CEI UPM | Centro de Electrónica Industrial

# Boosting Flexibility and Computing Performance in Dynamically Reconfigurable FPGA-Based Embedded Systems


**Andrés Otero, Alfonso Rodríguez and Eduardo de la Torre**

{joseandres.otero, alfonso.rodriguez, eduardo.delatorre}@upm.es

UNIVERSIDAD POLITÉCNICA DE MADRID



cei@upm.es


POLITÉCNICA

# Goals

Provide a step-by-step tour on the design of Dynamically Reconfigurable Systems using commercial Xilinx FPGAs

Some Low-level Details

FPGAs

CGRAs

Possibilities

Main Problems

Existing Methodologies

Dynamic and Partial Reconfiguration

ARTICo$^3$

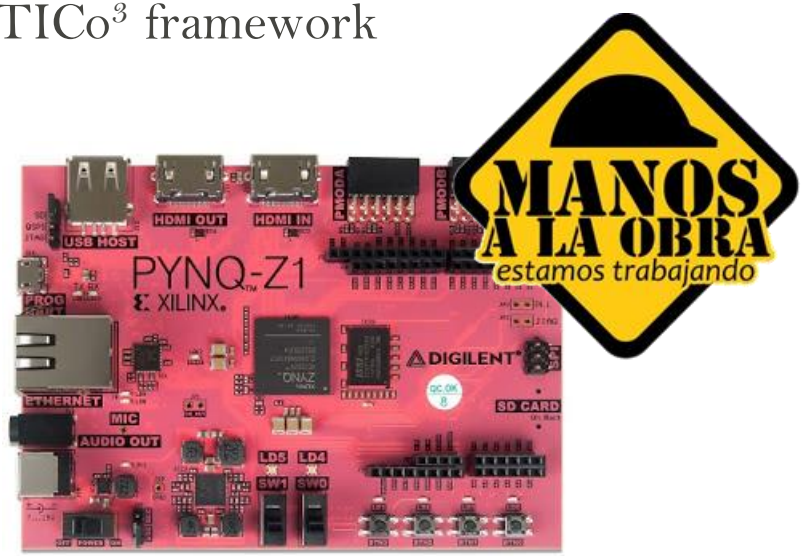
Reconfigurable Computing at CEI

# Agenda

1. Theoretical introduction to the implementation of dynamically reconfigurable systems

2. Practical overview of the Vivado Partial Reconfiguration flow.

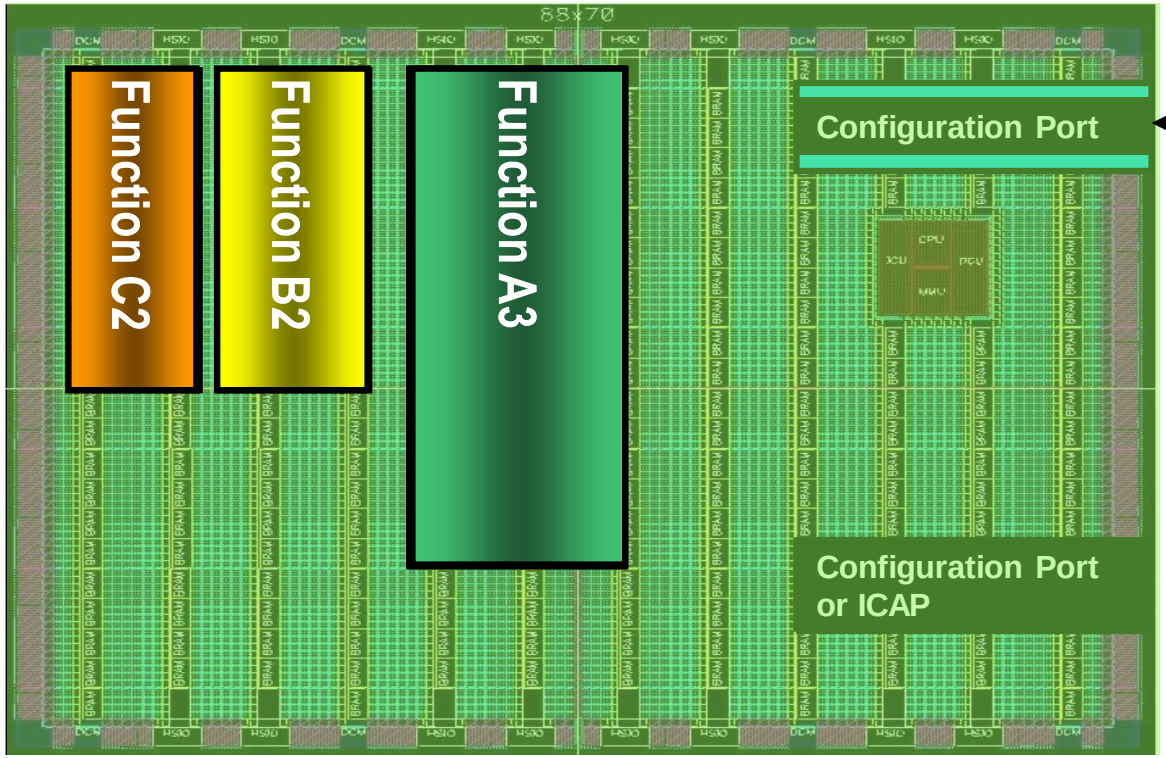
3. Implementing dynamically scalable and fault-tolerant solutions with the ARTICo$^3$ framework

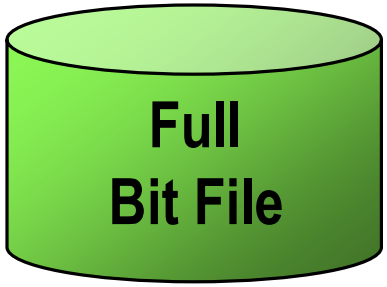


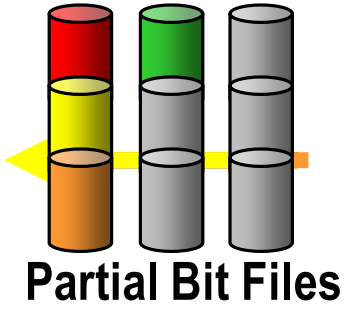
# Theoretical Introduction to the Implementation of Dynamically Reconfigurable Systems

# What is Dynamic and Partial Reconfiguration?

Dynamic and Partial Reconfiguration is the ability to dynamically change blocks of logic by writing partial bit files in the configuration memory of the device while the remaining logic continues working without interruption.



