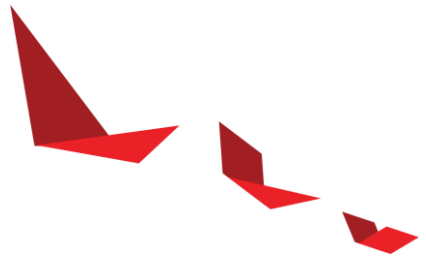
XILINX

Dynamic Function eXchange featuring Abstract Shell

David Dye
Xilinx Adapt
February 2021

Agenda



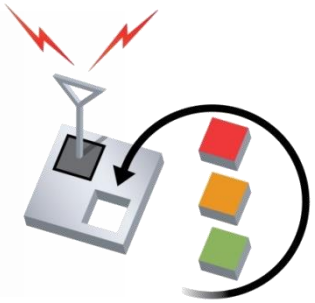
- Vivado Dynamic Function eXchange (DFX) solution
- New DFX flow capabilities
  - Nested DFX
  - **Abstract Shell**
- Customer Testimonial – Abaco Systems
- Forthcoming DFX Solutions
  - Block Design Containers
  - Versal DFX

Xilinx DFX Solution Overview

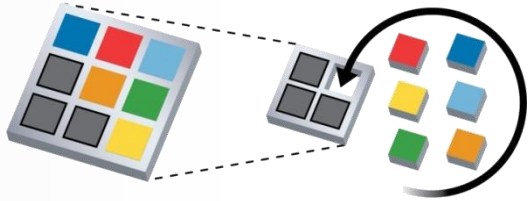
XILINX.

Dynamic Function eXchange Overview

- Expand System Flexibility
  - Swap functions on the fly
  - Perform remote updates while system is operational



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

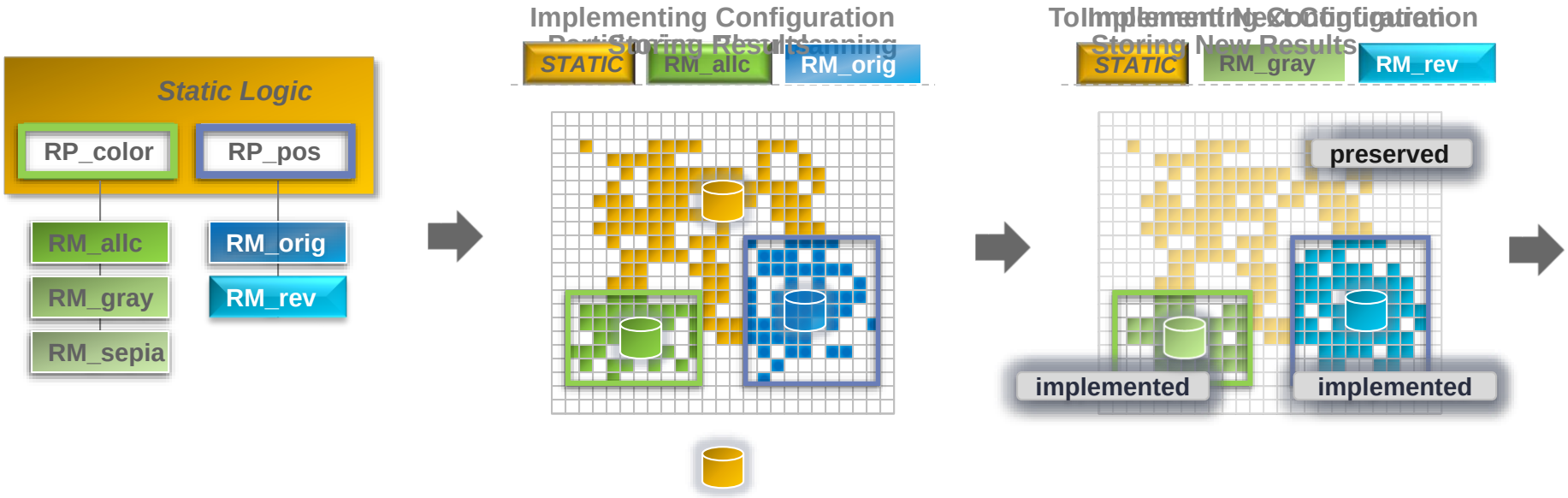


- Deploy Dynamic Systems
  - Dynamically load accelerated functions
  - Change features to systems in the field

XILINX.

Vivado DFX Tool Flow

- In-context multi-pass hierarchical place and route solution
  - First pass: route static plus one Reconfigurable Module (RM) per Reconfigurable Partition (RP)
  - Remaining passes: route new RMs in context of locked static image
  - Floorplan required to identify static vs. dynamic resources

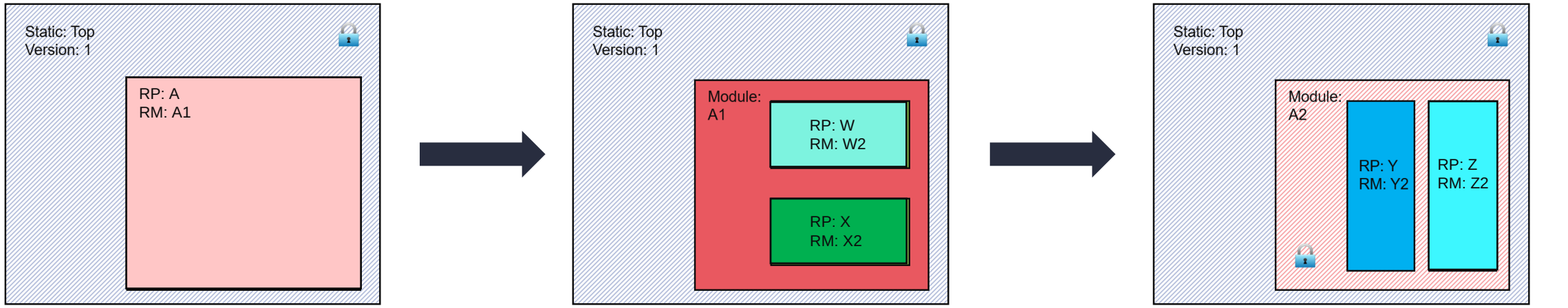


XILINX.

