XDF XILINX DEVELOPER FORUM

Designing for Acceleration: Methodologies for Creating Reconfigurable Applications

Presented By

David Dye
Senior Product Marketing Manager
10 December 2018

XILINX

Topics

> **Partial Reconfiguration Technology Review**

> **SDAccel and Dynamic Platforms**

> **Future Enhancements**

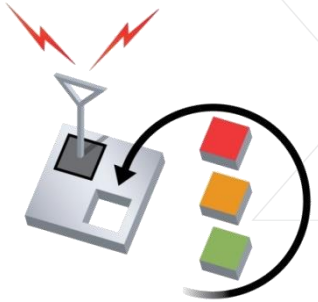
Partial Reconfiguration
Silicon and Software Technology

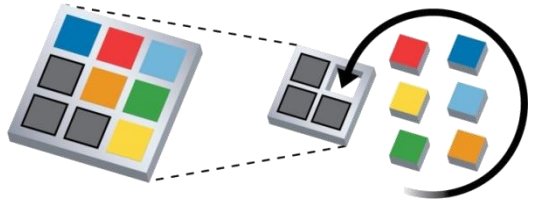
Partial Reconfiguration is mainstream

- **In production since 2010**
  - Starting with Virtex-4, through UltraScale+
  - Continuous silicon improvements with each generation
  - Major advancement with the introduction of Vivado

- **System Flexibility**
  - Swap functions on the fly
  - Enables hardware acceleration platforms
  - Perform remote updates while system is operational

- **Cost and Size Reduction**
  - Time-multiplexing hardware requires a smaller FPGA
  - Reduces board space
  - Minimizes bitstream storage

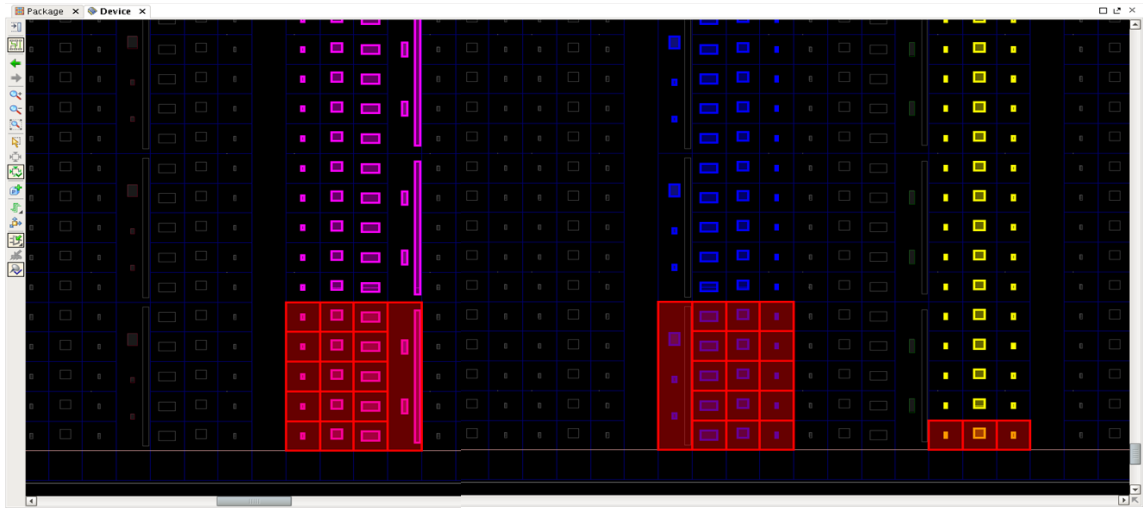




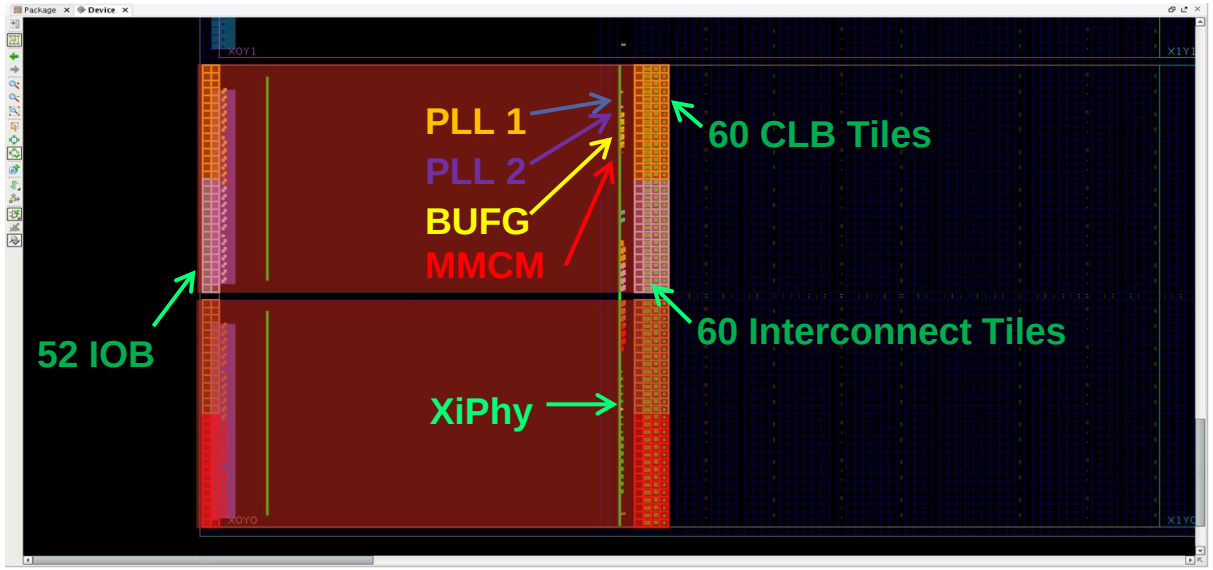
Reconfigurable granularity in UltraScale/+

UltraSCALE
UltraSCALE+

- **Reconfiguration tiles for core fabric based on shared interconnect**
  - Base regions combined two columns of resources with single INT
  - Bitstream granularity is larger, matching clock region height
- **IO and special elements have coarser rules, shared with column of CLBs**
  - IO and clocks: 1 bank
  - Transceivers: 1 quad
  - PCIe, CMAC, Interlaken: 1 clock region



CLB–BRAM CLB–DSP CLB–CLB

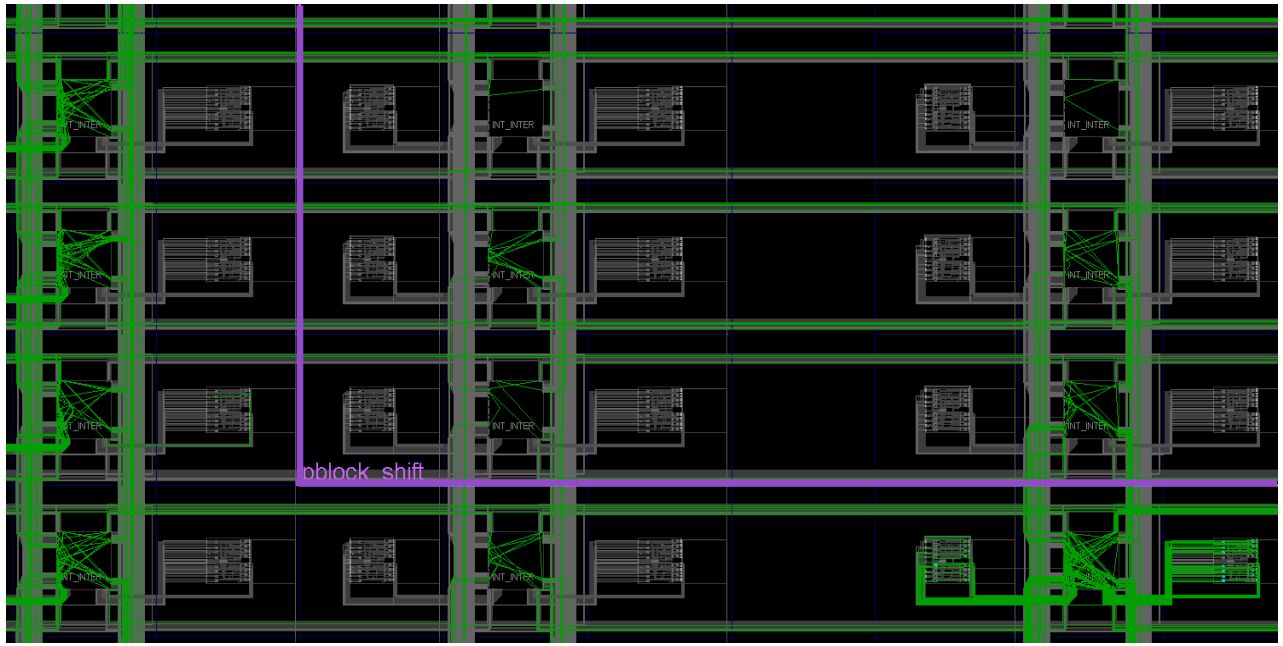


Dedicated silicon features provide design safety

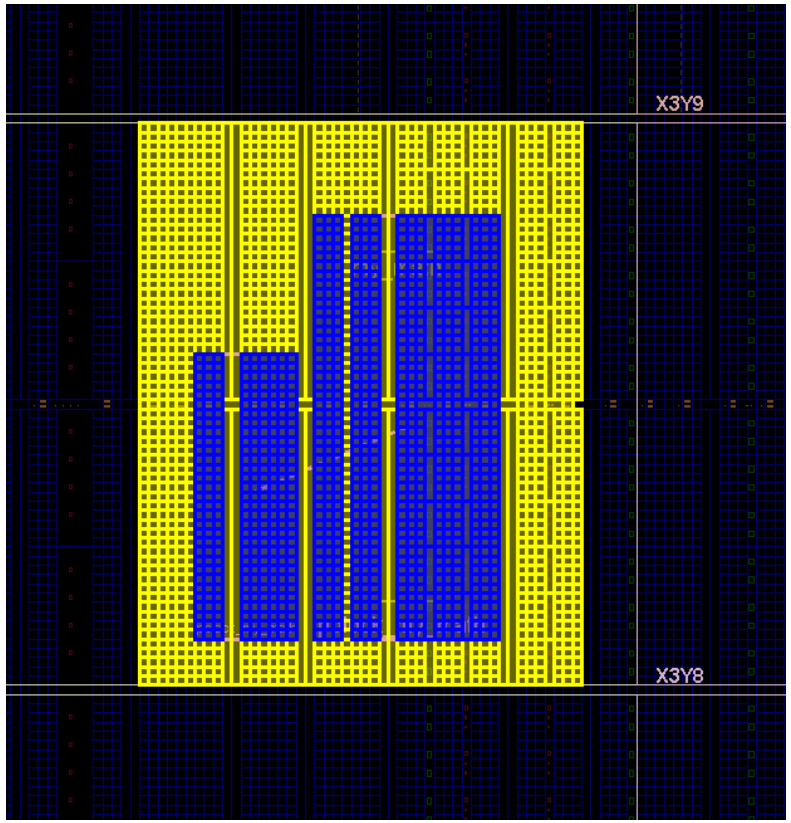
- **Dedicated initialization for reconfigured regions**
  - Reconfigured region is masked and receives GSR for all synchronous elements in partial bitstream
  - Behaves just like the initial configuration of the device
- **Encryption and Compression natively supported**
  - Authentication is also supported for Zynq SoCs
- **CRC checking for partial bitstreams**
  - Standard single CRC at the end of a partial bitstream can report errors
  - 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

Glitchless transitions and routing flexibility

- **Static routes pass through reconfigurable regions without disruption**
  - Routing structure and global controls designed for glitchless static operation

