Dynamic Function eXchange in Vivado 2019.2

March 2020

XILINX

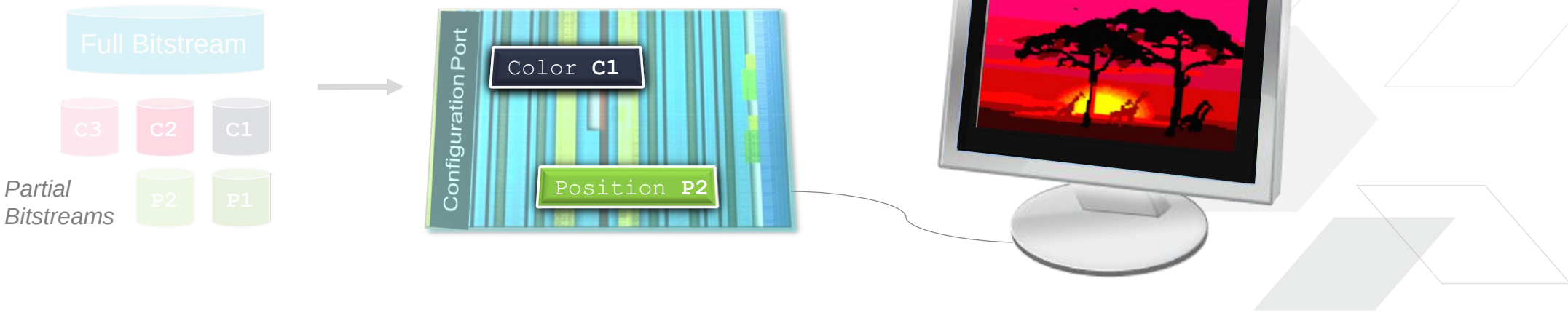
Agenda

- Benefits, Real World Customer Results
- The Dynamic Function eXchange Flow
- Maximizing Design Success
- Abstract Shell for DFX (Early Access)

Benefits, Real World Customer Results

Dynamic Function eXchange Technology

Dynamic Function eXchange dynamically modifies logic blocks while the remaining logic operates without interruption



Dynamic Function eXchange

Technology Timeline

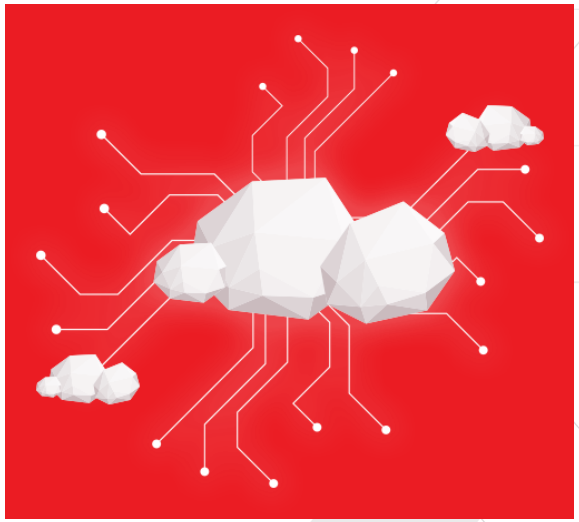
- **Most* 7 Series, Zynq and UltraScale supported in Vivado**

- **UltraScale+ support in production now**
  - For nearly all devices; most recent in EA

* Spartan-7 and smallest Artix-7 devices will not be supported

Dynamic Function eXchange is the Solution

- **Partial Reconfiguration (PR) has been rebranded as Dynamic Function eXchange (DFX)**
- **What is DFX? The ability to deliver new capabilities to silicon on demand**
  - DFX is the feature, what you can achieve with Xilinx
  - DFX has many pieces:
    - Silicon capabilities
    - Vivado design flow
    - IP and IP Integrator flow
    - FPGA Manager and xilfpga library
- **In Vivado, there are minimal changes to low-level terminology**
  - Tcl commands, scripts, DRCs remain unchanged
  - Forward migration is critical for current users
- **Existing utility IP continue to work**
  - Supported across all FPGA and SoC architectures
  - IP names may change in a future release, with forward migration



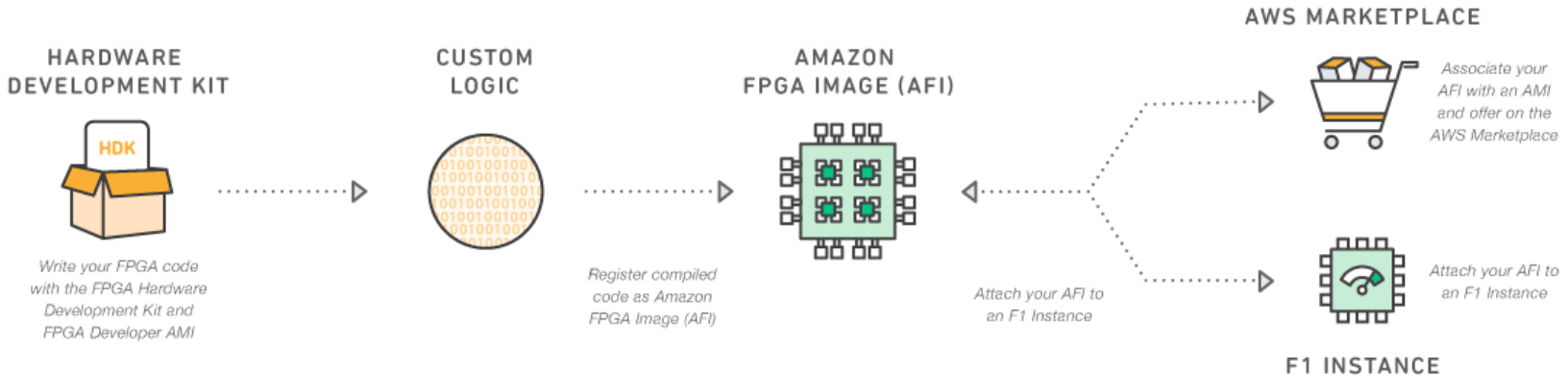
Broader Dynamic Function eXchange Access

- **DFX licenses are included with all System Edition and Design Edition seats**
  - As of the spring of 2017, PR has been available at no extra charge for all paid editions
- **In Vivado 2019.1, Partial Reconfiguration license checking has been removed**
  - That's right, no one needs a PR license any more, even WebPack users

Customer Example

Cloud Computing

- **Amazon Web Services EC2 F1 Instances (UltraScale+ SSI)**
  - Powered by the Xilinx Reconfigurable Acceleration Stack
  - Deploy acceleration kernels in the cloud across many F1 instances