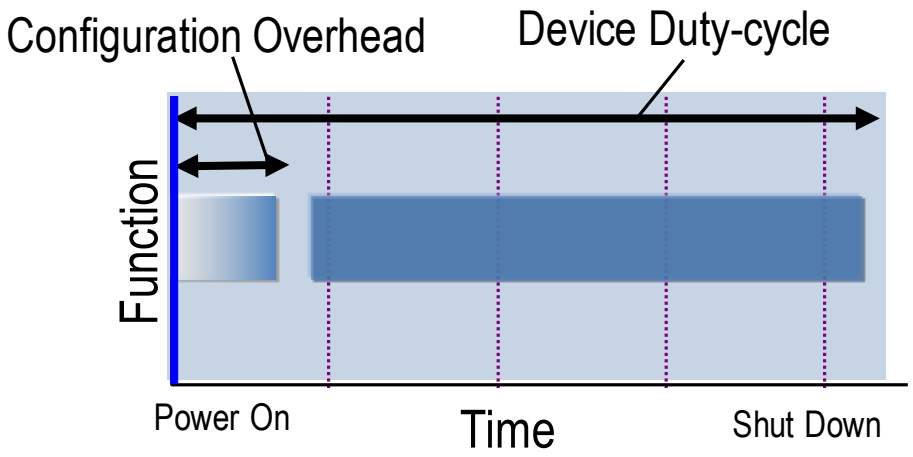



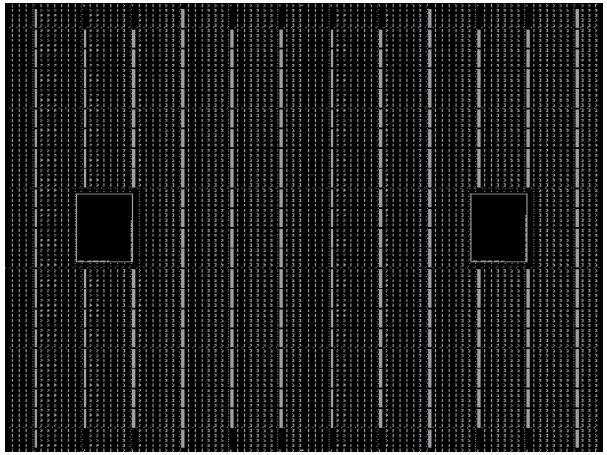




Hardware Update at Run-time, similar to context switch in Microprocessors

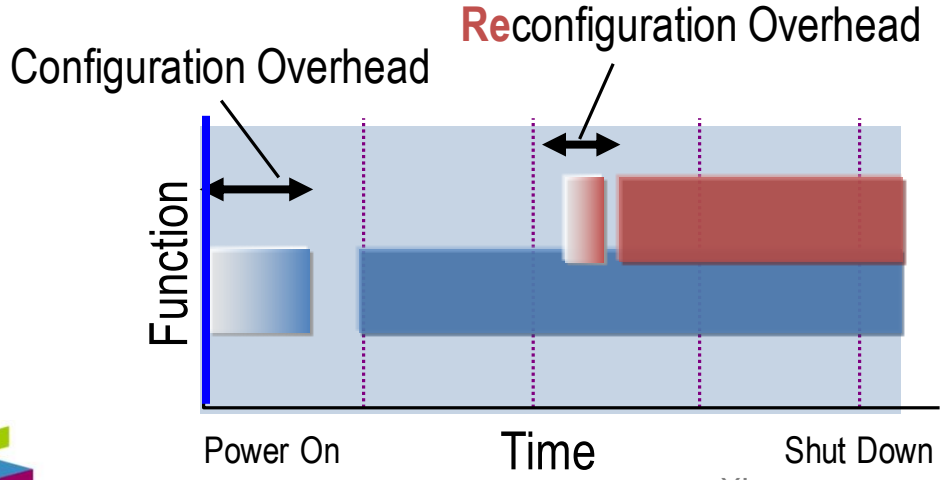
# A new Operation Mode for FPGAs

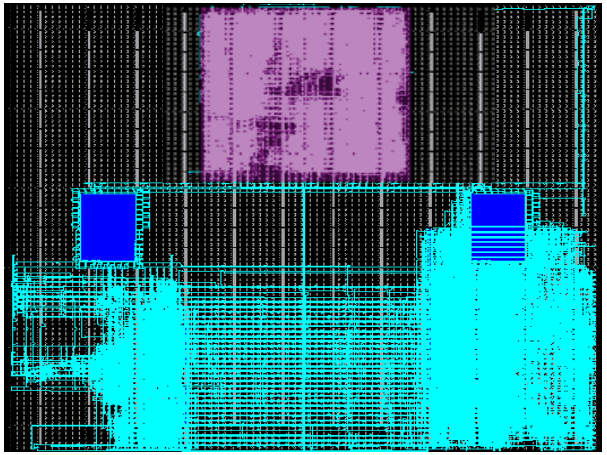
## Typical Operation Mode



## Dynamic and Partial Reconfiguration

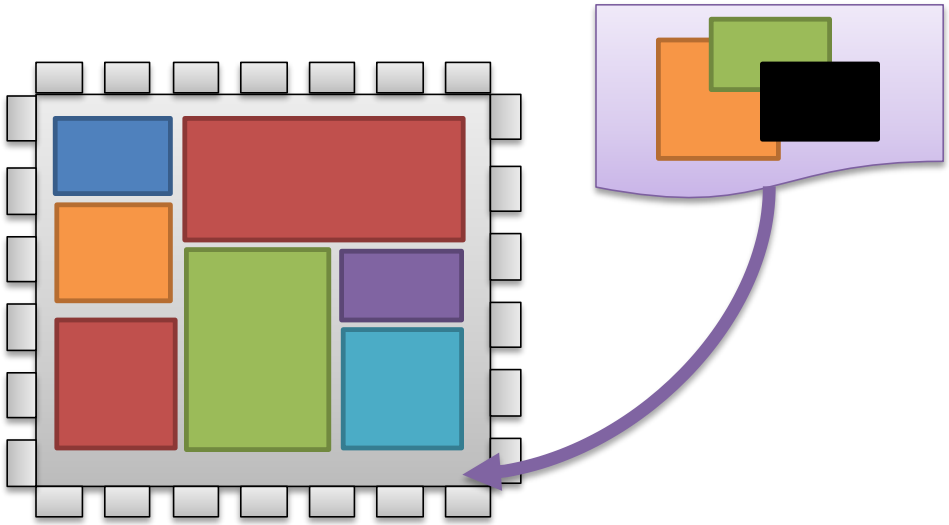


# DPR Enables Reconfigurable Computing

WIKIPEDIA
The Free Encyclopedia

**Reconfigurable Computing** is a computer architecture combining some of the flexibility of software with the high performance of hardware by processing with very flexible high speed computing fabrics like field-programmable gate arrays (FPGAs). The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the datapath itself in addition to the control flow. On the other hand, the main difference with custom hardware, i.e. application-specific integrated circuits (ASICs) is the possibility to adapt the hardware during runtime by "loading" a new circuit on the reconfigurable fabric.





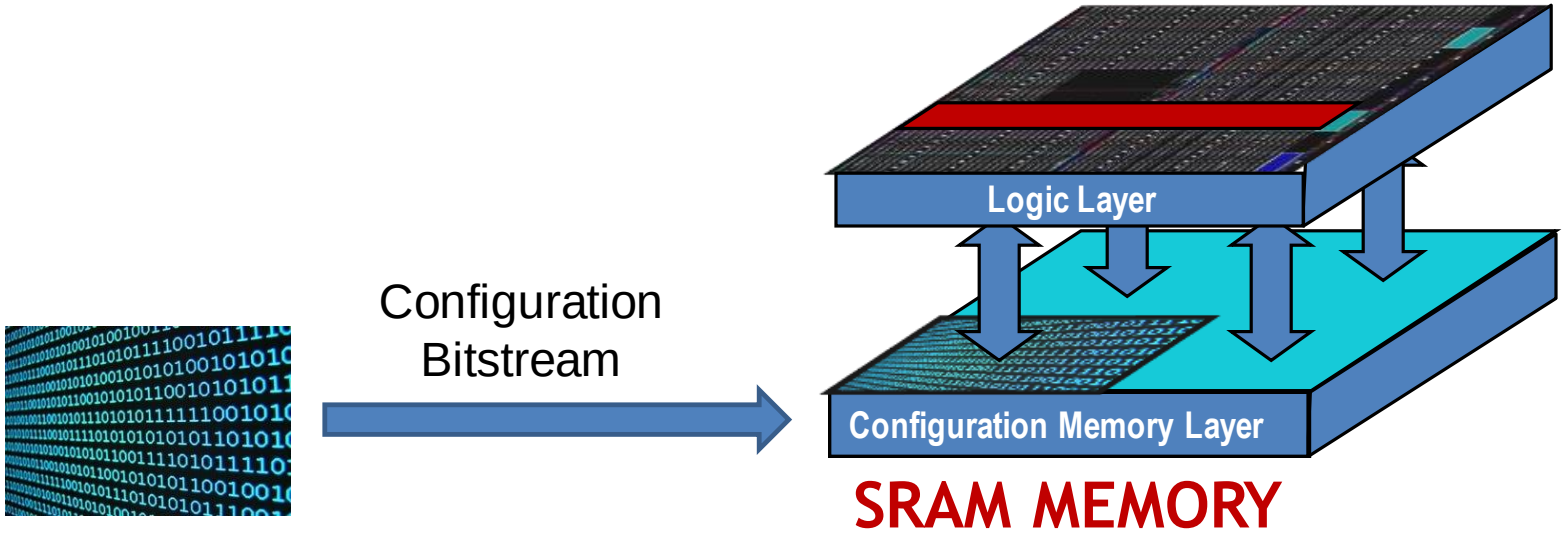
Reconfigurable computing combines some of the flexibility of software with the high performance of hardware.

# Reconfigurable Computing – Why?

- Size and Cost Reduction by silicon reuse
  - Time multiplexing allows implementing systems in smaller and cheaper devices.
- Power Consumption Reduction
  - Use of smaller / fewer devices
  - Switch-off the unused tasks → Reduction of dark silicon
  - More optimized and specific Hardware Designs
- System Flexibility, Allows Adaptive Systems
  - Adaptable Security Standards and Algorithms
  - Communication Systems with upgradable and adaptable Protocols
  - Biological Computing
  - Artificial Intelligence

# Reconfigurable Computing – How?

- We can think on FPGAs as two layered devices:
  - *Application layer*, which contains the whole pool of computing resources
    - Logic, Routing, I/O
  - *Configuration layer*, which selects what to use and how to connect it




Configuration memory controls function computed on logic layer

# Reconfigurable Computing – How?

**Theoretically, reconfiguring an FPGA**

**is as simple as changing a SW program:**

## It's just writing in a memory…

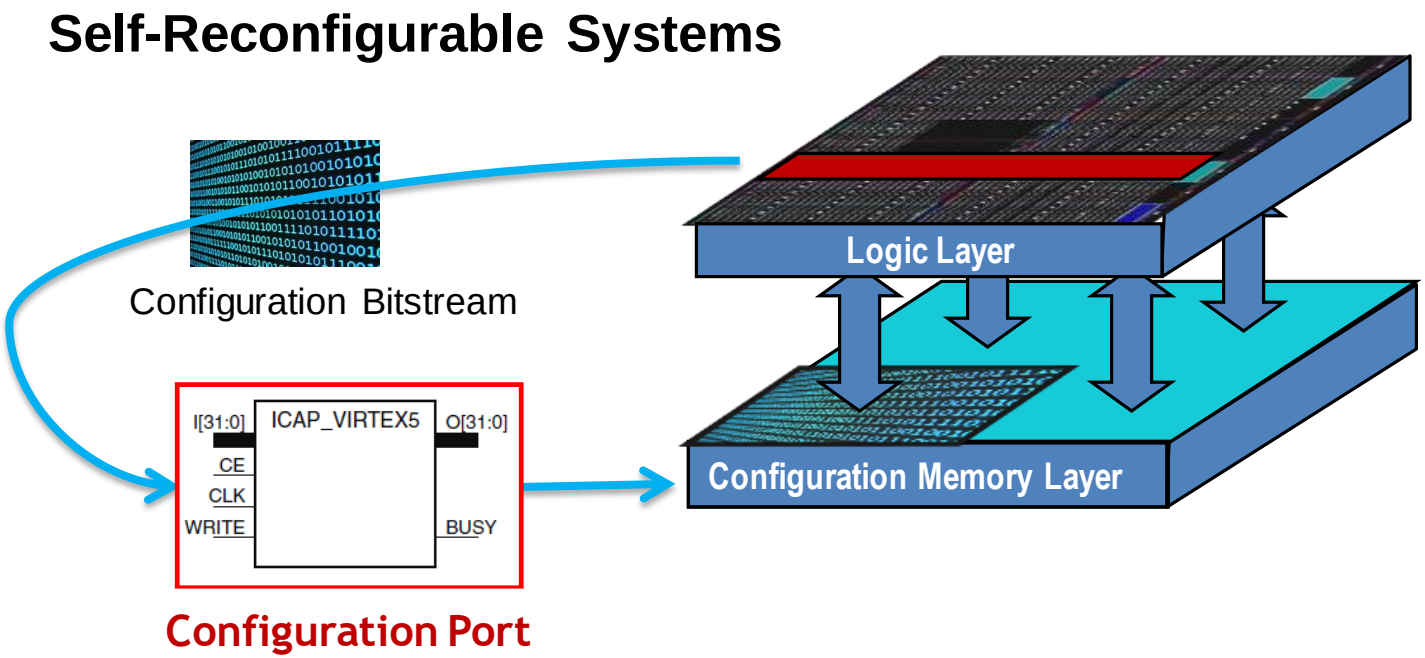
# Reconfigurable Computing – How?

## ... BUT dealing with a lot of low-level details that are device dependent ..

- To Dynamically reconfigure an FPGA…
  - Configuration Ports and Interfaces
  - Configuration Registers
  - Bitstream format

# Reconfiguration Ports

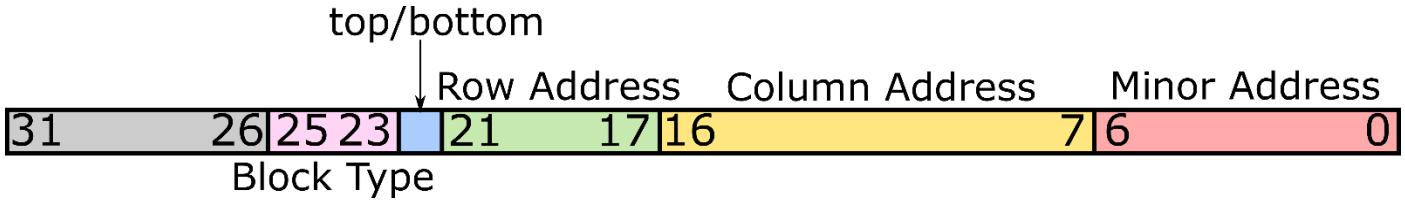
- Reconfiguration ports provide access to the configuration memory.
  - Internal ports are embedded in the device and accessible from the configurable logic. In Zynq:
    - **ICAP** (Internal Configuration Access Port) ← From FPGA
    - **PCAP** (Processor Configuration Access Port) ← From processor logic
  - External configuration: **JTAG, SelectMAP, SPI**, Serial Configuration …

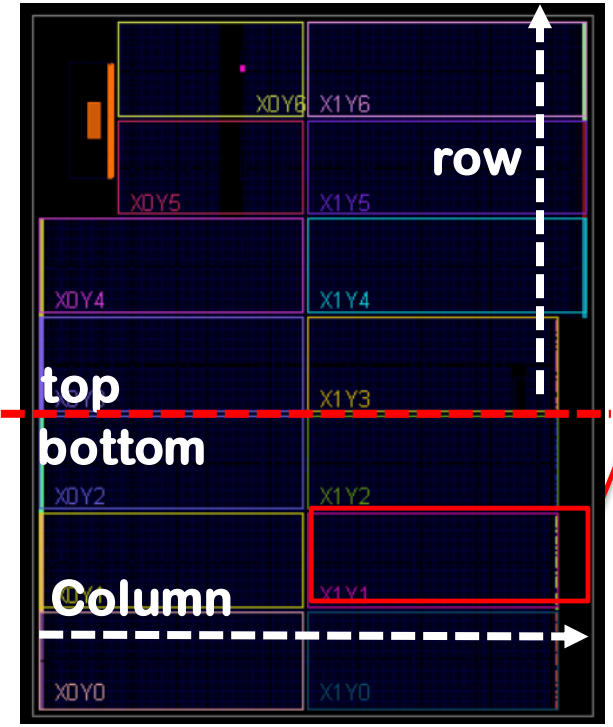


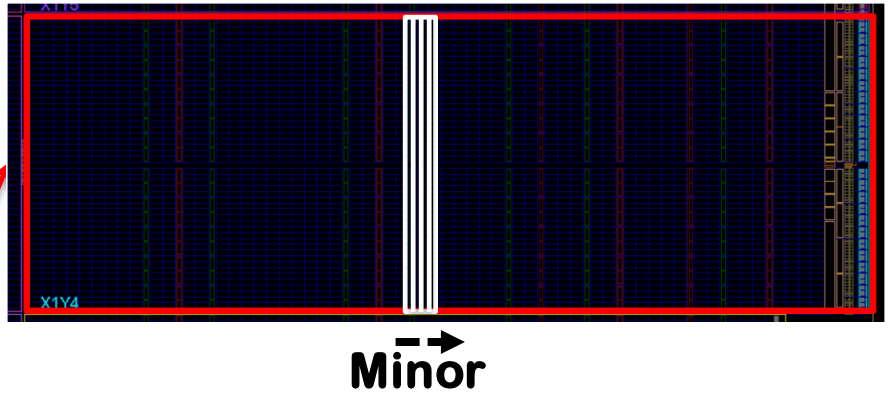
# Configuration Memory: Addressing Scheme

- The minimum addressable unit in the configuration memory is called **frame**.
- One reconfigurable frame defines the finest granularity for partial reconfiguration.

