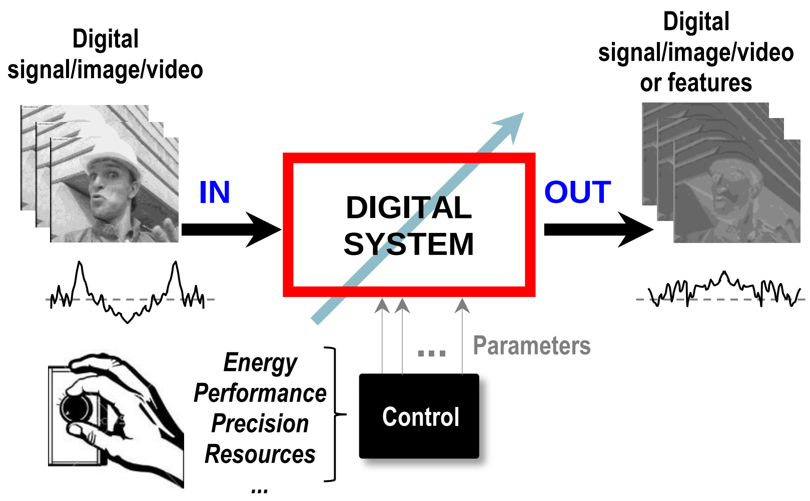
# Unit 6 – Dynamic Partial Reconfiguration

## INTRODUCTION TO SELF-RECONFIGURABLE SYSTEMS

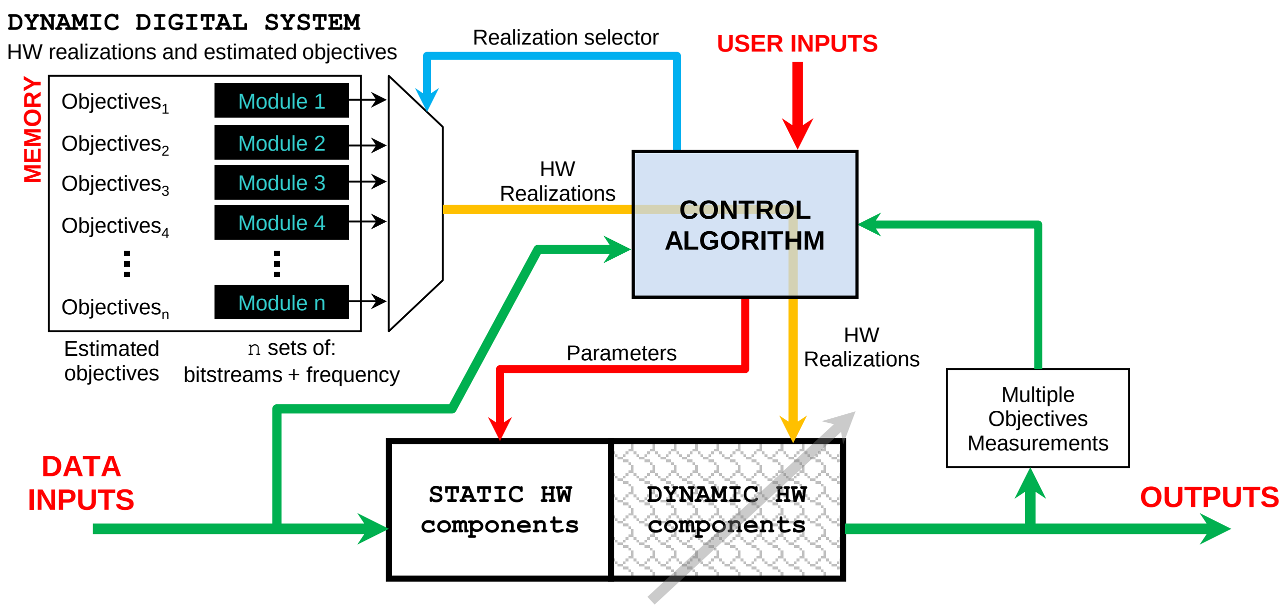
### MOTIVATION

- Digital systems can be characterized by a series of properties (or objectives):
  Energy, Performance, Accuracy, Hardware footprint, Bandwidth, etc.

