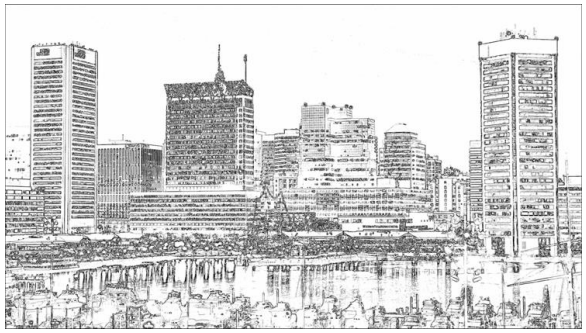
PRESENTED BY

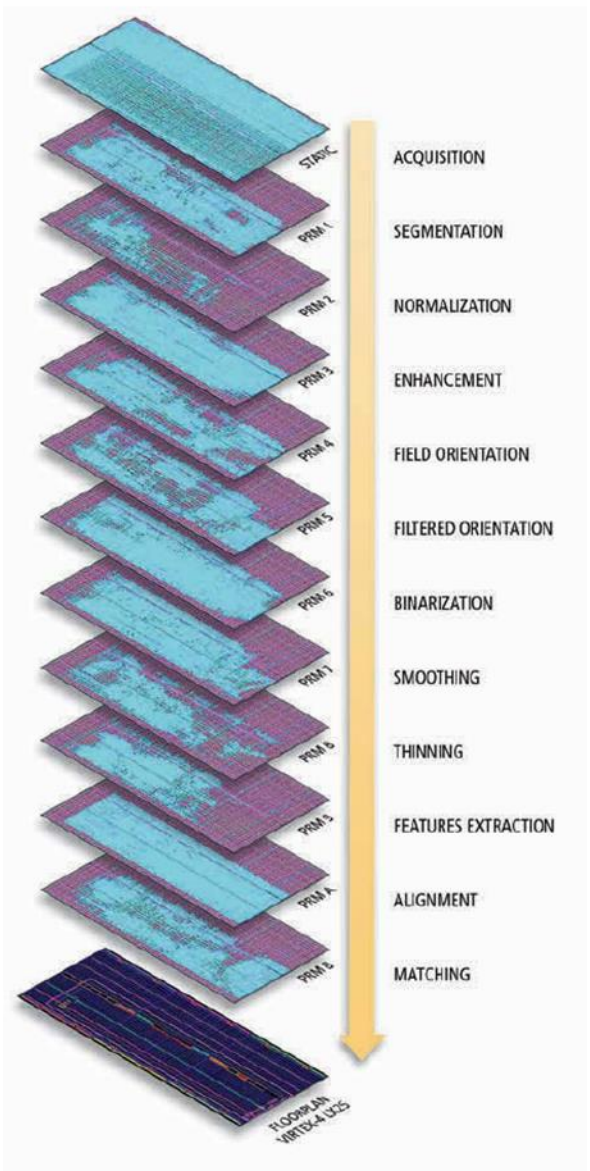
Tom Curran

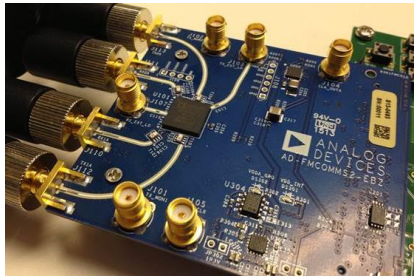
Partial Reconfiguration in Zynq

Accelerating Your Success™

Do More With The Same Hardware







Objectives

- Know the benefits of partial reconfiguration
- Learn the design flow required
- Understand the design considerations required for success with partial reconfiguration

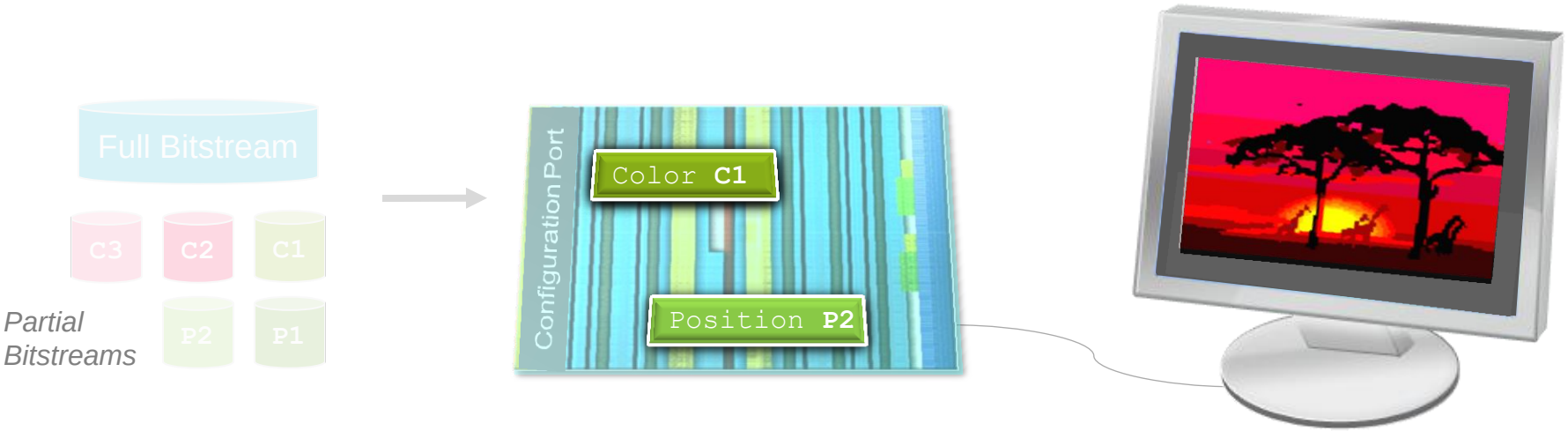
Agenda

- Overview of partial reconfiguration
- Benefits
- Design flow
- Design considerations
- Design examples
- Next steps

Overview

Partial Reconfiguration Technology

Partial Reconfiguration dynamically modifies logic blocks while the remaining logic operates without interruption



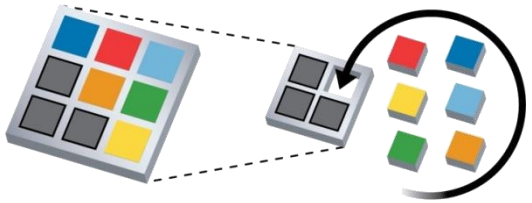
What Problems Does It Solve?

- System cost, size, and power constraints
  - Multiplex hardware functions
- Evolving protocol and industry standards
  - Reprogramability as standards evolve
- Mission critical uptime
  - Update on the fly while system still running
- Long design implementation cycle times
  - Accelerate development with focus on reconfigurable partitions

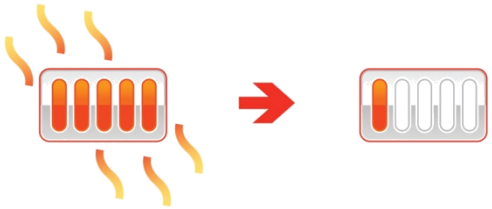
Benefits

Cost, Size, And Power Reduction

- Cost and size reductions
  - Reduced board space requirements
    - Save money on PCB design (NRE costs)
    - Save money in production (BOM costs)
  - Minimizes primary bitstream storage
    - Use a less expensive configuration memory
  - Time multiplexing hardware requires a smaller FPGA

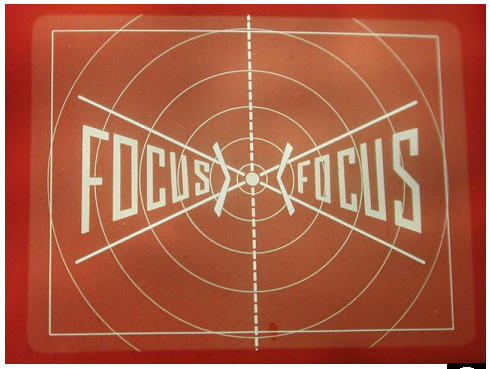


- Power reductions
  - Reduces static power
  - Load functions on demand
  - Swap out power hungry tasks
  - Less expensive power supply