Xilinx Series 7



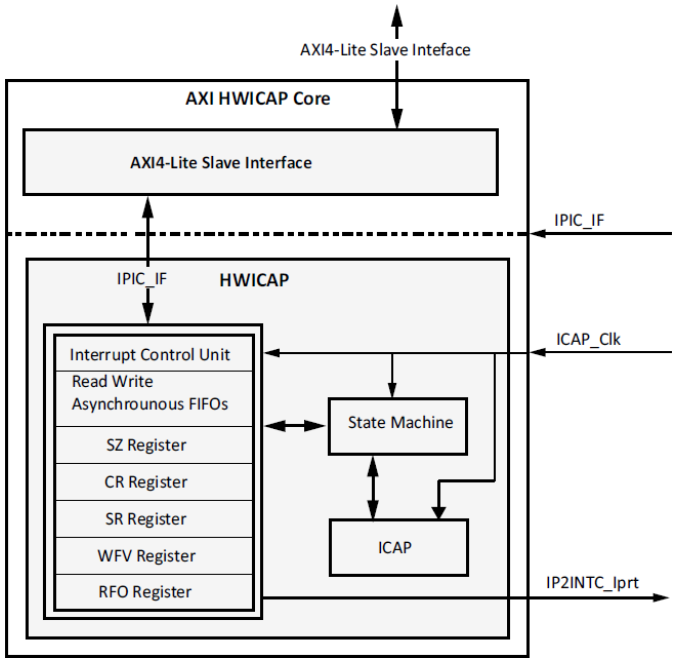

Granularity in Xilinx Series-7 is
- Slice region: 50 CLB high by 1 CLB wide
- BRAM region: 10 RAMB36
- DSP region: 20 DSP48

In Ultrascale / Ultrascale+
- Slice region: 1 CLB high by 2 CLB wide

CEI UPM

# Reconfiguration Ports: AXI HWICAP

- The core is a wrapper for the ICAP port enabling partial reconfiguration in a processor-based system.
  - Accessible through software API in xhwicap.h
- Provides the interface necessary to transfer data to and from the ICAP device Primitive
- Required data is first stored within a Write FIFO, from where it can be sent to the ICAP
- The data that is read from the ICAP is stored in the Read FIFO before it is read by the application
- Two clock domains
  - The s_axi_aclk for the AXI-Lite interface
  - The ICAP_Clk for the data exchange between the core and the ICAPE
    - Recommended speed of ICAP_Clk is 100 MHz



# Reconfigurable Computing – How?

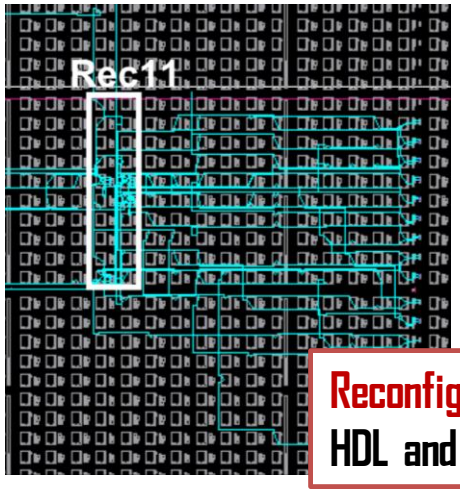
## ... BUT dealing with a lot of low-level details that are device dependent ..

- To Dynamically reconfigure an FPGA…
  - Configuration Ports and Interfaces
  - Configuration Registers
  - Bitstream format

## .. and so we need specific Design Methodologies and Tools!

# DPR Design Flow Overview: bottom-up strategy

- Partial Reconfiguration requires the isolation of the different Reconfigurable Modules. **Bottom-up strategies are needed!**
- The **bottom-up strategy** is to implement each reconfigurable module and the static system independently, so:
  - For each module a netlist (checkpoint) is generated
  - Changing a module involves implementing only the affected module.
  - Referred to as out-of-context (OOC) synthesis in Xilinx Vivado

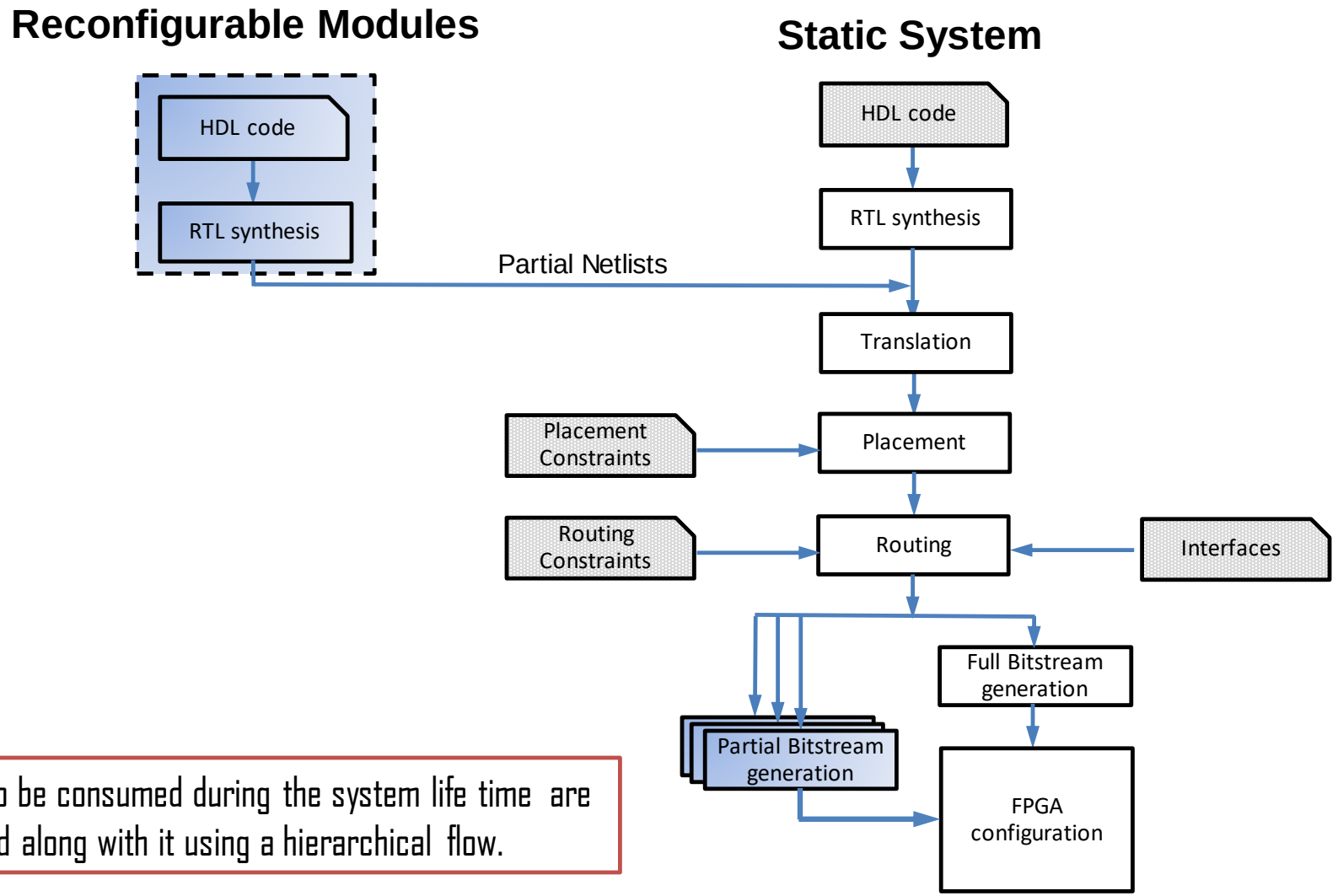



Necessary for Partial Reconfiguration

Reconfigurable IOs described as ports in HDL and mapped to external IOBs

- Top-down synthesis, where design is flattened is not (generally) used for DPR.

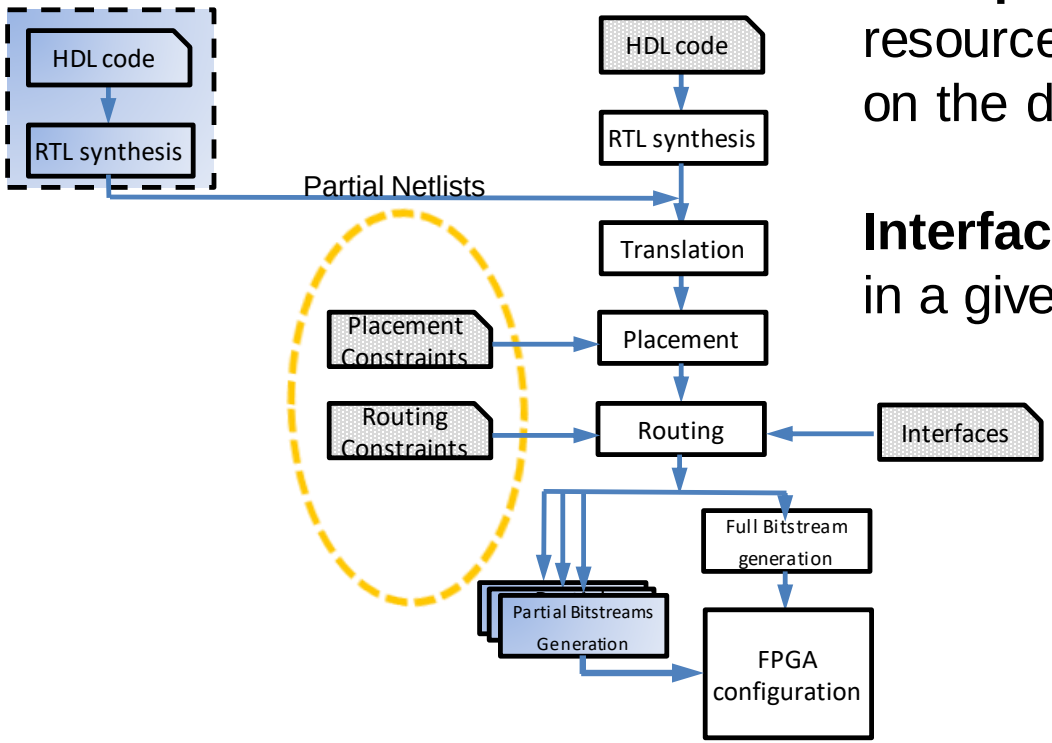
# Typical DPR Design Flow Overview




Cores to be consumed during the system life time are designed along with it using a hierarchical flow.