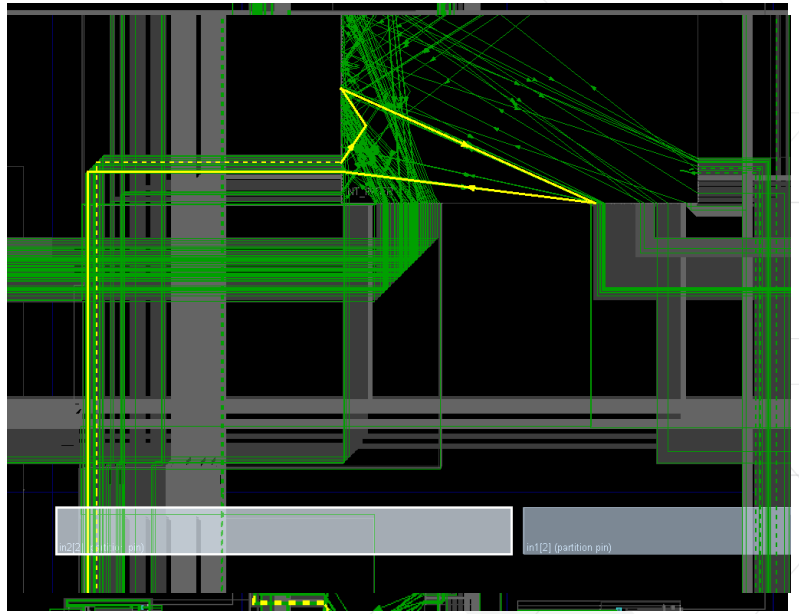


- **Expanded routing regions provide solution flexibility**
  - Alleviates congestion near corners and edges

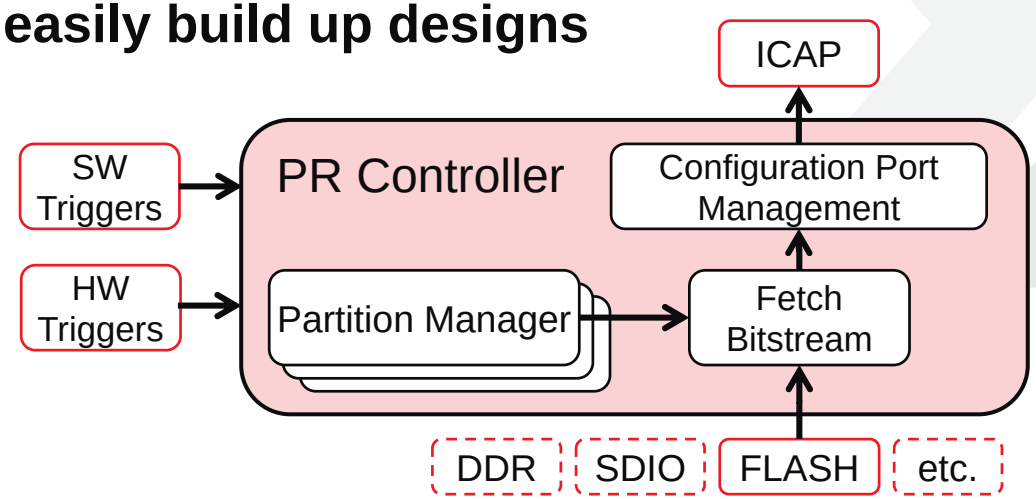


Capable Vivado toolset for compilation

- **Major software overhaul for PR completed in 2013**
  - PR included in all paid editions and kits since 2017
- **Technology improvements for performance and efficiency**
  - Example: Partition Pins
    - No overhead between static and dynamic logic
    - Vivado optimized location and distribution

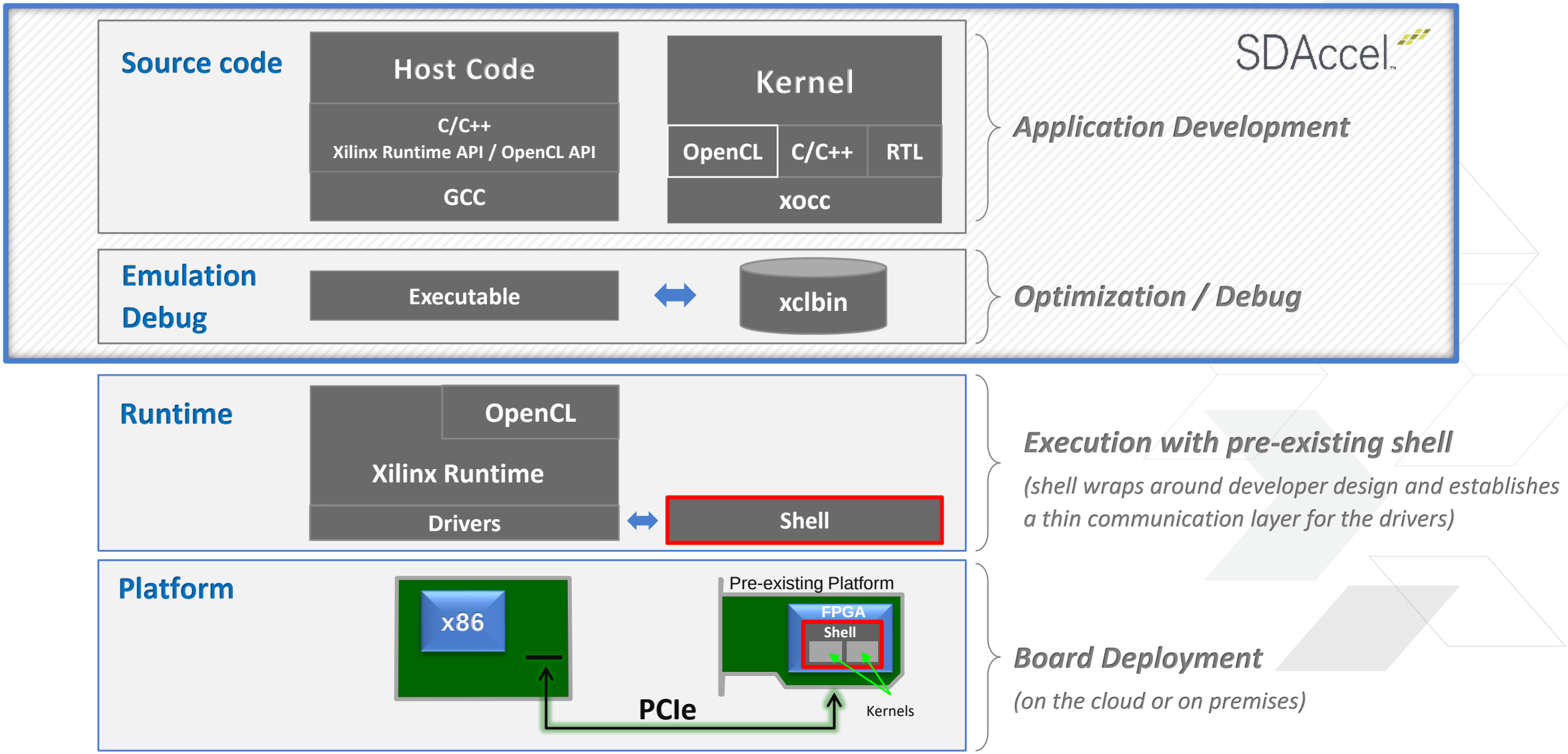


- **Partial Reconfiguration IP helps users easily build up designs**
  - Partial bitstream delivery management
  - AXI transitional behavior
  - Logical isolation
  - Version compatibility

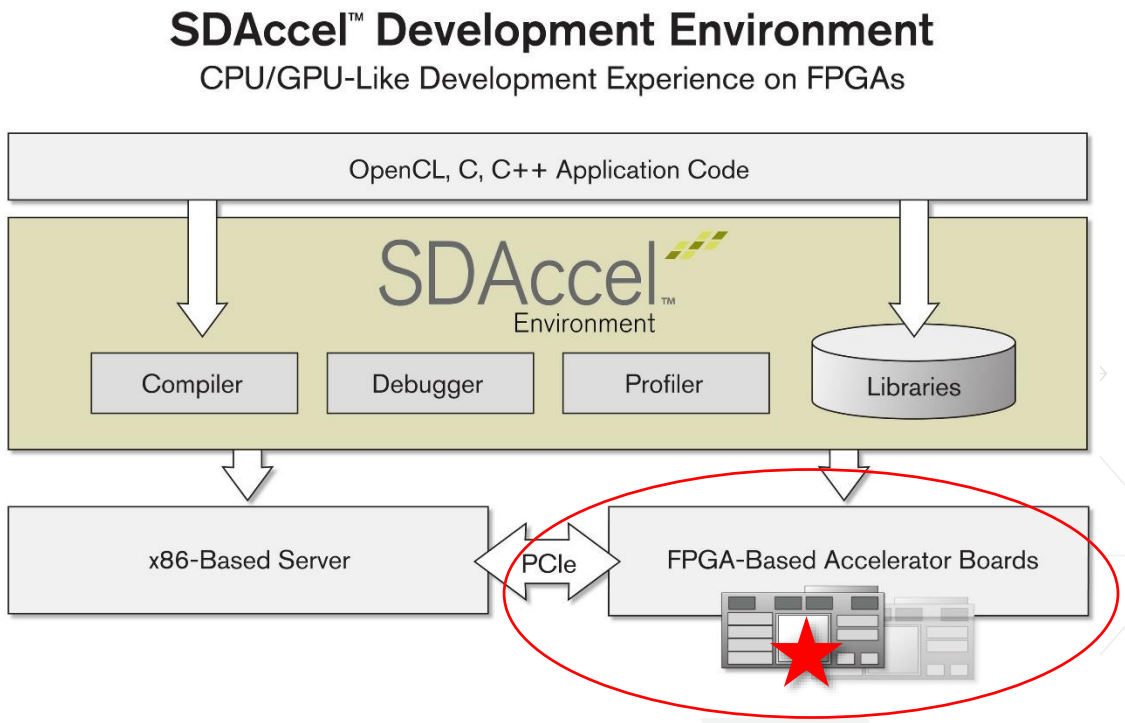


Evolution of SDAccel Acceleration Platforms

SDAccel software development environment

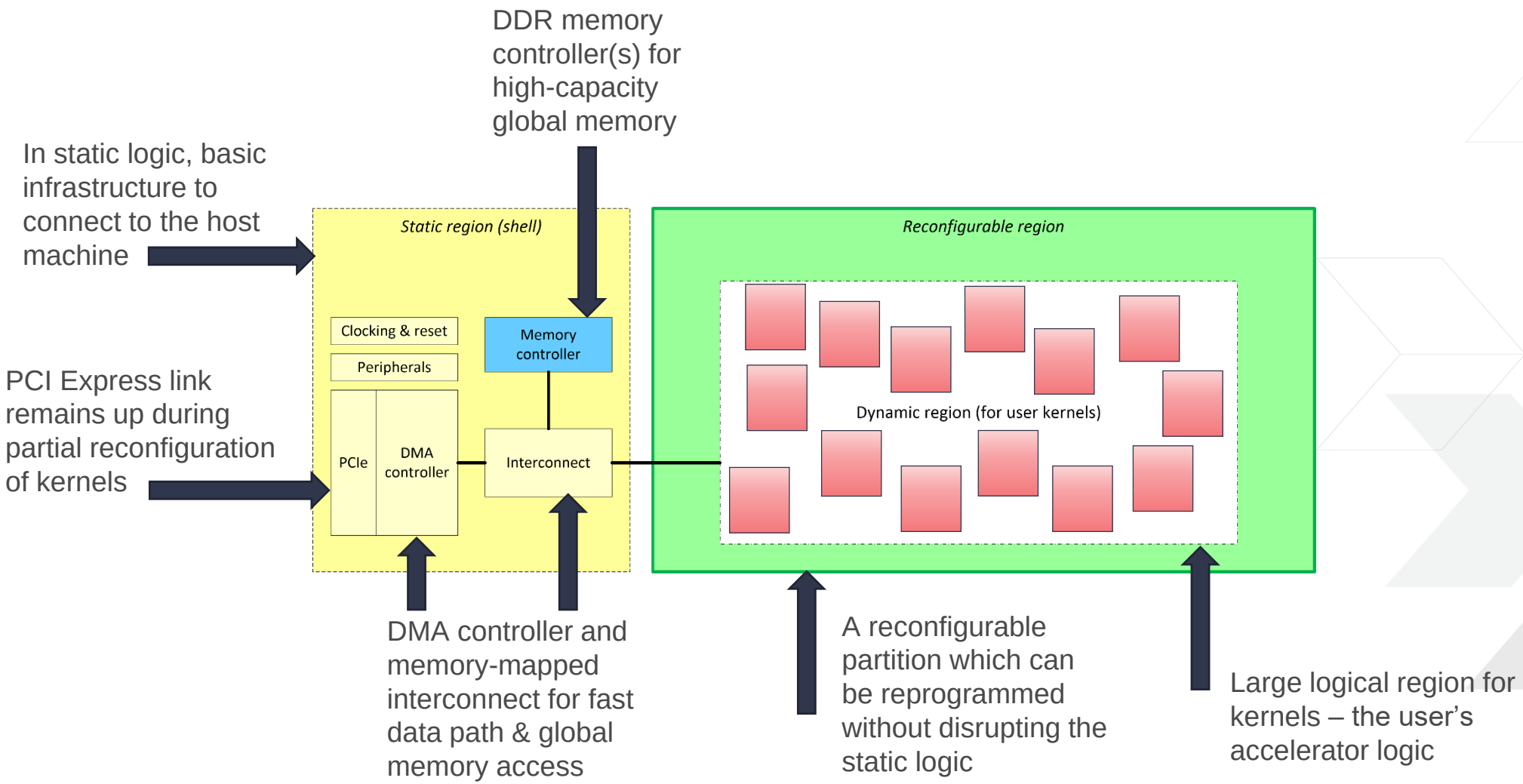


What is a platform?