New DFX Solution Capabilities

Nested DFX – Expand Silicon Flexibility

- DFX within DFX: Subdivide an existing DFX region into multiple lower-level DFX regions
- Implementation flow follows hierarchical order
  - Change scope of static / reconfigurable boundary down to a lower level

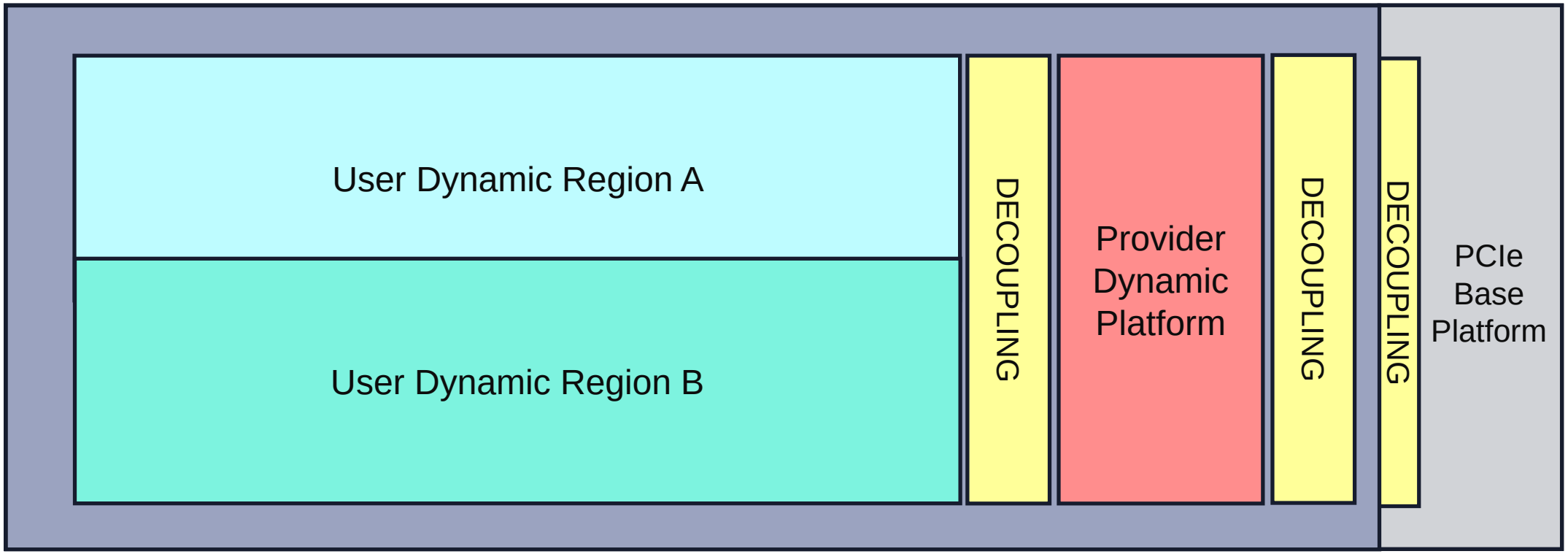


1st order implementation **2nd order implementation, ver 1** **2nd order implementation, ver 2**

- New in Vivado 2020.1 for UltraScale and UltraScale+ devices
  - Tcl script only; project mode support on roadmap

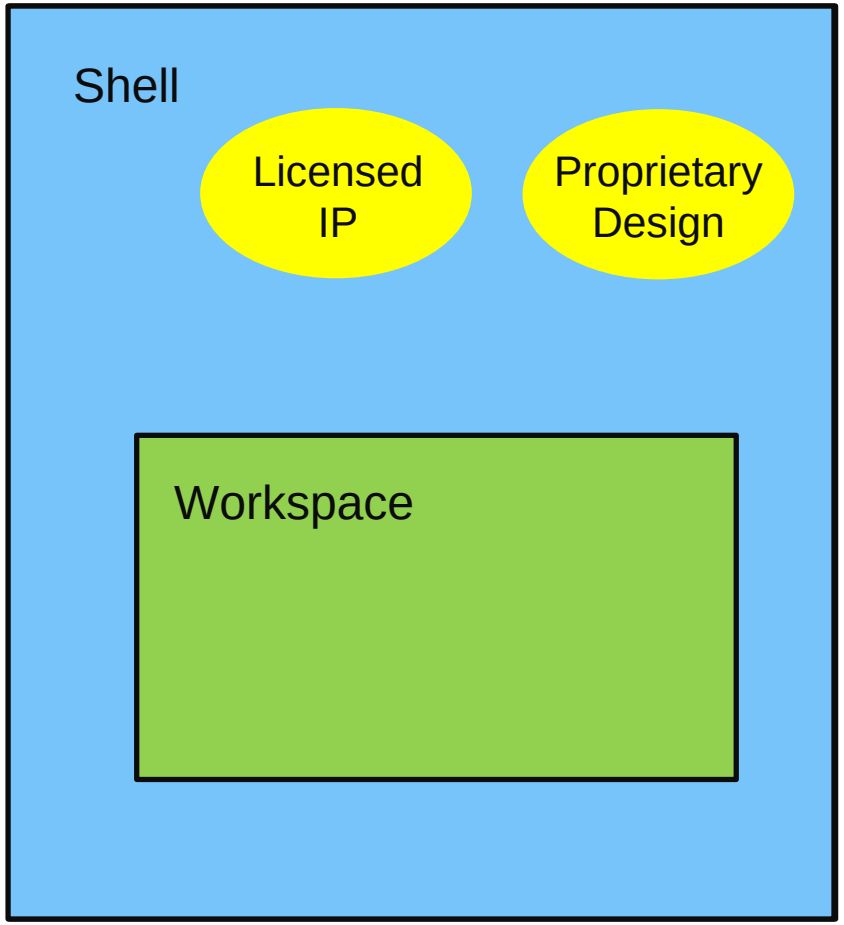
Nested Dynamic Function eXchange

- Technology enables Dynamic Platforms



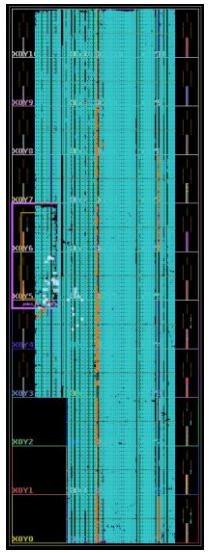
Abstract Shell for DFX

- Abstract Shell = Static Design limited to a minimal interface around target Reconfigurable Partition
  - Everything needed to implement Reconfigurable Modules must be included in DCP
- Key Solution Details
  - Write complete RP interface (logical, physical) in Abstract Shell
  - Must achieve sign-off timing from Abstract Shell environment
  - Generation of partial bitstreams from Abstract Shell environment
- Key Benefits
  1. Faster runtime and lower memory usage
  2. Compile all Reconfigurable Modules in parallel
  3. Hide proprietary static design information
  4. Bypass IP license check tags for static design
- Production release in Vivado 2020.2 for UltraScale+

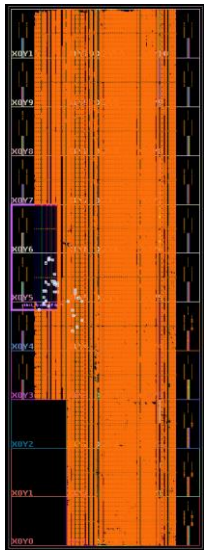


Standard DFX flow vs Abstract Shell flow

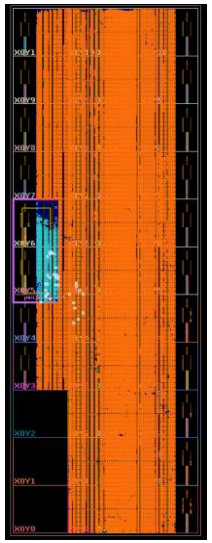
Standard DFX flow



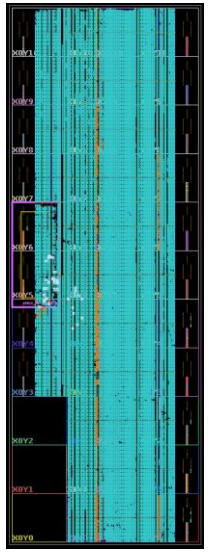
```
update_design -black_box
   -cell <RM cell>
lock_design -level routing
write_checkpoint -force
   static_bb.dcp
```



