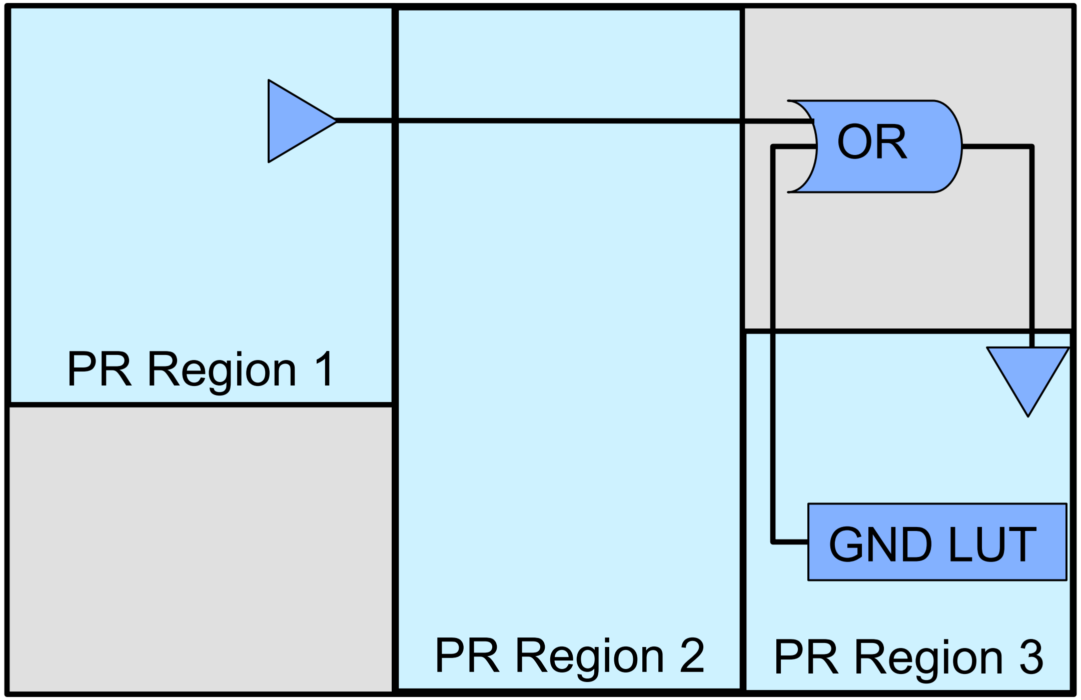
Route-Throughs At The Top Level

- HW supports reconfiguration of individual routing muxes
  - Enables routing through PR regions
  - Simplifies / removes many floorplanning restrictions
  - Quartus records routing reserved for top-level use and prevents PR instances from using these wires