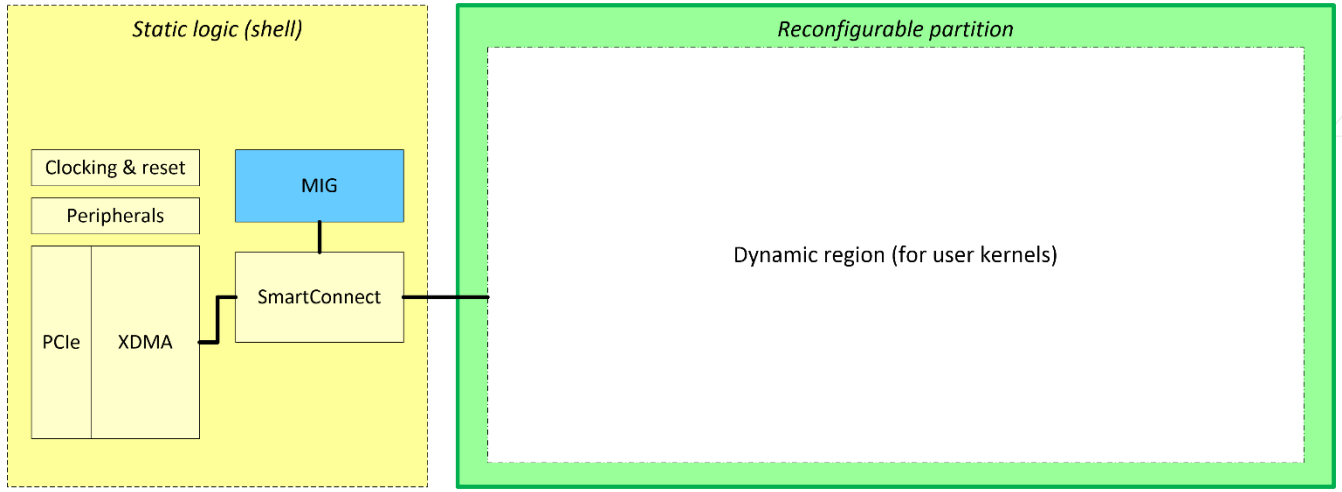
- **A platform is the hardware design which operates on an accelerator board's FPGA to enable integration of the board and host computer with the user's accelerator (kernel) logic**
  - Think "Device Support Archive", not *just* its "shell"
- **Examples of platform content:**
  - Persistent PCI Express link to the host computer
  - DMA controller and interconnect for fast memory access
  - Memory controllers for DDR4 SDRAM global memory
  - Security and board management features
  - A Partial Reconfiguration-enabled design and floorplan
- **In most acceleration platforms, a small static region (shell) provides this core functionality while a large reconfigurable region is available for user kernels**
  - The platform also implements the bridging between the two regions

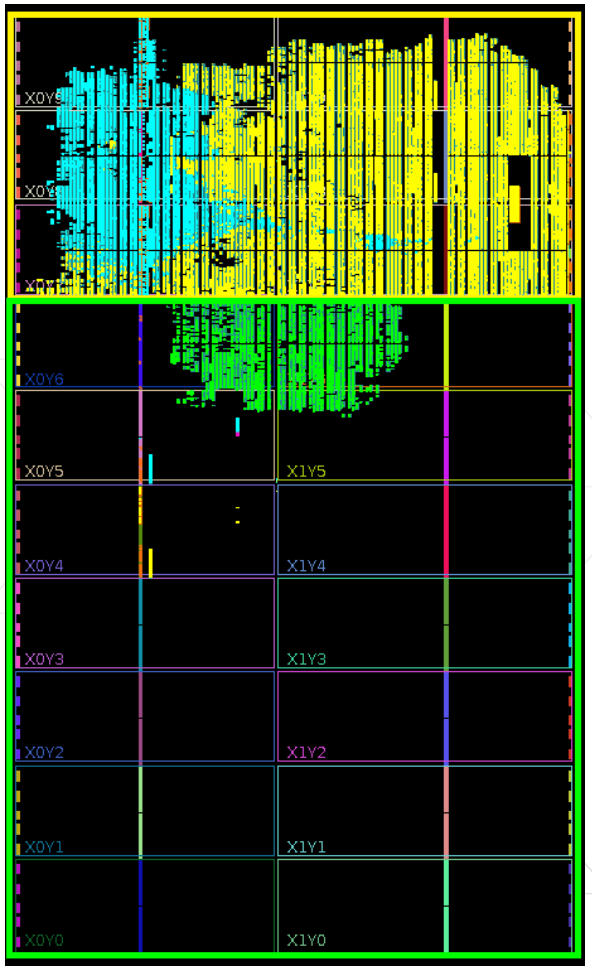
Block diagram of a prototypical platform



Where it started
Virtex-7 690T platform with Partial Reconfiguration

- Natural static/PR partitioning and simple device floorplan
- Relatively large rectangular dynamic region
- Single memory controller in the shell
- A good starting point, but not much bandwidth

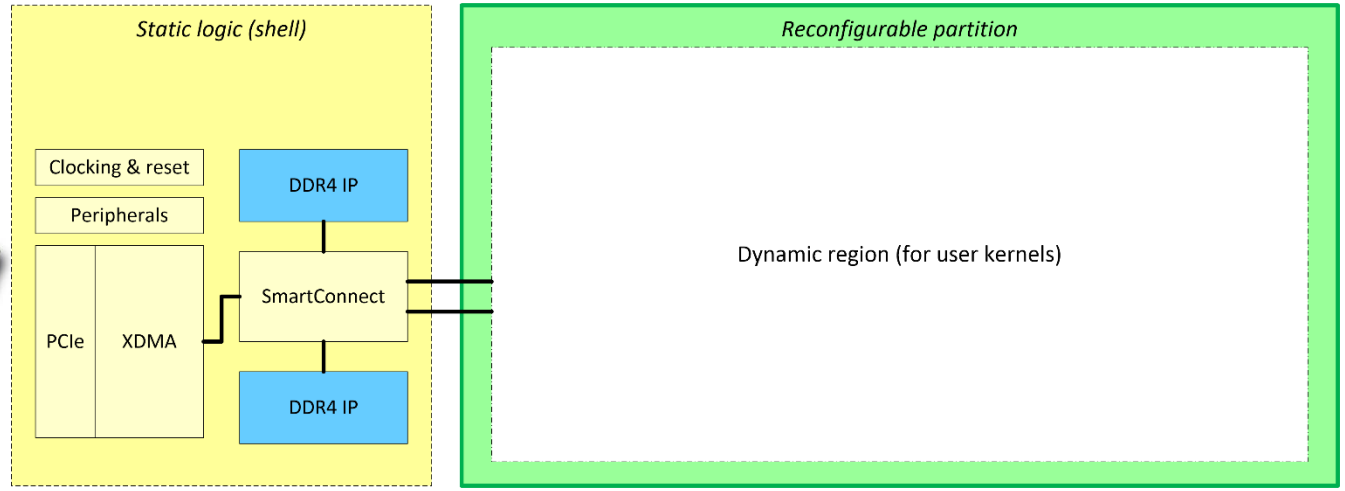


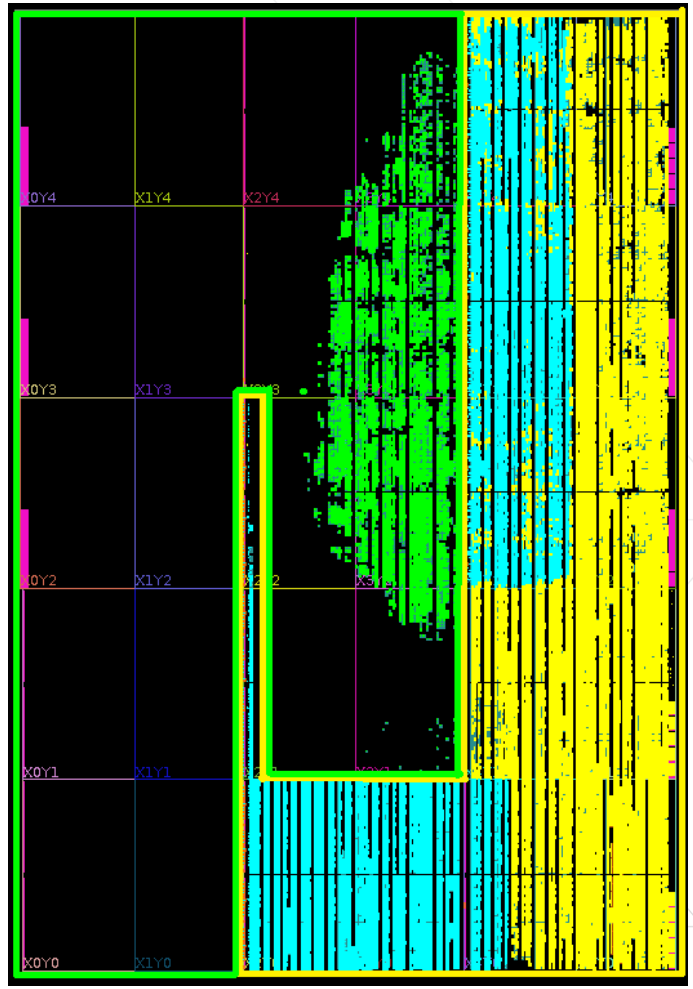


The first UltraScale platform

Kintex UltraScale KU060 platform with Partial Reconfiguration

- Second memory controller and DDR4 for more bandwidth
- But awkward memory controller locations
- Shell is relatively large for the device