Accelerated Development

- Focus on reconfigurable portions of design
  - No need to rebuild entire design
  - Much shorter implementation cycle
- Introduce fixes and enhancements much faster
- Tweak algorithms and improve system performance
- Analogous to software engineers debugging over NFS and Ethernet
- Greater designer productivity
- Improved design scalability



https://www.flickr.com/photos/editor/283989913

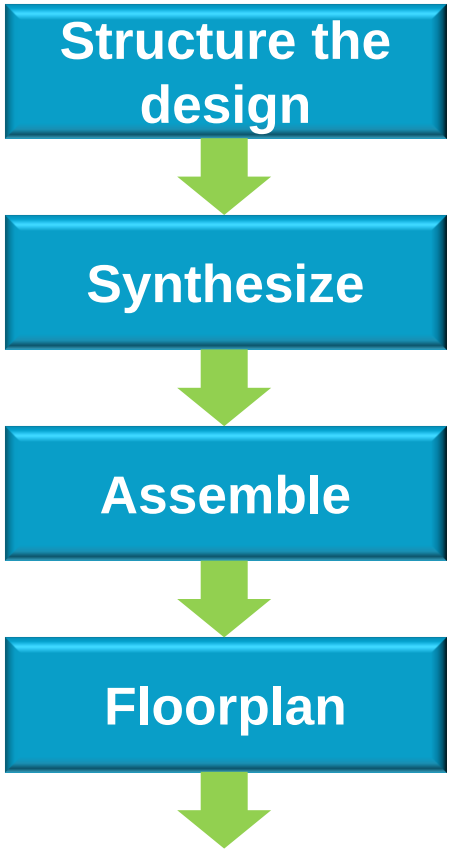
https://www.flickr.com/photos/lemanslive/4508022900

Design Flow

First, Some Terminology

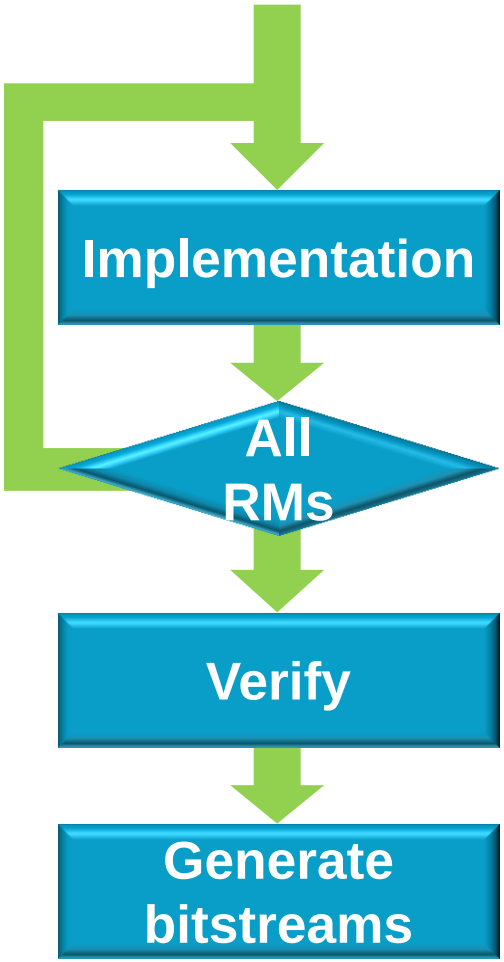
- Reconfigurable Partition (RP)
  - The physical location of FPGA resources selected for partial reconfiguration
- Static logic
  - Everything but the RP(s)
  - The part of the design that doesn't change
- Reconfigurable Module (RM)
  - Logic that lives in the RP
  - Defined by hardware interfaces and ports
  - Functional variants for associated RP
    - Different protocol, task, filter, etc.

Design Flow Chart (1 of 2)



- Structure the design
  - Separate functions into hierarchical blocks
  - Identify functions to be made into partitions
  - Identify set of signals that will become RP interface
- Synthesize
  - Bottom-up
  - Static “top” and RMs synthesized seperately
- Assemble static design with RM variants
  - RMs replace black boxes in static “top”
- Floorplan the RPs and run DRCs
  - Define regions and logic resources to be included

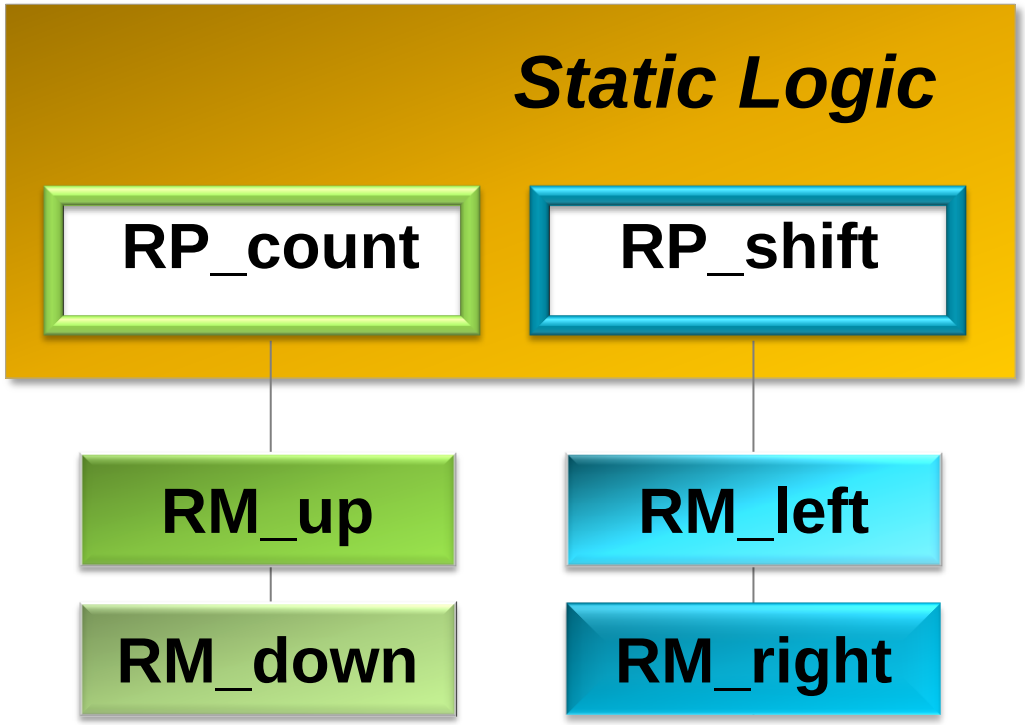
Design Flow Chart (2 of 2)



- Implementation
  - Configurations for static logic and all reconfigurable modules
  - Repeat for all modules
- Verify all configurations
  - Ensure that static portions match identically
- Generate bitstreams
  - Full and partial for each configuration

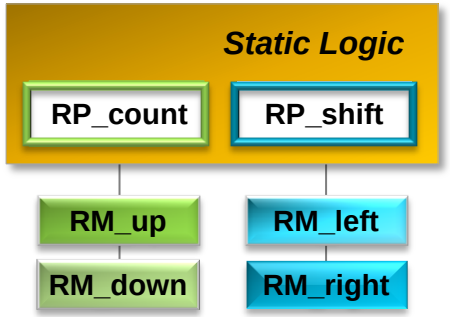
Simple Design Example

- Static Logic
- Two RPs
  - Counter
  - Shifter
- Two RMs per RP
  - Count up, down
  - Shift left, right



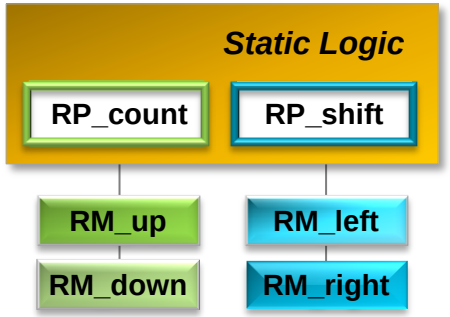
Structure The Design

- Partition the design into functions
  - These functions become the RPs
- Partitions are key to partial reconfiguration
  - Allow clear separation of static logic and RMs
  - Floorplan to constrain resources to be reconfigured
  - Analogous to “chip-to-chip within the chip”

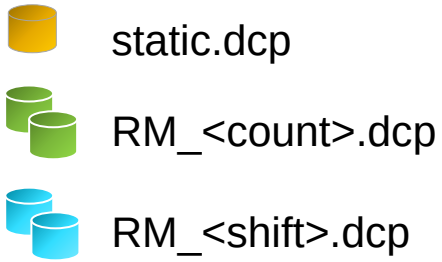


Synthesize The Design

- Iterations for static logic and each RM
  - Top module is static
  - Everything but RPs is static



- Synthesize bottom-up
  - Vivado non-project mode
  - Synthesize static logic in “default” mode
  - Synthesize RMs in “out of context” mode



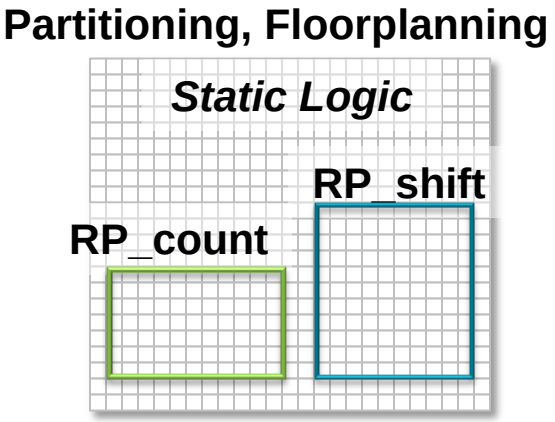
- Save design checkpoint for each iteration