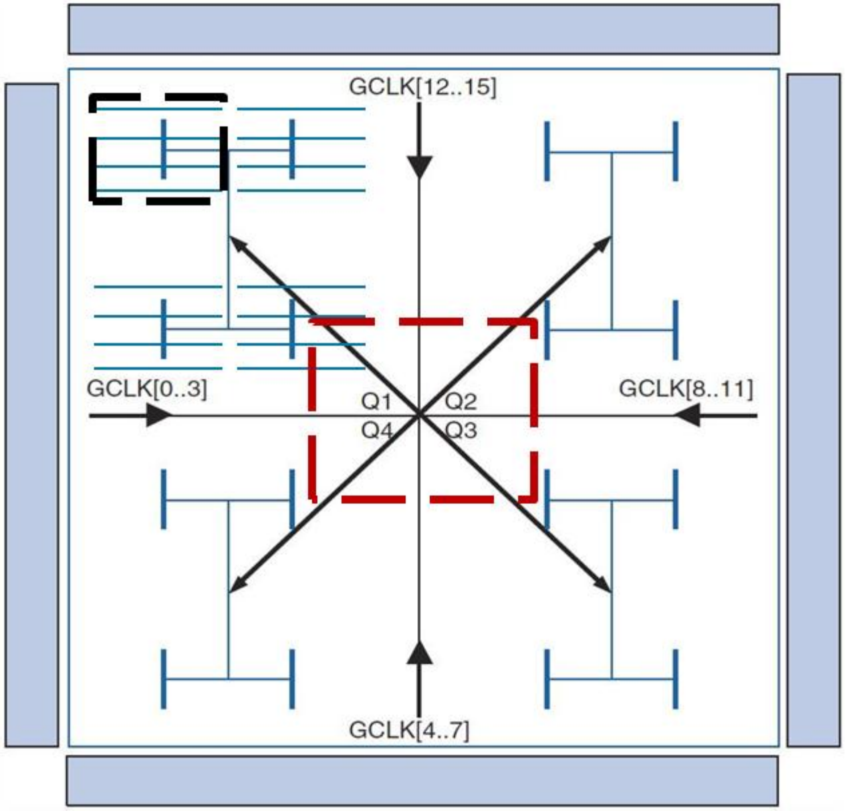
Inter-Region Communication

- SW design decisions make this easy to support
  - Anchor LUTs & inserted OR gate are routed in the static compile
  - Resources used by the static compile are disallowed by all personae compilations