# DPR Design Flow Overview: Specific Steps

**Temporal Partitioning** the design into independent functionalities to be considered as one of the reconfigurable modules.

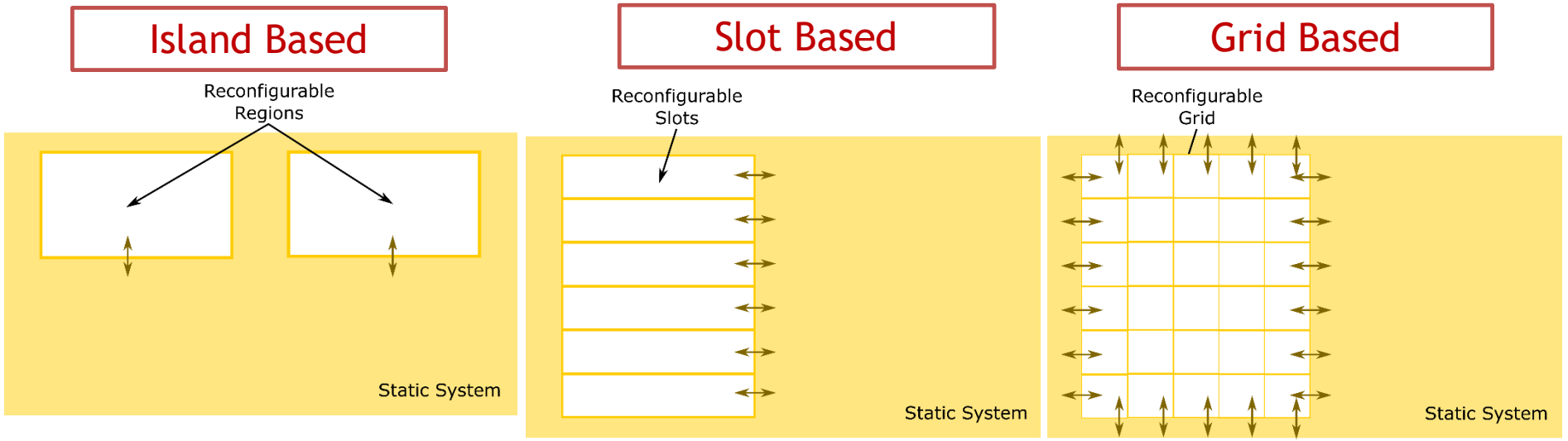



**Floorplanning –** allocate physical resources to the reconfigurable regions on the device

**Interface -** All the RMs to be configured in a given RR must share the same.

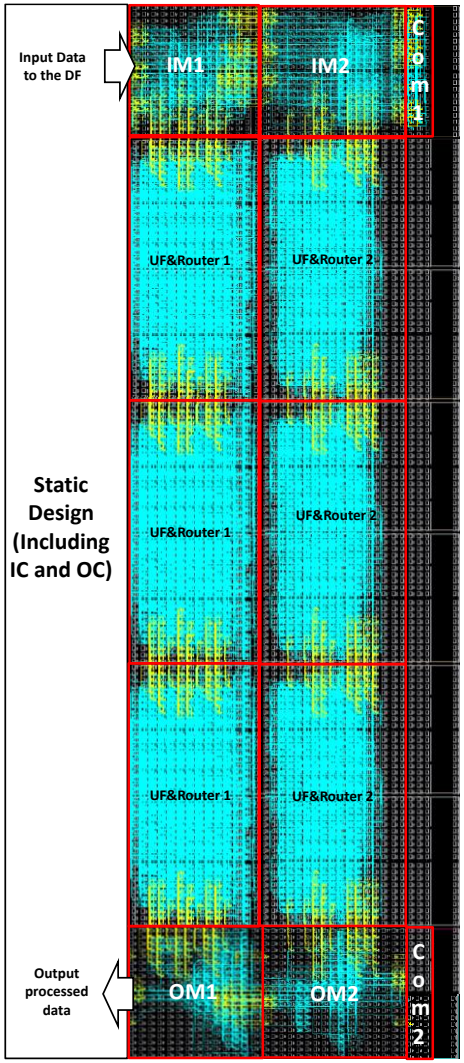
# Placement Flexibility: Virtual Architectures

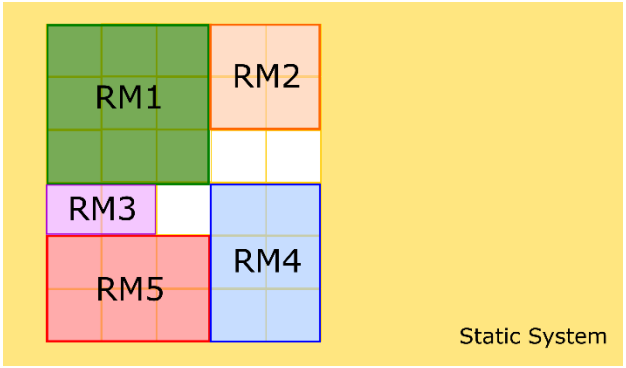
**Virtualization** of FPGA physical architecture that includes a **resource distribution** and defines **how modules are interconnected**.

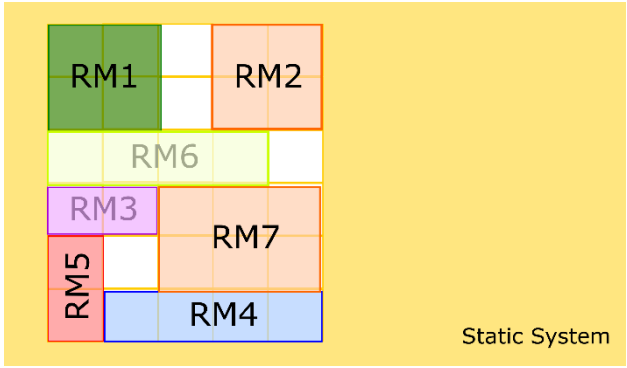



- **Island Style** – Reconfigurable Regions can not be merged
  - Suffers from fragmentation
- **Slot Style** – Also suffers from fragmentation, but less…
  - Strictly defines all slots to be equal and to use special resources to access the on-chip communication.
  - Optimal slot size depends on the modules and communication costs.
  - Slots can be grouped
- **Grid Style** – Reduced fragmentation, difficult run-time implementation

# Virtual Architectures: 2D Grid Architectures



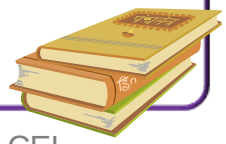




- **2D placement** provides a higher **flexibility** and **reduces internal fragmentation** but with a higher communication cost.
  - ¿Buses? ¿NoCs? ¿Point-to-point connections?

**Scalable Deblocking Filter**

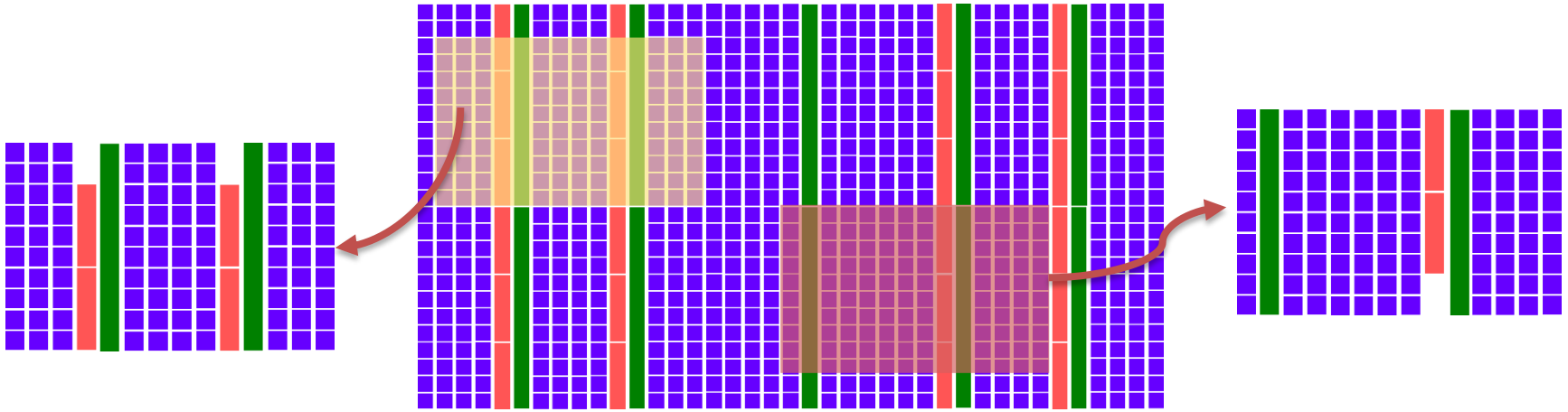
Device: FPGA V5-LX110T

Array 3×2 Fus



# Placement Flexibility: Relocation

- **Relocation** involves both Design and Run-time issues:
  - At run time: Reduces the memory footprint needed to store a modular and regular architecture (**M** vs. **MxN**, *being M the number of RM and N the number of RRs*)
    - … but bitstream header must be modified at run-time
  - At design time: Arrangement of resources must be considered when floorplanning a Reconfigurable System.

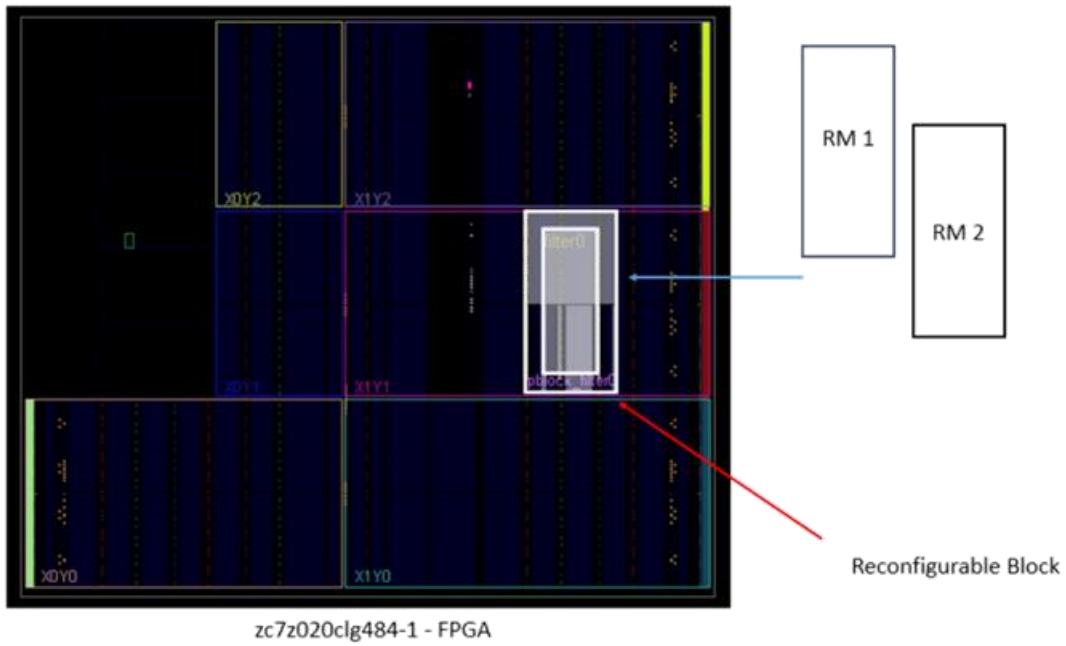


The main requirements to enable Relocation are:

- identical (size and resources arrangement) origin and destination region
- identical relative Communication Interfaces
- Identical routing between the static part and relocatable regions

# Placement Flexibility: pBlocks

**Pblocks -** Physical constraints that define the reconfigurable regions.



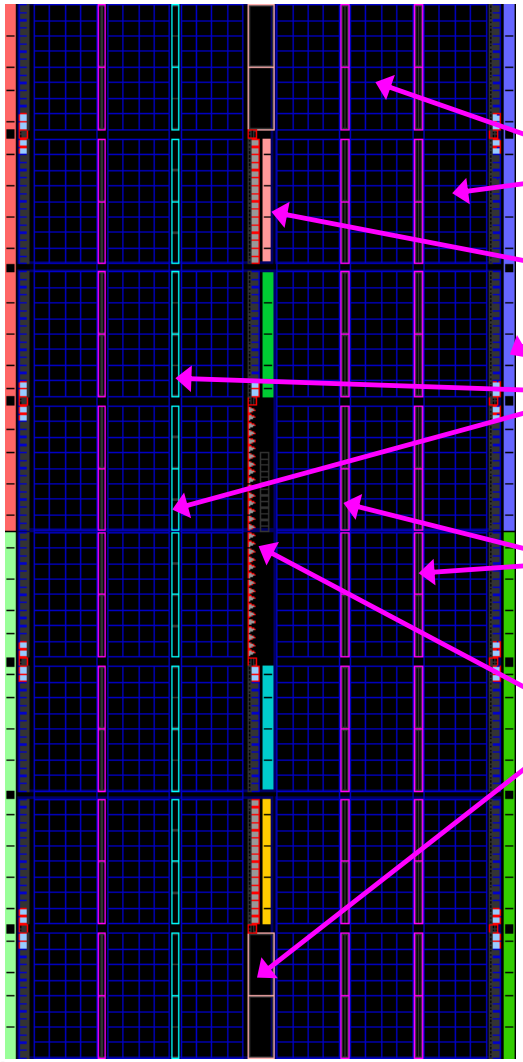




Figure 1. Virtex-4 LX15 FPGA layout

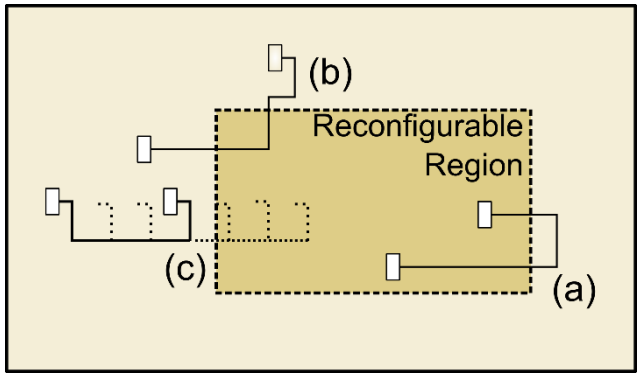
```
set_property SNAPPING_MODE ON [get_pblocks ]
```

The implementation tools automatically see the corrected Pblock ranges.

# Implementation of DPR Systems (I)

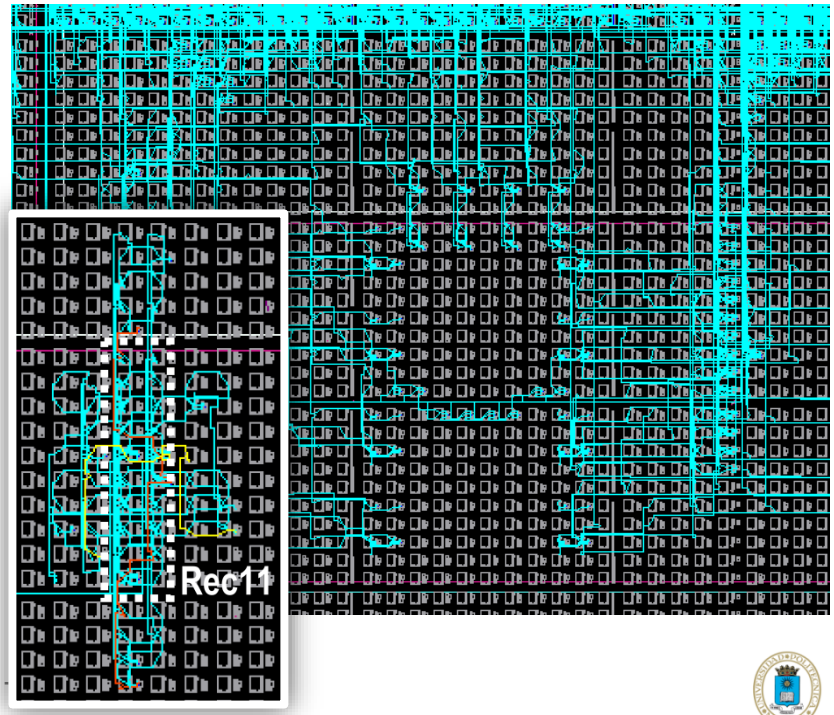
## Routing Conflicts

- Each reconfigurable module must be constrained within a bounded reconfigurable region without overlapping any other.
  - Logic area constraints supported by commercial tools.
  - Routing constraints are a problem




## Inter-module Communications

- Guarantee that identical wires are used to connect all the partial modules to be configured in a given region.
  - With a Low Area and Delay Overhead



# Implementation of DPR Systems (II)

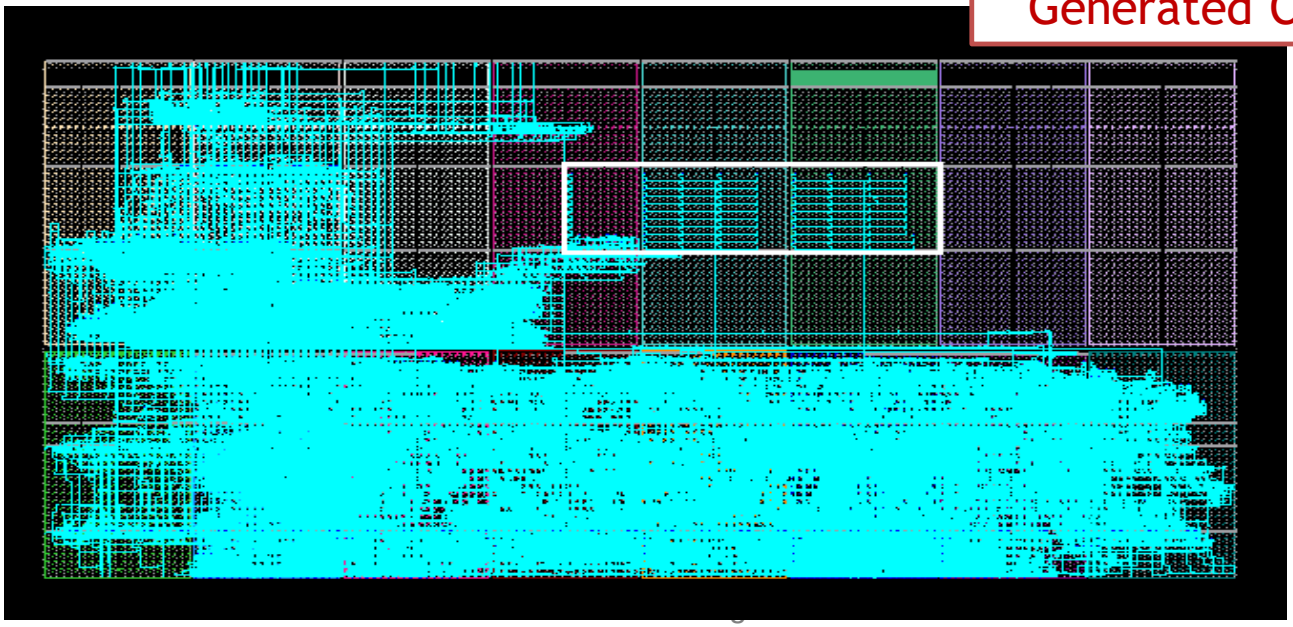
## Pre-route Global Clock Trees

- Activate all the clock trees used by reconfigurable modules

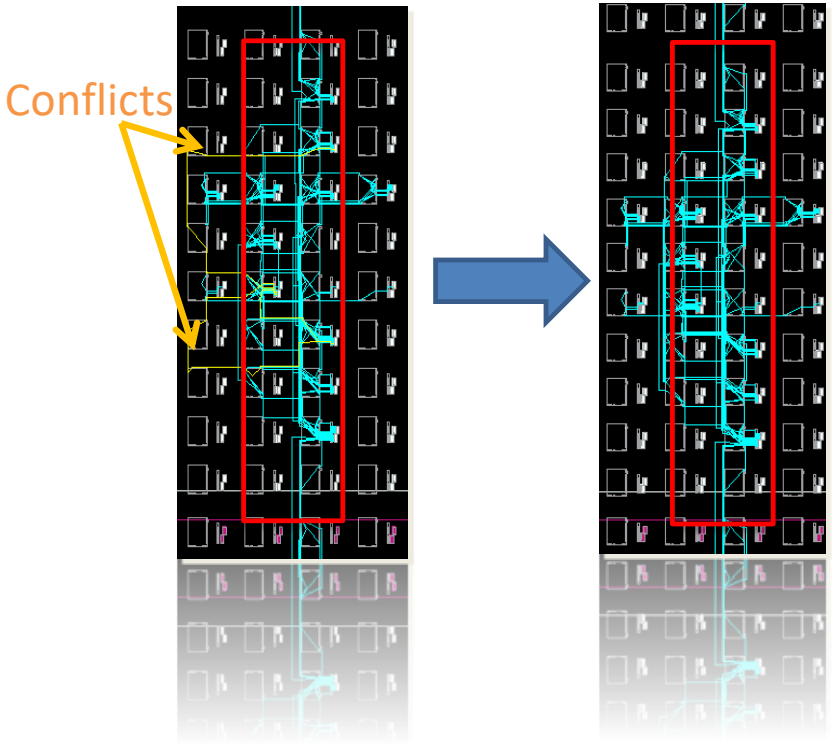
## Generation of Partial Bitstreams

- **Extract** the configuration corresponding to each **reconfigurable module**

Generated Clock Tree



# Routing Conflicts

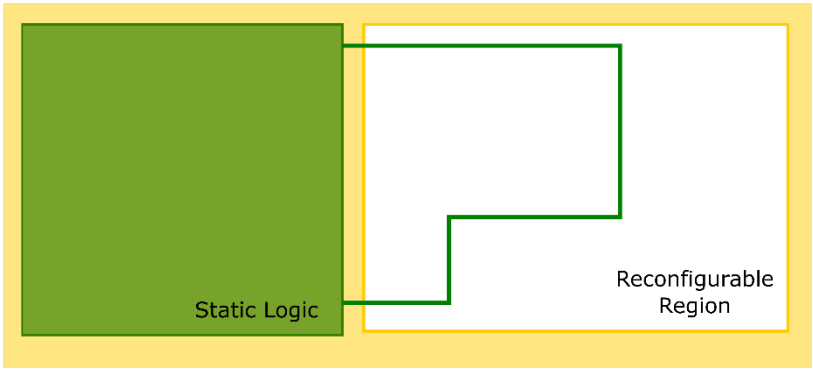


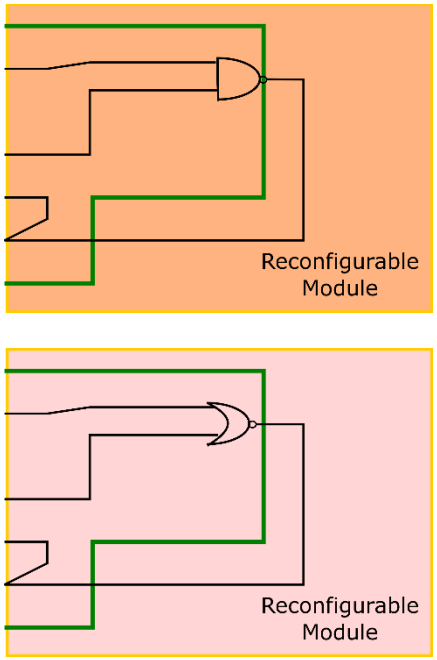

Each reconfigurable module must be constrained within a bounded region without overlapping any other.