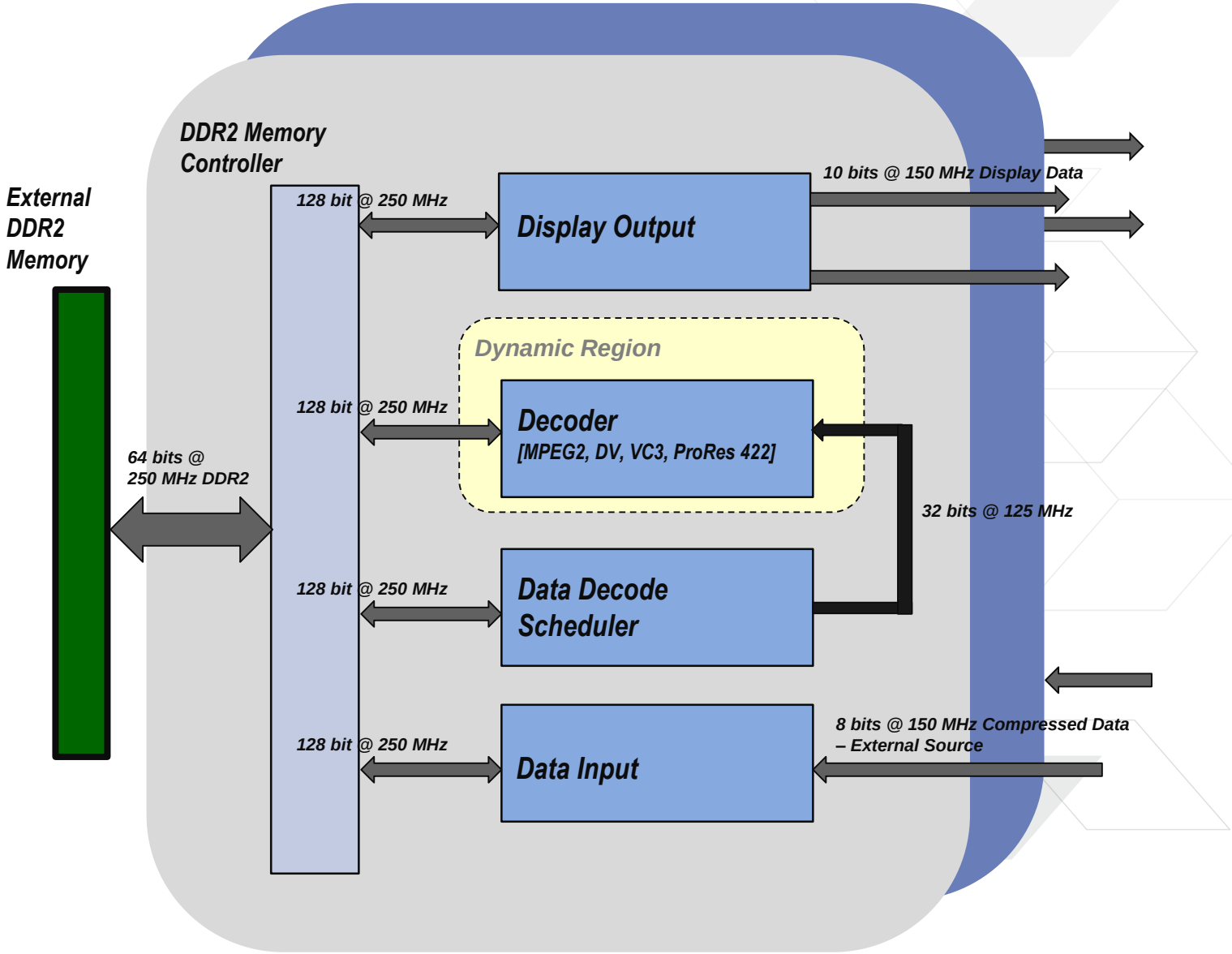


- **F1 partners have solutions across a wide range of applications**
  - Edico Genome, Maxeler, National Instruments, NGCodec, Ryft, TeraDeep and more
  - Learn more here: https://aws.amazon.com/ec2/instance-types/f1

Customer Example

Flexible Video Processing

- **Dynamically swap video decoders**
  - One channel remains up while the other changes
- **Customer released a "flat" version first**
  - Two decoders per channel
  - No partial reconfiguration
- **Expanded functionality with existing hardware**
  - Deployed new bitstreams for more decoders without changing hardware

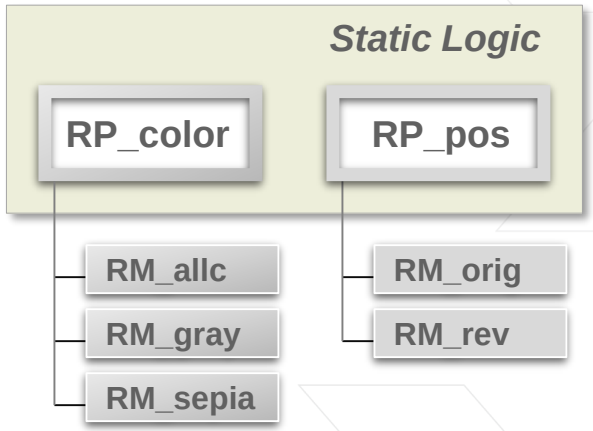


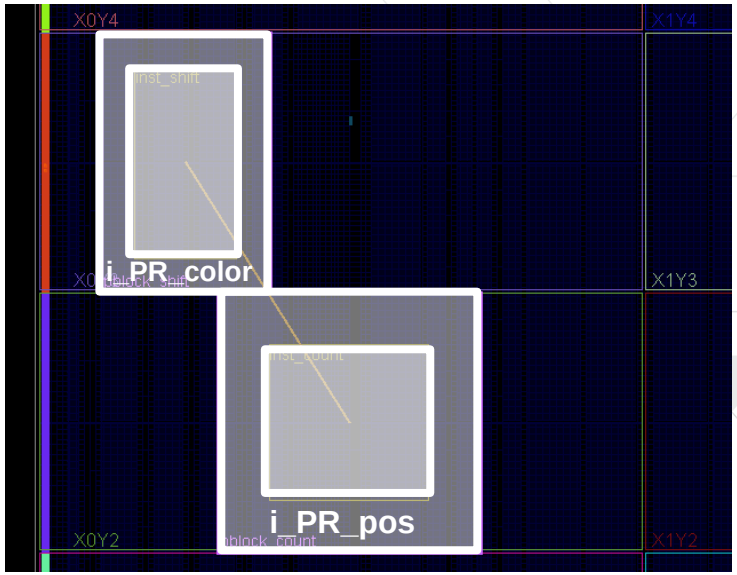
The Dynamic Function eXchange Flow

Intuitive Design Flow

Project Creation and Floorplanning

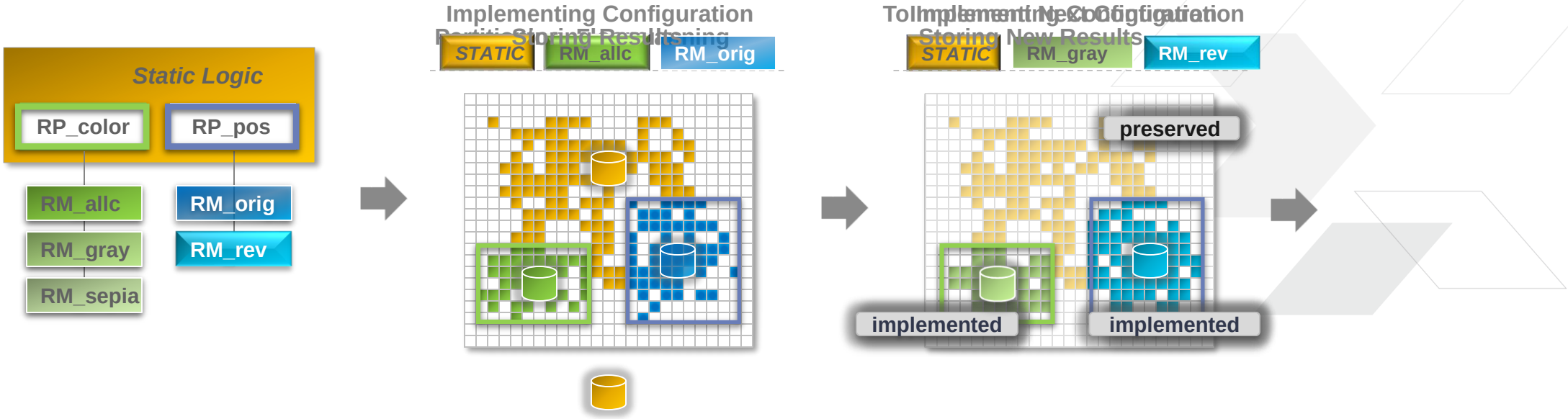
- **Structure your design**
  - Static Logic (unchanging design)
  - Reconfigurable Partitions (RP)
    - Instances to be reconfigured
  - Reconfigurable Modules (RM)
    - Functional variations for each RP
- **Synthesize bottom-up**
  - `synth_design -mode out_of_context`
- **Define resources to be reconfigured**
  - Pblocks map design modules to physical regions
    - Define XY ranges and resource types
- **Mark pblocks as reconfigurable**
  - `HD.RECONFIGURABLE` initiates flow