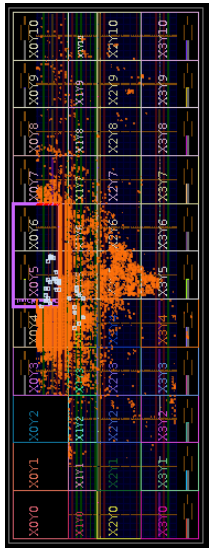
```
add_files static_bb.dcp
add_files rm2_synth.dcp
link_design
#Implement
```



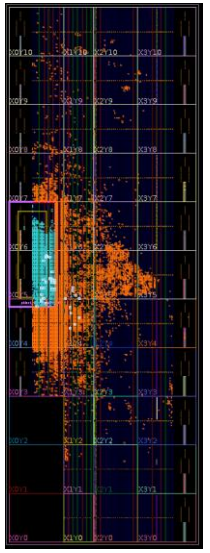
Abstract Shell flow



```
write_abstract_shell -cell
   <RM cell> abs_shell.dcp
```



```
add_files abs_shell.dcp
add_files rm2_synth.dcp
link_design
#Implement
```



Identical first implementation flow

Update_design and lock_design routines embedded in write_abstract_shell

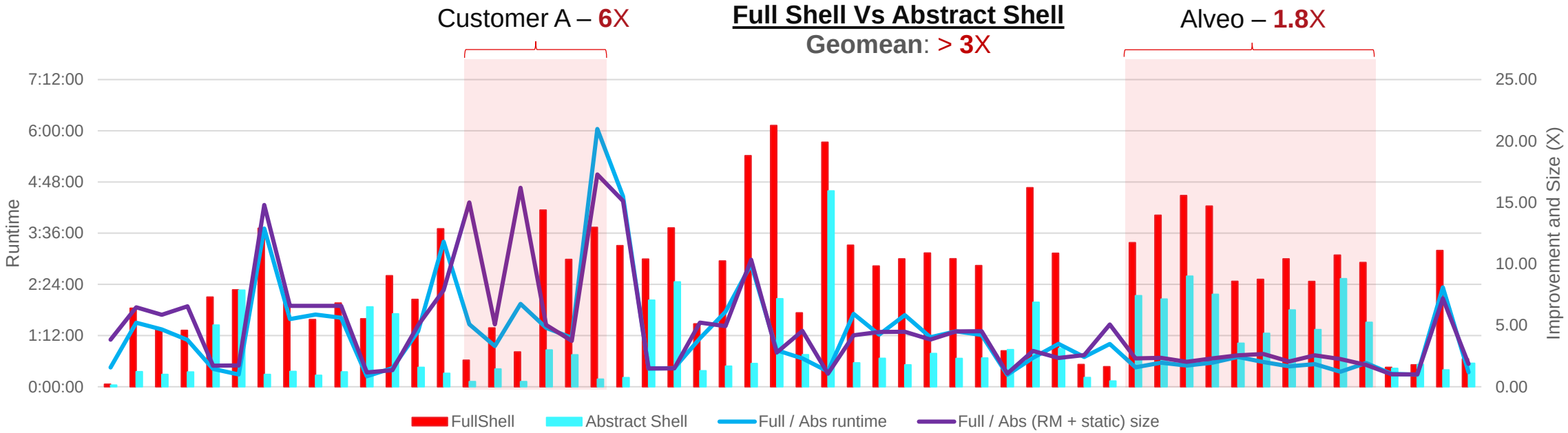
Identical second implementation flow, but different starting checkpoint

Routed DCP after first implementation

Intermediate DCP to second implementation

Second Implementation Routed DCP

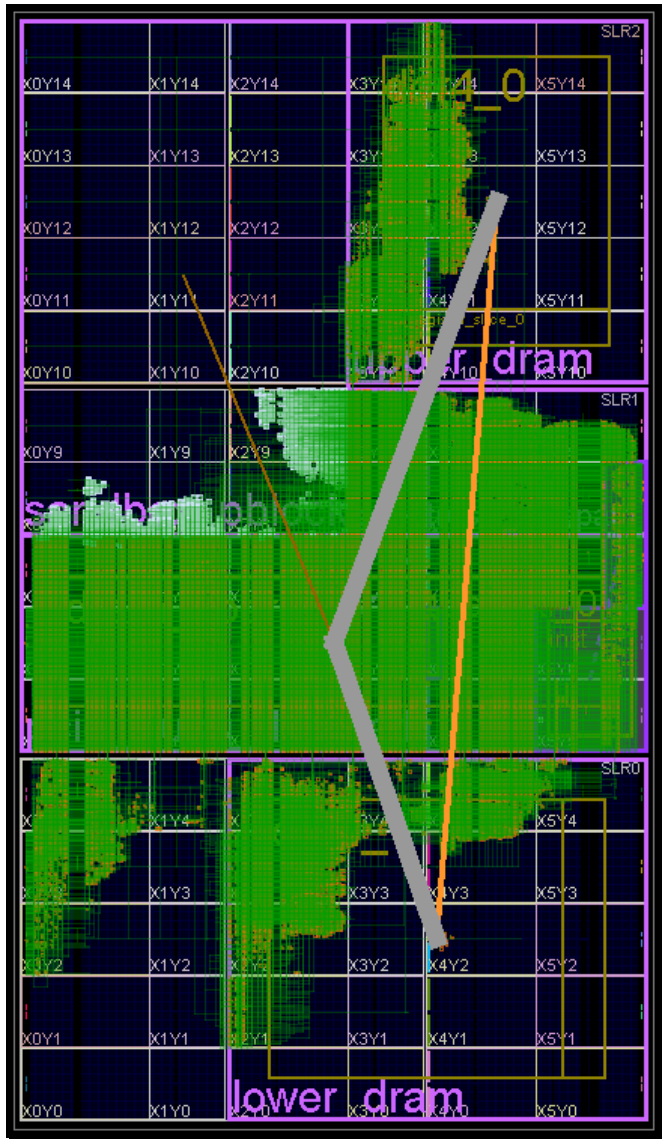
Abstract Shell Reduces Implementation Runtime