- Allow static Routes crossing the Reconfigurable Regions
  - Prevents Relocation.
- **Re-route conflicting Nets** or use **Blocking Nets** to prevent routes exiting the reconfigurable regions.

# Routing Conflicts: The XILINX Solution

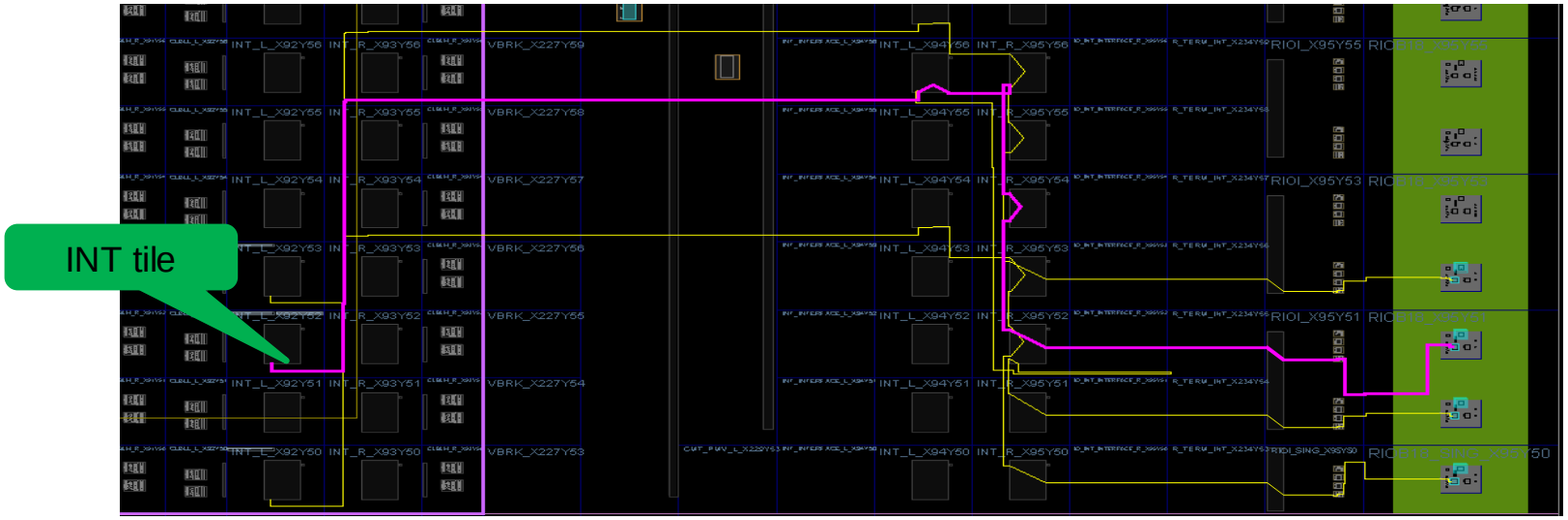
- Static routes (**feed-through** routes) are maintained for all RMs
  - Reconfigurable Regions can have both static and RM routes within them
  - Static routes have precedence with all others being implemented in contex
  - It is not compatible with module Relocation.
- RM routes must be fully contained within the RP boundary
  - For larger RMs it may be more difficult to route and/or meet timing
  - *Recommendations: Make the RP boundary 5–10% larger than an equivalent flat design*



# Inter-module Communications: The XILINX Solution

Partition Pins




- Crossing Wires (mid-route) are automatically identified by the tool, similar to Virtual Borders. Not under the control of the user.
- No overhead (LUTs or FFs) at reconfigurable partition interface

**Similar approach provided by Virtual Borders (CEI-UPM)**

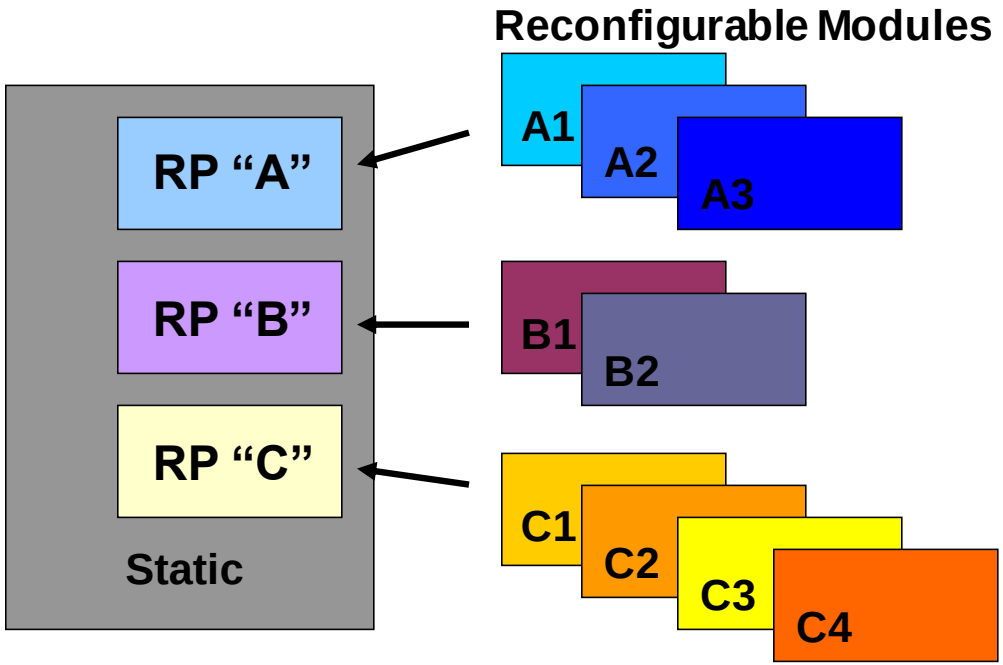
# Practical overview of the Vivado Partial Reconfiguration flow.

# Some Definitions…

- Partition
  - A logical block (entity or instance) to be used for design reuse
  - User determines implementation versus preservation for each block
- Reconfigurable Partition (RP)
  - Design hierarchy instance marked by the user for reconfiguration
- Reconfigurable Module (RM)
  - Portion of the logical design that occupies the Reconfigurable Partition
  - Each RP may have multiple Reconfigurable Modules
- Static Logic
  - All logic in the design that is not reconfigurable

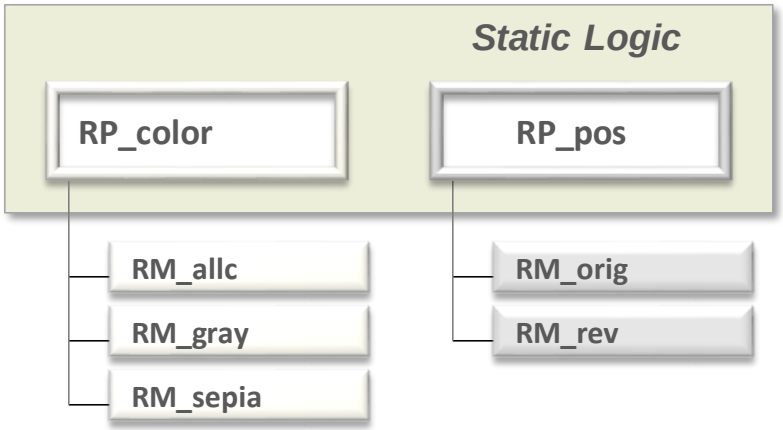
# Some Definitions…

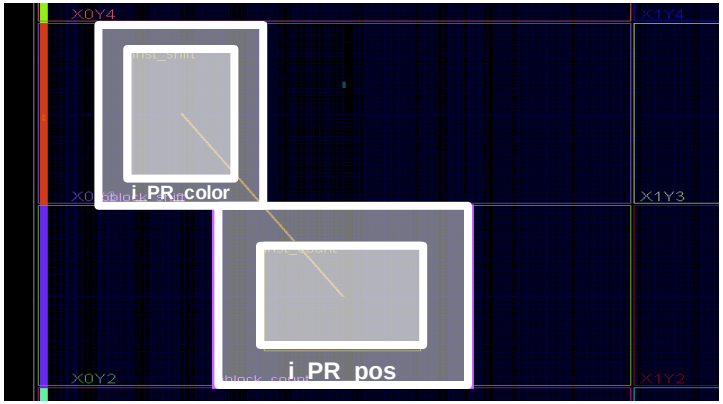
- Configuration
  - A full design image consisting of Static Logic and one Reconfigurable Module for each Reconfigurable Partition
  - Any combination of RM for each RP where it can be reconfigured is a valid configuration (Relocation not allowed with Xilinx Flow)



# Summary of the Design Flow

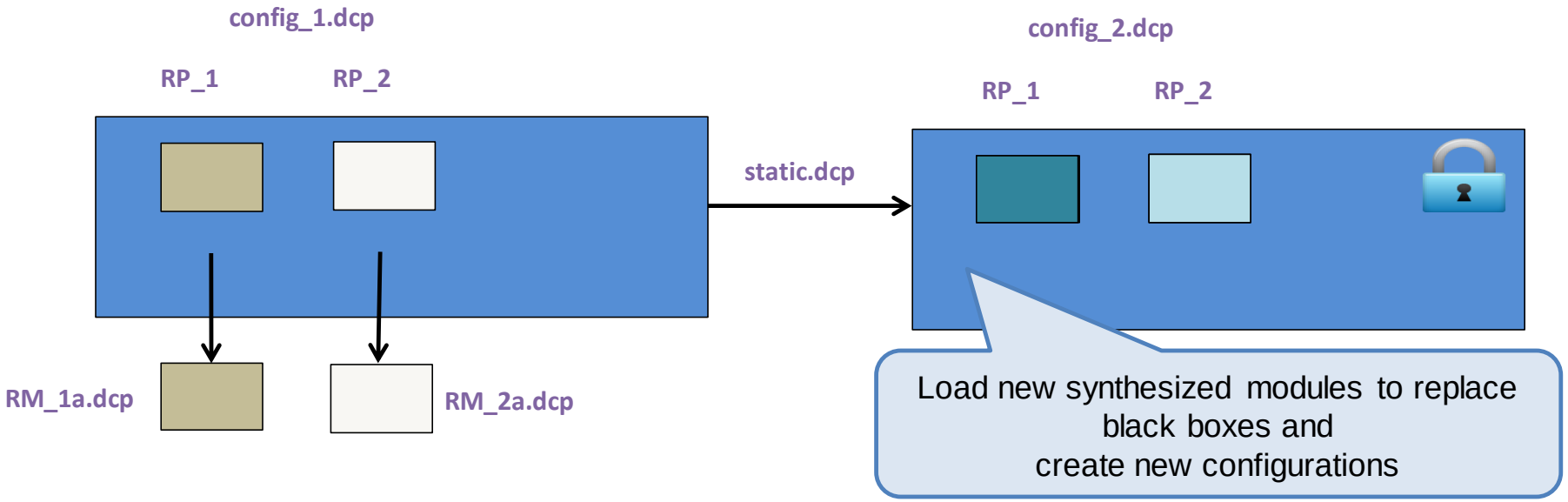
- Structure your design
  - Static Logic (unchanging design)
  - Reconfigurable Partitions (RP)
    - Instances to be reconfigured
  - Reconfigurable Modules (RM)
    - Functional variations for each RP
- Synthesize bottom-up
  - `synth_design -mode out_of_context`
- Define resources to be reconfigured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