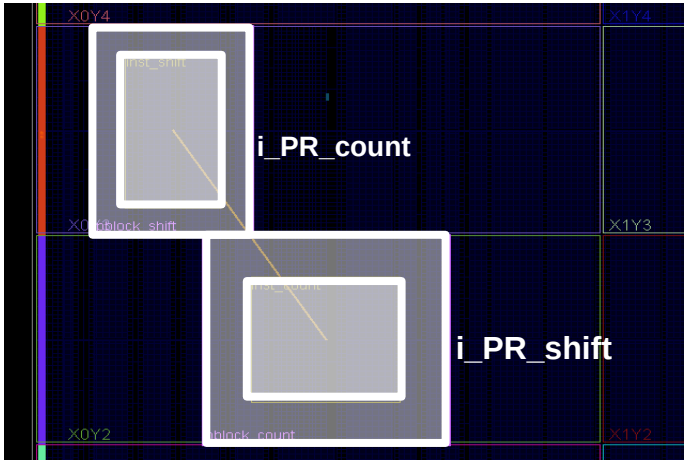
Assemble The Design

- Vivado GUI interactive TCL command session
  - No support for “project mode” for partial reconfiguration yet
- Open static top synthesis checkpoint
  - Black boxes for RMs
  - Ignore warnings regarding unmatched instances
- Load synthesized checkpoint for first RM variants
- Declare the RM as being reconfigurable
  - Vivado checks for PR license
- Save the assembled design for initial configuration

Build The Design Floorplan (1 of 2)

- Must create a floorplan to define the regions that will be partially reconfigured
- RPs are made of one or more Pblocks
- Draw Pblock for logic to be constrained
  - Define XY size and resources
  - Exact size and shape matters
  - Pblock can cross clocking region
  - Resources such as block RAM and DSP48 slices can be added if needed
  - Statistics tab shows resource requirements for loaded module





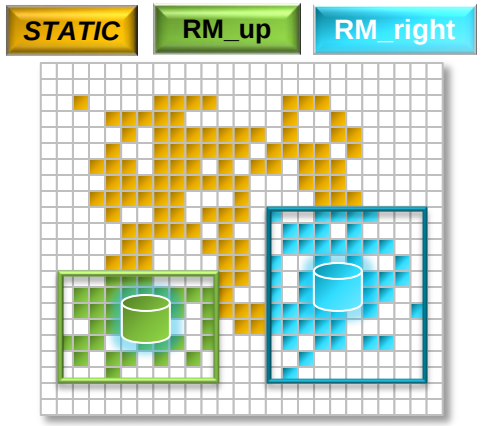
Build The Design Floorplan (2 of 2)

- Set Pblock property to RESET_AFTER_RECONFIG
  - Utilize the dedicated initialization of the logic in this module after reconfiguration has completed
- Run Design Rule Checks for Partial Reconfiguration
- Adjust size of Pblocks as needed to pass DRC
  - Two methods
    - Manually stretch Pblock boundaries
    - Use SNAPPING_MODE feature in Pblock properties
- Save Pblocks and associated constraints

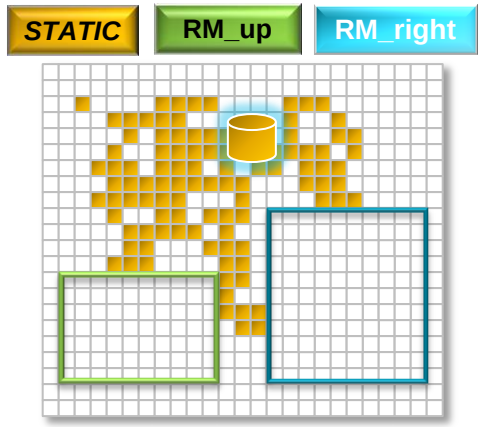
Implement The First Configuration

- Load the top level constraint file
- Optimize, place, and route the design
- Save the full design checkpoint
- Save the checkpoint for the RMs
- Clear out the RM logic from the design
- Lock down all static placement and routing
- Save the checkpoint for the static design

Implement Configuration

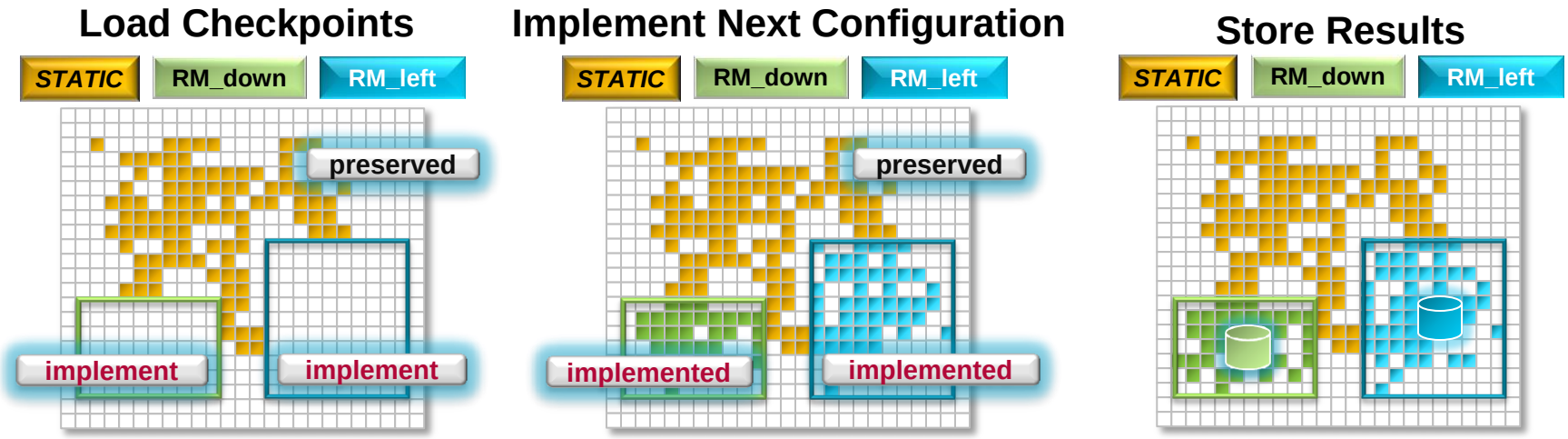


Lock Down Static Design



Implement The Second Configuration

- Load synthesized checkpoint for second RM variants
- Optimize, place, and route the design
- Save the full design checkpoint
- Save the checkpoints for the RMs



Generate Bitstreams

- Verify all configurations
  - Ensure that static portions match identically
  - They MUST match so bitstreams can be used safely
- Load the first configuration
- Generate all bitstreams for the design
  - Full and partial bitstreams created automatically
- Repeat for all configurations

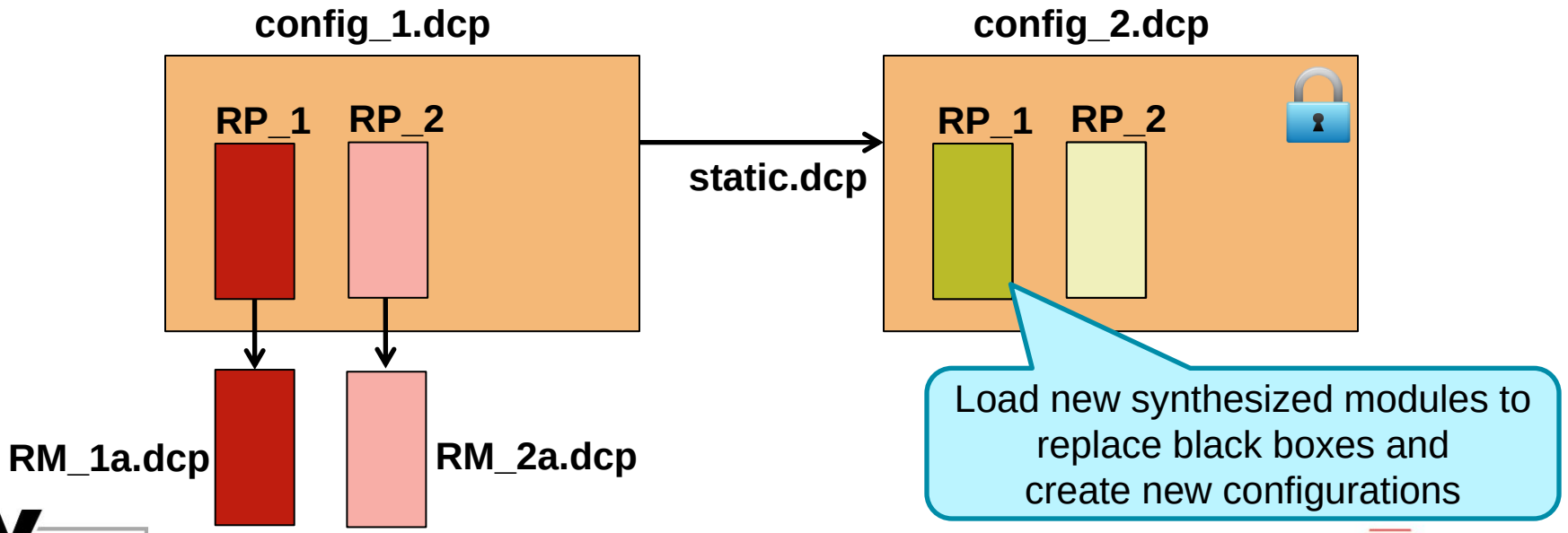
Generate Blanking Bitstreams

- Optionally generate full bitstream with black boxes plus blanking bitstreams for RMs
  - Blanking bitstreams can be used to reduce power
- Load the static design checkpoint
- Generate blanking bitstreams for the design
  - Buffers inserted on RM outputs to tie logic to steady state