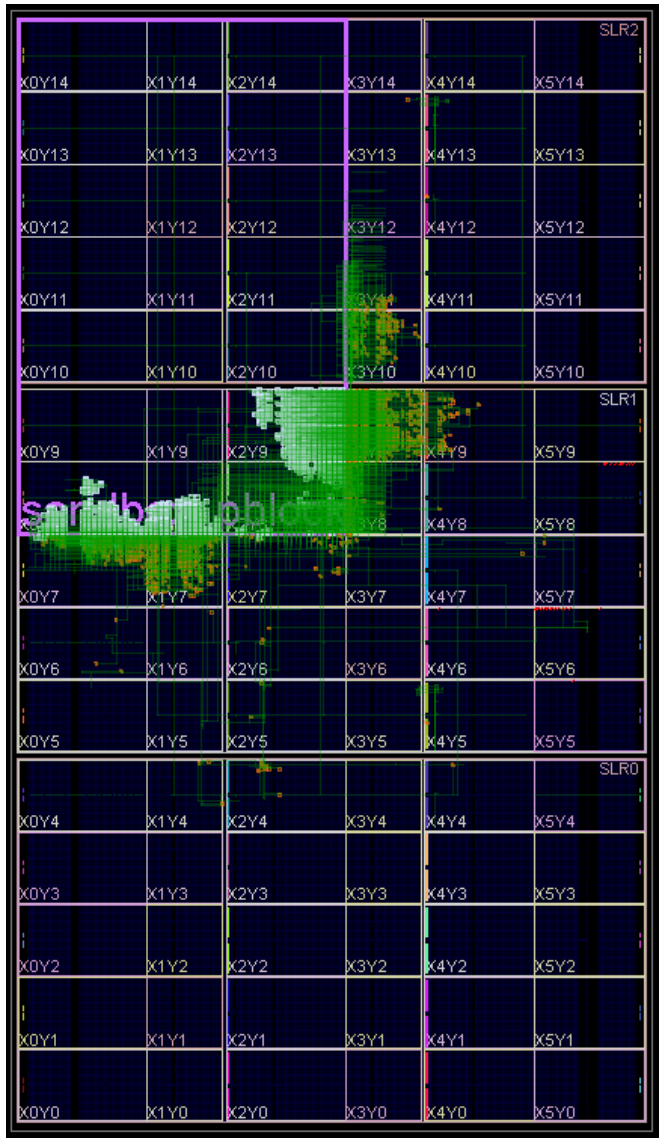


- Average improvement across varied suite: 3X faster
- Runtime improvements depend on (static + RM) design size
  - Alveo platforms have smaller static portions compared to general DFX (e.g. Customer A) designs
  - Second order dependencies: Isolation of RM from static, tool algorithm heuristics
- Memory Usage: 10x Design Checkpoint (.dcp) size reduction

Customer Example Design

- VCU118 (VU9P) design
  - 3 SLR device, 1 RP
  - 2 memory interfaces + PCIe
  - File size 331 MB
- Full Shell (locked static)
  - File size 266 MB
  - 97 min to implement RM
  - 21385 MB peak memory
- Abstract Shell
  - File size 10 MB
  - 27 min to implement RM
  - 14446 MB peak memory

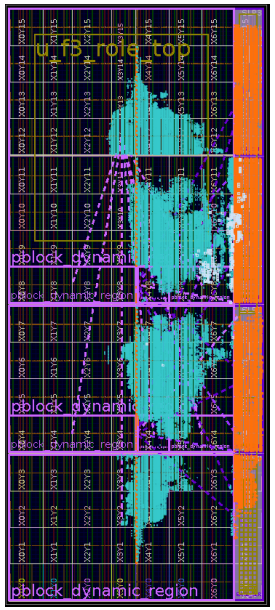


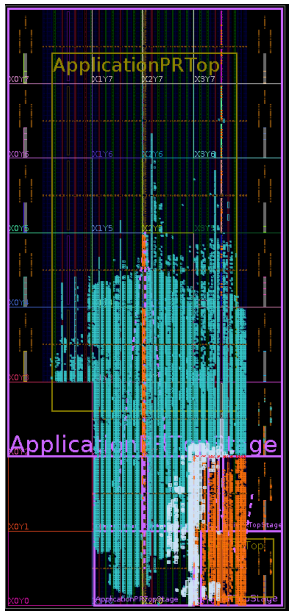


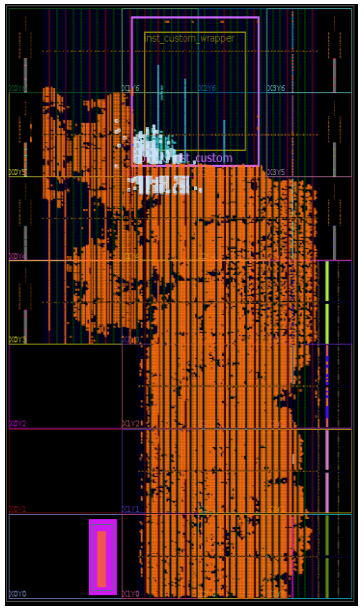
XILINX.