# Summary of the Design Flow

- 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 as their own checkpoints



# Summary of the Design Flow

- Place and Route all design configurations
  - Apply full design constraints in-context
  - Use normal timing closure, simulation and verification techniques
  - Use scripted non-project flow or new RTL-based project flow
- Final Verification
  - Validates consistency of place and routed results across the entire system
- Generate Bitstreams
  - write_bitstream automatically creates all full and partial bit files by default
  - Selectively generate full bitstreams or specific partial bitstreams

# Implementing dynamically scalable and fault-tolerant solutions with the ARTICo$^3$ framework

# ARTICo³: Motivation



Independent of the task itself!

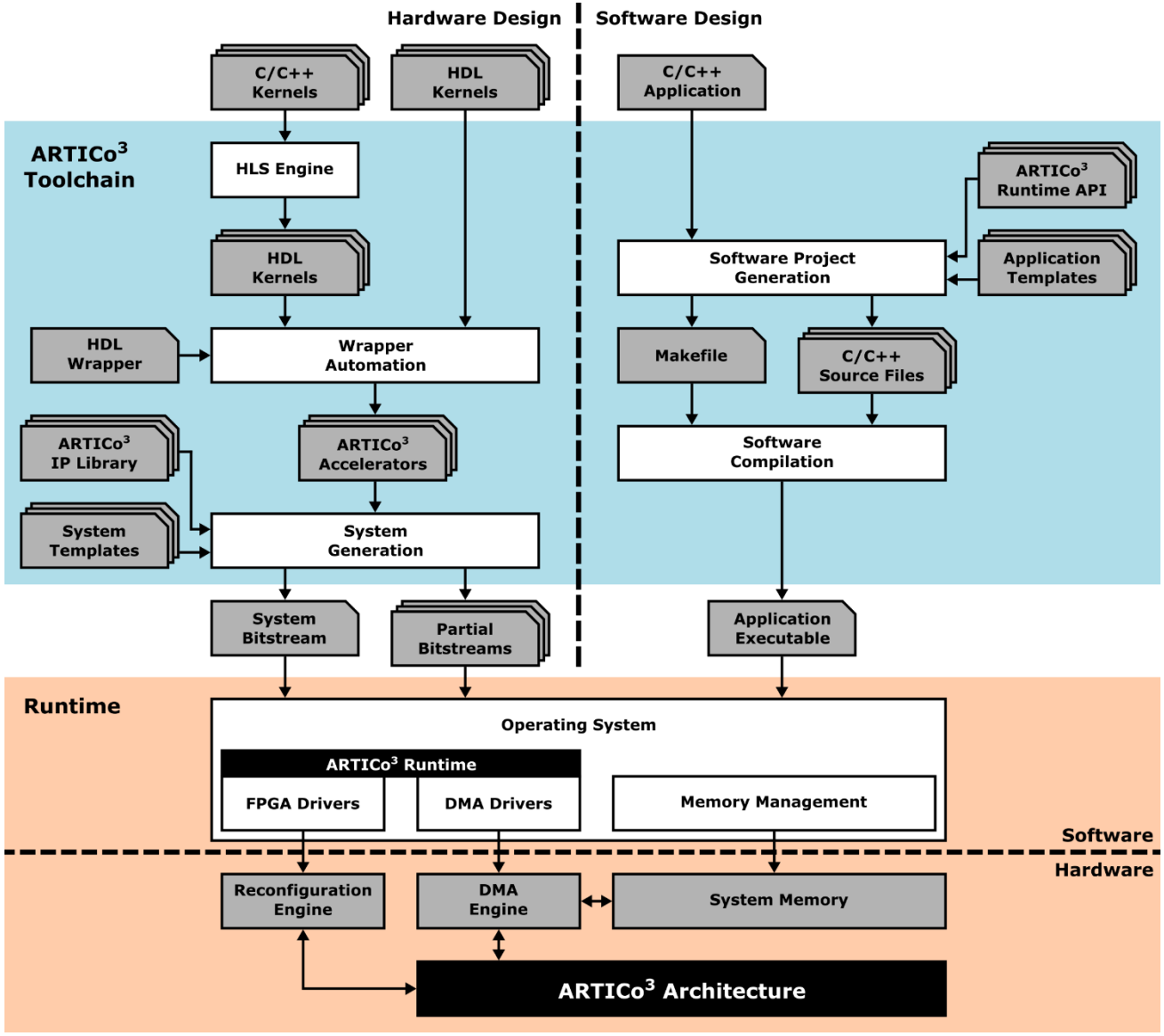

Put as many people as possible to work, and save time


Do not tell others what you are doing, protect from others

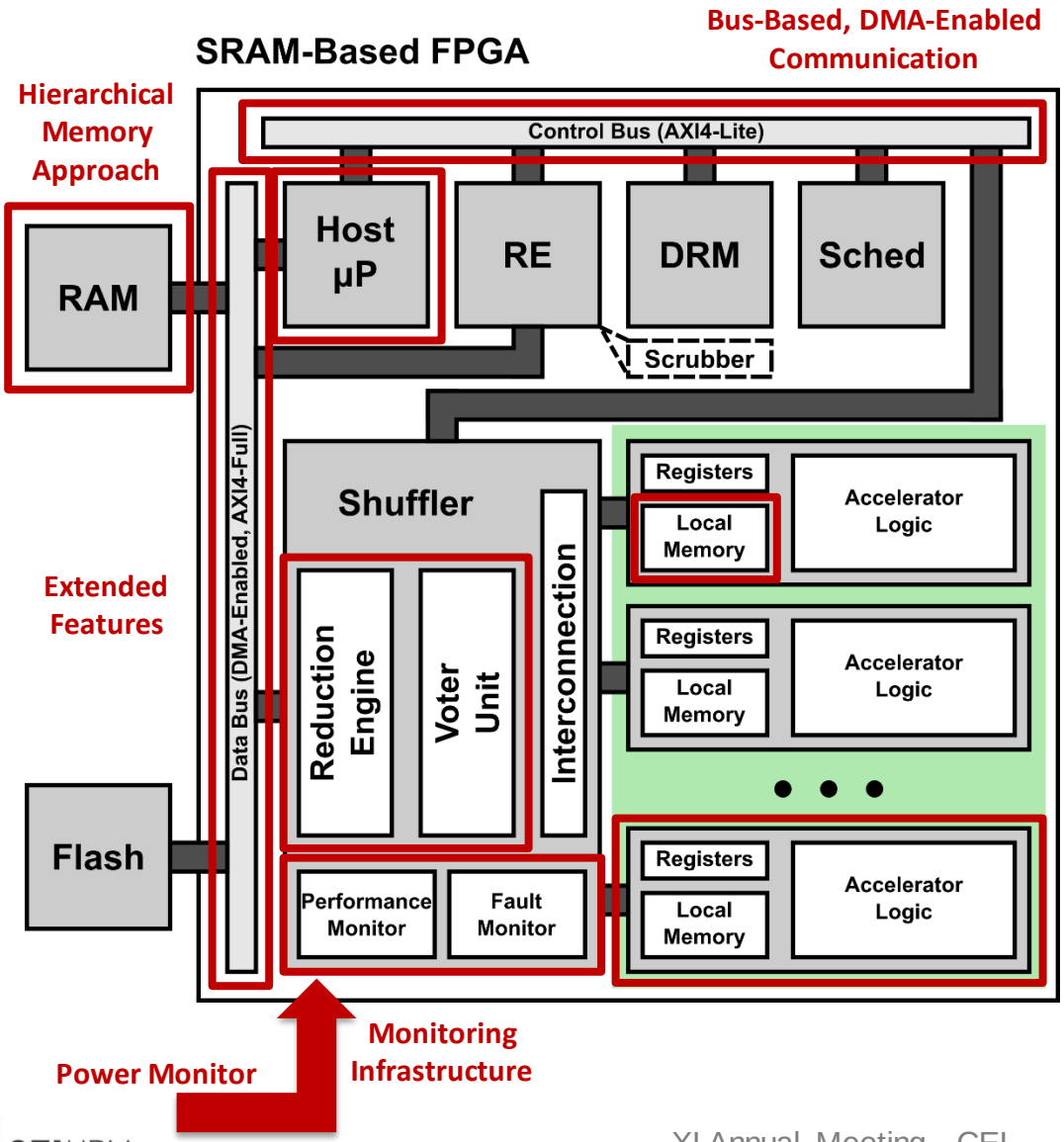

Let more than one do the same work and compare result

Dynamic and Partial Reconfiguration
(hardware acceleration + module replication)