Leverage Module Checkpoints for DFX

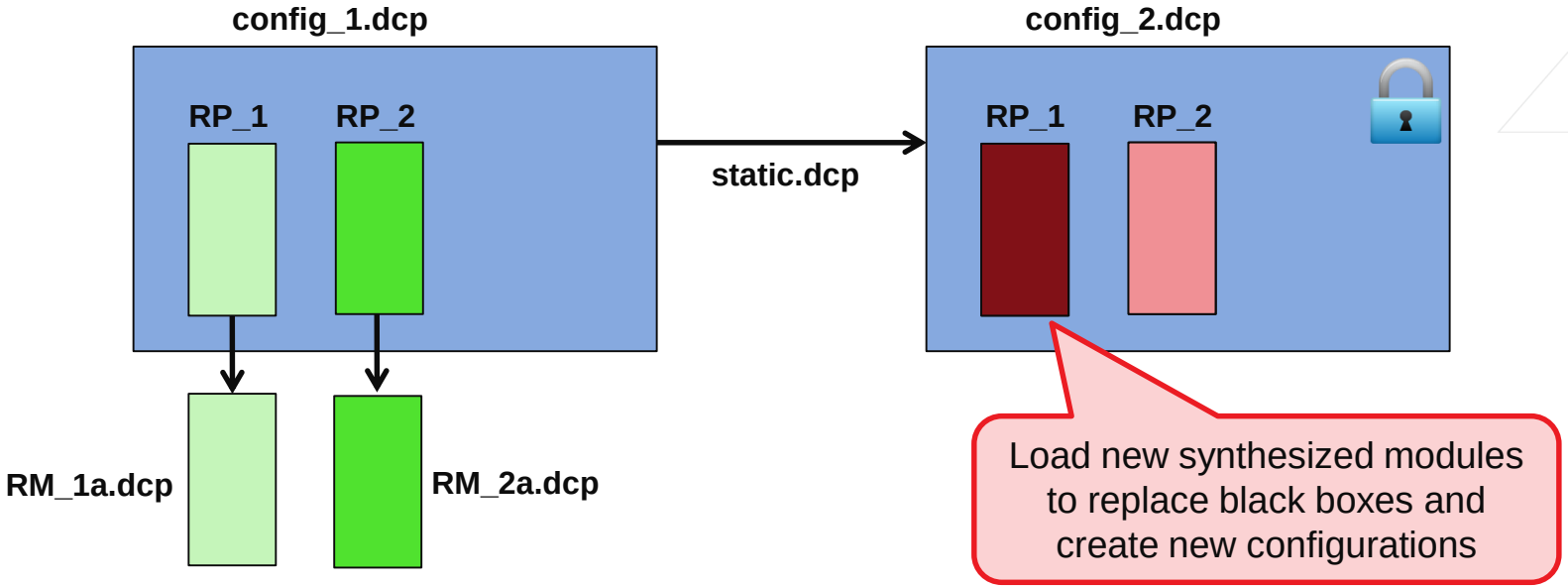
- **Partition methodology enables Dynamic Function eXchange**
  - Allows clear separation of static logic and Reconfigurable Modules
  - Floorplan to identify silicon resources to be reconfigured
- **Design preservation accelerates design closure**
  - Lock static design database while implementing new modules



Vivado Software Design Management

Checkpoints for Each Partition

- **Vivado stores design data in checkpoints**
  - Save full design as a configuration checkpoint for bitstream creation
  - 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
  - Reconfigurable modules can also be stored and reused as their own checkpoints



Intuitive Design Flow

Implementation

- **Place and Route all design configurations**
  - Apply full design constraints in-context
  - Use normal timing closure, simulation and verification techniques
  - Incremental Compile supported for DFX designs

- **Final Verification**
  - Validates consistency of place and routed results across the entire system

- **Generate Bitstreams**
  - `write_bitstream` automatically creates all full and partial bitstreams
  - Or, selectively write only full or only partial bitstreams

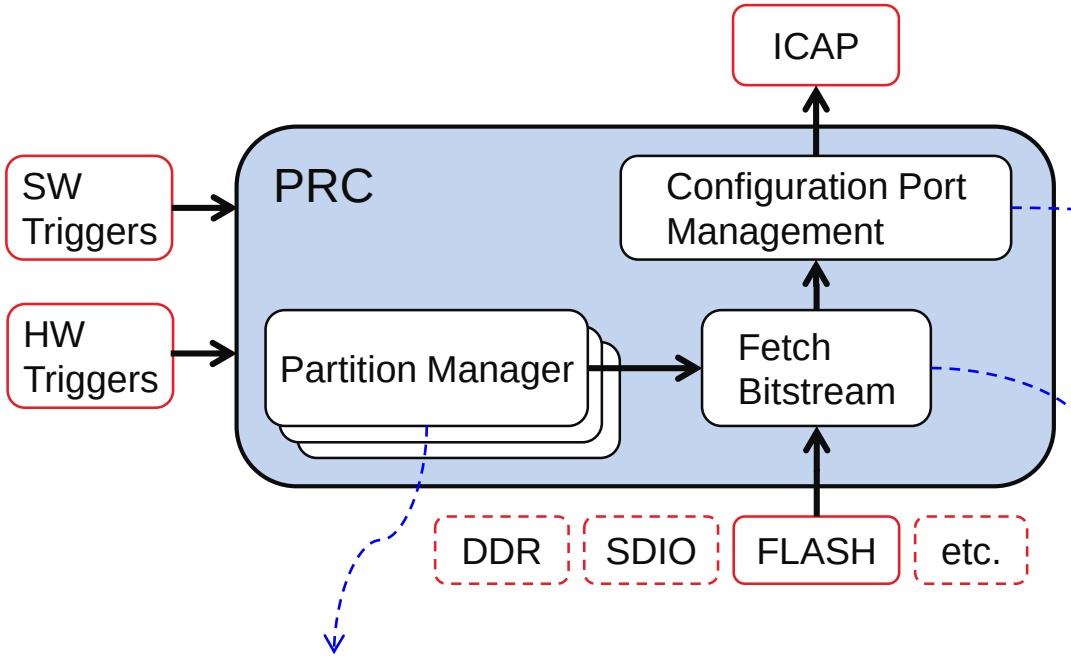
Leverages standard Vivado design flow

Partial Reconfiguration Controller (PRC) IP

Manage dynamic reconfiguration on any supported architecture

New in 2017.3:
Bitstream compression!
30-70% size reduction

- **Improves Partial Reconfiguration Ease of Use**
  - Reconfiguration management is currently a user problem
  - The PRC becomes the heart of the partial reconfiguration process



Arbitrates for access (ICAP/MCAP/PCAP/JTAG)

Drives protocol, detect & manage errors

Creates path between bitstream source & bitstream destination

Manages bitstream transfer

Quickly detects SW & HW requests to reconfigure

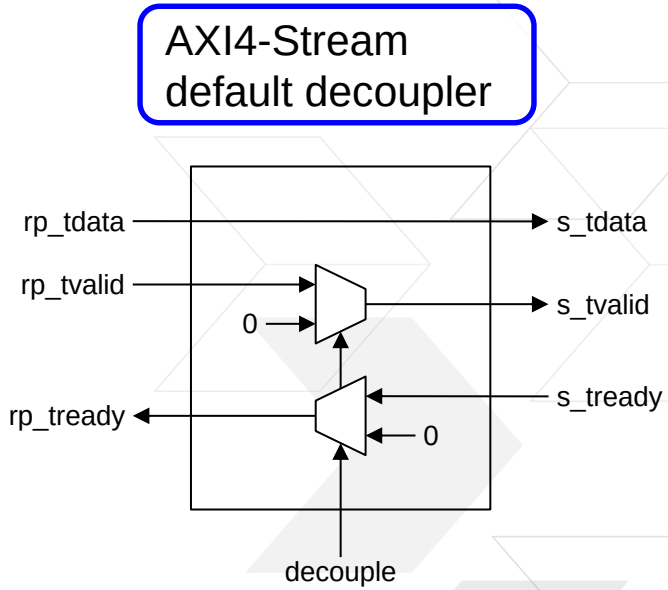
Sequences through configuration steps

Documented in PG193

Partial Reconfiguration Decoupler IP

Easy isolation of reconfiguring regions

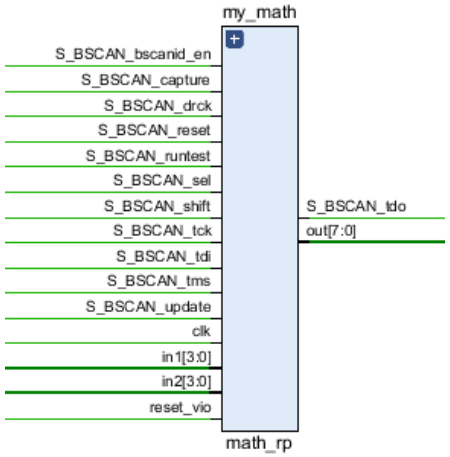
- **Easily isolate reconfiguring regions of your design**
  - Ensures that indeterminate state of RP will not disrupt the static design
  - Decoupling logic built from simple muxes
- **Interface Aware**
  - One click addition of any interface type registered in Vivado
  - One click interface connection in IP Integrator
  - Full control of interface decoupling
  - Optimized for AXI-based interfaces
  - Create custom interfaces to meet specific design needs
- **Advanced schemes supported**
  - Clock domain crossing, phased decoupling, etc.
  - Compatible with PR Controller IP