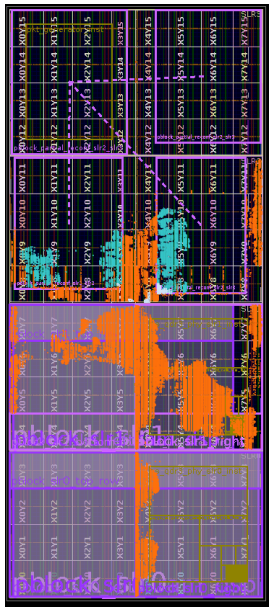
More Examples of Customer Designs

Full Shell RM result

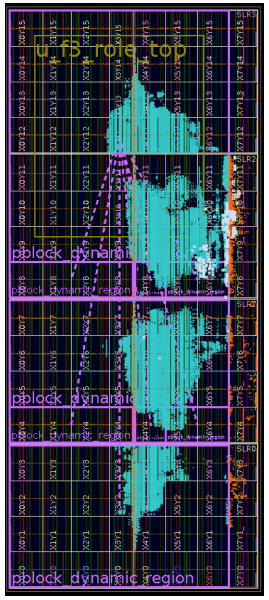


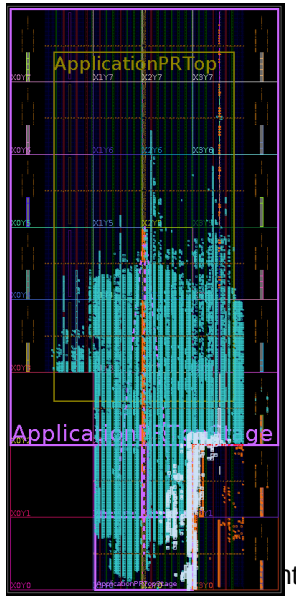


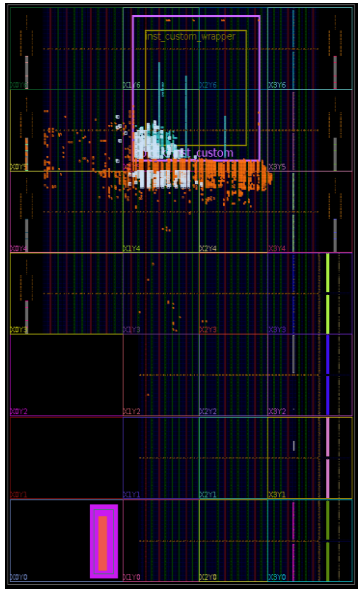


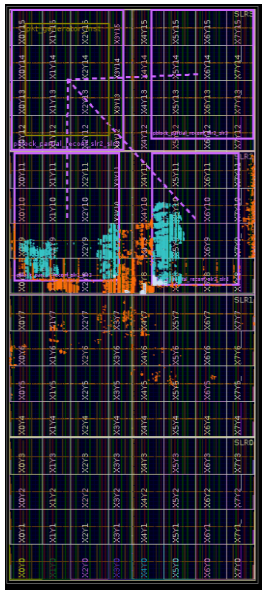


Abstract Shell RM result



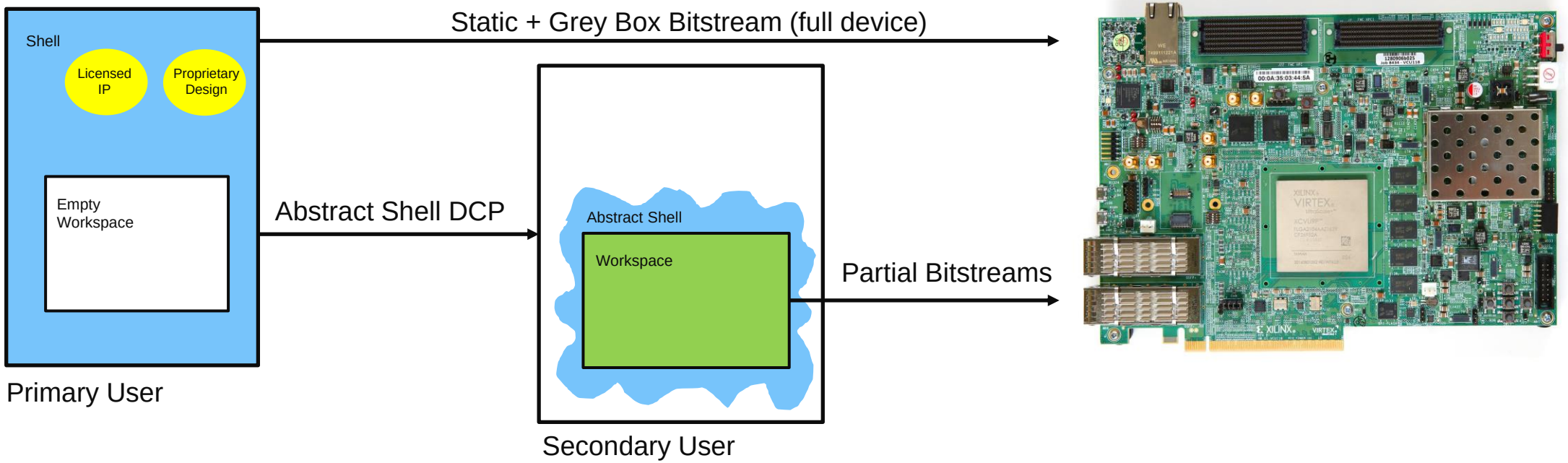






XILINX.

Multi-User Scenario – Secondary User Programming Path

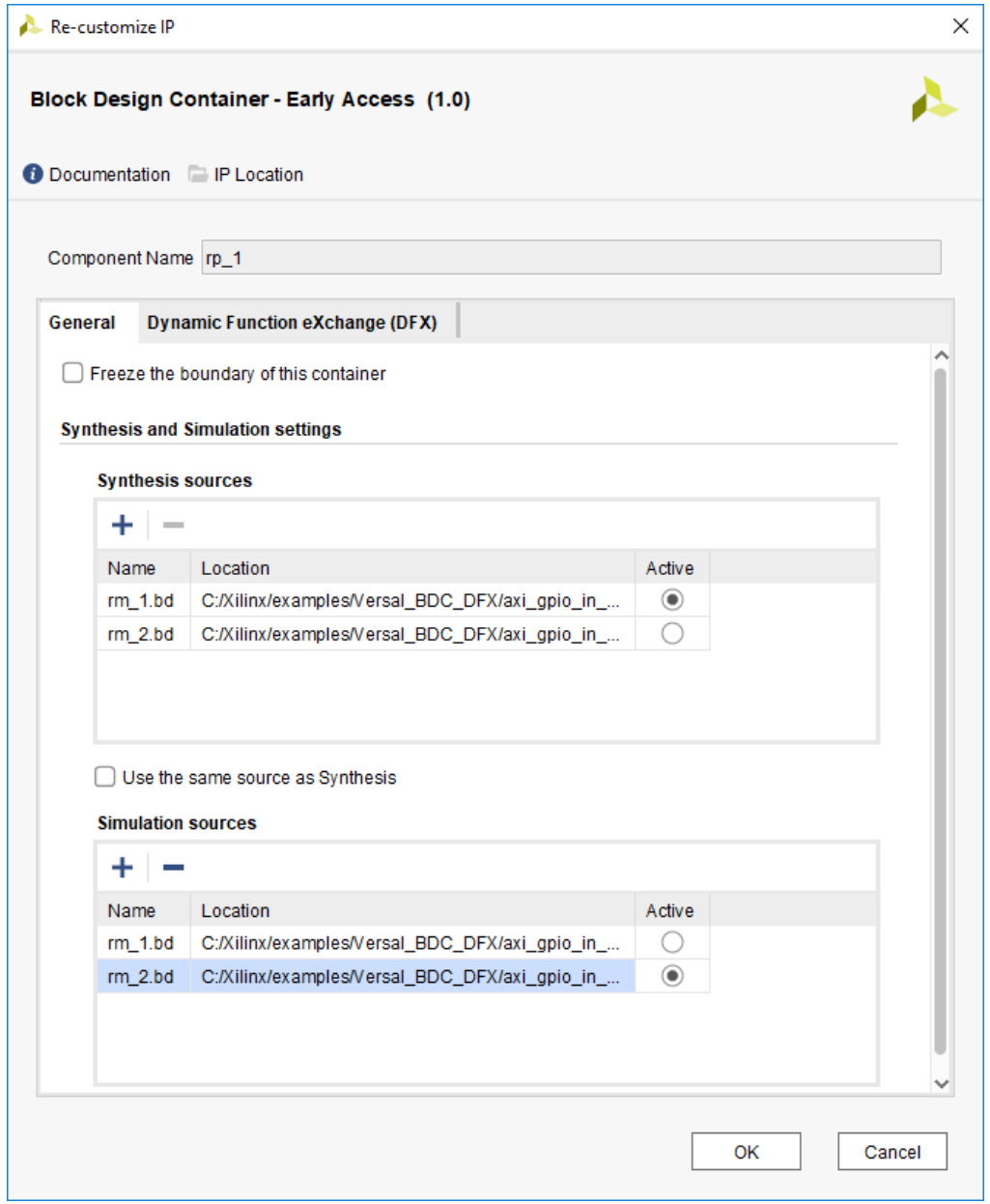


- **Primary Customer** shares Abstract Shell checkpoint and static design bitstream
  - Runtime details (e.g. Decoupling) managed in static design and loading firmware
  - All must be updated if anything in shell is updated, unless part of shell is in a second RP
  - IMPORTANT: Primary customer must have redistribution rights for any IP in static design
- **Secondary Customer** programs first with delivered static bitstream
  - Then with their own partial bitstreams