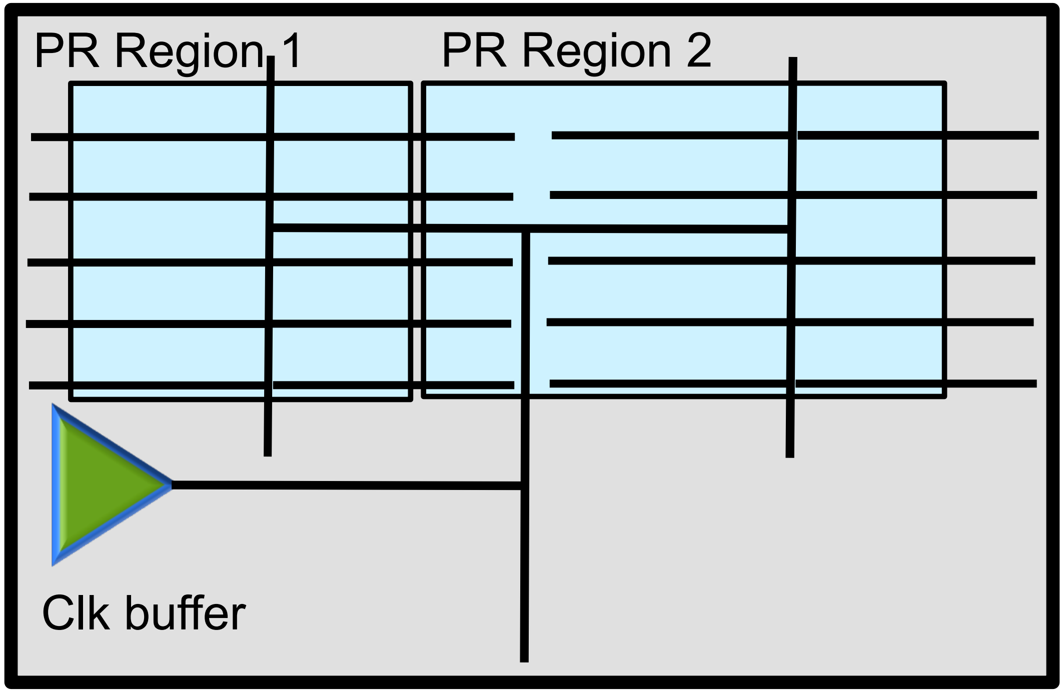


Global Networks On Stratix V Devices

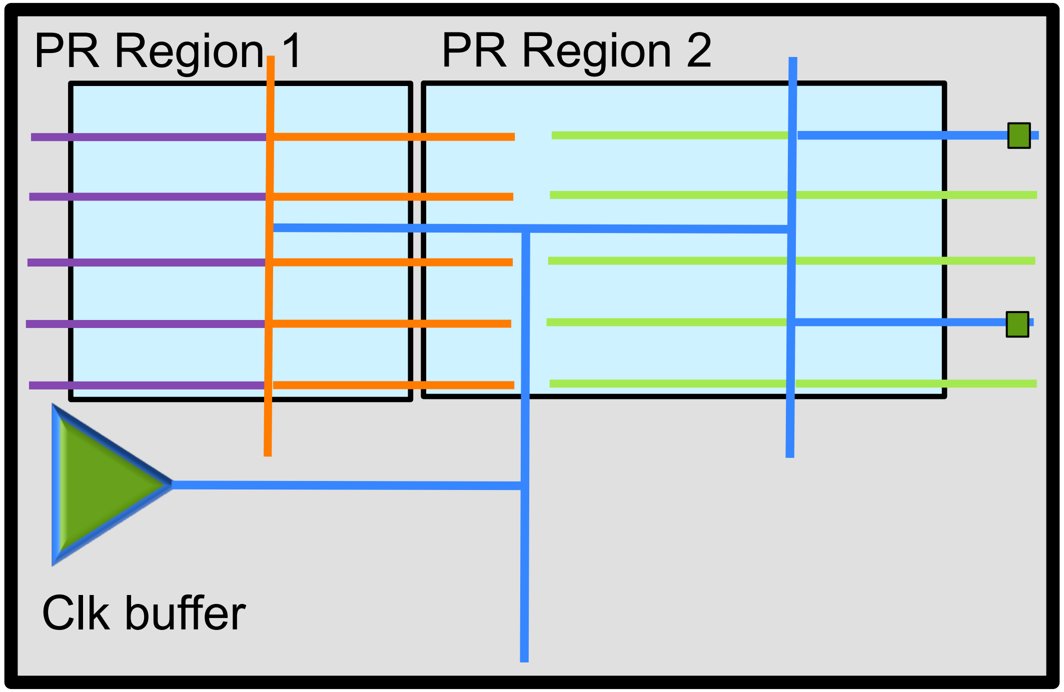
- 16 global networks
  - Balanced H-tree
  - Reaches every block
  - Only 6 unique clocks can drive each "row clock region" (~12x1)
- PR designs should consider clocking regions
  - Match region size to clock boundaries = easier problem
  - Quartus supports any PR region
  - P&R engine assumes clock feeds every location in PR region