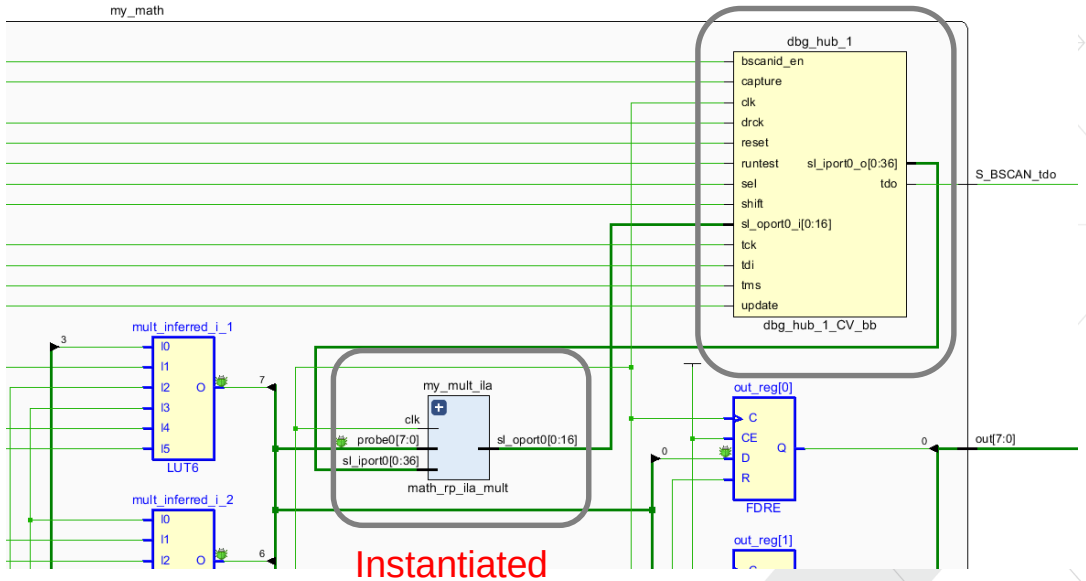
Documented in PG227

Debugging Dynamic Function eXchange Designs

- **Vivado Debug Cores are supported in Reconfigurable Modules**
  - Instantiate ILA, VIO cores, or use debug cores embedded within IP
  - Debug Hub automatically inferred by RP port names
    - Or inserted by attributes if necessary



Inferred



Instantiated

- **Debug flow within Hardware Manager improved**
  - Probe (LTX) files automatically included during partial bitstream loading

Vivado DFX Software Roadmap

Plans subject to change

- **Project support will continue to evolve**
  - Infrastructure will be used to support general partition-based flows
  - Goal is to have users seamlessly build DFX designs in IPI
- **Advanced features will improve ease of use**
  - Long team goal is to make DFX a ubiquitous part of the Xilinx solution
  - Ongoing research to improve overall QoR and routability of DFX designs
  - Analysis tool will examine all RMs and suggest optimal RP; later: automated floorplanning
- **Advanced features are currently in development**
  - Nested DFX allows one (or more) reconfigurable region(s) inside a reconfigurable region (2020.1)
  - Abstract Shell strips away much of the static design for security and runtime reduction goals (EA)

Maximizing Design Success

Top Things to consider when designing for DFX

1. **Think “FPGA inside an FPGA”**
   - Each Reconfigurable Partition (RP) is like its own FPGA within the fabric
2. **Simplify IO of each RP as much as possible**
   - Reducing number of ports, especially clocks, will improve routability
3. **Build the design requirements up incrementally based on target shell floorplan**
   - Add requirements one at a time to solve challenges incrementally
4. **Consider reconfiguration events early in the design process**
   - Partial bitstream delivery and decoupling are designer’s responsibility
5. **Consider floorplanning of RP to leverage expanded routing feature**
   - This allows better routability around the edges of RP pblocks

General DFX Strategy

- **Build up the design requirements one at a time**
  - Check the implementation results at each stage
  1. Implement the design using bottom-up (OOC) synthesis
  2. Add standard pblocks for the reconfigurable partitions
  3. Add EXCLUDE_PLACEMENT to pblocks
  4. Add CONTAIN_ROUTING to pblocks
  5. Mark pblocks as HD.RECONFIGURABLE (remove other properties)

- **If goals are not met at any step, iterate before proceeding**
  - Examine design structure, timing constraints (false paths), floorplan
  - Ensure enough slack in timing and area to accommodate both DFX rules and future Reconfigurable Modules (RMs)

Dedicated Silicon Features Provide Design Safety

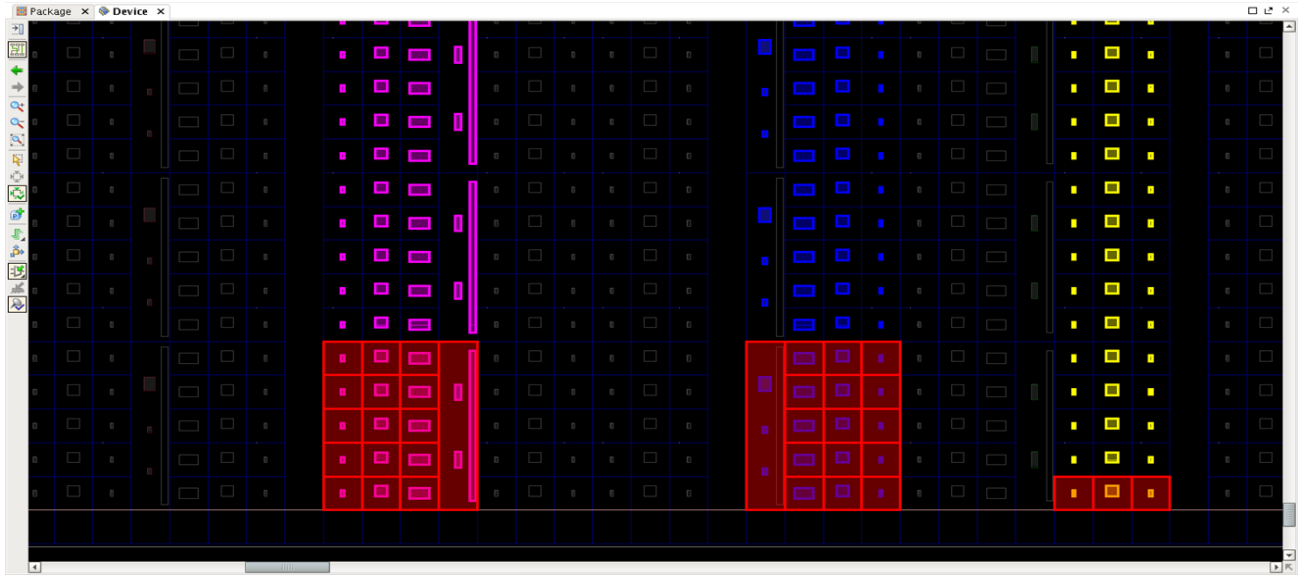
- **Dedicated GSR for reconfigured regions**
  - Set RESET_AFTER_RECONFIG property (always on for UltraScale and UltraScale+)
  - Reconfigured region is masked and receives GSR for all synchronous elements in partial bitstream
    - Eliminates user requirement for initializing logic
- **CRC checking 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
  - 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
- **Encryption and Compression also natively supported**
  - Authentication only supported for Zynq devices

UltraScale Silicon Advancements

- **Everything except configuration frame is reconfigurable**
  - Nearly all resource types can be dynamically reconfigured
    - Including clocks, IO and transceivers (limited changes and coarse granularity)
  - Two-stage configuration process will be necessary
    - Clear existing module before loading next one
  - Configuration frame includes ICAP, BSCAN, STARTUP, etc.
- **Fine-grained GSR controls**
  - Reconfigure and initialize very small building blocks
  - Great flexibility in design floorplanning
- **Dedicated access to configuration status**
  - PRDONE and PRERROR on internal configuration port (ICAP)
- **All production silicon supported as of Vivado 2016.1**
  - VU440 available without parameter as of 2018.3

Reconfigurable Granularity in UltraScale

- **Reconfiguration tiles based on shared interconnect**
  - Base regions combined two columns of resources with single INT