Design Considerations

Vivado Design Management

- Vivado stores design data in checkpoints
  - Save full design as a configuration checkpoint for bitstream creation
  - RMs can also be stored as their own checkpoints
  - Save static-only checkpoint to be reused across multiple configurations
    - Routed static checkpoint can remain open in memory
    - Results are locked at the routing level



Silicon Features For Partial Reconfiguration

- Dedicated GSR for reconfigured regions
- CRC checking for partial bitstreams
- Interconnect tiles

Dedicated GSR For Reconfigured Regions

- Set RESET_AFTER_RECONFIG property

- Reconfigured region receives a masked GSR for all synchronous elements in partial bitstream
  - Eliminates user requirement for initializing logic
  - Requires vertical alignment to clock region boundaries

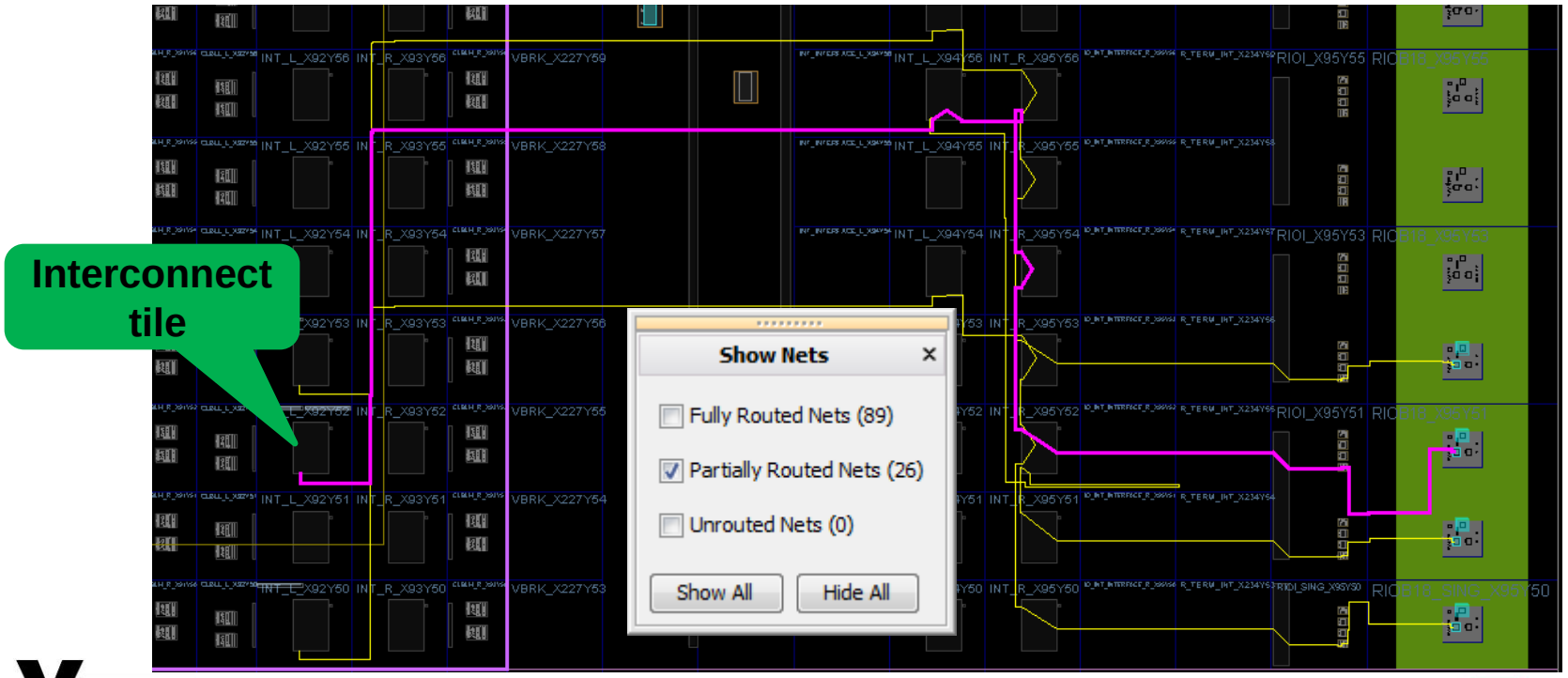
CRC Checking For Partial Bitstreams

- Standard single CRC at the end of a partial bitstream can report errors
  - But the errors have already been sent to the active device
  - Cannot recover in the same way as a full configuration
  - Monitor PCAP status bits to detect error

- Per-frame CRC feature injects CRC checks at intervals in partial bitstream
  - Failures are found and reported before bad frames are loaded into the device
  - Steps can be taken to recover from corrupt bitstream without disrupting the existing functioning design
  - Upcoming feature in Vivado 2014.3

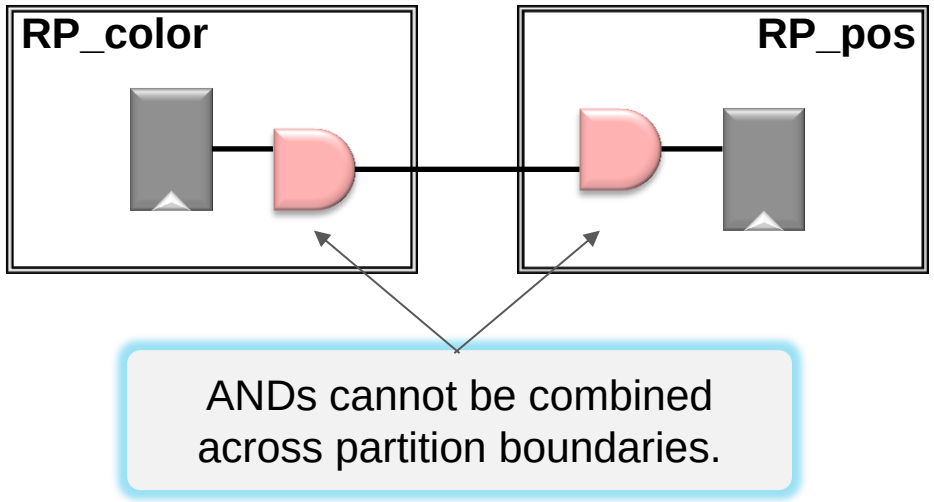
Reconfigurable Partition Interface To Static Logic

- Partition Pins are junctions between static and reconfigured logic
  - Interface wires can be broken at interconnect tile site
  - Anchor mid-route between static and reconfigurable logic
  - No overhead at reconfigurable partition interface
  - Decoupling logic still highly recommended



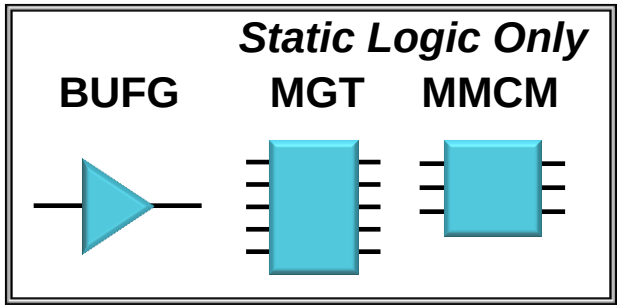
Partition Boundary Recommendations

- Always register partition boundaries
- Keep related logic together



Not Everything Can Be Reconfigured

- Some component types CAN be reconfigured
  - CLB, BRAM, DSP, and routing resources
- Other components CANNOT be reconfigured
  - Clocking resources
    - BUFG, BUFR, MMCM, PLL, etc.
  - I/O resources
    - ISERDES, OSERDES, IDELAYCTRL, etc.
    - MGTs and related components
  - Architecture feature components
    - BSCAN_STARTUP, XADC, etc.