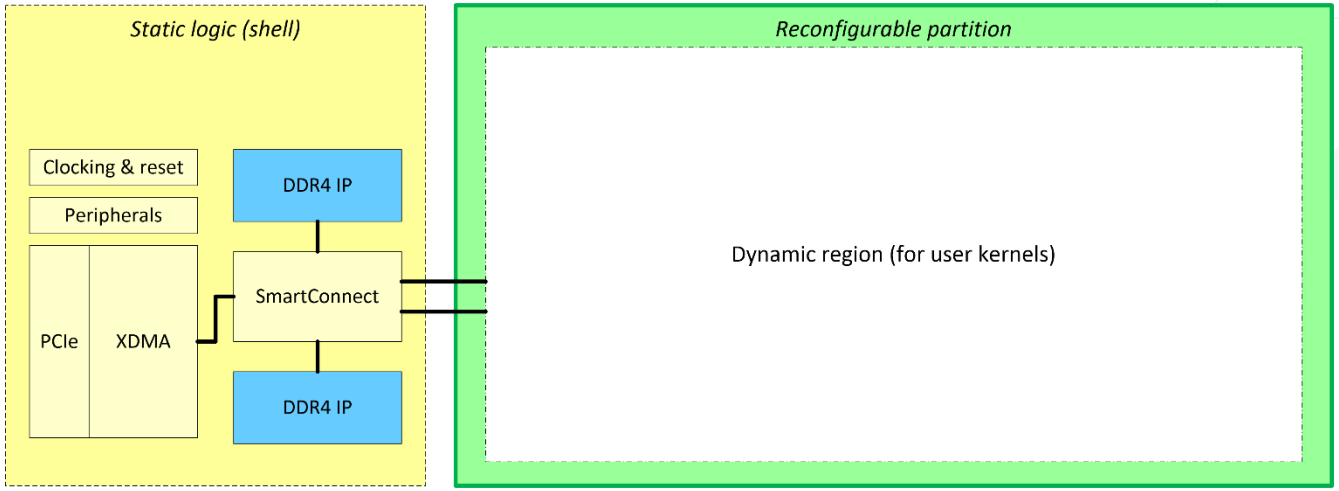


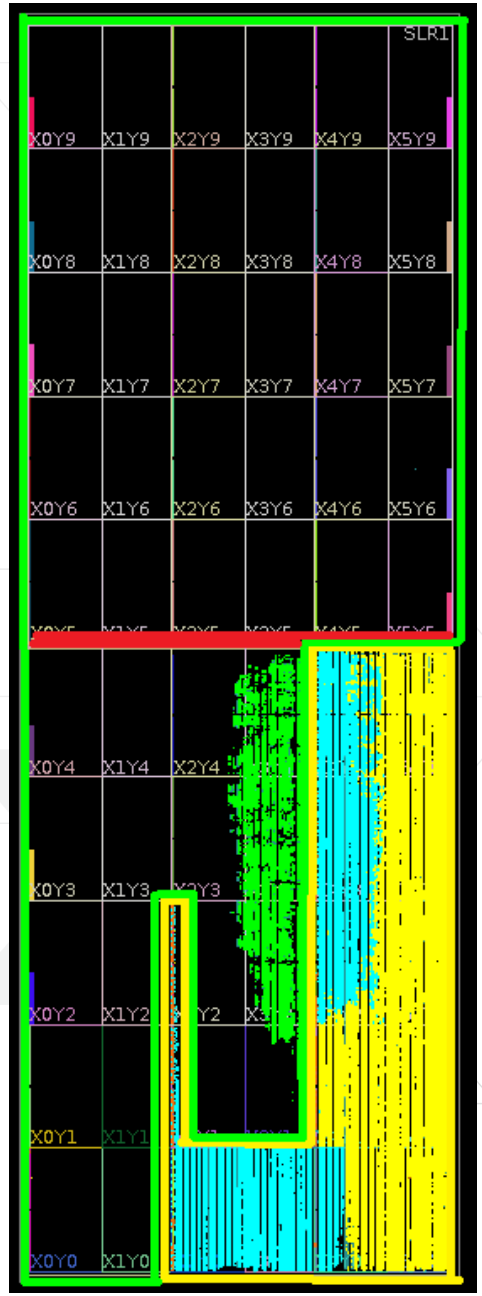


Larger SSI device, same shell

Kintex UltraScale KU115 platform with Partial Reconfiguration

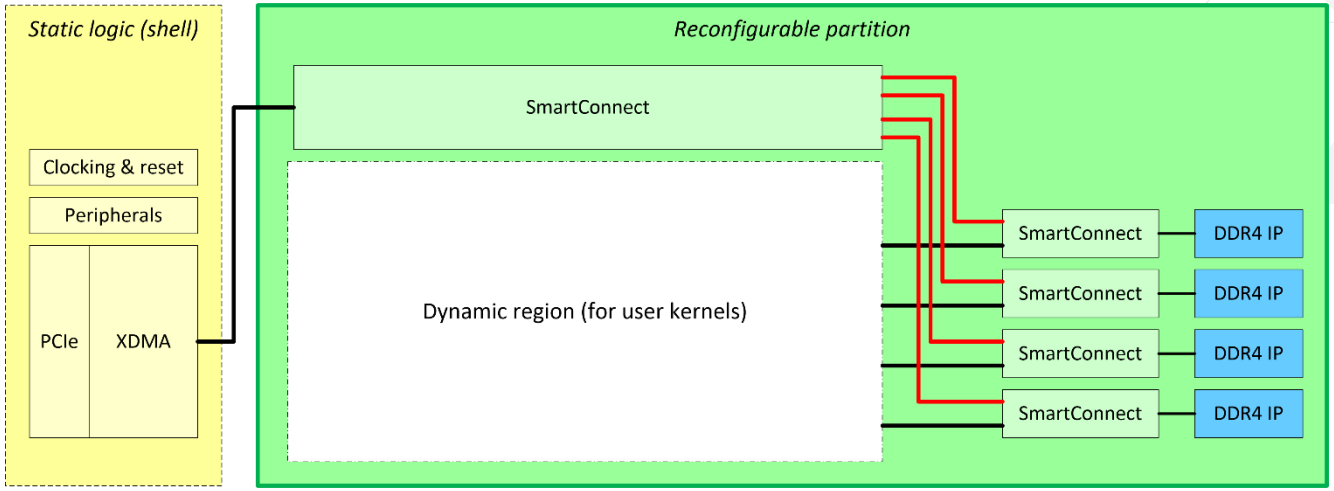
- Almost the same platform as the KU060 variant, on larger device
- Second SLR offered >2x the available area for kernels
- But unmanaged SLR crossings proved problematic
- Still 2 DDR4 channels, despite enough HPIO banks for 4 channels

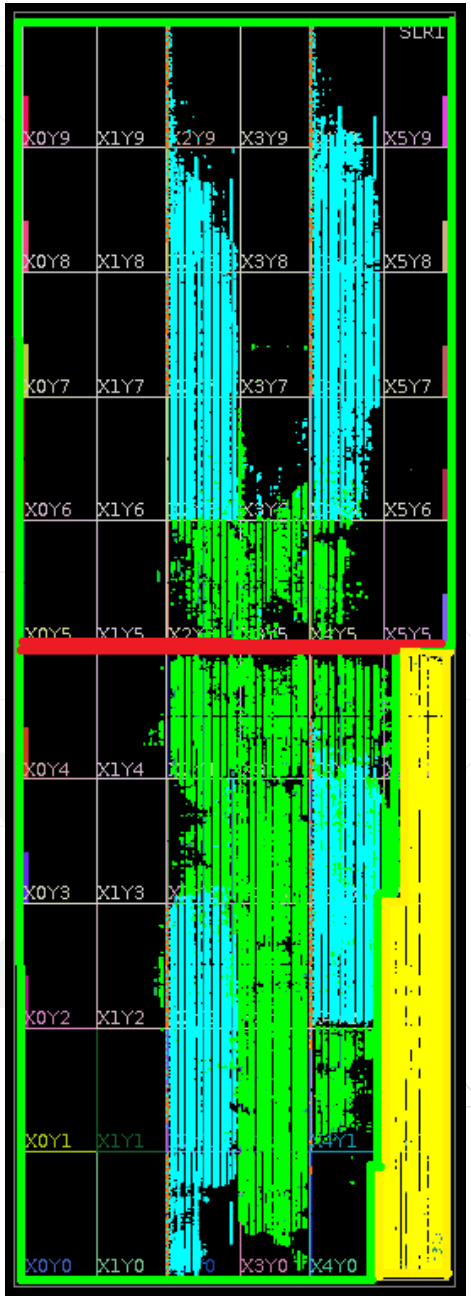




A new kind of platform
Kintex UltraScale KU115 platform with *Expanded* PR

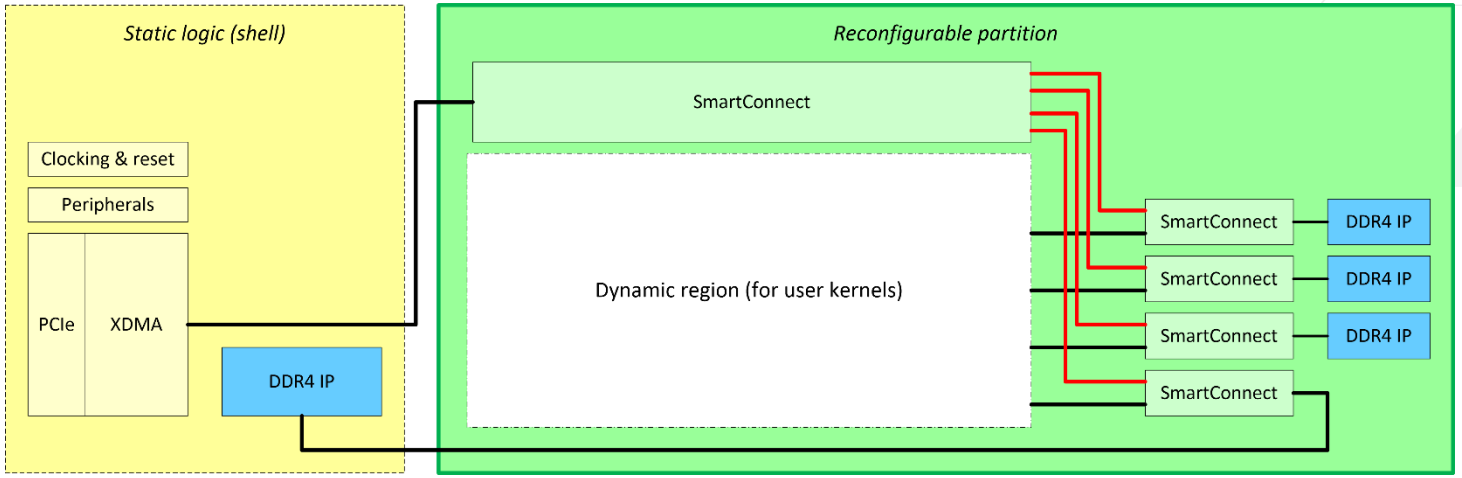
- **First Xilinx-built board for SDAccel**
- **Four DDR4 memory controllers for 2x bandwidth**
- **… but no good floorplan with the standard platform approach**
- **Adopted a new approach: *Expanded* Partial Reconfiguration**
  - Static shell is much smaller
  - Distributed interconnect better manages SLR crossings
  - Dynamic region for kernels now a subset of the reconfigurable module

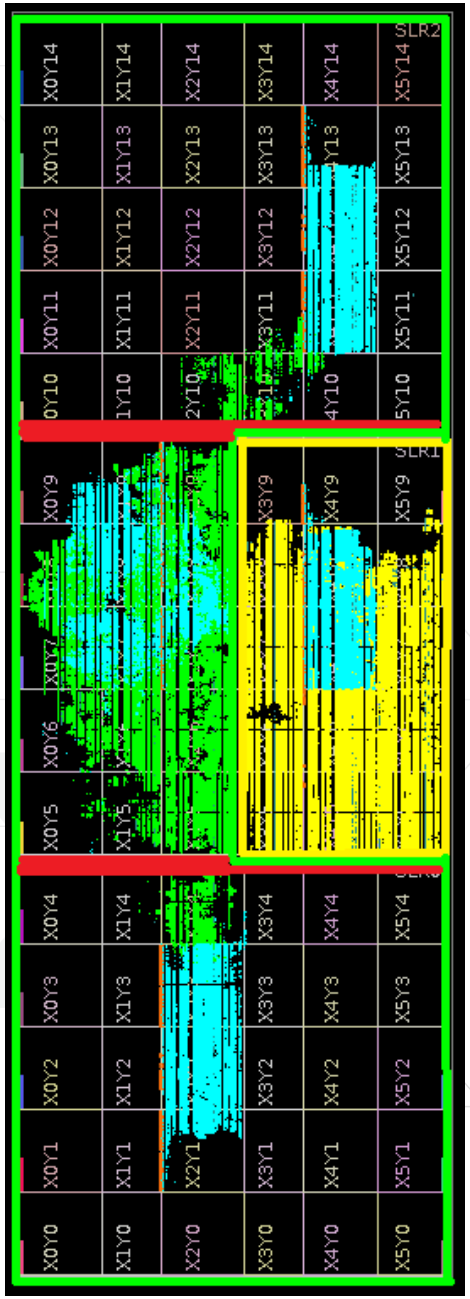




Pushing the limits
Virtex UltraScale+ VU9P platform with *Expanded* PR

- **Same approach as in KU115-based expanded PR platform**
- **Now with 3 SLRs and more interconnect logic**
- **But several inefficiencies becoming more prominent**
  - Just too much static logic for simple use cases
  - Automatic placement of kernels sometimes inconsistent with manually-placed platform infrastructure
  - Still no variability of platform contents … DIMMs?, debug IP?, etc.