Forthcoming DFX Solutions

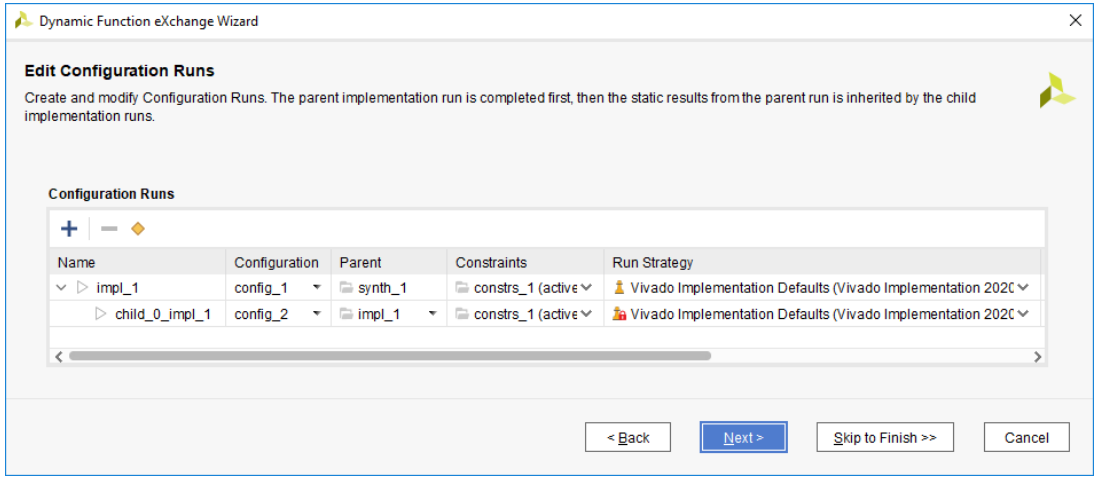
Block Design Containers in IP Integrator

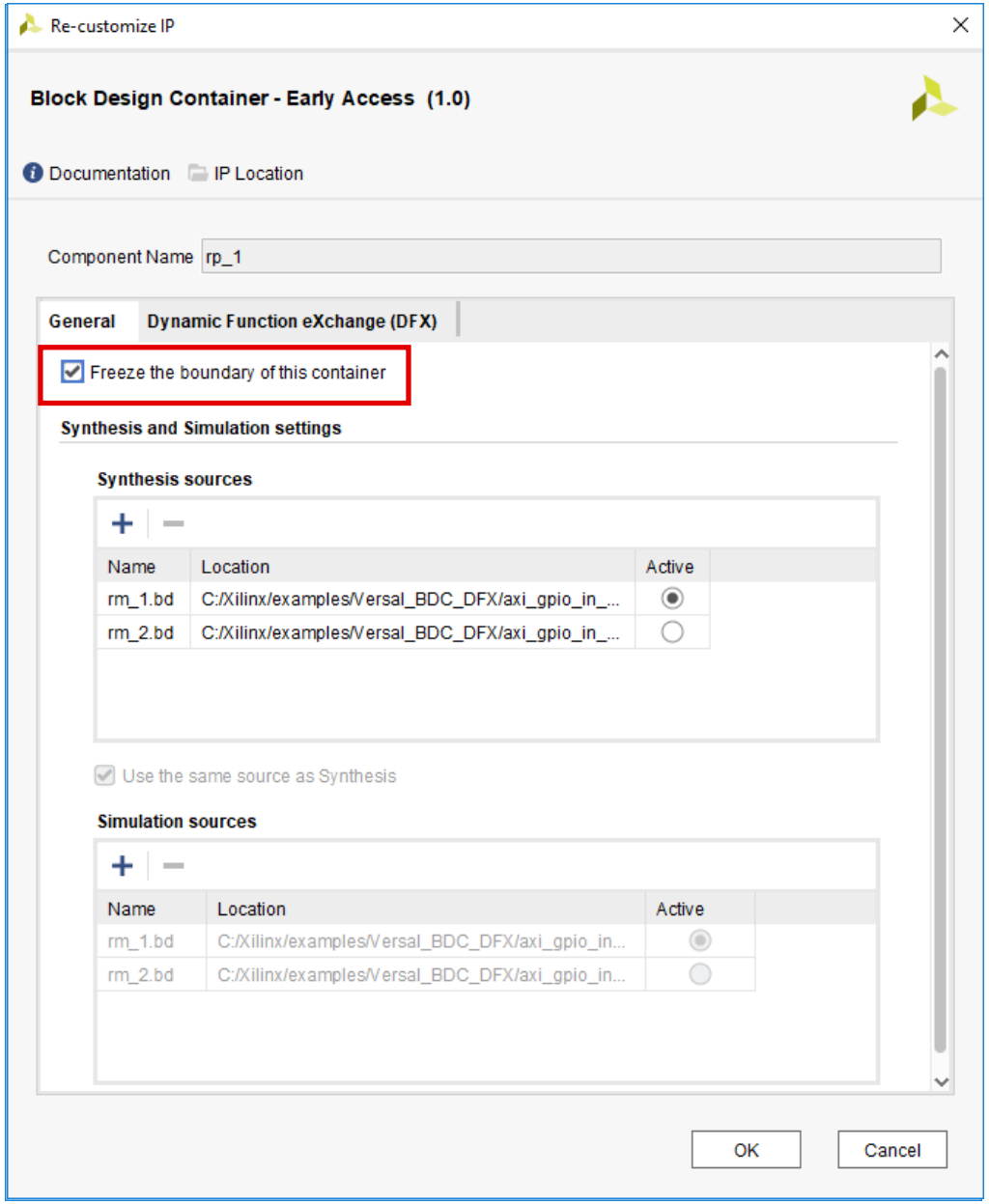
- Instantiate or create one Block Design inside another
  - Enables Modular Design for reusability
  - Allows Team Based Design
  - Enables DFX Flow
- Block Design Container capabilities
  - Build designs bottom-up or top-down
  - Load different versions to active status
  - Differentiate between synthesis and simulation
  - Modify address information from top level



Block Design Containers enable DFX

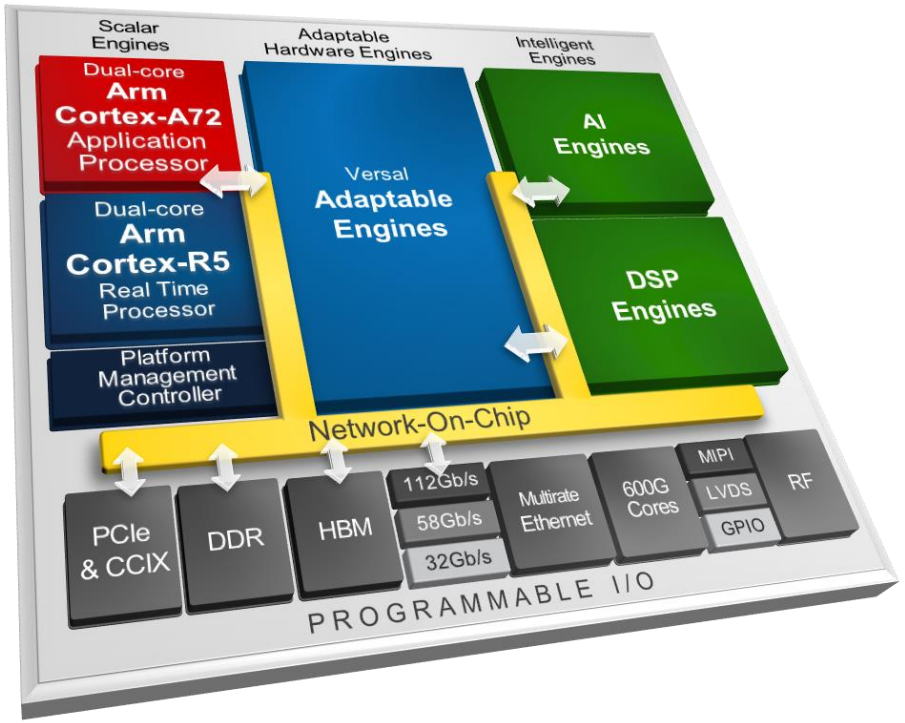
- BDC can be set as Reconfigurable
  - Establishes logical instance with multiple design variants
- Use DFX Wizard to manage system
  - Define Configurations
  - Create and Manage Runs
  - Set strategies and options