Sharing Global Wires Between Regions

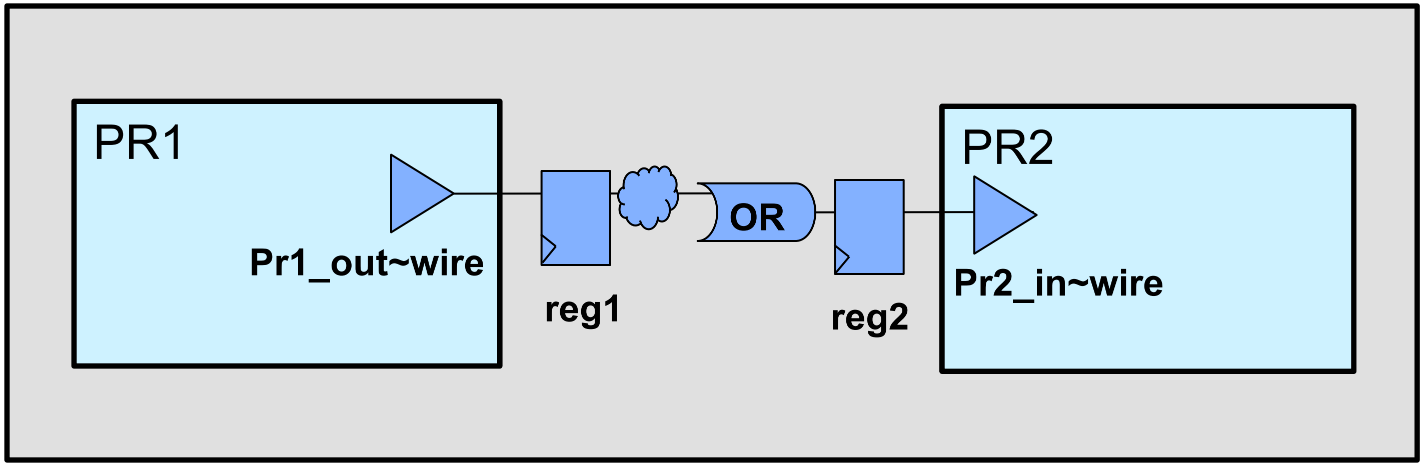


Sharing Global Wires Between Regions



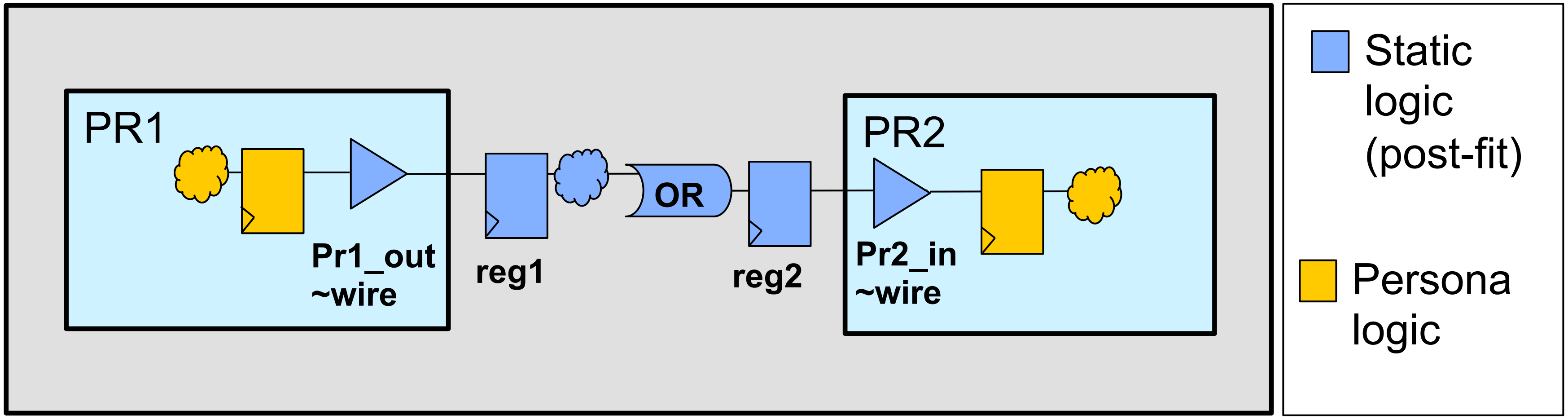
🟩	Static logic
🟦	Wires in the base revision bitstream
🟩	Wires that may used in a persona bitstream targeting region 2
🟪	Wires that may used in a persona bitstream targeting region 1
🟧	Wires that must be in the base revision bitstream because any persona in region 1 or 2 may use them

Simple Timing Closure (Base Compile)



- Add registers at all PR partition boundaries
  - Fitter routes intra-region connections for minimum delay
    - pr1_out~wire → reg1
    - reg2 → pr2_in~wire
  - 📫 OR gate can be placed before ‘reg2’ (assuming no clock enable)