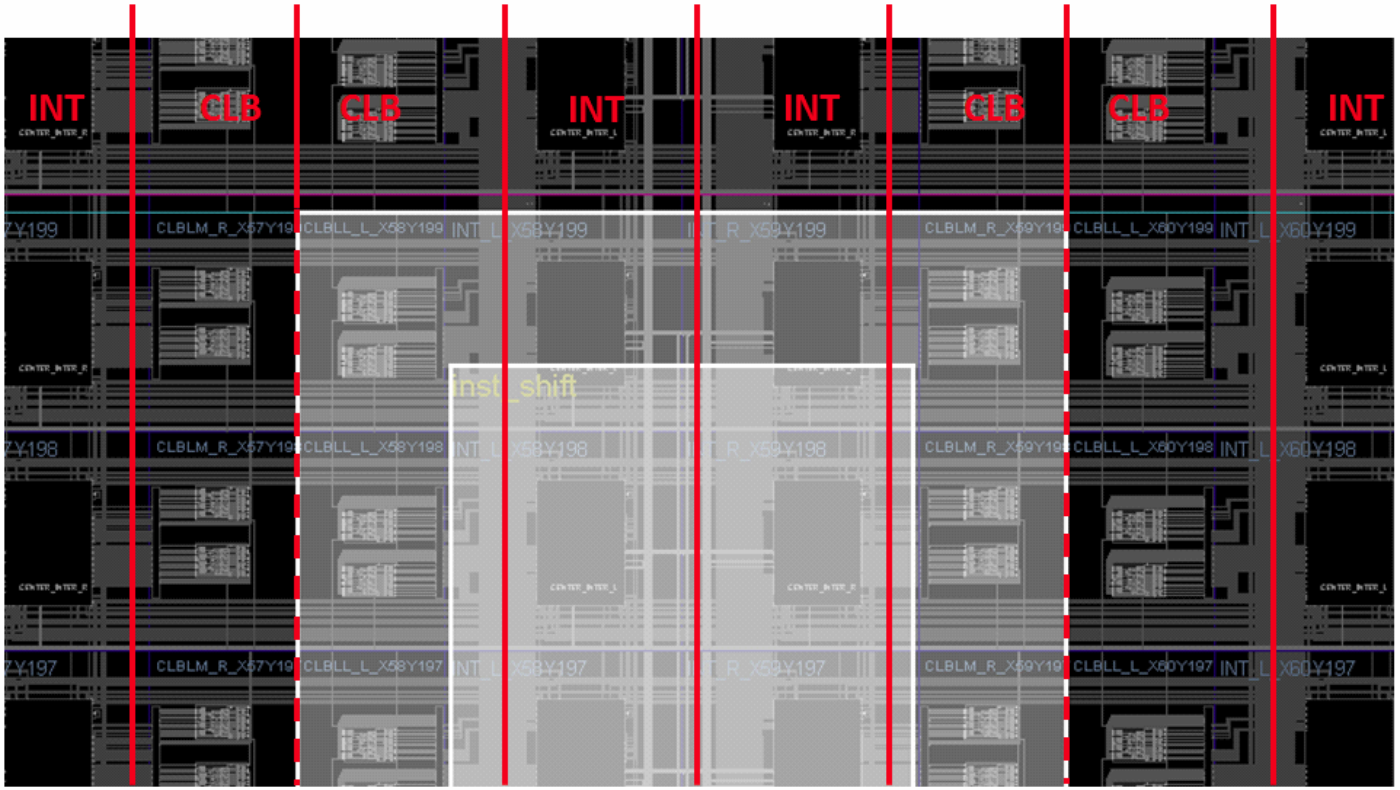
- RP MUST be floorplanned
  - RM must be contained within RP and meet timing
- Keep partition pins to minimum to avoid routing congestion

Pblock Boundary Considerations

- The width and composition of a Pblock must not split interconnect columns
- The Pblock must not overlap any other Pblock in the design
- Nesting of RPs is currently not supported

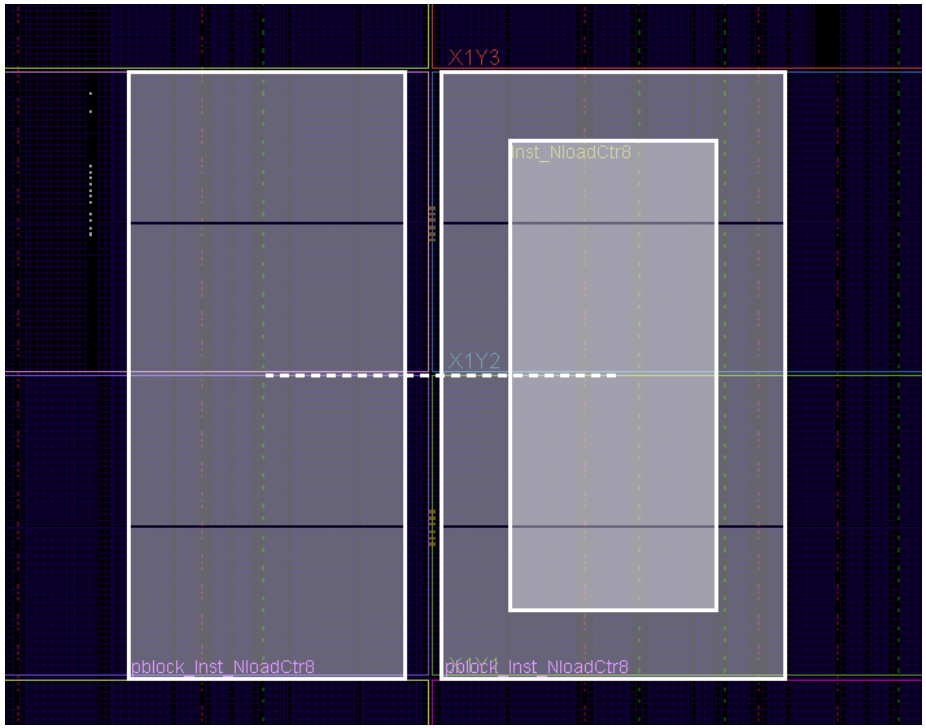
Optimal Pblock Boundaries

- Must contain only CLB/SLICE, DSP and BRAM sites
  - Optimally split between CLB columns on edges



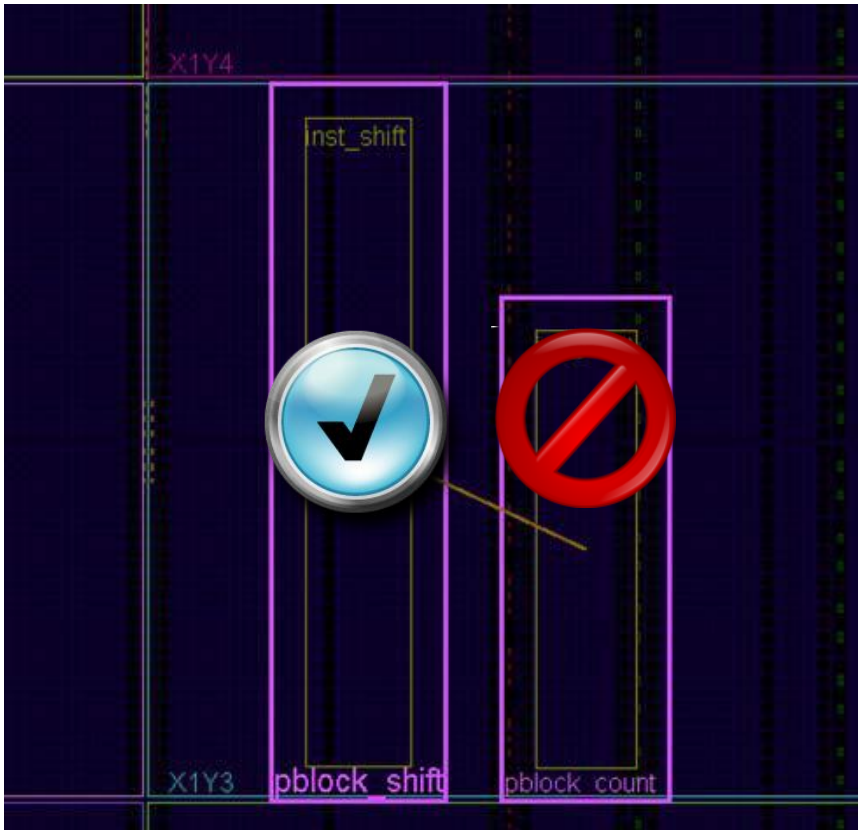
Use Multiple Pblocks To Avoid Splitting Interconnect

- Multiple Pblock rectangles for each component type may be used to create the RP
  - Should be contiguous to avoid routing problems
  - Gaps for non-reconfigurable resources are permitted



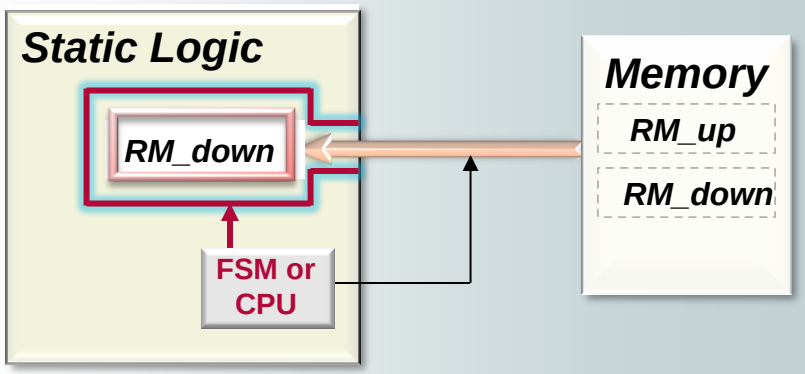
Align Pblocks To Clock Region Boundaries