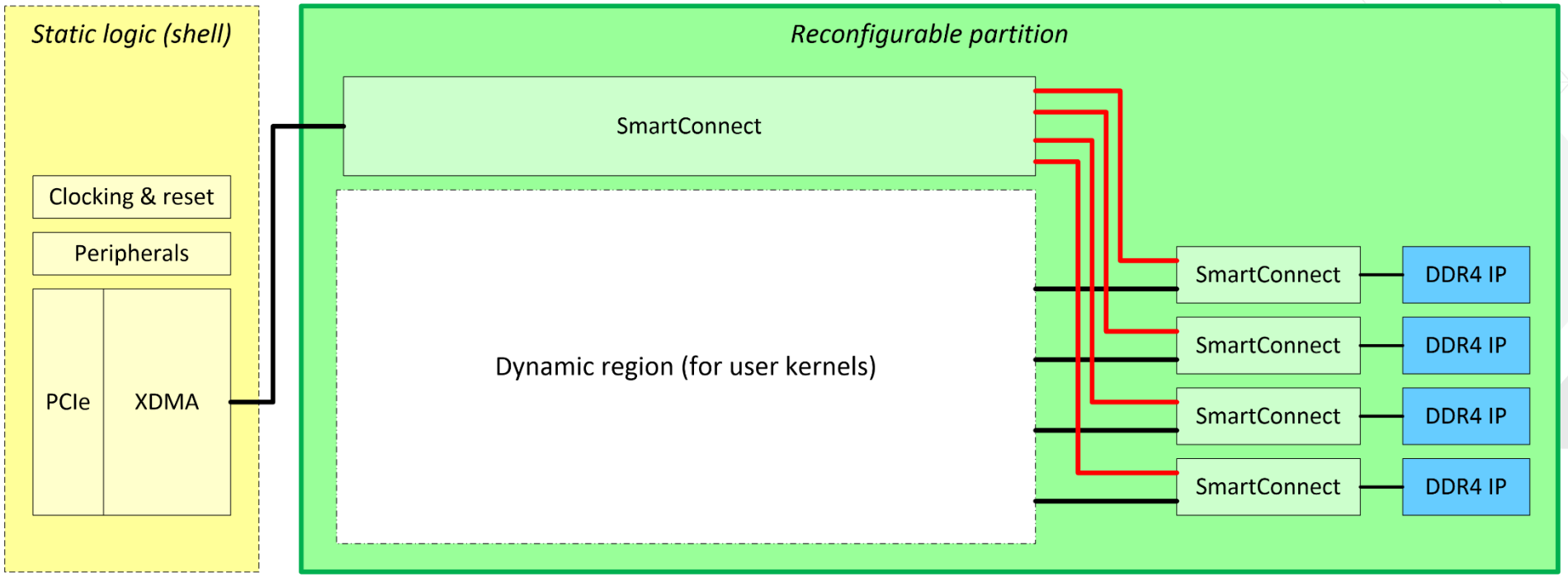


The reality

- **VU9P platform with Expanded Partial Reconfiguration is a powerful, full-featured platform**
- **And maybe it is perfect for the user**
  - Exactly the right number of DDR4 memory channels vs. free fabric resources
  - Just the right performance tuning
  - Works perfectly with user kernels (no debug necessary)
  - Performance profiling taps are in the right places
  - Ideal logic placement for kernels relative to platform interconnect
- **But if not…**
  - For a different number of DDR4 channels: **new platform**
  - For different performance tuning / memory parameters: **new platform**
  - Any issues requiring debug IP inserted at the problematic location: **new platform**
  - Modified performance profiling tap points: **new platform**
  - Different interconnect topology or for required kernel placement: **new platform**

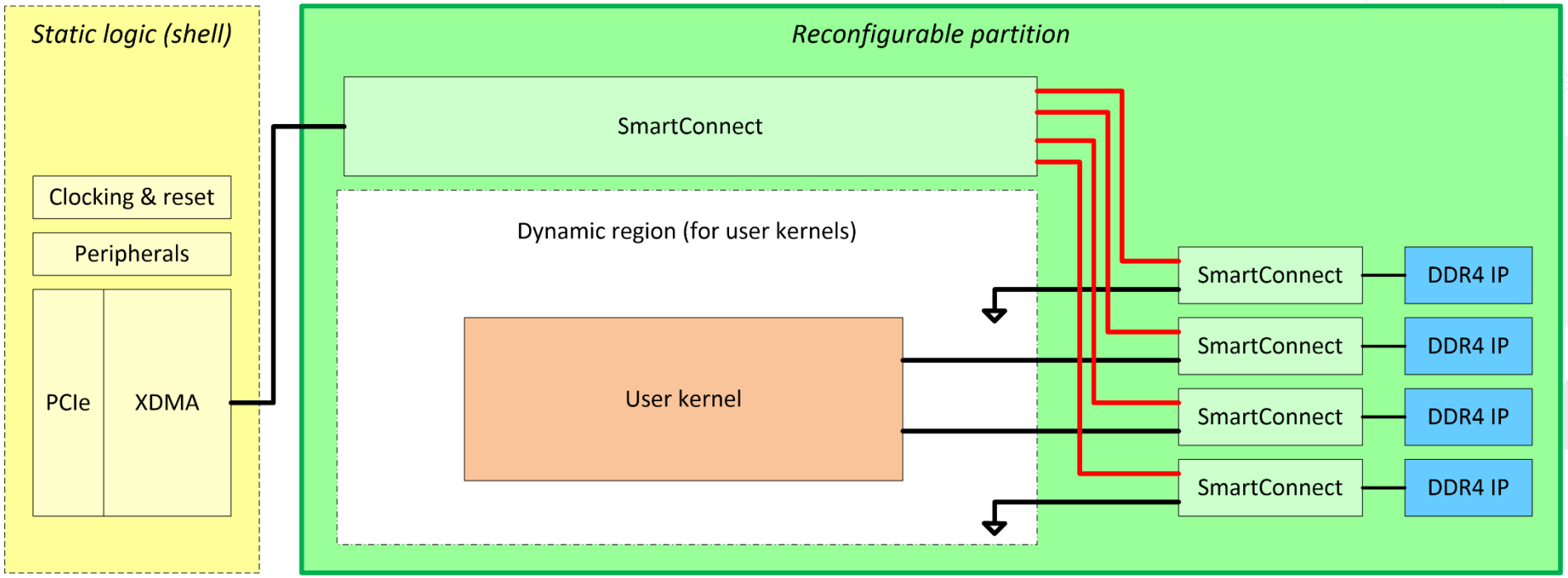
A better way

> **Platform logic in the reconfigurable module was fixed for every kernel build…**



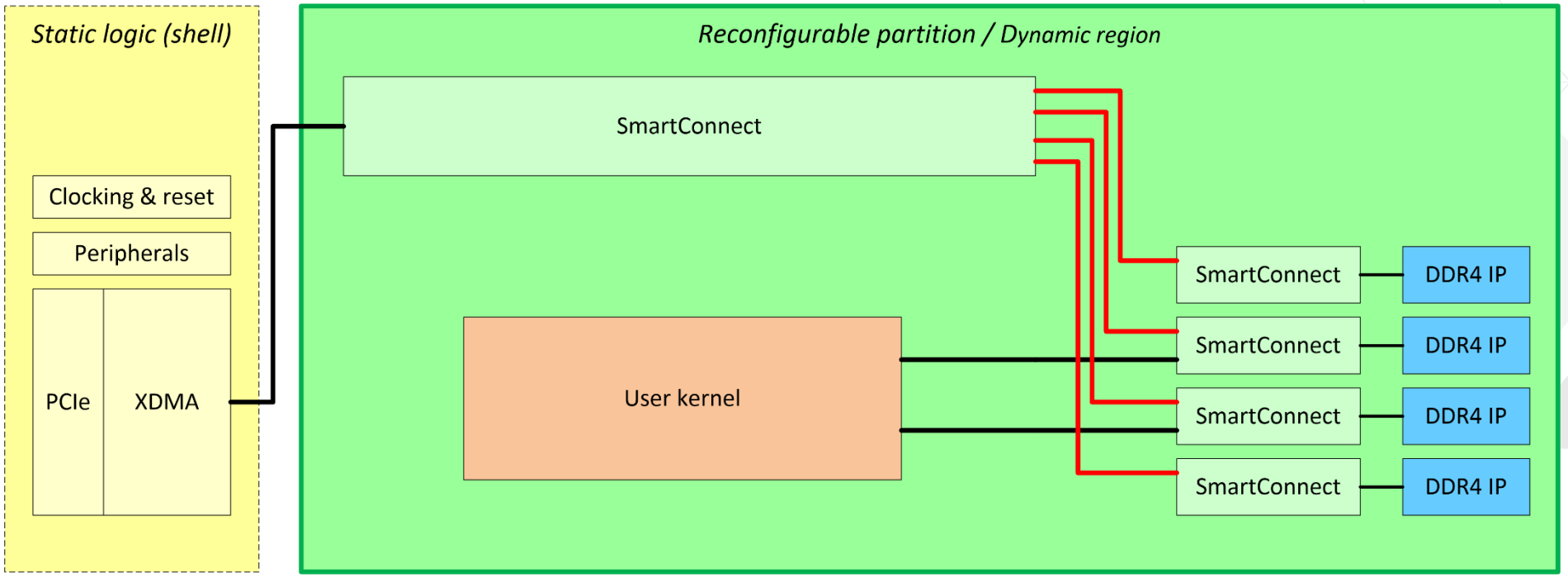
A better way

- Platform logic in the reconfigurable module was fixed for every kernel build…
- …leading to wasted FPGA area, power, and Vivado runtime when unused