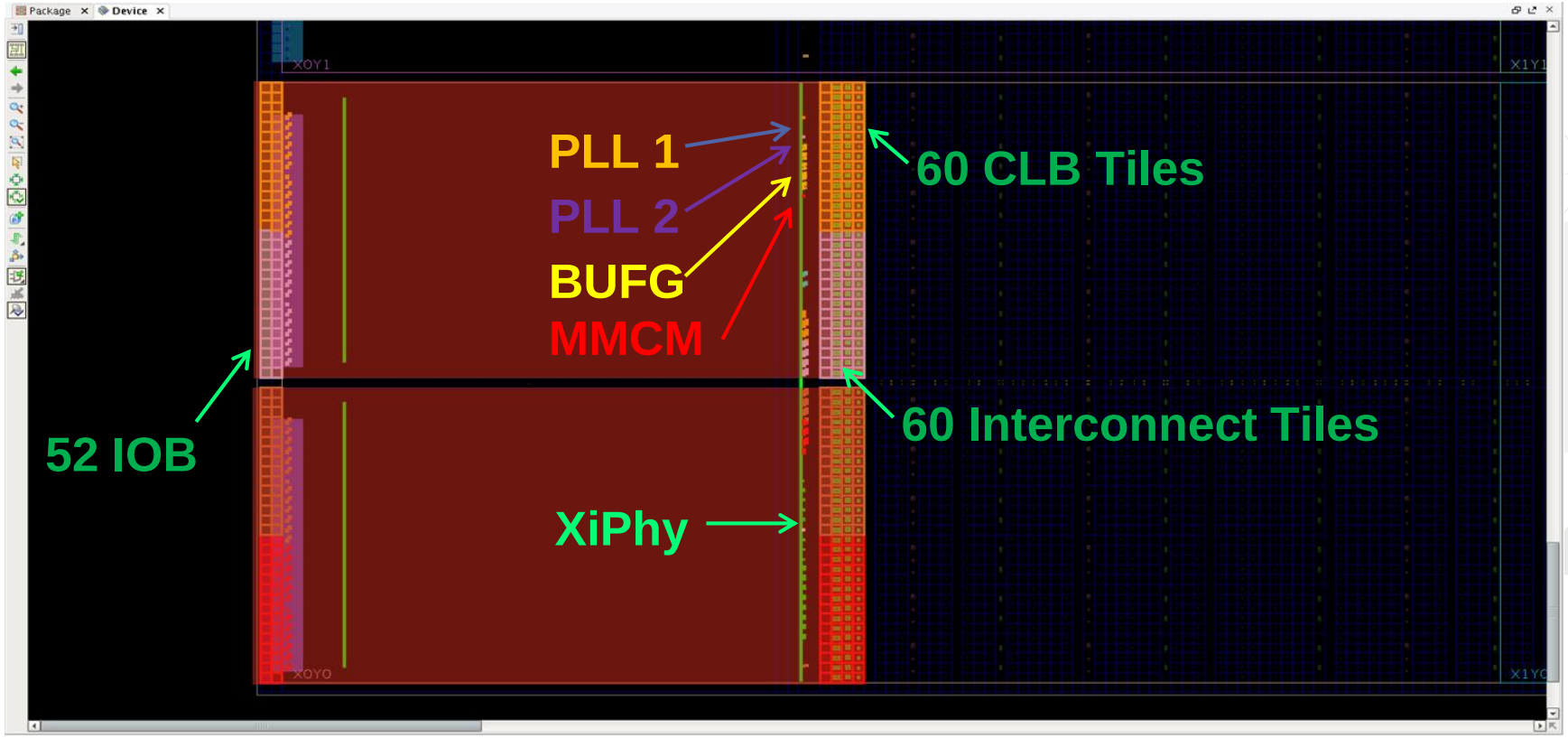


- **New elements have coarser rules, shared with column of CLBs**
  - IO and clocks: 1 bank
  - Transceivers: 1 quad
  - PCIe, CMAC, Interlaken: 1 clock region

Reconfigurable Granularity in UltraScale

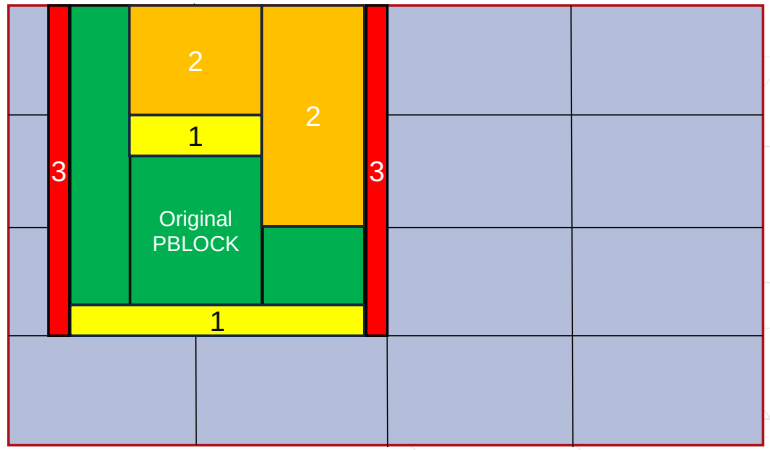
IO Frame Example

- **Entire IO bank, plus clocking & XiPhy, plus shared CLB column**
  - All these elements must be reconfigured together
  - IO held by GTS during reconfiguration



Dynamic Function eXchange Efficiency Improvements

- **Expanded Routing for UltraScale and UltraScale+**
  - Allows Reconfigurable Module to use routing resources outside Pblock
  - Improves routability and performance 4-5% on average
    - Most effective for non-rectangular Pblocks, as they have more corners
  - Pblocks should be drawn to allow for expansion
    - Keep Pblocks away from frame boundaries
    - Do not abut Pblocks of different Reconfigurable Partitions



1. Vertical frame alignment (**YELLOW**)
2. FILL: cover the entire RM PBLOCK tiles (**ORANGE**)
3. Left/Right 2-Progammable Unit expansion (**RED**)

- **Feature added in 2016.3 (UltraScale), expanded in 2017.1 (UltraScale+)**
  - Will not be retrofitted to 7 series – restricted reconfigurable elements reduce effectiveness
  - Not customizable – cannot modify how expansion is done, it's either on (default) or off

UltraScale+ Advancements

Continuing the evolution of DFX

- **Improvements compared to UltraScale:**
  - No clearing bitstream requirement
  - Reconfiguration via QSPI and sync BPI interfaces
- **DFX is a key part of initial silicon design and testing**
  - Validation of component reconfiguration in earliest silicon
  - With each process node, DFX features tested earlier and more broadly
- **Software support now has production status for most US+ devices**
  - Bitstream generation is gated for engineering silicon
  - Zynq RFSoC and VU+ HBM devices now in production
  - VU+ PAM-4 is in beta with the VU29P

Architectural Comparisons

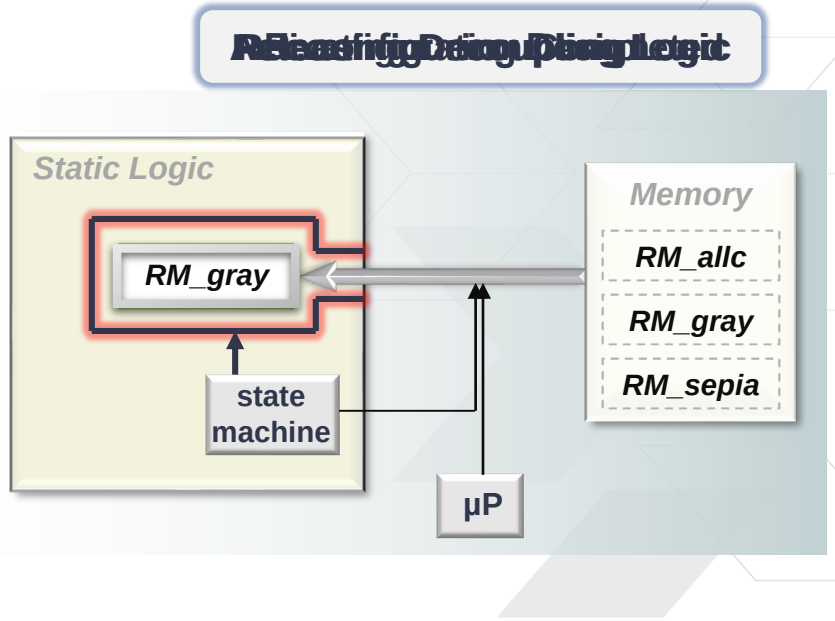
	UltraScale	UltraScale+
Device Support	All devices	All devices
Reconfigurable Elements	CLB, BRAM, DSP, IO, MGTs, clocks, PCIe, CMAC, ILKN, etc.	CLB, BRAM, DSP, IO, MGTs, clocks, PCIe, CMAC, ILKN, etc.
Reset After Reconfiguration (GSR)	Always On	Always On
Snapping Mode	Always On	Always On
Clearing Bitstreams	Required	No
Expanded Routing	Yes	Yes
DFX pins on ICAP	Yes	Yes
Multi-SLR regions for SSI	Yes	Yes
QSPI and sync BPI modes	No	Yes
Tandem Configuration	Yes, all devices	Yes, all devices eventually
Tandem w/ Field Updates	Yes, clearing required	Yes, with Reconfigurable Stage Twos*