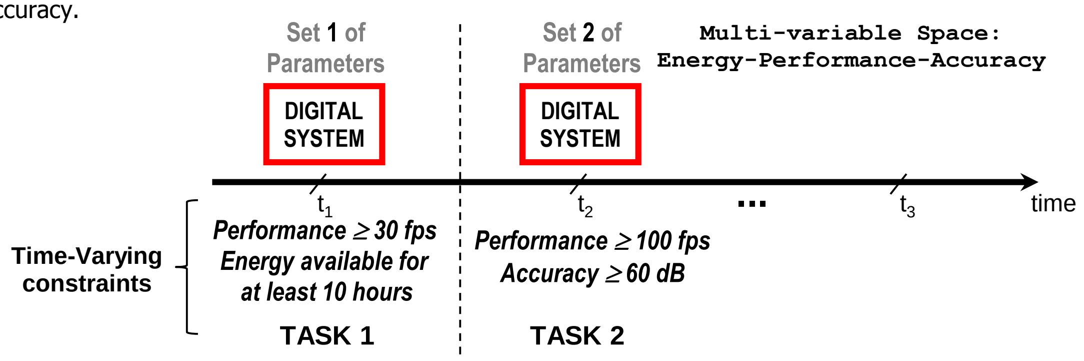


- **Dynamic Reconfigurable Computing Management**: The ability to control the aforementioned properties at run-time. We can deliver a dynamically self-adaptive system (by dynamic allocation of resources and dynamic frequency control) that satisfies time-varying simultaneous requirements.

- **Dynamic Reconfigurable Computing Management** allows us to control digital system properties at run-time:



- The system can then carry out independent tasks in time. For example:
  - ✓ Task 1: A video processing system is asked to deliver real time performance at 30 frames per second on limited battery life that will also need to operate for at least 10 hours.
  - ✓ Task 2: The video processing system is asked to deliver performance at 100 frames per second at some minimum level of accuracy.

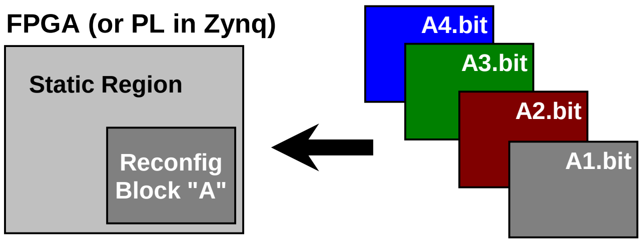


# DEVELOPMENT OF RUN-TIME HARDWARE ALTERABLE SYSTEMS

## TECHNOLOGIES

### DYNAMIC PARTIAL RECONFIGURATION (DPR)

- Dynamic Partial Reconfiguration (DPR) enables the run-time allocation and de-allocation of hardware resources by modifying or switching off portions of the FPGA (or PL inside the Zynq-7000) while the rest remains intact, continuing its operation.
- The operating design is modified by loading a partial bitstream configuration file. After a full bitstream configuration file configures the FPGA (Full Reconfiguration), partial bit files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured (this is called the static region). The figure illustrates the idea where the Block A (user-defined reconfigurable region) can be modified by any of the partial bit files (`A1.bit`, `A2.bit`, `A3.bit`, or `A4.bit`). The static region remains functioning and it is completely unaffected by the loading of a partial bit file. This is akin to multiplexing FPGA resources over time.



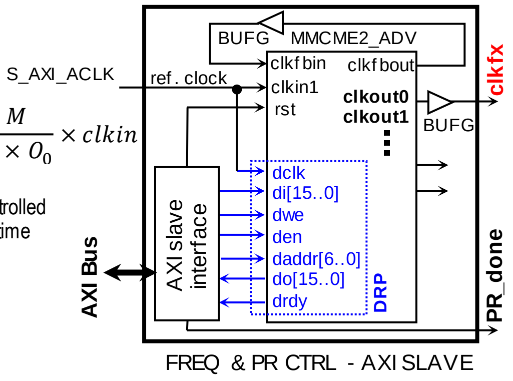
- This technology can dramatically extend the capabilities of FPGAs. In addition to potentially reducing size, weight, power, and cost, Dynamic Partial Reconfiguration enables new types of FPGA designs that provide efficiencies not attainable with conventional design techniques. The main FPGA vendors, ALTERA and Xilinx provide commercial support for this technology.
- **Xilinx devices**: The Reconfigurable Region can be dynamically reconfigured by writing on:
  - ✓ The Processor Configuration Access Port (PCAP) inside the PS. This is the preferred method for Zynq devices.
  - ✓ The Internal Configuration access port (ICAP) inside the PL. Here, an AXI interface is commonly built around the ICAP (e.g.: Xilinx Partial Reconfiguration Controller, custom-built controller) in order to easily write partial bitstreams to the ICAP; this method is less favored for Zynq devices, it can be useful in FPGAs with soft-core processors.
  - ✓ JTAG interface: This is manual configuration. Partial bitstreams are downloaded via the Vivado Hardware manager.