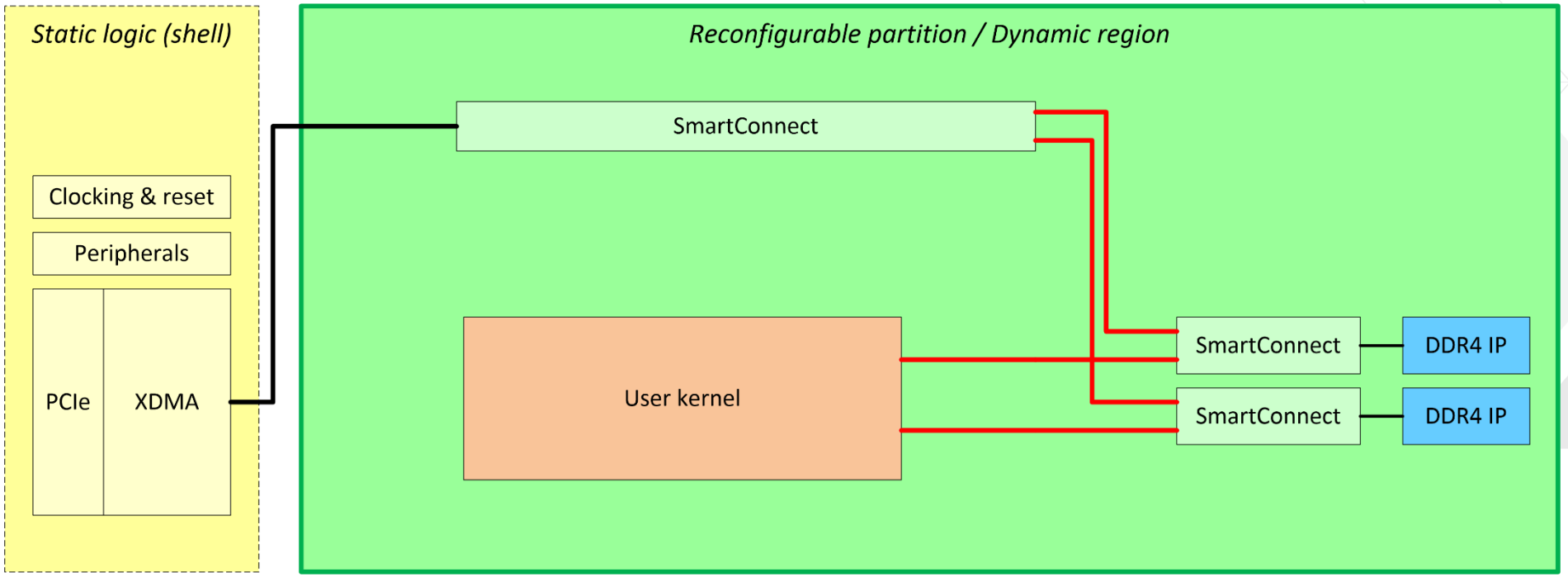
A better way

- Platform logic in the reconfigurable module was fixed for every kernel build…
- …leading to wasted FPGA area, power, and Vivado runtime when unused
- But if the entire dynamic region – not just the user kernels – could be modified by SDx…



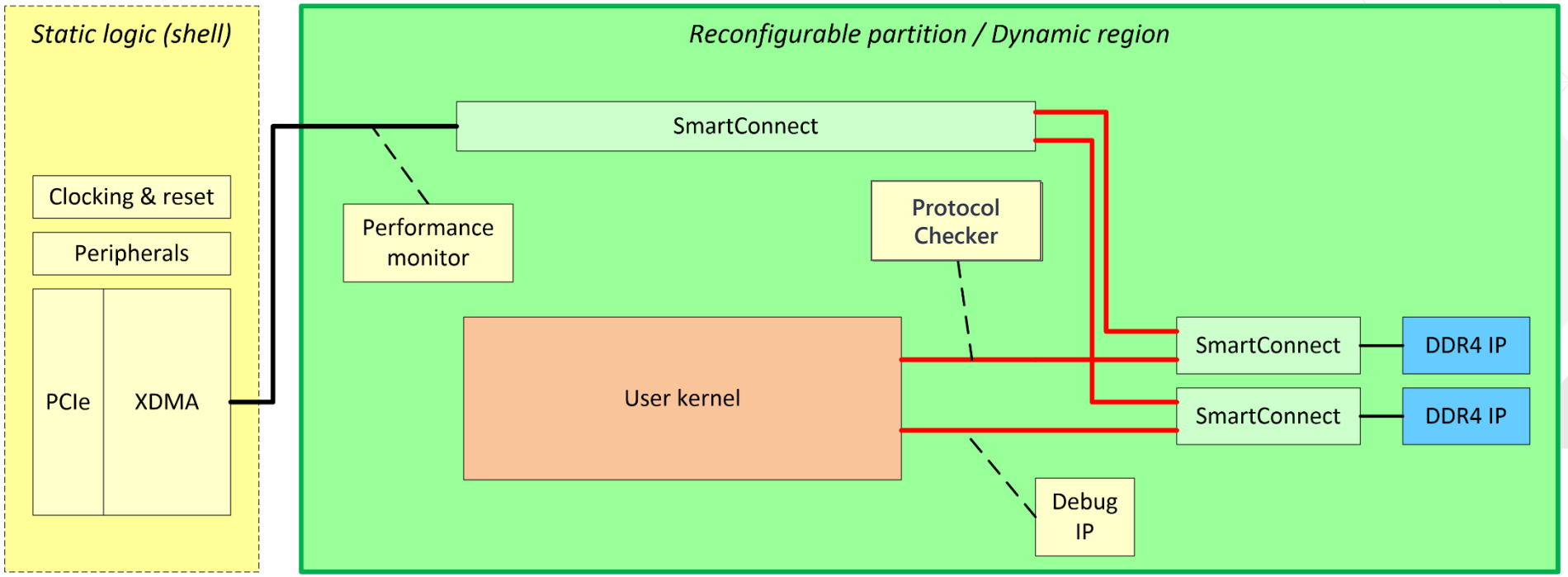
A better way

- Platform logic in the reconfigurable module was fixed for every kernel build…
- …leading to wasted FPGA area, power, and Vivado runtime when unused
- But if the entire dynamic region – not just the user kernels – could be modified by SDx…
- …then the platform infrastructure could best suit the kernels <u>for every build, automatically</u>



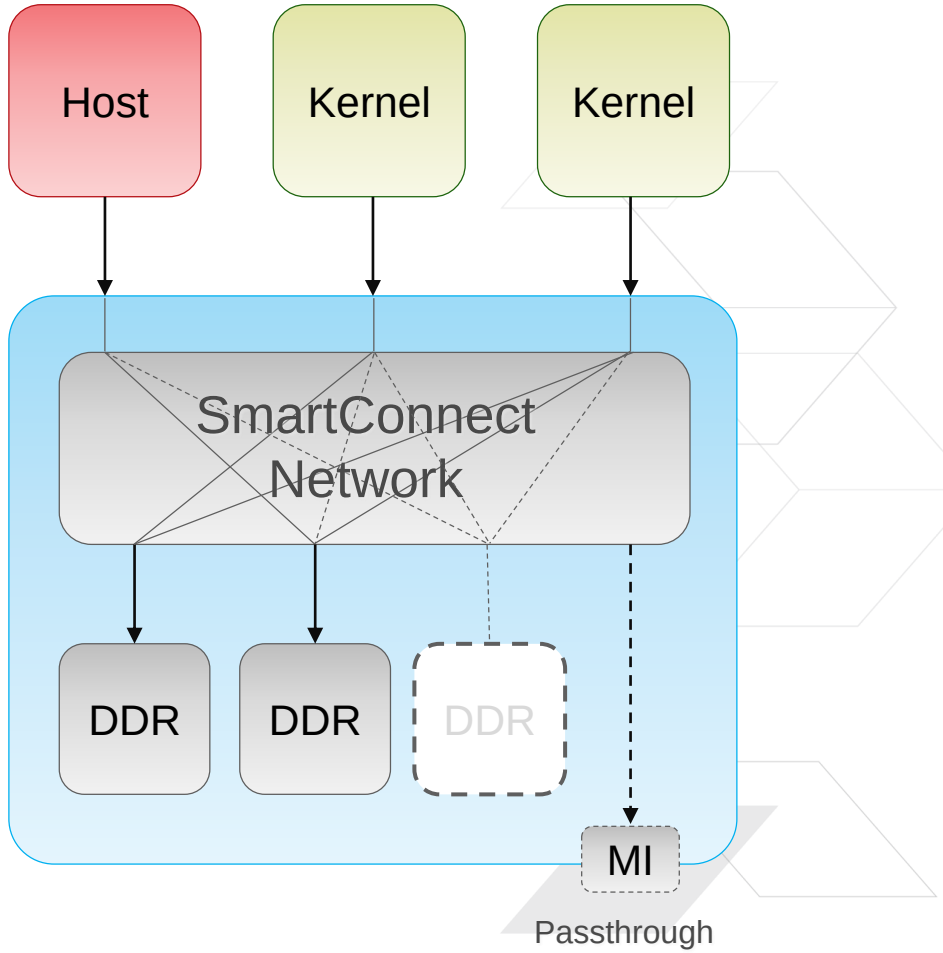
A better way – Dynamic Platforms

- Platform logic in the reconfigurable module was fixed for every kernel build…
- …leading to wasted FPGA area, power, and Vivado runtime when unused
- But if the entire dynamic region – not just the user kernels – could be modified by SDx…
- …then the platform infrastructure could best suit the kernels <u>for every build, automatically</u>
- *And* we'd have the framework for SDx to dynamically build up just what the user needs



Memory subsystem Hierarchical IP

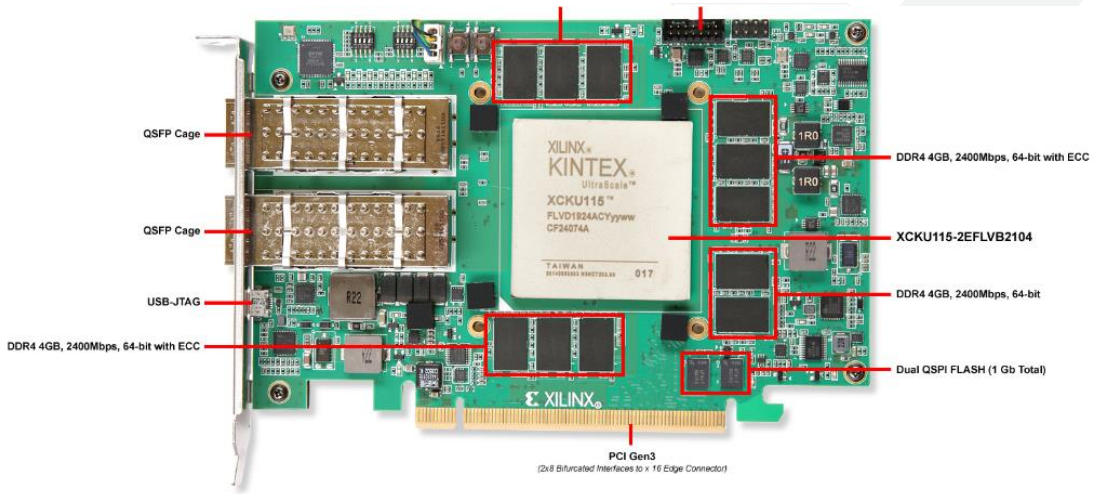
- **Encapsulate and abstract DDR-based memory resources and associated data-interconnect instances**
  - Data-driven dynamic generation of interconnect and MIG instances based on application use
- **Simplify dynamic region design**
  - Single IP instance versus hand constructed network of SmartConnect and MIG instances
- **Simplify linking of kernels**
  - Resource based model of platform memory
  - Resource-Connection API hides IP-specific details of attaching Kernel's AXI interface to memory subsystem

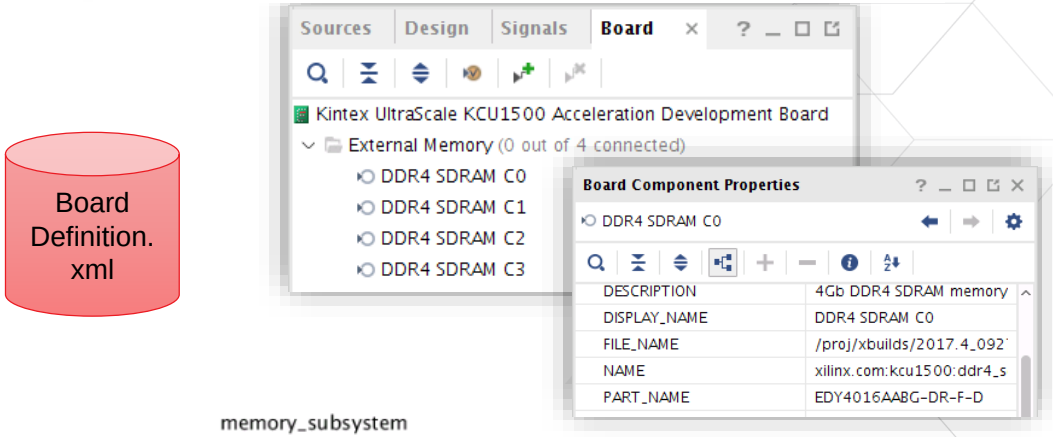


Board-driven DDR resources

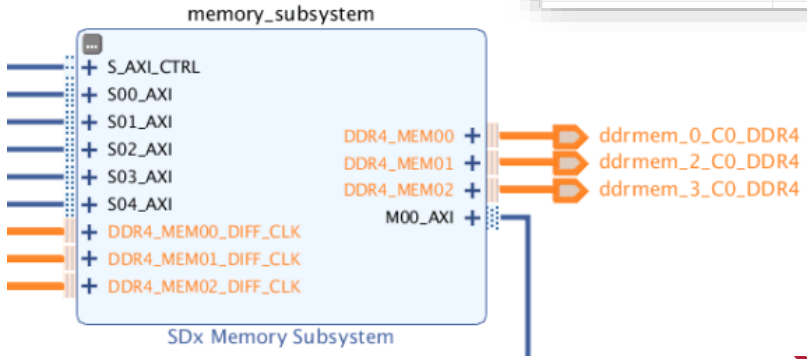
- **What DDR resources are available in the platform?**
  - Avoid manual configuration specified by each DSA developer