– Any feature not mentioned is consistent across all families

* See AR 71877 for a known issue

Dynamic Function eXchange Control Sequence

- **Initiation of reconfiguration implemented by the designer**
  - Off-chip microprocessor or other controller, or …
  - On-Chip state machine, processor or other logic …
- **Activate decoupling logic and reconfigure**
  1. Shut down activity in dynamic region
  2. Disconnect the reconfigurable region from static
  3. Deliver Partial Bitstream
  4. Region is automatically initialized*
  5. Release decoupling logic when reconfiguration is complete
- **Standard configuration mechanism are used**
  - Partial bitstream is the same format as full bitstream
  - Deliver via standard configuration ports



Dynamic Function eXchange Collateral

- **Learn about Dynamic Function eXchange**
  - User Guide UG909, Tutorial UG947
    - New UltraScale+ lab: PR from DDR4
  - XAPP1338 shows fast PR over PCIe
  - XAPP1231 shows Zynq solution
  - XAPP1261 shows PR + SEM
  - XAPP1292 shows PR over TFTP
  - XAPP1321 shows fast recalibration of DDR
  - PR Design Hub in DocNav
- **Training and Support**
  - Training Course available via ATPs
  - QuickTake Video for UltraScale+
    - One for UltraScale features as well

Partial Reconfiguration in Vivado

V2018.1 - Published 2018-04-09

XILINX ALL PROGRAMMABLE

Getting Started

Introduction	Published
Partial Reconfiguration Home Page	
Vivado Design Suite User Guide: Partial Reconfiguration	2017-12-20
Vivado Design Suite Tutorial: Partial Reconfiguration	**2018-04-04**
Partial Reconfiguration for UltraScale+	**2017-04-19**
Partial Reconfiguration for UltraScale	**2014-11-25**
Partial Reconfiguration in Vivado (7 Series)	**2013-12-20**
Key Concepts	**Published**
What Does Partial Reconfiguration Software Flow Look Like?	2017-12-20
Can I Use the Vivado IDE in Project Mode for Partial Reconfiguration?	2017-12-20
How Do I Program the Full and Partial BIT files?	2017-12-20
What Are the Key Design Considerations for Partial Reconfiguration with 7 Series Devices?	2017-12-20
What Are the Key Design Considerations for Partial Reconfiguration with UltraScale and UltraScale+ Devices?	2017-12-20
How Do I Floorplan My Reconfigurable Modules?	2017-12-20
Can I Use Project Flow for Partial Reconfiguration?	2017-12-20
When Do I Need to Use a Clearing BIT file for UltraScale Devices?	2017-12-20
Frequently Asked Questions	**Published**
How Do I Obtain a License for Partial Reconfiguration?	
Can I Use Vivado Debug Cores in a Partial Reconfiguration Design?	2017-12-20
How Do I Convert a Design from a Standard Flow to the Partial Reconfiguration Flow?	
How Do I Use the SNAPPING_MODE Property for Partial Reconfiguration?	
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 in Partial Reconfiguration Flow?	
How Do I Update BRAM with ELF file for Partial Reconfiguration when MicroBlaze is Inside of the Reconfigurable Module?	

Partial Reconfiguration Resources

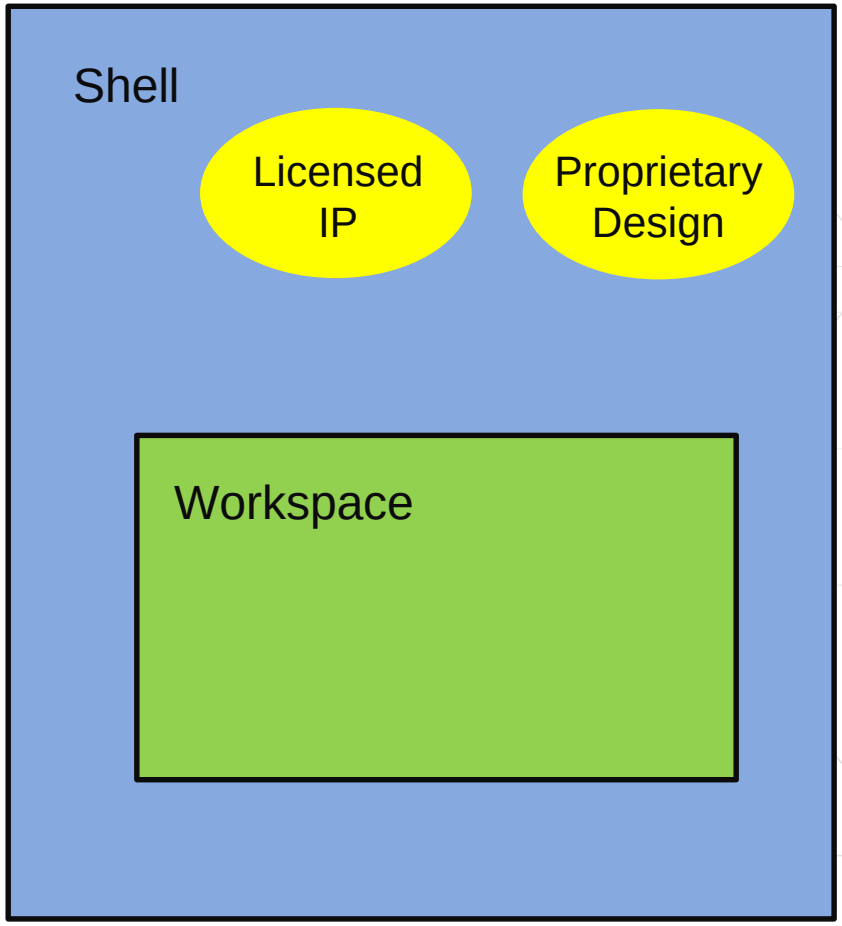
Partial Reconfiguration IP	Design Files	Published
Partial Reconfiguration Controller Product Page		
Partial Reconfiguration Decoupler Product Page		
Partial Reconfiguration AXI Shutdown Manager Product Page		
Partial Reconfiguration Bitstream Monitor Product Page		
Application Notes	**Design Files**	**Published**
Loading Partial Bitstreams using TFTP	Design Files	2016-10-05
Partial Reconfiguration of a Hardware Accelerator with Vivado	Design Files	**2015-03-20**
Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices	Design Files	**2015-06-19**
Local Partial Reconfiguration Using Embedded Processing for 3D ICs	Design Files	**2016-03-02**