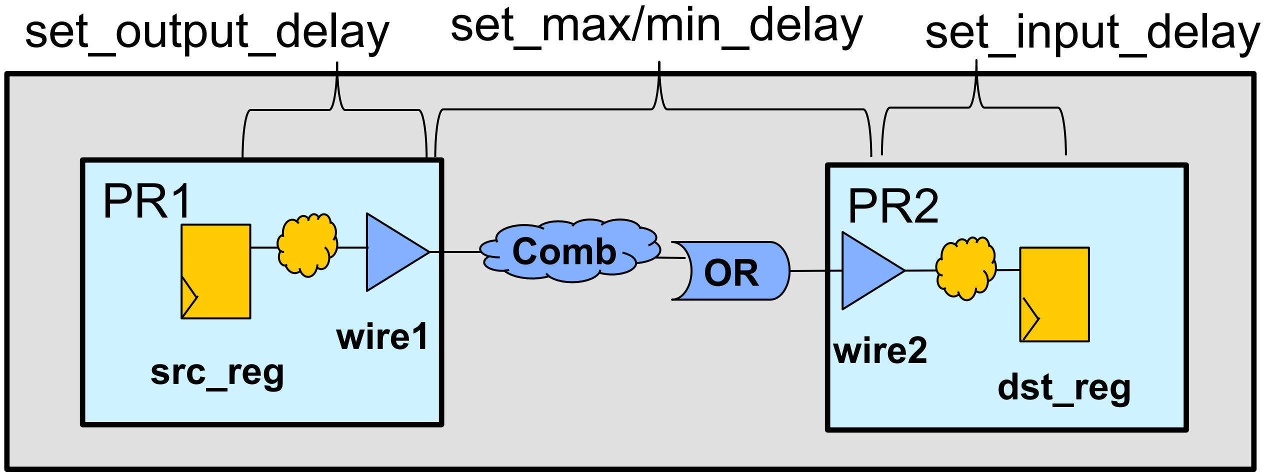
Simple Timing Closure (Persona Compile)



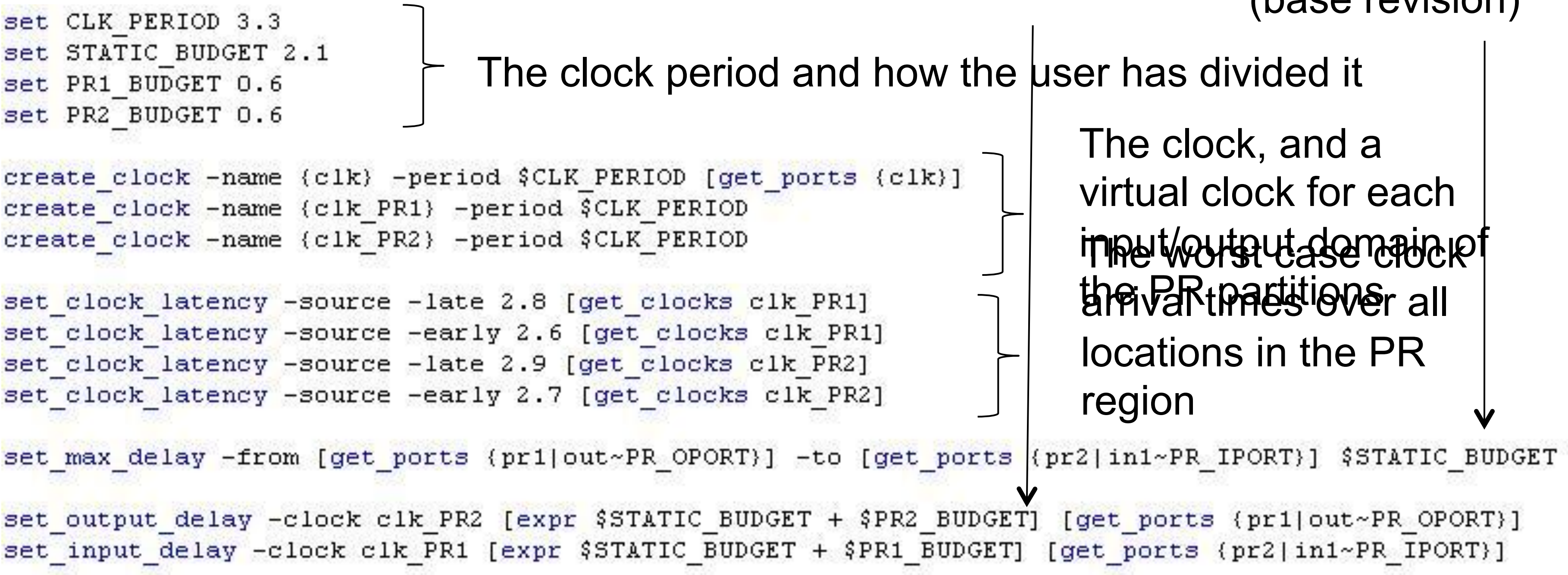
- Add registers at all PR partition boundaries
  - Compiler sees reg->reg paths and optimizes accordingly
  - Each path has at least 1 “free” connection in persona compiles
    - Can add more delay, if required, to meet hold timing constraints