- **Board infrastructure provides a better experience for DSA developer**
  - Xilinx standard approach for board modeling
  - Board components document which DDR memory resources are available
  - Board infrastructure presets can be embedded in the board definition to provide a working configuration for the memory controller IPs

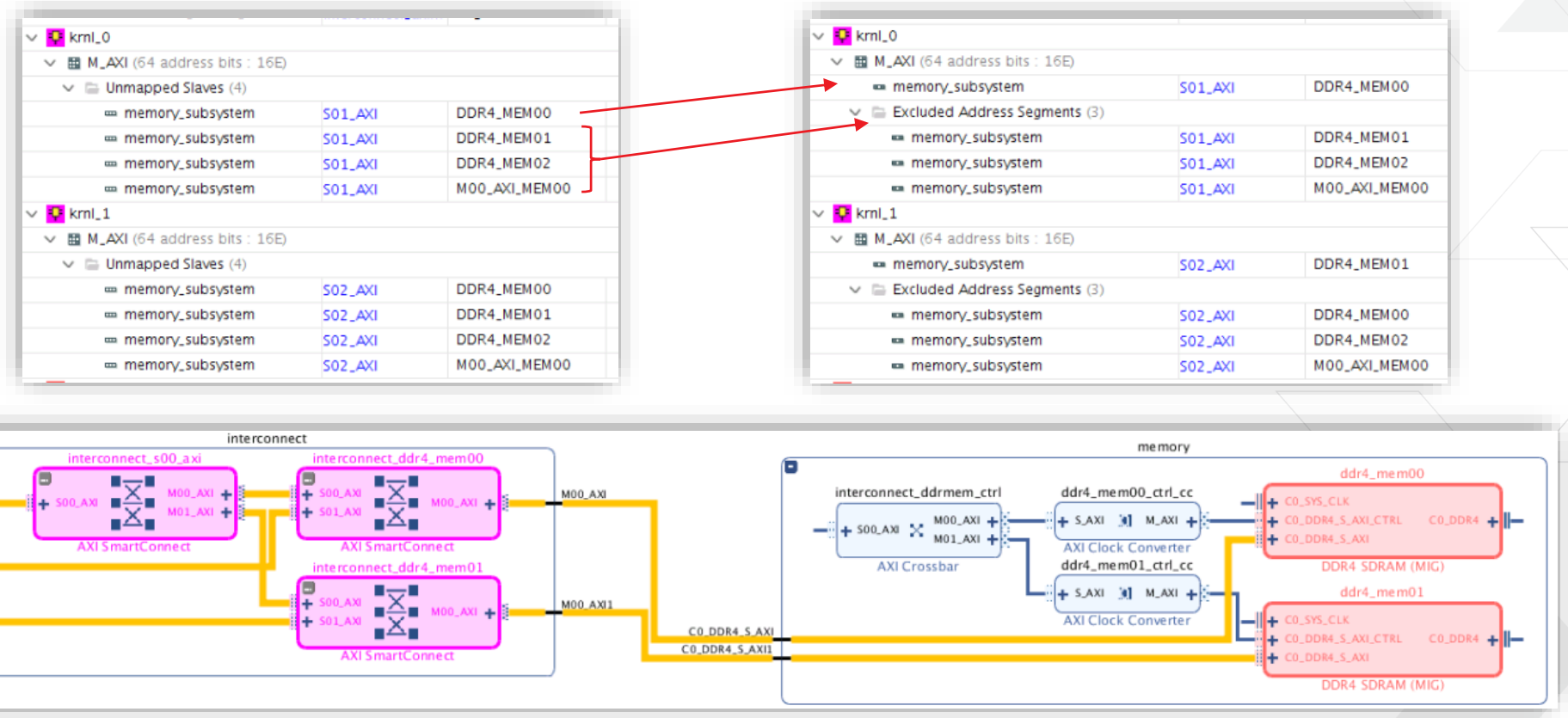




Board Definition. xml

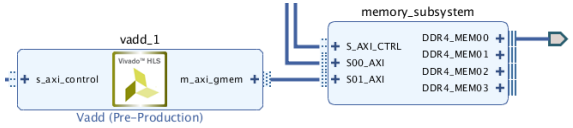


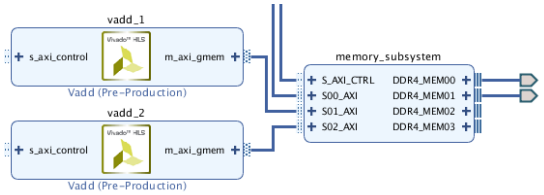
Memory mapping and dynamic system topology

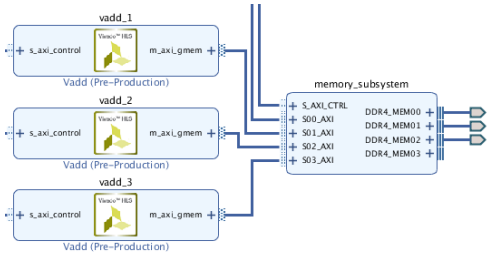


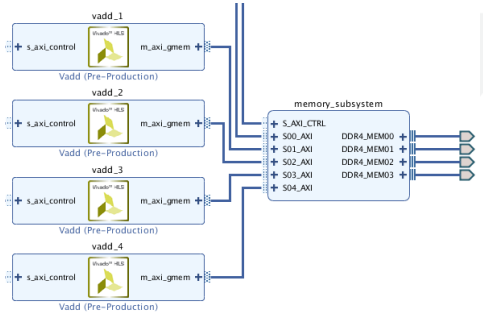
- **AXI Interfaces advertise the set of available DDR memory resources within the memory subsystem to IPI's address editor**
  - Dynamic region memory controllers not actually instantiated at this point
  - Only create memory controllers for the memory resources that actually get mapped into Kernel's address space