Abstract Shell for Dynamic Function eXchange

XILINX

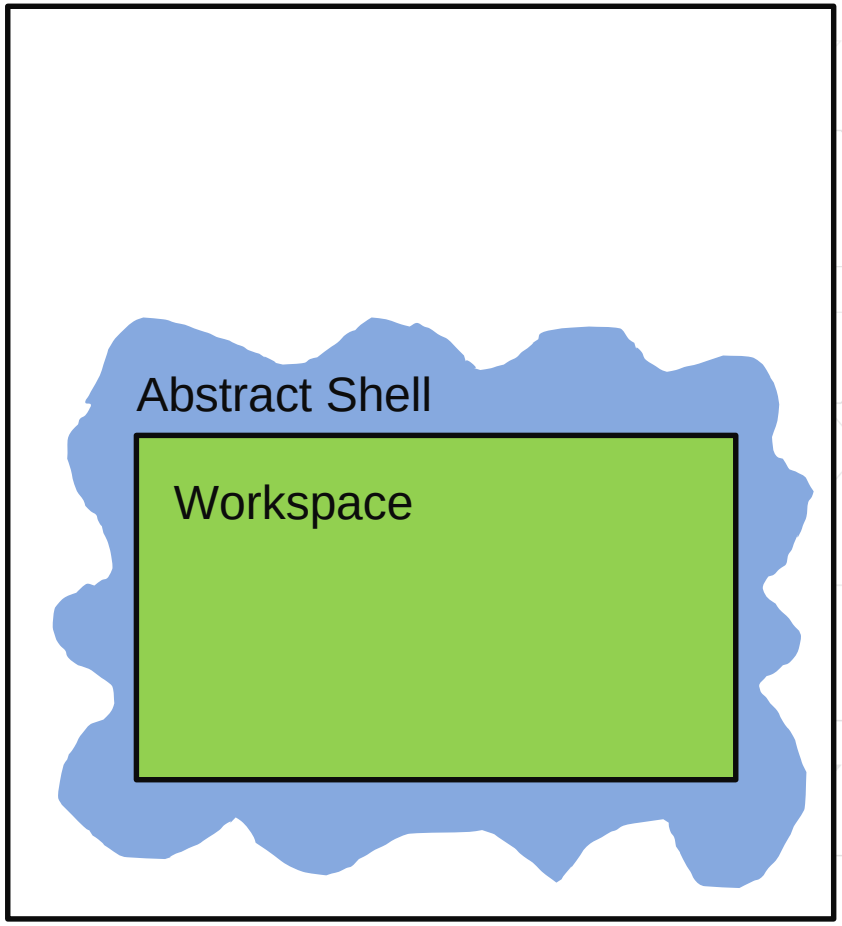
Primary Use Case for Abstract Shell

- **Challenge: How to deliver environment for end user without sharing critical static information?**
- **Main use case is for multi-customer environment**
  - Primary customer builds design Shell (static)
  - Secondary customer receives shell to implement logic in Workspace(s) (a.k.a. reconfigurable module(s))
  - Primary and secondary customers want to share as little information as possible with each other
    - Due to IP licensing challenges or proprietary information
- **Abstract Shell = Static design checkpoint limited to a minimal interface around Reconfigurable Partition**
  - Licensed IP / proprietary design must not be exposed



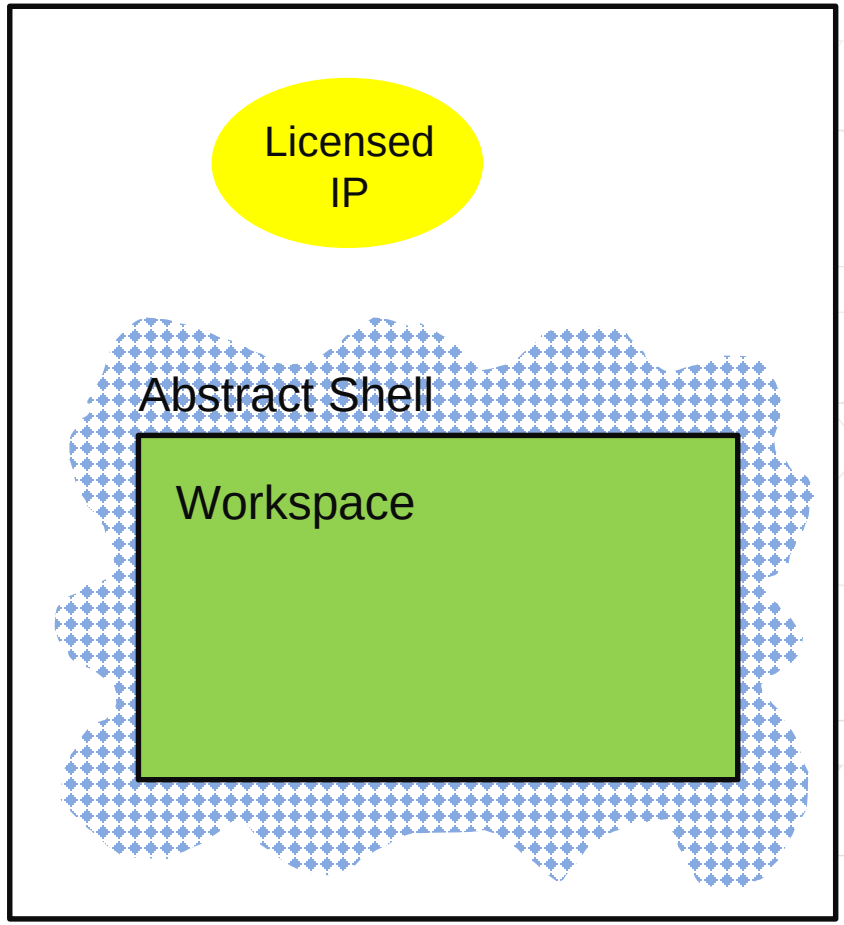
Abstract Shell Design Flow Goals

- **Inclusion of “complete” RP interface with Abstract Shell checkpoint**
  - Everything needed to implement Reconfigurable Modules must be included in DCP
  - Sign-off timing from Abstract Shell environment
- **Bypass IP license check tags**
  - Opening the Abstract Shell DCP cannot trigger license check
- **Generation of partial bitstreams from Abstract Shell environment**
  - Must not require full design reassembly
- **Faster runtime and lower memory usage**

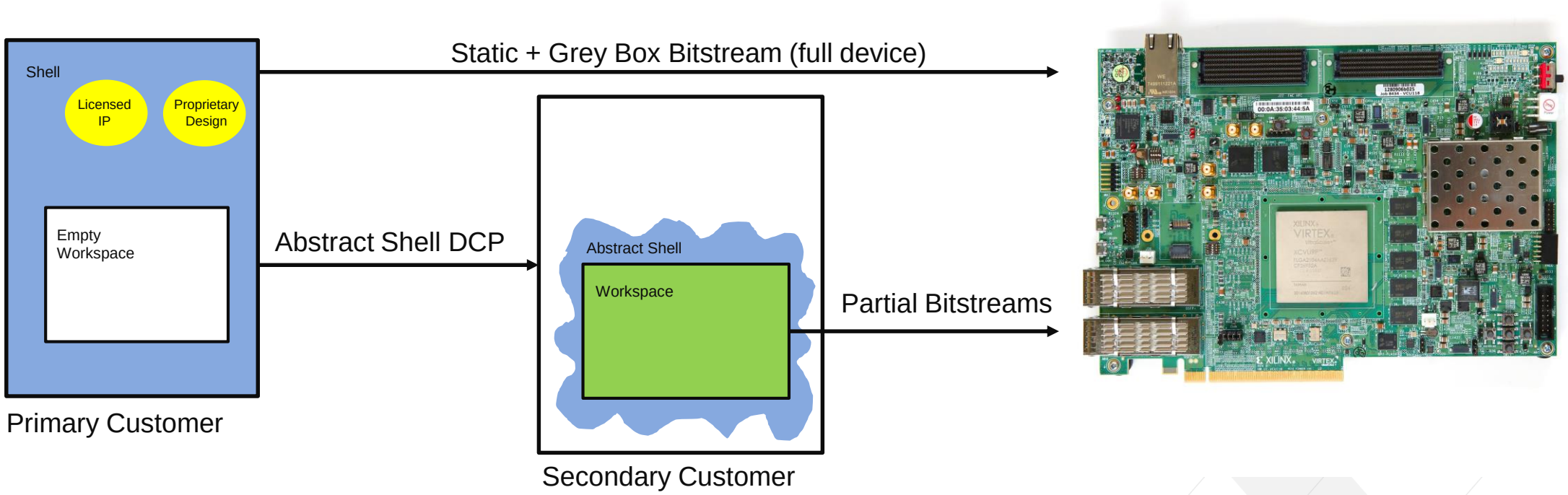


Solution in Vivado through 2020.1 – Early Access

- **All logic in Abstract Shell removed**
  - Except path to first/last synchronous element, needed for timing closure
  - Only routing information maintained
- **No IP logic = no need for IP licensing**
  - Checkpoints will be smaller and faster as well
  - No need to restrict placement of licensed IP
- **Runtime and memory improvements**
  - Goal of 50% runtime reduction – will vary by design
- **Production planned for 2020.2**
  - UltraScale+, single RP to start
  - No project mode (will consider in 2021)