- Pblock height must align to clock region boundaries if using the RESET_AFTER_RECONFIG property
- Aligned to top **and** bottom of clocking region



Partial Reconfiguration Control Sequence

- Initiation of reconfiguration implemented by the designer
  - State machine, MicroBlaze, or Zynq PS
- Activate decoupling logic and reset
  - Disconnect the reconfigurable region from the static region
  - Deliver Partial Bitstream
    - Region is automatically initialized if RESET_AFTER_RECONFIG is selected
  - Release decoupling logic when reconfiguration is complete



- Standard configuration mechanism are used
  - Partial bitstream is the same format as full bitstream
  - Deliver via JTAG, ICAP, or Zynq PCAP configuration ports

Decoupling Logic Considerations

- RM must ignore data from static logic during reconfiguration
  - Decouple clock and other inputs to RMs
    - Prevent spurious writes to memories during reconfiguration
- Static logic must ignore data from RMs during reconfiguration
  - Data from RM will be invalid until reconfiguration is complete and logic is reset
- Should be included in static logic portion of design
- Up to the designer to implement

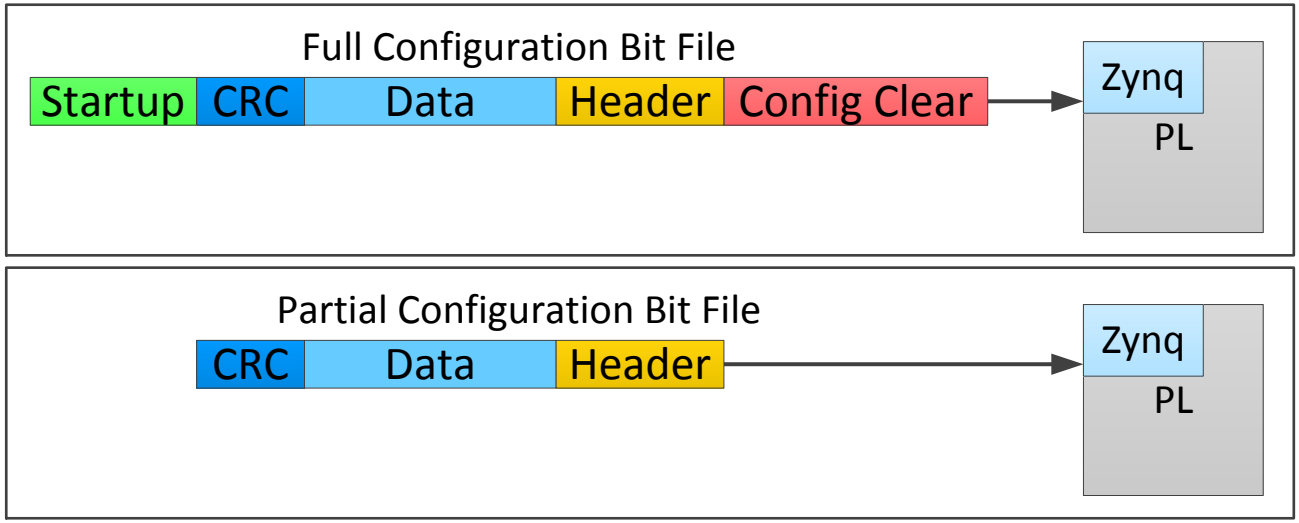
Decoupling Logic Schemes

- Register all outputs from the Reconfigurable Module
  - Use an enable signal to isolate the logic
- Use muxes on the outputs
- Disable AXI interface
- Register interface / mailboxes

Decoupling Control

- Pay special attention to AXI interfaces
  - No pending AXI transactions at reconfiguration time
    - AXI interconnect can become hung
- Disable interrupts during reconfiguration
  - Reconfiguration can trigger spurious interrupts
    - Servicing such an interrupt can hang the system

Downloading A Partial Bitstream

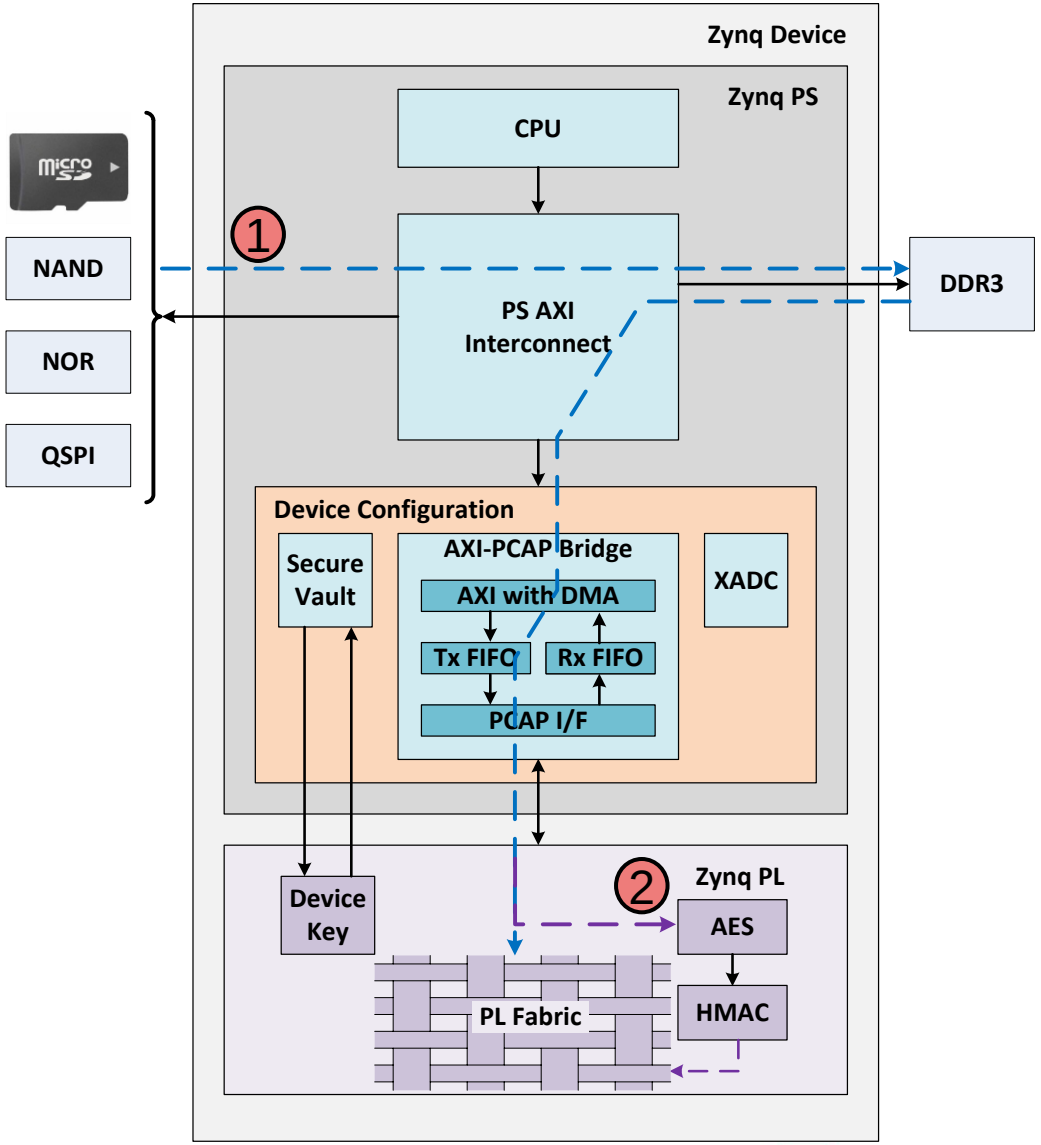


- Partial bit file contains (with default settings)
  - Frame address
  - Configuration data
  - Final CRC value

- If RESET_AFTER_RECONFIG is set, DONE will pull low then pull high when reconfiguration completes

Zynq PL Partial Reconfiguration

- User application loads partial bitstream into DDR3 and then into the PL through the PCAP via DevC DMA
- AES encryption can also be used



PR Usage In Linux

- Tell running Zynq device you plan to partially reconfigure
  - Set "`is_partial_bitstream`" device attribute via sysfs
  - `# echo 1 > /sys/devices/amba.0/f8007000.devcfg/is_partial_bitstream`
- Write the bitstream to DevC
  - DevC function initiates DMA transaction
  - Interrupt signals transfer is completed
  - `# cat /<path_to>/<filename>.bit > /dev/xdevcfg`
- Linux commands built into driver and user application