### DYNAMIC FREQUENCY CONTROL

- The mixed-mode clock managers (MMCM) inside the 7-Series FPGAs (Artix-7, Virtex-7, Zynq-7000 PL) provide a wide range of clock management features. (more info on *UG472: 7 Series FPGAs Clocking Resources - User Guide*)
- The Dynamic Reconfiguration Port (DRP) can adjust a clock frequency and phase at run-time without loading a new bitstream. In the figure, CLKFX is connected to one of several output clocks (clkout0). The frequency of CLKFX is controlled by M, D, and $O_0$. (more info on *XAPP888: MMCM and PLL Dynamic Reconfiguration*)

$$clkfx = \frac{M}{D \times O_0} \times clkin$$

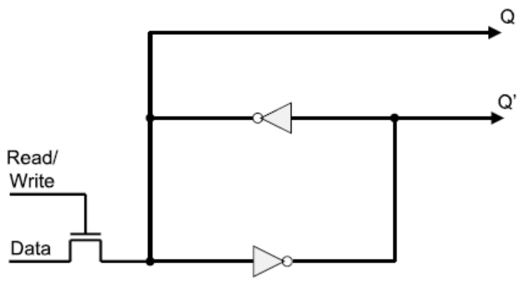
$O_0$ is controlled at run-time



- You can instantiate the Xilinx primitives `MMCME2_ADV` and `BUFG` (In Vivado: Project Manager → Language Templates → VHDL → Artix-7 → Clock Components). M and D are design parameters of `MMCME2_ADV`. The value of $O_0$ can be modified at run-time. This is how we can dynamically modify the frequency of CKFX.

### Technology that enables reconfiguration (full/partial) of FPGAs

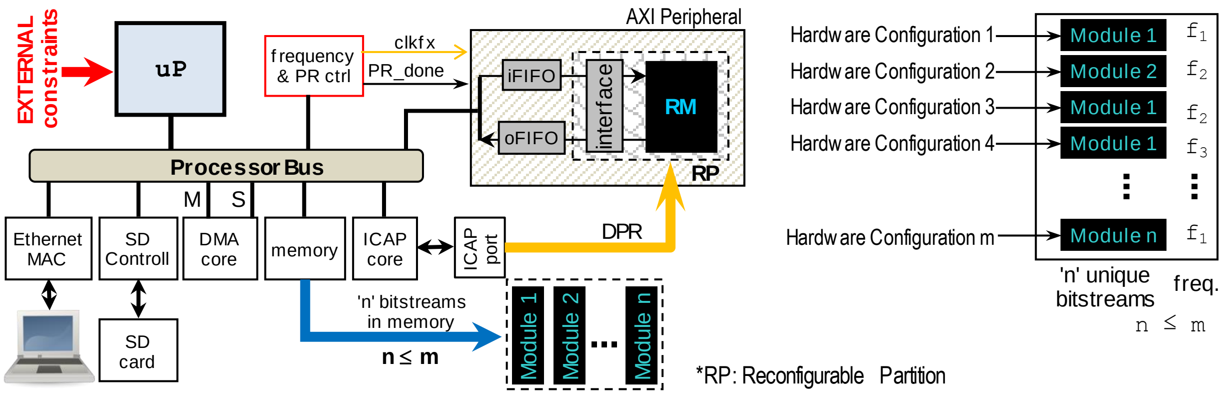
- Xilinx and ALTERA use a memory-based paradigm for computation of Boolean functions as well as for the realization of interconnections. Among the programmable technologies available, we can list SRAM, EEPROM, and Flash-based. SRAM devices, the dominant technology for FPGAs, are based on static CMOS memory technology, and are re-programmable and in-system programmable.
- In a SRAM-based FPGA, the states of the logic blocks, I/O blocks, and interconnections are controlled by the output of the SRAM cells. The basic SRAM configuration is constructed from two cross-coupled inverters and uses a standard CMOS processor. A new connection or function is implemented by a change on the SRAM cell values. Moreover, the device can be rapidly reconfigured in-circuit (when mounted on the circuit board with the other components) and on-the-fly (while the device is operating).