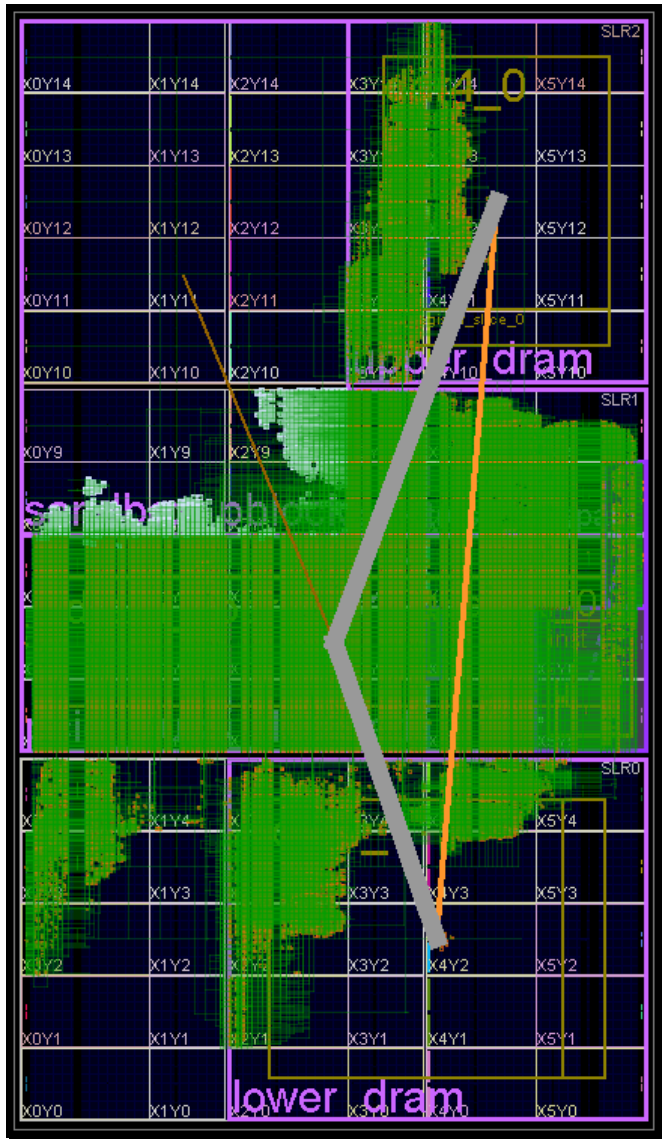
Customer Path

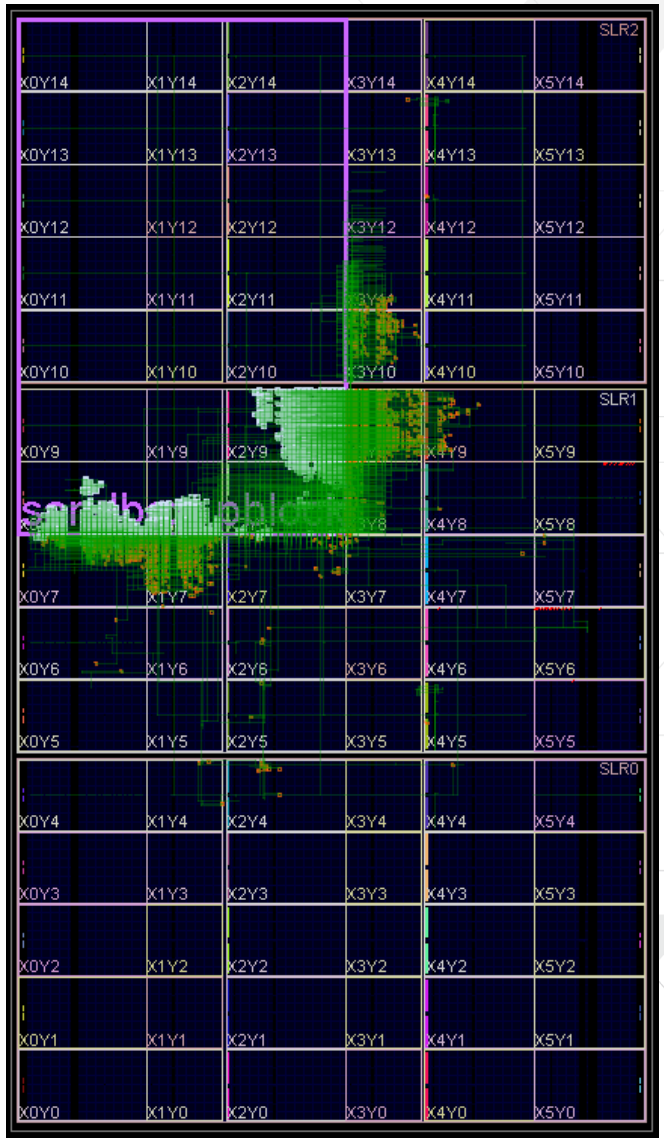


- **Primary customer shares abstract shell checkpoint and static design bitstream**
  - All must be updated if anything in shell is updated, unless part of shell is in a second RP
  - Runtime details (e.g. Decoupling) managed in static design and loading firmware
- **Secondary customer programs first with delivered static bitstream**
  - Then with their own partial bitstreams

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



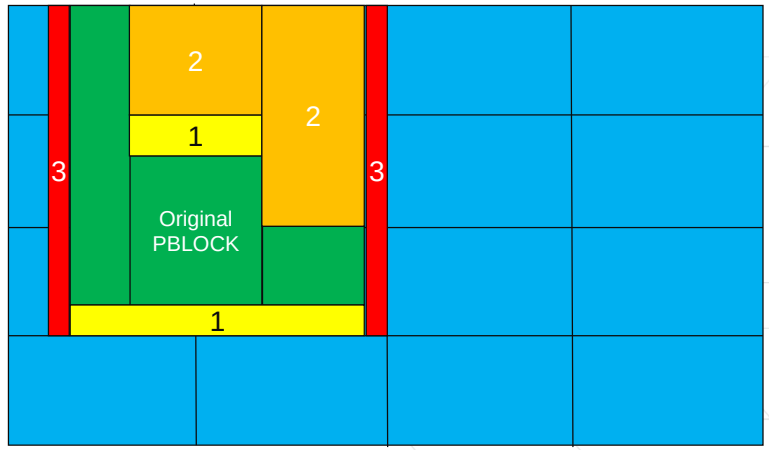


Additional Details

- **Secondary users can confirm timing closure without the full static design**
  - PR_Verify can also be done, comparing new RM checkpoint with base abstract shell
  - Complete designs can always be reassembled by merging RM and full static checkpoints
- **Primary customer must provide guidance for secondary customers**
  - Details include:
    - Workspace instantiation template
    - Run scripts / project environment
    - Interface timing requirements
    - Runtime details – how to decouple region, load partials, reload drivers, etc.
  - Bitstream version checking will be a key detail for shell, to keep version compatibility aligned
    - Manage in bitstream loader functionality
- **Documentation provided with each EA release**
  - Initial design configuration is identical to standard DFX design flow
  - Then, create abstract shell checkpoints when saving locked static design
    - **`write_abstract_shell -cell <foo> <abstract_static_only>.dcp`**

Expanded Routing for Abstract Shell

- **Expanded Routing for UltraScale and UltraScale+**
  - Allows Reconfigurable Module to use routing resources outside Pblock
  - Improves routability and performance 4-5% on average
  - Recommendation: draw Pblocks to allow for expansion
    - Keep Pblocks away from frame boundaries
    - Do not abut Pblocks of different Reconfigurable Partitions
    - Use hd_visual scripts to see expanded region:
      `source pblock_<instance>_Routing_AllTiles.tcl`



1. Vertical frame alignment (**YELLOW**)
2. FILL: cover the entire RM PBLOCK tiles (**ORANGE**)
3. Left/Right 2-Progammable Unit expansion (**RED**)

- **Most static shell logic in expanded region will be removed from the Abstract Shell**
  - Partial bitstream generation will include this entire region, but goal is to limit programming to routing resources only
    - Routing necessary to understand what resources are available for RM routing
  - By design this keeps all sensitive static design details out of this expanded region
  - Only remaining logic would be synchronous elements on boundary paths, for timing closure

Summary

Dynamic Function eXchange:

- **Reduces cost, size, power of your system**
- **Provides system flexibility**
- **Intuitive Vivado design flow**

Adaptable.
Intelligent.