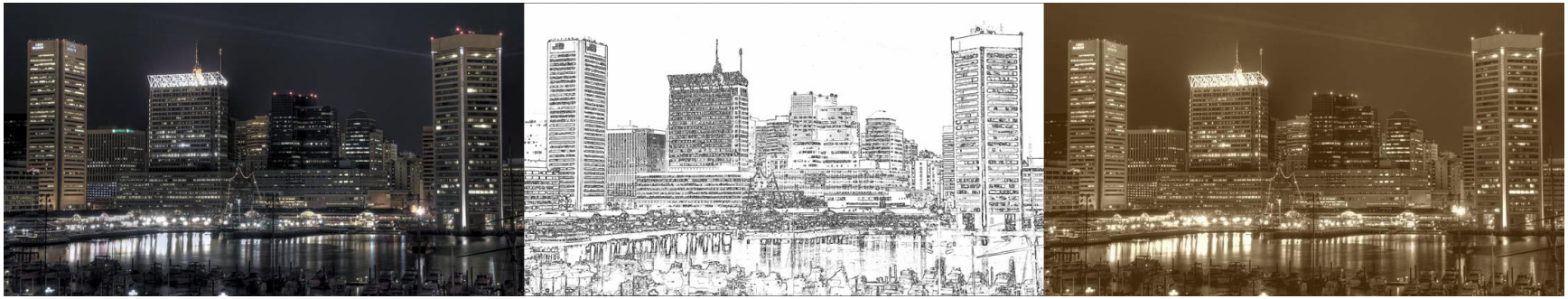
Design Performance Considerations

- Application of PR rules affects design performance
  - Routing containment
  - Exclusive placement
  - No optimization across reconfigurable module boundaries
- Density and performance will be lower for PR design vs. equivalent flat design
- Design performance for PR designs will vary from design to design and is based on many factors
  - Number of reconfigurable partitions
  - Number of interface pins to these partitions
  - Size and shape of Pblocks

Design Examples

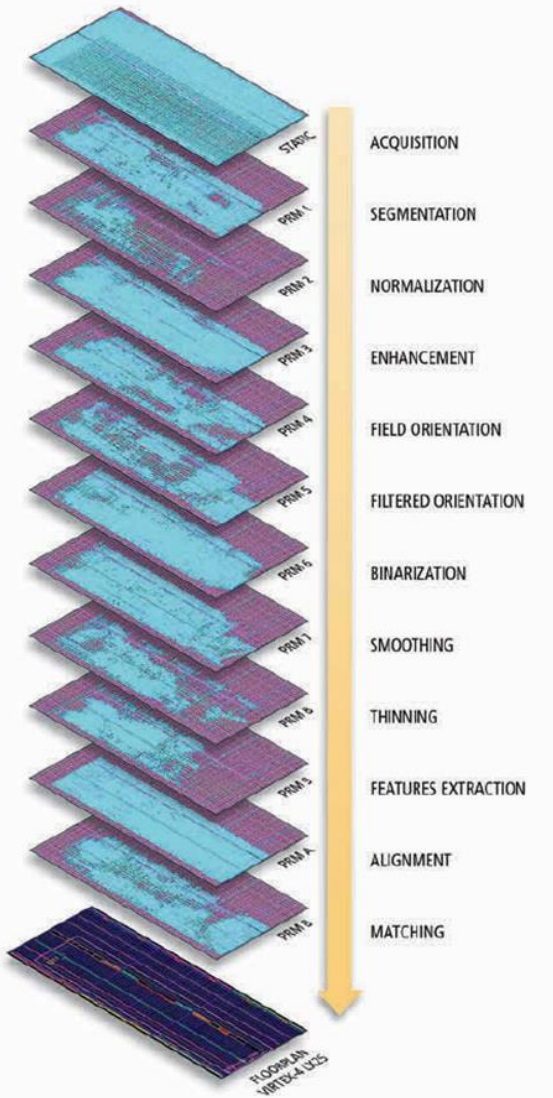
Flexible Video Processing

- Sobel and sepia video filter RMs for a single RP
- Zynq PL accelerators developed in Vivado HLS
- Control path in Zynq PS, data path in the PL
- Software control in Linux
- Filter IP uses a generic register interface and address map
  - Greatly simplifies the software driver architecture
- Partial Reconfiguration via Zynq PCAP



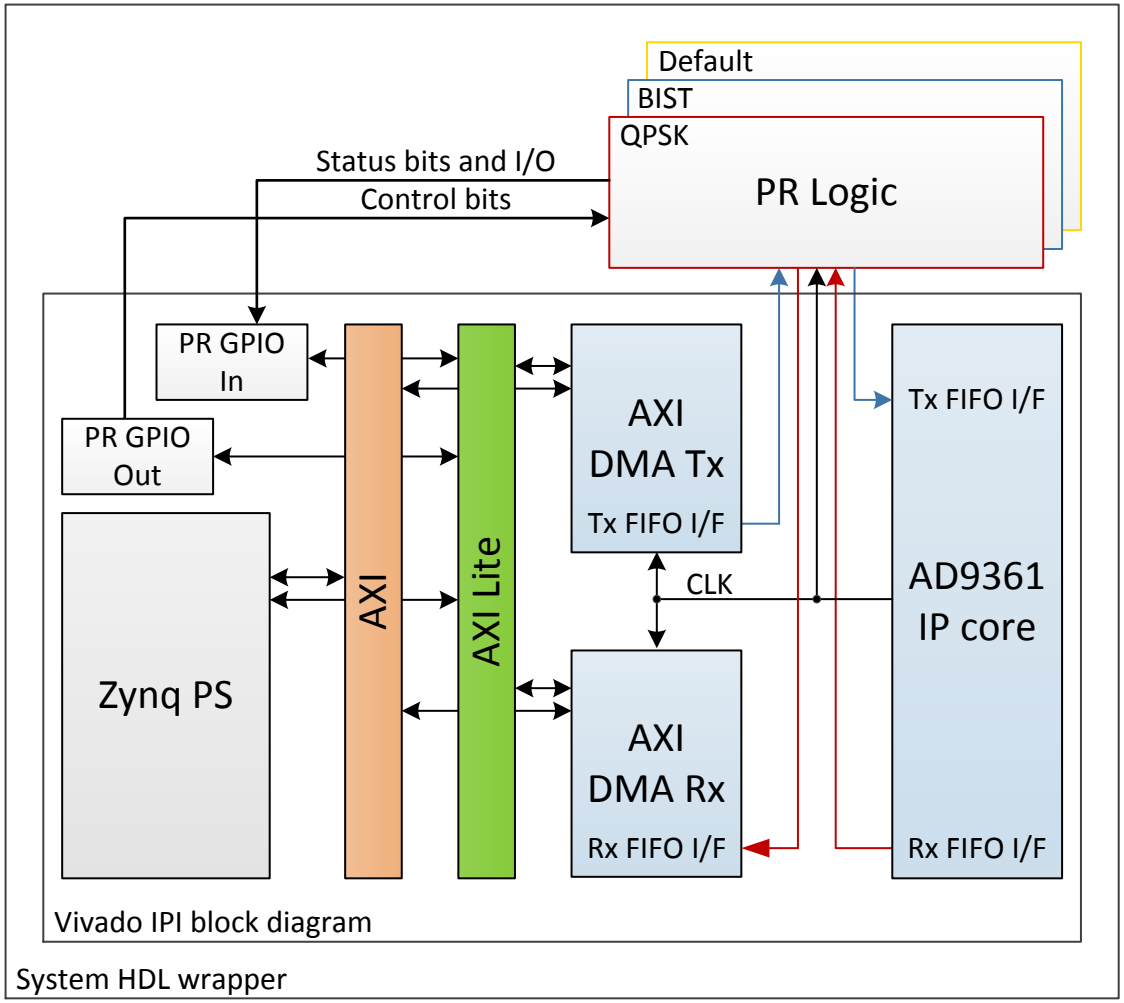
Biometrics Sequential Processing

- Automatic Fingerprint Authentication System
  - Sequence of 12 different functions required
  - No two were needed at any one time
- Load in functions on demand
  - Steps could be skipped or revisited
- Processing time greatly improved
  - PC-based Software solution: 3.77 sec
  - Xilinx MicroBlaze solution: 143.19 sec
  - Xilinx Partial Reconfiguration: 0.7 sec
    - Includes 10ms total for all reconfiguration events
- Efficient use of resources
  - "Flat" solution requires 39k flops, 52k LUTs
  - PR solution uses region with less than 10k of each