Key Advancements in Versal for DFX

- Configuration Performance Increase
  - Maximum bandwidth 8X faster than Zynq US+ MPSoC (6.4GByte/sec)*
  - Capable of reconfiguring 1M logic cells in under 8ms
- Hardened Memory Controllers + DDRIO moved to periphery
  - Memory controllers remain active while fabric is reconfigured
  - Moving DDRIO out of fabric improves ease of floorplanning
- Hardened PCIe + DMA
  - PCIe enumerated with minimal programming
  - PCIe can remain active during reconfiguration
- Floorplan granularity twice as fine
  - Programming images aligned to half clock region height

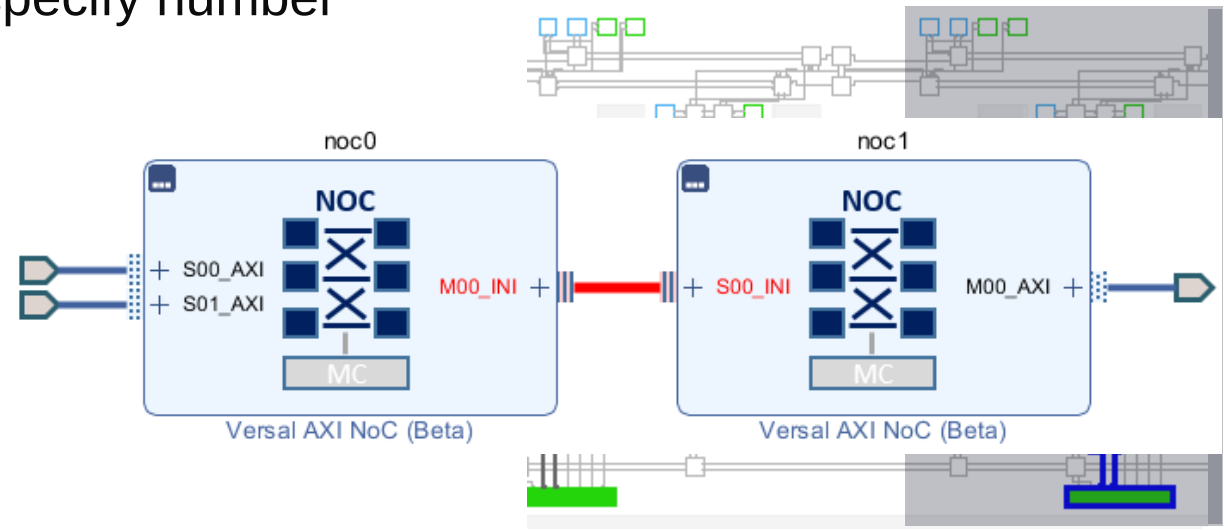


* When using an external interface such as PCIe, DDR memory, etc.

XILINX.

Versal DFX – Network on Chip (NoC)

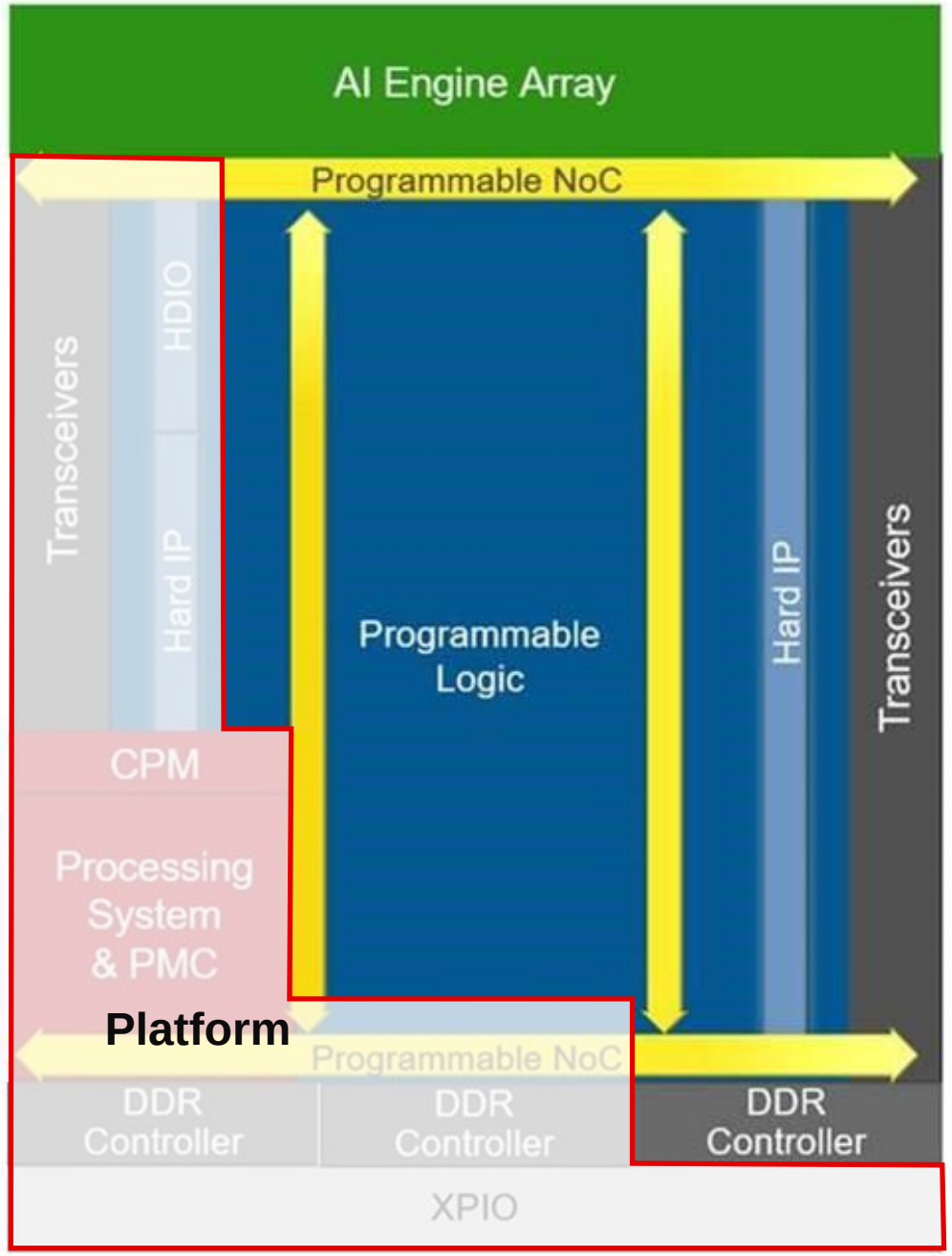
- NoC can be split between static and reconfigurable functions
  - Any static part of the NoC continues to operate during reconfiguration
  - One or more endpoints and associated paths can be reconfigured
- NoC Represented as multiple instances of Hierarchical IP
  - Each instance represents a subset of the master and slave units
  - Each instance is customized individually to specify number and type of ports, connectivity and QoS
- INI connects different sections of NoC
  - Logical/physical boundary established
- Quiescing traffic occurs automatically during reconfiguration



XILINX.