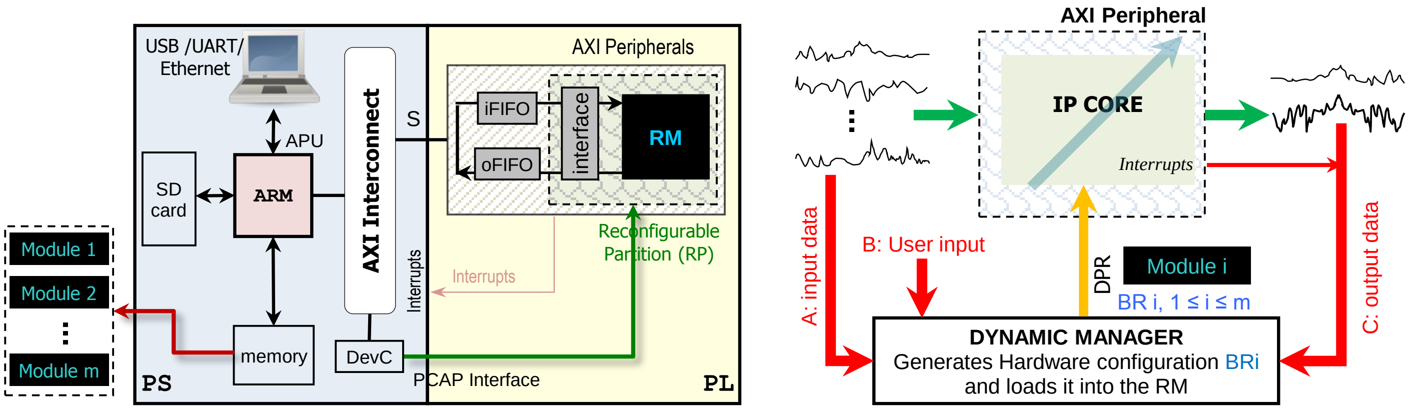
- A major disadvantage of SRAM programming technology is its large area. It takes at least five transistors to implement a SRAM cell, plus at least one transistor to serve as a programmable switch. Furthermore, the device is volatile, i.e., the configuration of the device stored in the SRAM cells is lost if the power is cut off. Thus, external storage or non-volatile devices such as EEPROMs, Flash devices are required to store the configuration and load it into the FPGA at power on.

## IMPLEMENTATION DETAILS

- **Reconfigurable Partition (RP)**: Region in the FPGA fabric (or PL fabric) that can be modified at run-time. It used to be called Partial Reconfigurable Region (PRR). There can be several RPs in a design.
- **Reconfigurable Module (RM)**: Variant for each reconfigurable Partition.
- The figure depicts an embedded All-Programmable SoC system that supports Dynamic Partial Reconfiguration (DPR) and Dynamic Frequency Control (DFC). It includes an AXI custom peripheral, which contains a Reconfigurable Partition (RP).
- In general, we can have several AXI custom-built peripherals, where each peripheral can have its own Reconfigurable Partition (RP). Moreover, within each peripheral, there can be several RPs.
- The figure depicts a generic dynamically reconfigurable embedded system that reconfigures via ICAP. Also, many peripherals (memory controller, DMA controller, Ethernet, SD controller) are part of the FPGA fabric. Here, we would have to instantiate every peripheral into the FPGA fabric. The microprocessor (uP) can be hard-wired (ARM, PowerPC) or soft-core (MicroBlaze).



- Zynq-7000 devices contain a PS unit that includes the ARM as well as many peripherals (including the PCAP Interface that has a dedicated DMA path). This is much simpler to handle as the designer only requires to deal with the software drivers (no need to instantiate the peripherals):



- The AXI Peripheral contains the proper interface to the AXI bus. In addition, in AXI4-Full there is an interface to the iFIFO and oFIFO. This interface is usually outside the Reconfigurable Partition (RP), but in general it can be inside it.
- Each hardware configuration is represented by a partial bitstream file and a frequency of operation. Every single hardware configuration has to be pre-computed prior to final system implementation.