# The ARTICo³ Framework

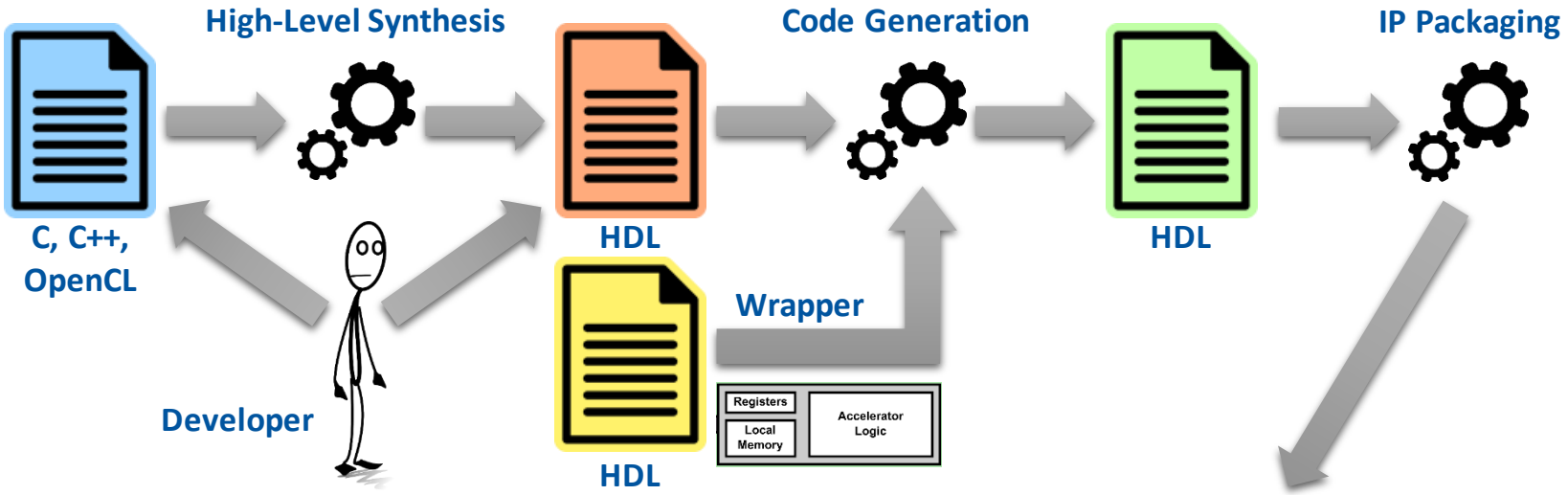


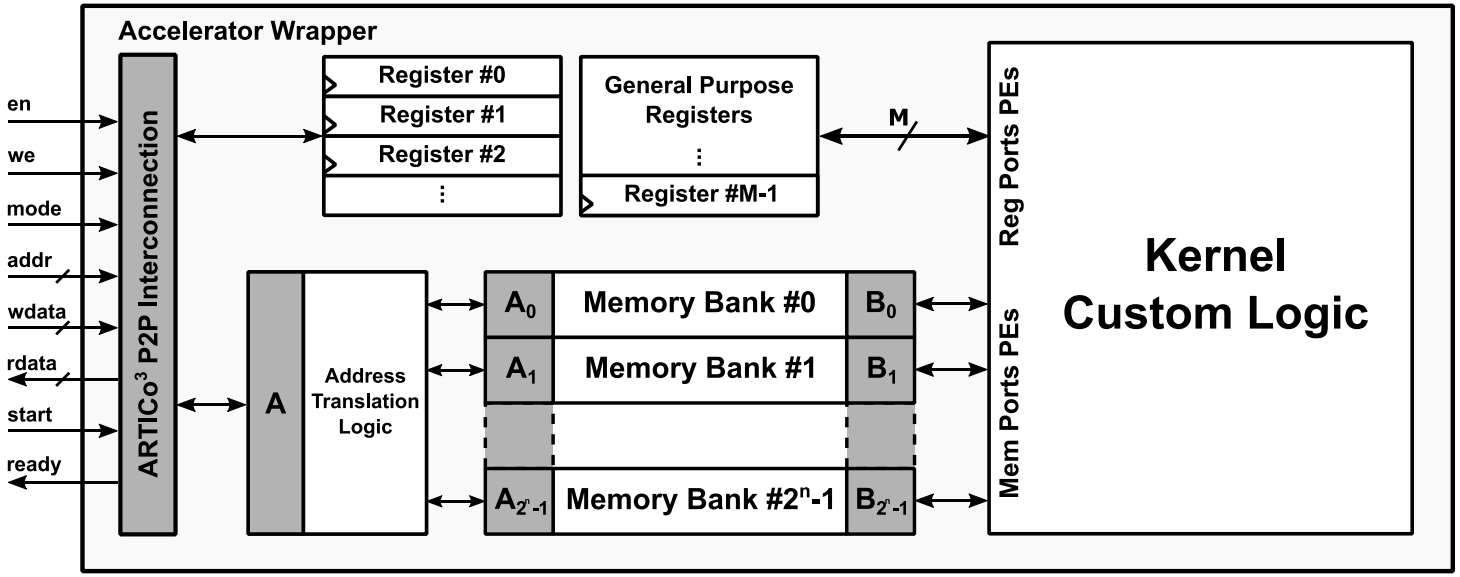
# ARTICo³ Architecture



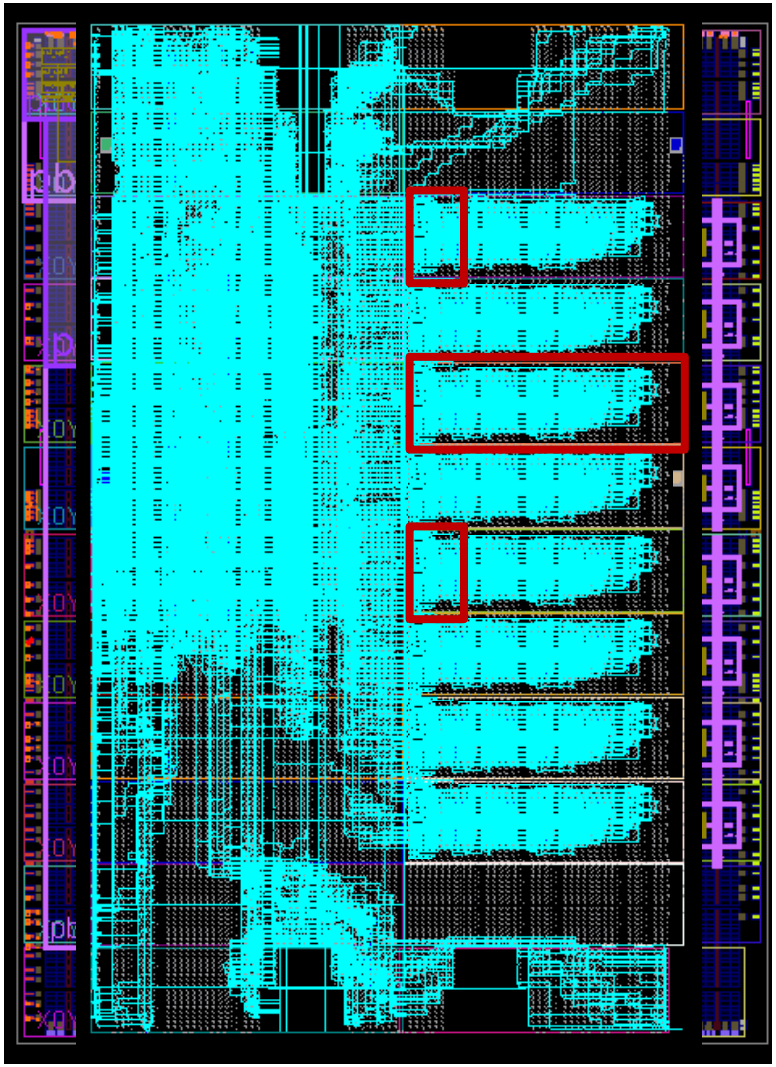

- Reconfigurable
- Reconfigurable
- Management
- Performance
- Consumption and
- Reliability

# ARTICo³-Compliant Accelerator Design



# DPR-Compatible Floorplanning



Low-Level Constraints

Design Placement

Design Routing

Technology Dependencies

"Homogeneous" Fabric Layout

Common Interfaces

Hardware "Copy & Paste"

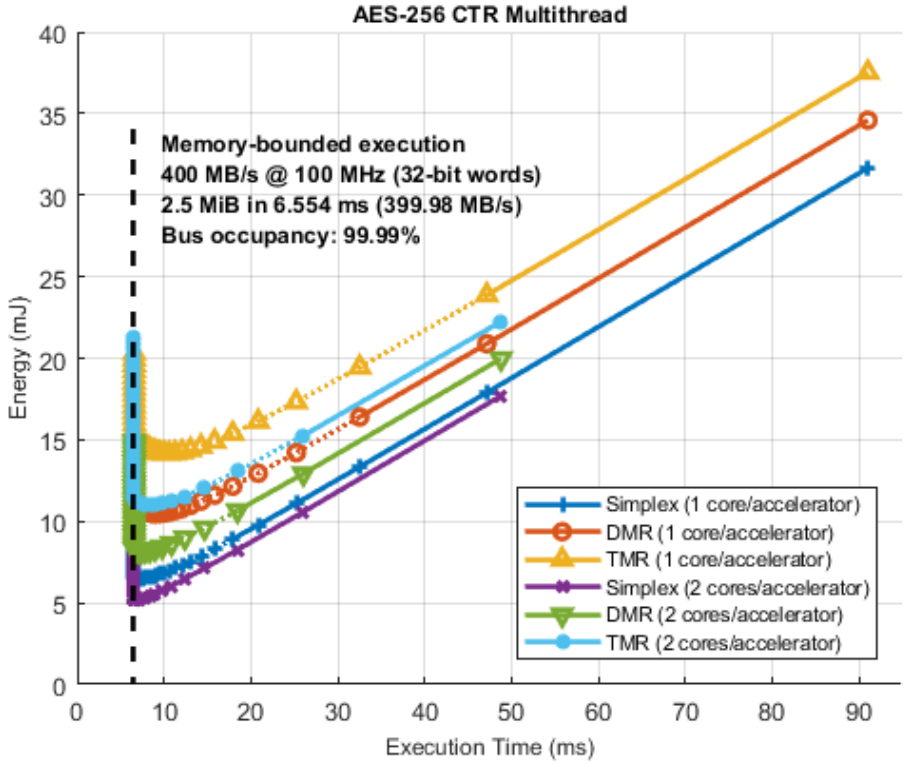
# Runtime Support

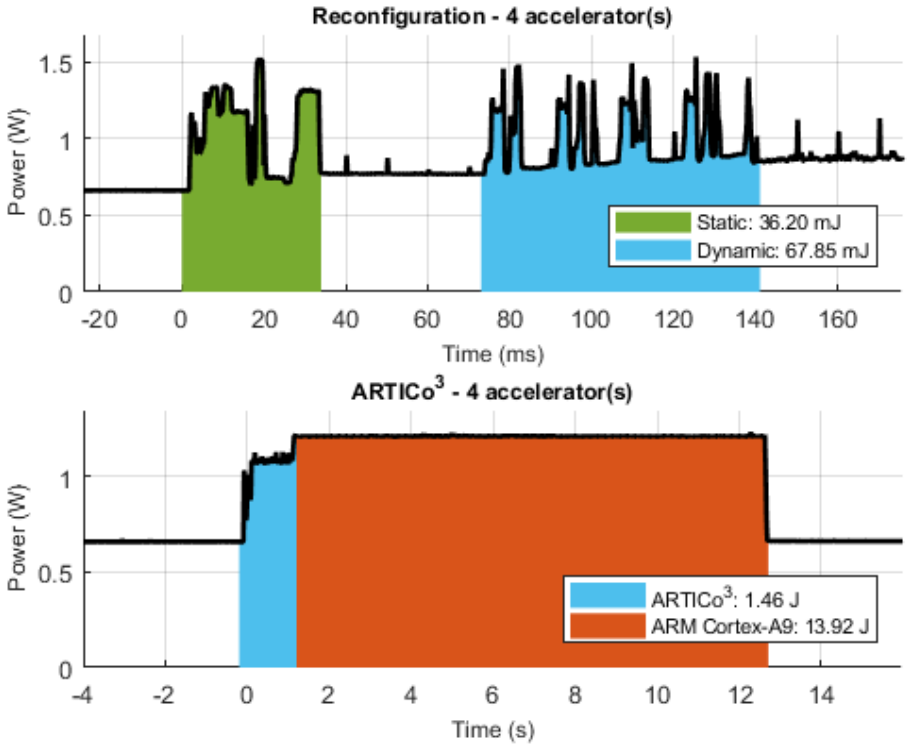
**Run-Time Adaptation**

Computing Performance

Energy Consumption

Fault Tolerance



# Tutorial Outline

- The ARTICo$^3$ repo
  - Where are architecture, toolchain and runtime?
  - I don't know how to set up an embedded Linux, can I use ARTICo$^3$?
  - Open Source (not available yet!): **https://github.com/XXX/artico3.git**

- Demo applications
  - Dummy HLS-based kernel generation
    - ARTICo$^3$ project structure and configuration
    - Memory-based I/O, register-based I/O
    - Host application development: ARTICo$^3$ API
  - Matrix multiplication (yes, we know…)
    - Test of a prebuilt example