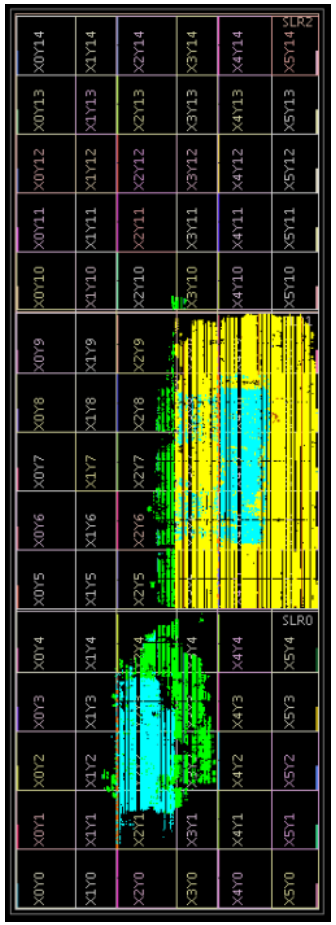
Only build what you need



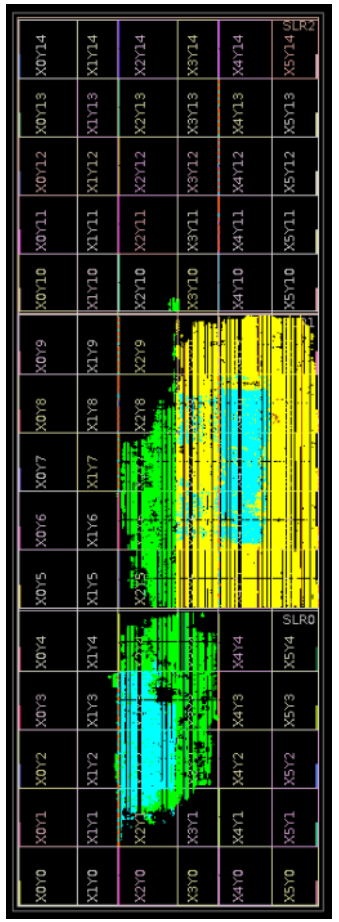




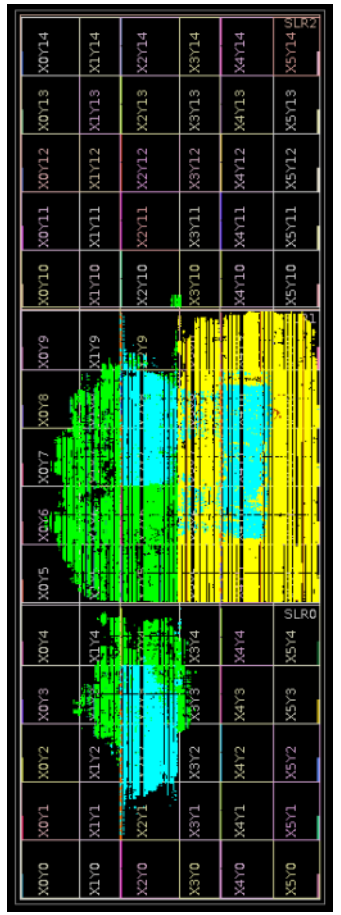




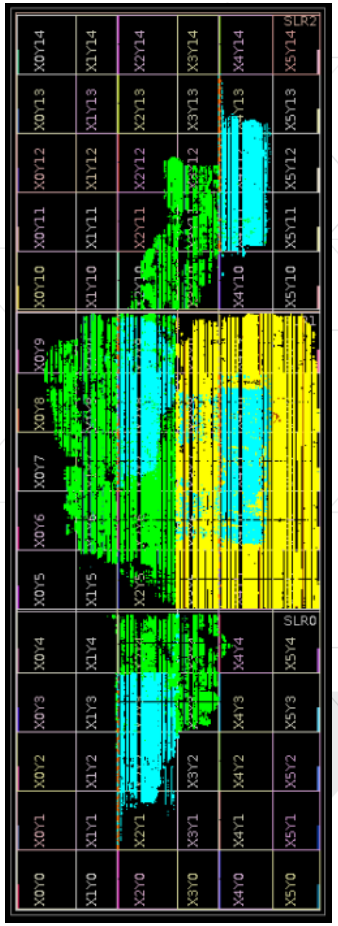
1 DDR



2 DDRs



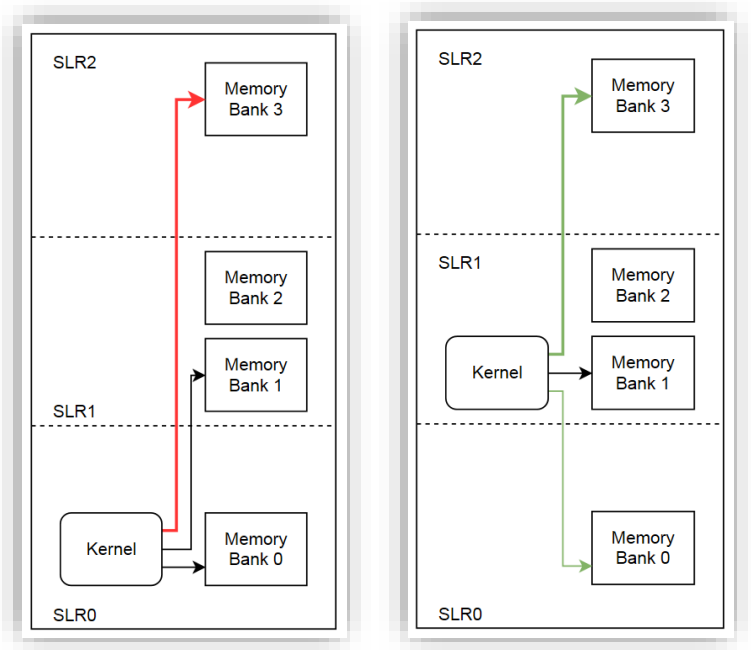
3 DDRs



4 DDRs

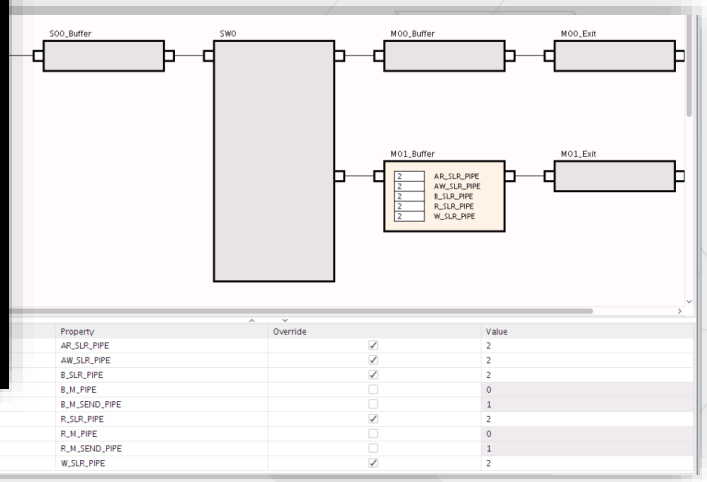
Future Enhancements for Partial Reconfiguration and Platform Design

SLR automation




```
set_property CONFIG.SLR_ASSIGNMENTS SLR0 [get_bd_cells vadd_0]
set_property CONFIG.SLR_ASSIGNMENTS SLR1 [get_bd_cells vadd_1]
set_property CONFIG.SLR_ASSIGNMENTS SLR0 [get_bd_cells vadd_2]
```

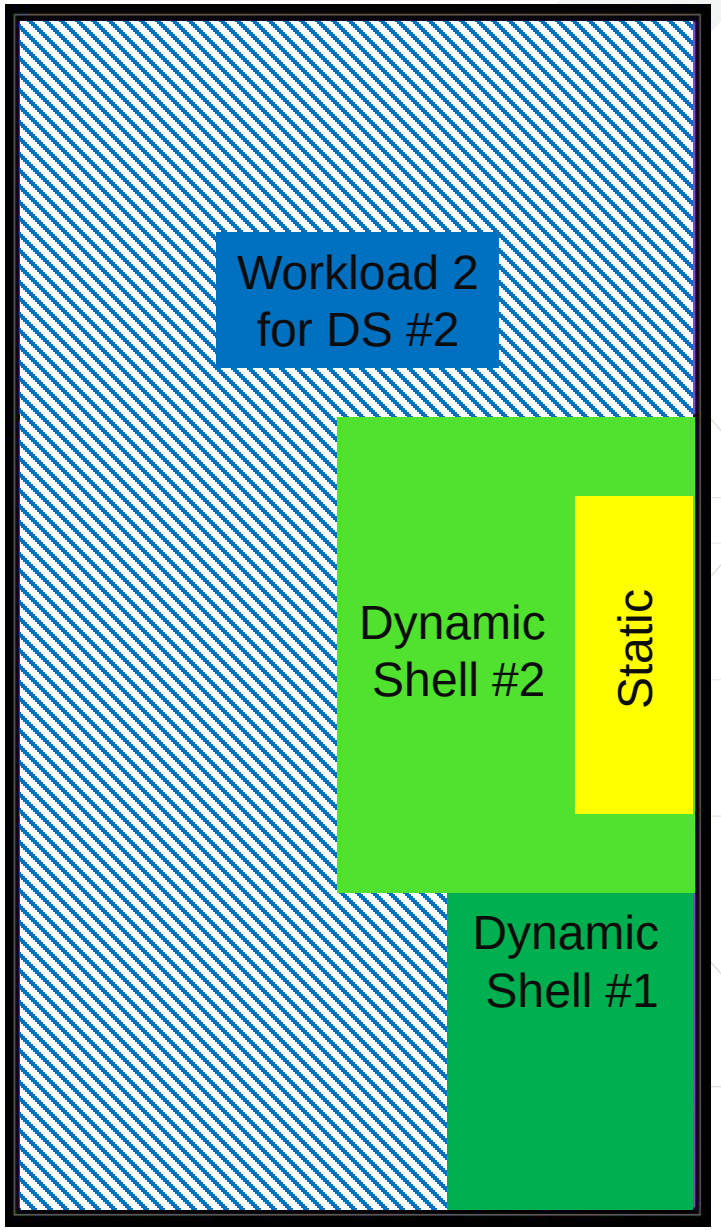
```
# ----------------------------------------------------------------------
# SLR-crossing PBLOCK assignments
# ----------------------------------------------------------------------
# S00_AXI:SLR1 -> interconnect_s00_axi:SLR1
add_cells_to_pblock [get_pblocks pblock_dynamic_slr1] \
  [list \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_aw_node/inst/inst_si_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_ar_node/inst/inst_si_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_w_node/inst/inst_si_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_r_node/inst/inst_mi_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_b_node/inst/inst_mi_handler \
  interconnect/interconnect_s00_axi/inst/s00_entry_pipeline \
  interconnect/interconnect_s00_axi/inst/s00_axi2sc \
 ]
add_cells_to_pblock [get_pblocks pblock_dynamic_slr1] \
  [list \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_aw_node/inst/inst_mi_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_ar_node/inst/inst_mi_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_w_node/inst/inst_mi_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_r_node/inst/inst_si_handler \
  interconnect/interconnect_s00_axi/inst/s00_nodes/s00_b_node/inst/inst_si_handler \
 ]
# interconnect_s00_axi:SLR1 -> M00_AXI:SLR0
```



- **FaaS platforms commonly use SLR-based devices**
  - Sometimes a kernel doesn't reside in the same SLR as the memory bank(s) it accesses
- **Memory Subsystem automates SLR crossings and IPI SLR automation provides a simple way for the user to allocate kernels to SLR**
  - Memory Subsystem samples IPI's SLR annotations to determine how to apply SLR crossing resources
  - SDx 'software developer' can avoid low level details of XDC placement constraints and microarchitectural options for SLR timing pipes

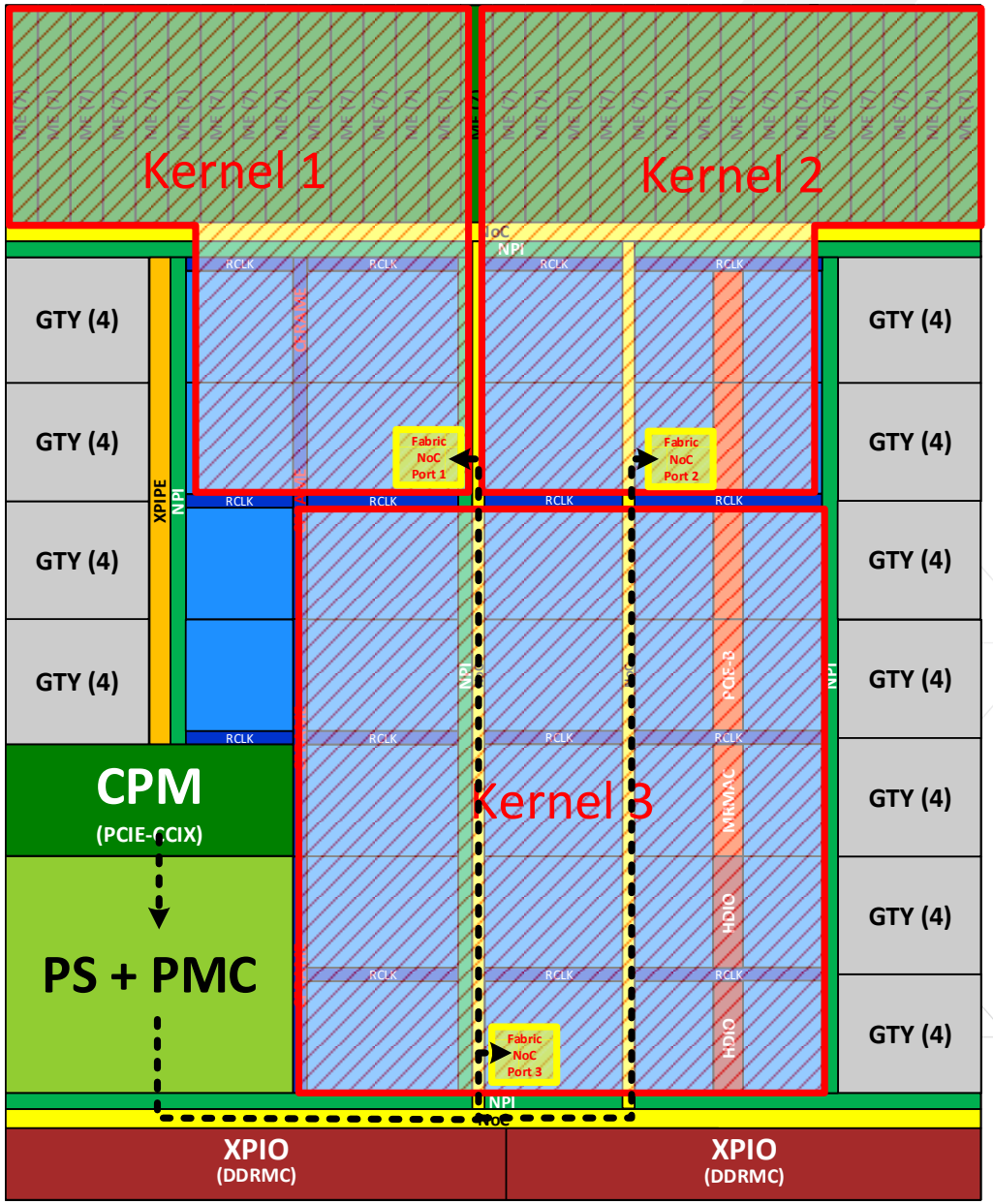
Dynamic Shell

- **Support multiple platforms on a single shell**
- **Example**
  - Locked static design
  - Dynamic shell with specific features (e.g. DMA)
  - Multiple workloads for this shell
- **New shell features loaded without changing static**
  - New dynamic shells can have:
    - Different sizes and floorplans
    - Different features
    - Different interfaces to workload space
    - Optimized for user application
  - Workloads must be compatible with a specific dynamic shell
    - Bitstream manager must keep track of current state



The next generation: ACAP

- **Hardened Memory Controllers + DDRIO**
  - Allows DDR memory controllers to remain active while fabric is reconfigured
  - Moving DDRIO out of fabric improves ease of PR floorplanning
- **Hardened PCIe + DMA**
  - PCIe interface can be brought into operation with minimal programming
- **NoC Ports in Fabric**
  - High bandwidth datapath to PMC for PR
- **Configuration Performance Increase**
  - Max bandwidth 8X faster than Zynq US+ MPSoC (6.4GByte/sec)*
  - Capable of reconfiguring 1M logic cells in under 8ms



* When using an external interface such as PCIe, DDR memory, etc.

Summary

Robust Silicon Technology

> Decades of investment coupled with extensive validation of dynamic configuration solutions

Evolution of Platform Development

> Steadily increasing optimization for dynamic acceleration

Exciting Future Ahead

> Enhanced flexibility for the most efficient use of silicon

XDF XILINX DEVELOPER FORUM