- A **Dynamic Manager** (software routine running on the ARM inside the PS) provides input data, retrieves outputs from the AXI Peripherals, and deals with constraints (automatically generated or external) and interrupts. More importantly, it is in charge of swapping hardware configurations based on a particular set of rules.

## DPR ISSUES

- For proper DPR operation, we need to address two issues that arise due to DPR (especially when the interface to the FIFOs is inside the RP):
  - ✓ The RP outputs toggle during DPR and they might cause erratic behavior if the PRR outputs are directly connected to 'sensitive' signals (e.g.: AXI ready/valid signals, FIFO write enable). Thus, they need to be disabled during Partial Reconfiguration (they are usually AND'ed with 0).
  - ✓ The RP flip flops are not automatically reset after DPR (unlike in full reconfiguration). Depending on the circuitry, this might not be an issue; however, in most cases we must reset all the flip flops inside the Partial Reconfigurable Region after Partial Reconfiguration. One way to do it is by using a `PR_done` signal to be asserted (via software) after the DPR process is completed.

## TIME AND MEMORY OVERHEAD

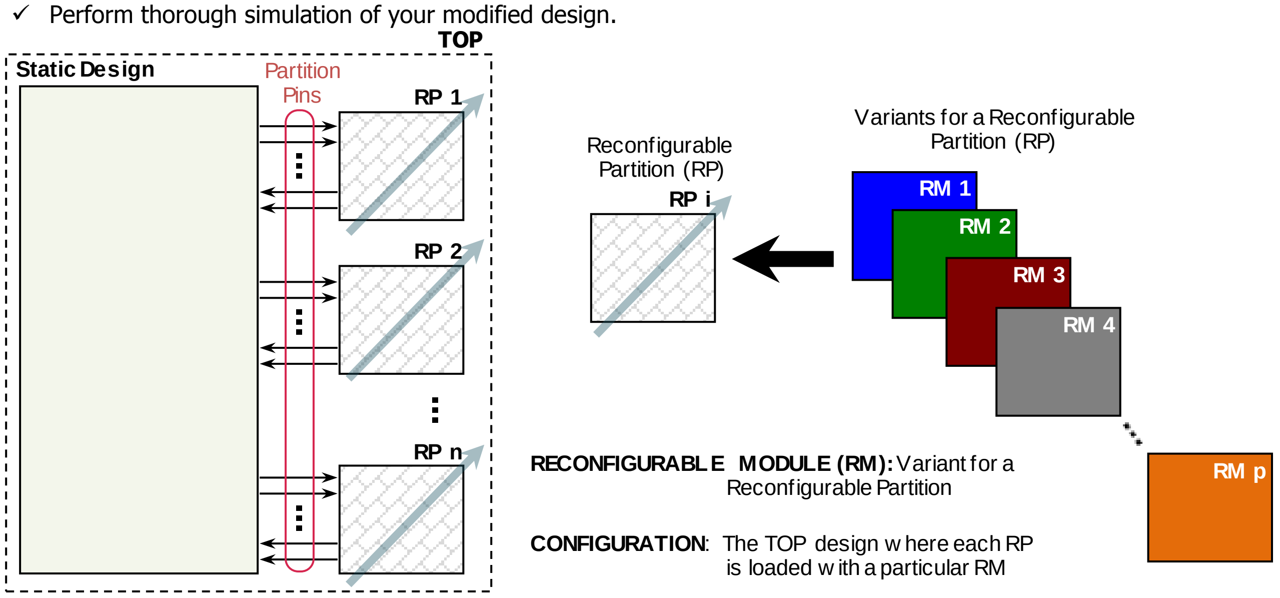
- **Reconfiguration Time Overhead**: This depends on the bitstream size, the design of the AXI interface design around the ICAP core, and the speed with which we can move data from memory to the ICAP core. Depending on the application, this overhead can be negligible or significant (reconfiguration speeds can range from KB/s to about 400 MB/s for a 100 MHz ICAP clock). In the case of the PCAP, the speed is more or less constant (~128 MB/s for a Zynq-700 device in the ZYBO Board); this high speed is achieved due to the use (by default) of DMA.
- **Memory Overhead**: The partial bitstream files are stored in memory. Depending on the application, the number of combinations can range from the MBs to the GBs and it can pose a significant challenge to the system design.