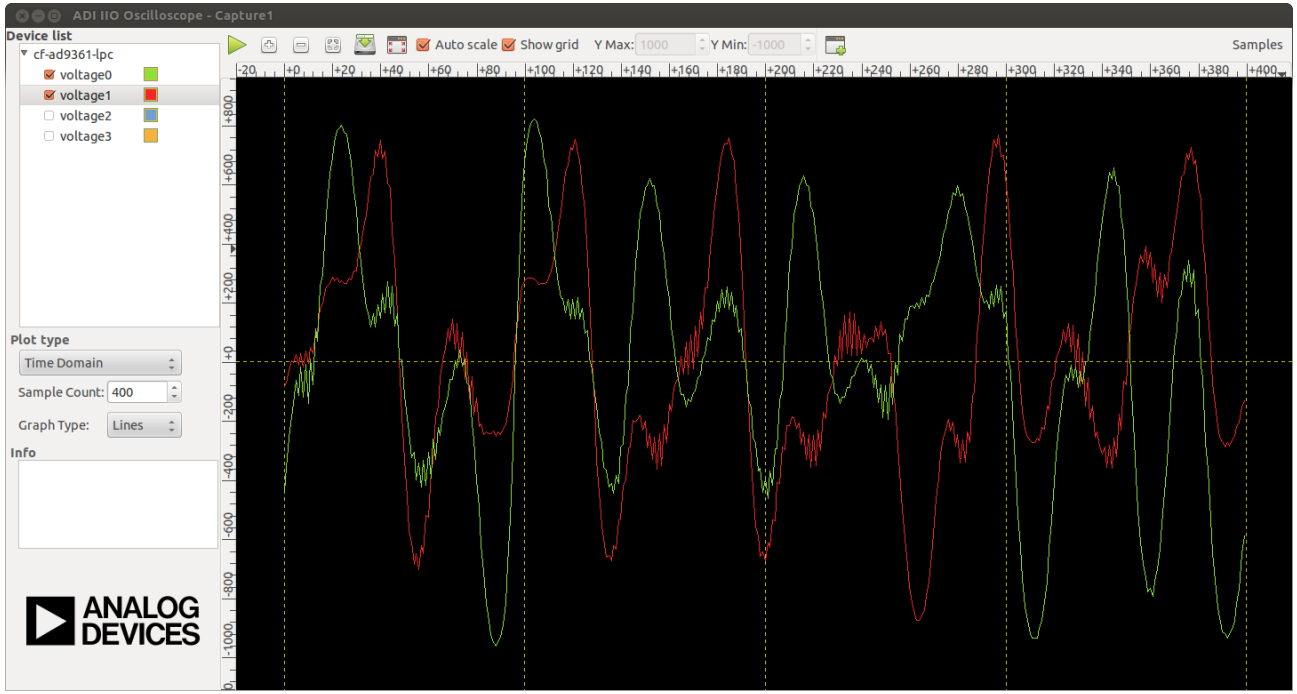
Read the detailed article in Xcell Issue 72

ADI Software Defined Radio



- Default is a bypass logic
  - Good starting point for new logic integration
- BIST has different internal generators for testing
  - Sine, PRBS, constant pattern
- QPSK modulator and demodulator logic
- IP blocks generated with MathWorks MatLab HDL Coder tools

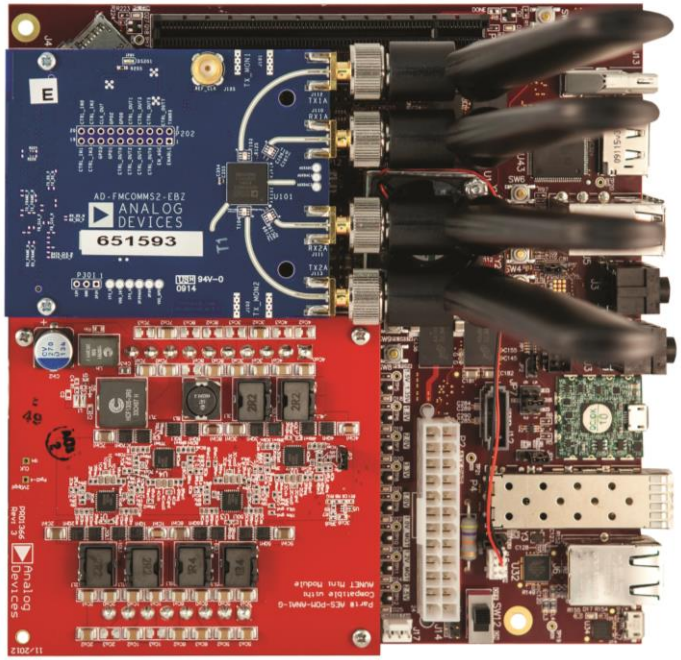
Demo Design Features

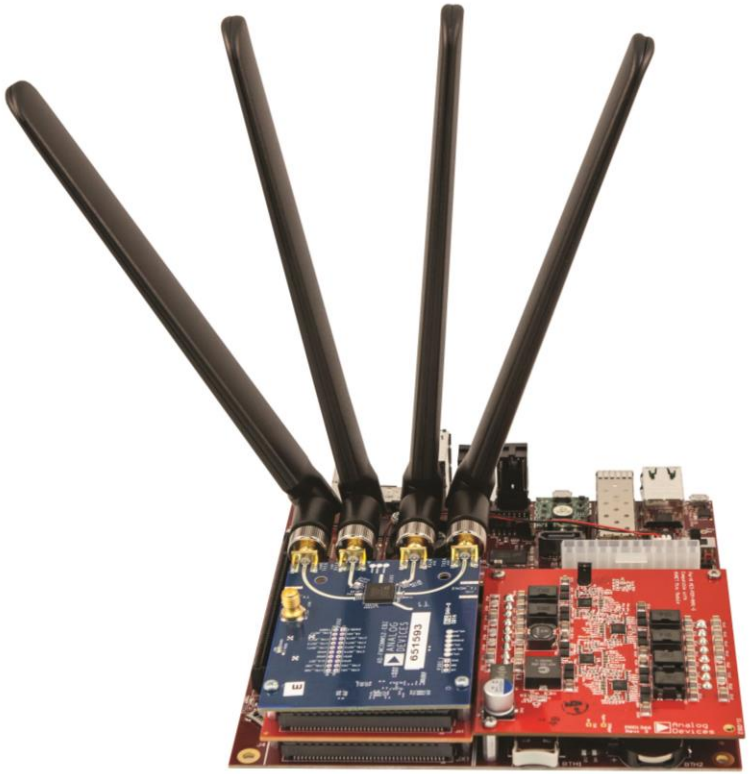


- Default (pass-through), BIST, and QPSK reconfigurable modules
- Control path in Zynq PS, data path in the PL
- Custom Linux application to select and setup the module
  - Write bitstream to Zynq PCAP
  - Set mode for module
- Ubuntu Linux running on Avnet Mini-ITX Zynq AP SoC development board

Demo Design Hardware And Software

- ADI FMCOMMS2 FMC module
  - www.analog.com/en/evaluation/eval-ad-fmcomms2/eb.html
- ADI IIO scope
  - wiki.analog.com/resources/tools-software/linux-software/iio_oscilloscope
- Avnet Mini-ITX
  - www.zedboard.org/product/mini-itx





Next Steps

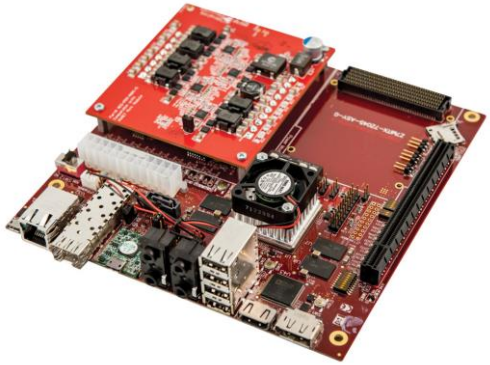
Additional Resources

- Contact your FAE for license and pricing information
- Documentation
  - User Guide UG909
  - Tutorial UG947
  - Application note XAPP1159
  - Xcell articles
    - Issues 72, 73, 75, 77, 78, 79, 80, 84, 85
- Video
  - Xilinx QuickTake Video
- Course presentation and forums
  - Go to www.xfest2014.com
  - Available September 15th

Avnet Zynq Development Kits

- Please visit our www.zedboard.org web site for information on Avnet Zynq development boards and SoMs

Mini Module Plus



Mini-ITX Motherboard

MicroZed


Thank
You
Asante
Maake
Kiitos
köszönöm
спасибо
Dankscheen
Vinaka
Shukria
Dhanyavadagalu
Manana
Dankon
Kam Sah Hammida
감사합니다
Dank Je
Maururu
Biyan
Matondo
Terima Kasih
Taiku
Tack
Diolch i Chi
Chokrane
Blagodaram
Ngiyabonga
Dziekuje
Juspaxar
Arigato
Grazie
Mochchakkeram
Gracias
Tingki
Paldies
Khap Khun Khop
Kia Ora
Kop Khun
Djiere Dieuf
Obrigado
Gratias Tibi
cảm ơn bạn
ধন্যবাদ
நன்றி
Ua Tsaug Rau Koj
Bedankt
Dakujem
Grazas
Děkuji
Nirringrazzjak
Suksama
Hvala
Welalin
Di Ou Mèsi
Merci
Danke
Go Raibh Maith Agat
ขอขอบคุณคุณ
Salamat
Najis Tuke
Eskerrik Asko
Misaotra
Rahmat
Nuwun
Matur
谢
谢
хвала