Advanced Timing Closure

- User explicitly manages clock period
  - Divide into three segments
  - Specify timing constraints for each segment in SDC file



The SDC File



```
set CLK_PERIOD 3.3
set STATIC_BUDGET 2.1
set PR1_BUDGET 0.6
set PR2_BUDGET 0.6

create_clock -name {clk} -period $CLK_PERIOD [get_ports {clk}]
create_clock -name {clk_PR1} -period $CLK_PERIOD
create_clock -name {clk_PR2} -period $CLK_PERIOD

set_clock_latency -source -late 2.8 [get_clocks clk_PR1]
set_clock_latency -source -early 2.6 [get_clocks clk_PR1]
set_clock_latency -source -late 2.9 [get_clocks clk_PR2]
set_clock_latency -source -early 2.7 [get_clocks clk_PR2]

set_max_delay -from [get_ports {pr1|out~PR_OPORT}] -to [get_ports {pr2|in1~PR_IPORT}] $STATIC_BUDGET

set_output_delay -clock clk_PR2 [expr $STATIC_BUDGET + $PR2_BUDGET] [get_ports {pr1|out~PR_OPORT}]
set_input_delay -clock clk_PR1 [expr $STATIC_BUDGET + $PR1_BUDGET] [get_ports {pr2|in1~PR_IPORT}]
```

Timing budget for PR compiles

Timing budget for the static region (base revision)

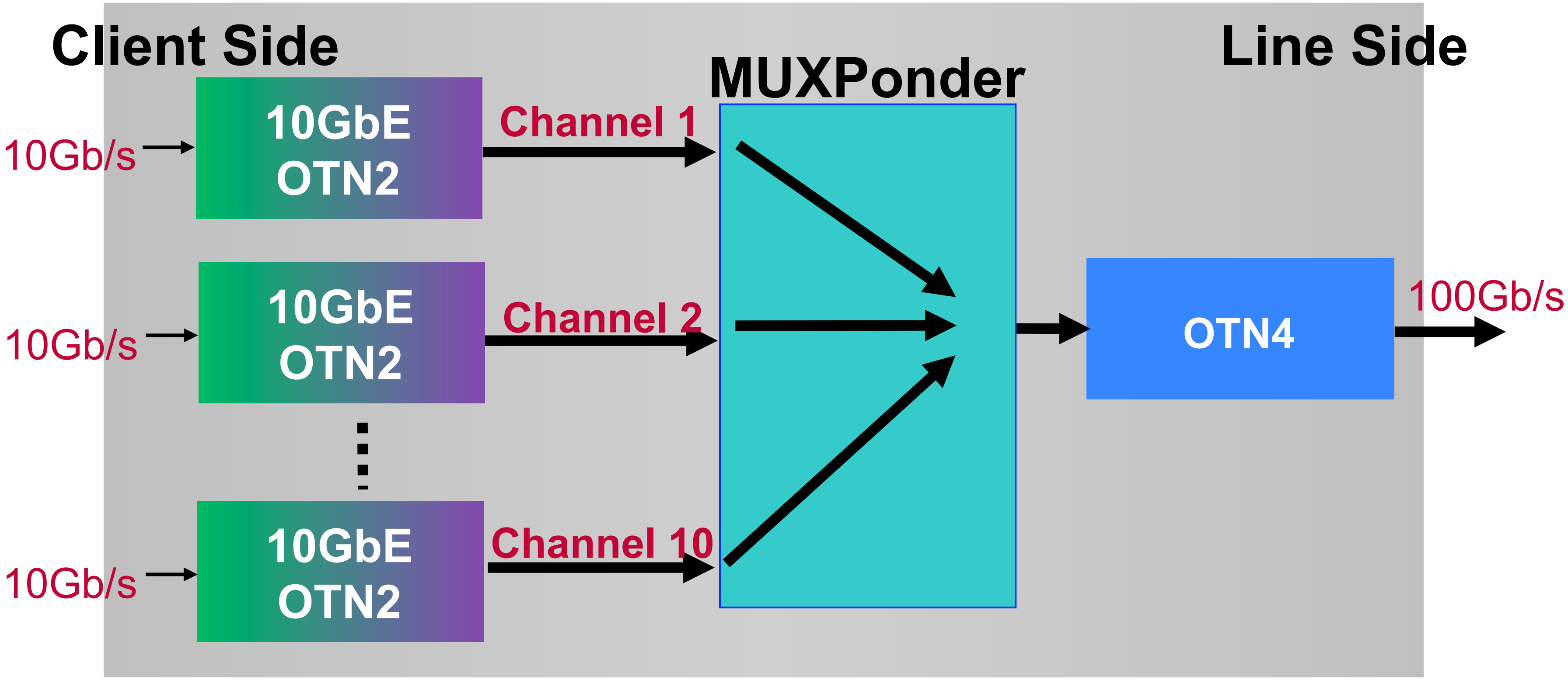
The clock period and how the user has divided it