## GENERAL APPROACH FOR SELF-RECONFIGURABLE SYSTEMS

- **Definition of objective functions**: Energy, Power, Performance, Accuracy, bandwidth.
- **Definition of the Dynamic Regions (PRRs)**: This depends on the application. The more PRRs, the more complex the system becomes.
- **Development and parameterization of high-performance hardware architectures**: Here, we should explore techniques that optimize the amount of computational resources, exploit parallelism and pipelining.
- **Design Space Exploration of the multi-objective space**: Parameterization allows us to quickly generate a large set of different hardware profiles by varying the design parameters. This helps to explore trade-offs among design parameters and the objectives.
- (optional) **Multi-objective optimization**: Not all points in the design space are optimal; here, we get rid of sub-optimal points.
- **Dynamic management based on simultaneous multi-variable requirements**: The system receives stimuli in the form of multi-variable constraints and reconfigures itself via DPR and/or Dynamic Frequency Control to satisfy the multi-variable constraints.

## GENERAL PR DESIGN STEPS

- Partition your application into Software and Hardware Components.
- Hardware: partition your design between the Static Region and Reconfigurable Partitions (RPs).
  - ✓ Create a Vivado Project for your hardware. Design partition usually requires rearrangement of hardware description files so that Reconfigurable Partition(s) are exposed in the code.
    - For each Reconfigurable Partition, determine how many Reconfigurable Modules (RMs) you plan to have. The process of generating different variations (RMs) for a Reconfigurable Partition should be straightforward in your HDL description (e.g.: using generic parameters in VHDL).
  - ✓ In a Zynq-7000 PSoC, the Static Region (or Static Design) can include the PS.
  - ✓ Perform thorough simulation of your modified design.

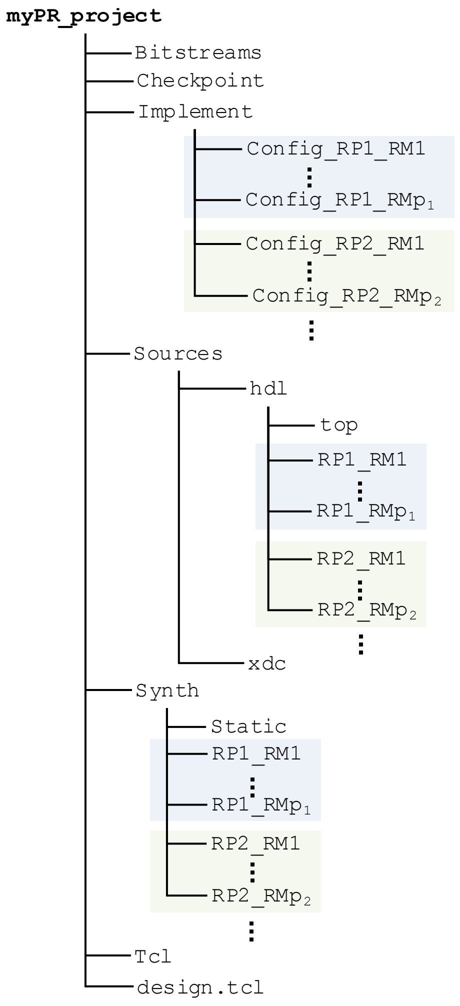

TOP
Static Design
Partition Pins
RP 1
RP 2
RP n
Reconfigurable Partition (RP)
RP i
Variants for a Reconfigurable Partition (RP)
RM 1
RM 2
RM 3
RM 4
RM p
RECONFIGURABLE MODULE (RM): Variant for a Reconfigurable Partition
CONFIGURATION: The TOP design where each RP is loaded with a particular RM

- Follow the Vivado$^{TM}$ Design flow for Partial Reconfiguration to generate: i) partial bitstreams (one per each RM variation for each RP), and ii) full bitstreams (one for an entire design with a unique RM for each RP). Two design flows are available:
  - ✓ GUI-based method
  - ✓ TCL-based non-project flow (we will cover this). See Embedded System Design for Zynq PSoC Tutorial → Unit 6 and 7.