Platform/Workload Model

- Platform remains operational at all times
- Remainder of the device is the reconfigurable area or “Workload”
- The Platform may consist of:
  - PS/PMC
  - PCIe (CPM) with XDMA
  - GTs + XPIPE
  - DDR Memory Interface (XPIO/XPHY/XPLL/DDRMC)
  - AXI Connections to fabric and AIE through NoC
- Note: These elements must be configured initially through a primary boot interface such as QSPI

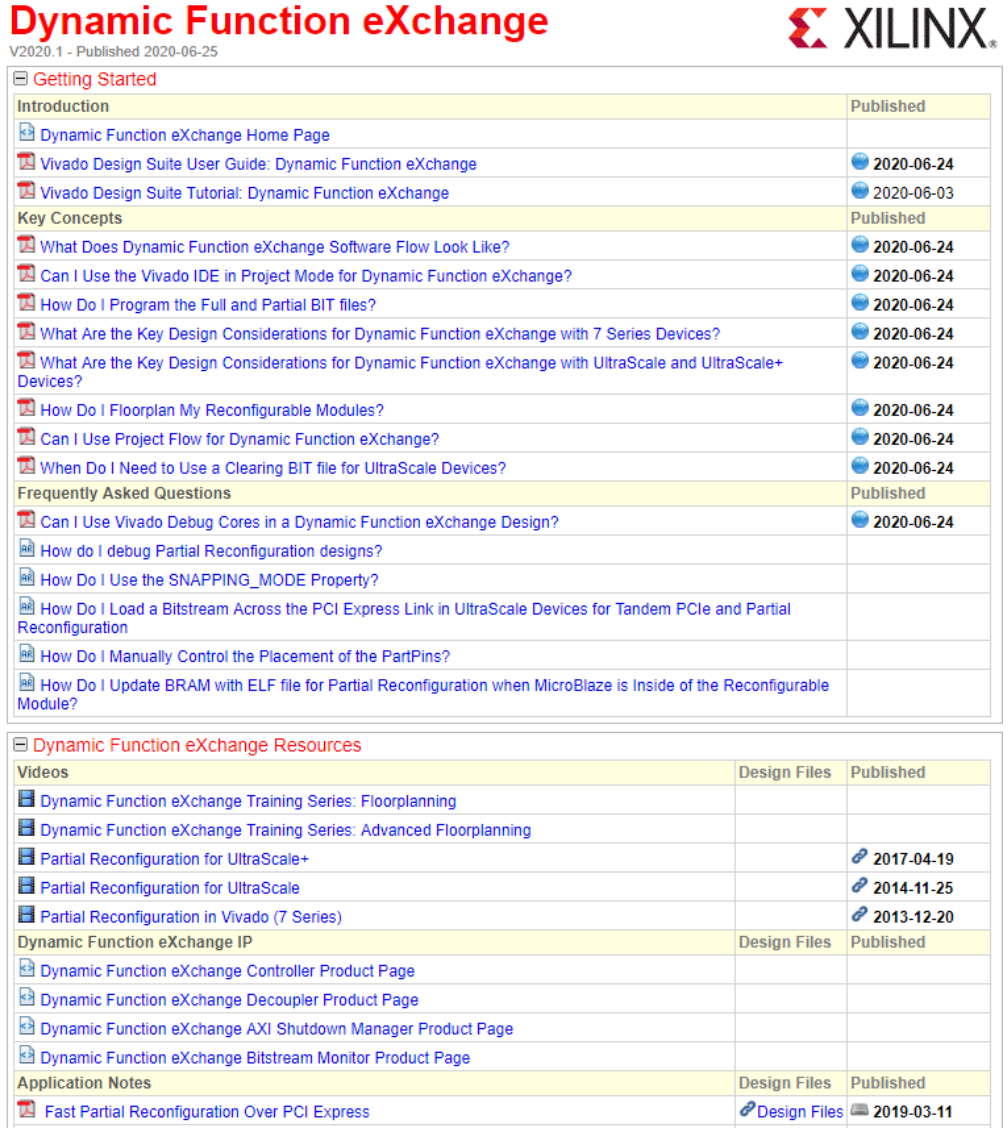


Dynamic Function eXchange Resources

- From the Xilinx.com home page:
  - Products ➔
  - Hardware Development ➔
  - Vivado Design Suite,
  - then click Dynamic Function eXchange

- Resources available:
  - Documentation
  - Tutorials
  - Application Notes
  - IP Product Guides
  - Videos

DocNav Design Hub



Dynamic Function eXchange

V2020.1 - Published 2020-06-25

XILINX

Getting Started

Introduction	Published
Dynamic Function eXchange Home Page	
Vivado Design Suite User Guide: Dynamic Function eXchange	2020-06-24
Vivado Design Suite Tutorial: Dynamic Function eXchange	2020-06-03
Key Concepts	**Published**
What Does Dynamic Function eXchange Software Flow Look Like?	2020-06-24
Can I Use the Vivado IDE in Project Mode for Dynamic Function eXchange?	2020-06-24
How Do I Program the Full and Partial BIT files?	2020-06-24
What Are the Key Design Considerations for Dynamic Function eXchange with 7 Series Devices?	2020-06-24
What Are the Key Design Considerations for Dynamic Function eXchange with UltraScale and UltraScale+ Devices?	2020-06-24
How Do I Floorplan My Reconfigurable Modules?	2020-06-24
Can I Use Project Flow for Dynamic Function eXchange?	2020-06-24
When Do I Need to Use a Clearing BIT file for UltraScale Devices?	2020-06-24
Frequently Asked Questions	**Published**
Can I Use Vivado Debug Cores in a Dynamic Function eXchange Design?	2020-06-24
How do I debug Partial Reconfiguration designs?	
How Do I Use the SNAPPING_MODE Property?	
How Do I Load a Bitstream Across the PCI Express Link in UltraScale Devices for Tandem PCIe and Partial Reconfiguration	
How Do I Manually Control the Placement of the PartPins?	
How Do I Update BRAM with ELF file for Partial Reconfiguration when MicroBlaze is Inside of the Reconfigurable Module?	

Dynamic Function eXchange Resources

Videos	Design Files	Published
Dynamic Function eXchange Training Series: Floorplanning		
Dynamic Function eXchange Training Series: Advanced Floorplanning		
Partial Reconfiguration for UltraScale+		2017-04-19
Partial Reconfiguration for UltraScale		2014-11-25
Partial Reconfiguration in Vivado (7 Series)		2013-12-20
Dynamic Function eXchange IP	**Design Files**	**Published**
Dynamic Function eXchange Controller Product Page		
Dynamic Function eXchange Decoupler Product Page		
Dynamic Function eXchange AXI Shutdown Manager Product Page		
Dynamic Function eXchange Bitstream Monitor Product Page		
Application Notes	**Design Files**	**Published**
Fast Partial Reconfiguration Over PCI Express	Design Files	2019-03-11

Thank You