The clock, and a virtual clock for each input/output domain of the PR partitions

The worst case clock arrival times over all locations in the PR region

- User will be provided the worst-case clock arrival times for each global signal entering PR region
  - Will be known at the end of the base revision
  - Generally, these numbers are a function of device/speedgrade

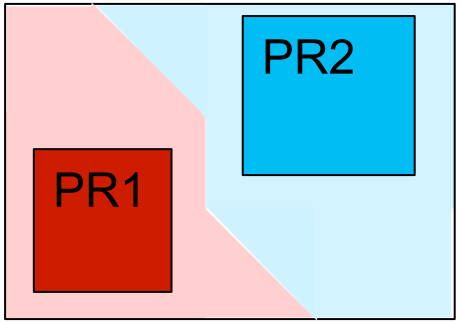
Upcoming Idea: Translatable Bitstreams



- 10 PR regions, 2 personae per PR region
  - But, there's only 2 unique personae across all regions
  - Users would prefer to generate 2 partial bitstream files, not 20

Upcoming Idea: Resource Negotiation

- Enforcing hard boundaries of routing resources for PR regions may increase fitting difficulty
  - Guarantees legality at the expense of optimization quality
- Option 1: Partition unused routing resources between PR regions
  - Downside: Increases cost of changing ‘base’



- Option 2: Allow PR regions to use all routing resources and iteratively resolve congestion using costs
  - Downside: Increased latency of overall compile
  - Downside: Significantly increases cost of changing ‘base’

Upcoming Idea: Minimize Overhead

- Current SW implementation approach works well for multi-modal applications
  - Overhead << PR region size

- Some advanced applications would want to create hundreds of very small regions
  - SW approach doesn't prohibit this design style
  - However, overhead added by SW reduces becomes limiting factor

Summary

- Altera devices have flexible hardware support for partial reconfiguration
  - Able to reconfigure individual LABs, RAMs, routing muxes
- Altera has provided a well-defined design flow for implementing multi-modal applications using PR
  - Intuitive design entry, simulation, and project cockpit (GUI)
  - User can manage complex timing situations between PR regions
- Quartus automatically handles implementation details efficiently & effectively
  - "OR"-gate insertion
  - Route throughs
  - Cross-region clocking