- SDK (PS+PL Projects): Launch SDK for your full Vivado project that includes both the PS and the PL. The partial bitstreams must be accessible to the Zynq-7000 Dev. Board (e.g.: SD card, USB, Ethernet). You need to be able to write on the PCAP port. The PR examples provided here include software applications that you can use as templates for your design.

## TCL-BASED NON-PROJECT FLOW

- This flow uses TCL commands and scripts. Script files automate FPGA simulation, synthesis, implementation, including Partial Reconfiguration. CAD tools for most vendors (Xilinx, Intel, Mentor Graphics) support TCL scripts and commands.



### Design Process using Tcl scripts

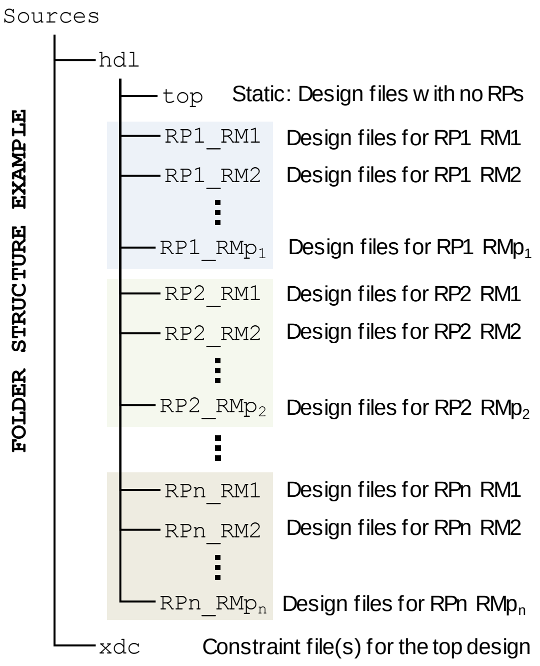
- We use Master scripts where the design sources, parameters, and structure are defined. They are available in the examples shown in this Unit:
  - ✓ `design_complete.tcl`: It compiles the entire design, from RTL to bitstreams (full, partial). Use it if the RP constraints (size, location) are known and merged with the I/O and clocking constraints in a `top.xdc` file.
  - ✓ `design.tcl`: If the RP constraints are unknown, this script synthesizes the modules of the design. After that, the rest of the process is to be completed with individual commands.
- The Master Tcl scripts define the design file hierarchy shown here.
- For your design, you need to edit the Master scripts to suit your needs.

### Design File Hierarchy for a PR (Partial Reconfiguration) Project

- This is shown in the figure for a project called 'myPR_project'. The directory structure underneath this top-level folder is described below. Note: a *checkpoint* is a snapshot of a design at any stage of the design process.
  - ✓ `\Bitstreams`: Empty folder, the target location for the bitstreams.
  - ✓ `\Implement`: This folder is the target location for checkpoints and reports for each of the design configurations. Implementation results for each configuration (each RM variation within each RP) are stored in a subfolder.
  - ✓ `\Sources`: Folder with the design sources (`.vhd, .v, .dcp, .xdc`, etc).
    - ▫ `\hdl`: Source code (VHDL, Verilog) is in this folder. There are folders for static logic (top) and for each RM variant.
    - ▫ `\xdc`: Constraints files, they are usually partitioned into: I/O and clocking constraints (`top_io.xdc`) and RP constraints (`pblocks.xdc`). Usually, the RP constraints are not known ahead of time.
  - ✓ `\Synth`: This folder contains empty folders that will store the post-synthesis checkpoints for all the modules of the design.
  - ✓ `\Tcl`: It contains all the lower-level Tcl scripts invoked by the Tcl scripts at the top level (`design_complete.tcl, design.tcl`).

### Design Steps

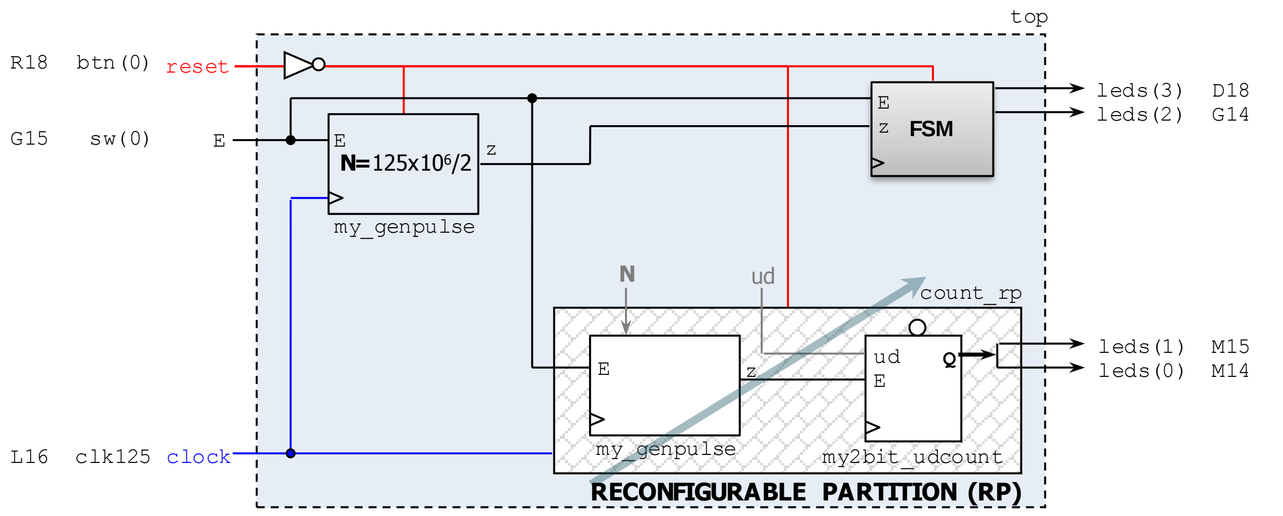
- Organize the design files for TCL-based PR flow in the `\Sources` folder. This folder structure is depicted in more detail in the figure.
  - ✓ Place the Static Design files (top-level logic without the RPs) on the `/hdl/top` folder, so that the top-level logic is synthesized with black boxes for the Reconfigurable Partitions (RPs).
  - ✓ Place the design files for each RM variation per RP in a unique folder. The design files are the same, but the top design file in the RP has different parameters for each RM.
  - ✓ Include constraint file(s) (`.xdc`): I/O and clocking constraints, as well as PR size constraints (if available beforehand).
- Synthesize the Static and Reconfigurable Modules separately.
- Generate a first complete design (Static + a Reconfigurable Module per RP) called *First Configuration*:
  - ✓ Load the Static design. Add in the respective RMs to the RPs for this *First Configuration*.
  - ✓ Create or load physical constraints (Pblocks) for the RPs.
  - ✓ Implement (Place & Route) the design. Save the fully routed design of this *First Configuration* as a checkpoint.
- Remove the RMs from this *First Configuration.* Lock the static placement and routing. Save this *Static-only design* checkpoint.
- Implement all the other configurations (all RM variations per RP):
  - ✓ Add new RMs to the *Static-only design*. Implement this new configuration and save it as a checkpoint.
  - ✓ To implement new configurations, you need to remove the RMs of the current configuration. You can also close the current project and load the *Static-only design* checkpoint.
  - ✓ Run a verification utility (`pr_verify`) on all *Configurations*.
  - ✓ Create bitstreams (full and partial) for each *Configuration* (including a *Configuration* with a black-box for each RPs).



## PR EXAMPLES

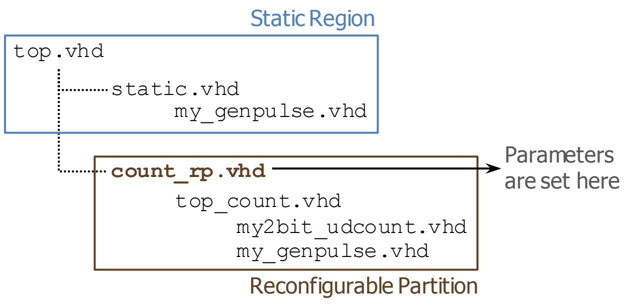
### CASE EXAMPLE: SIMPLE LEDS (ONLY PL)

- Step-by-step instructions on how to implement this project are available at Embedded System Design for Zynq PSoC Tutorial → Unit 6. The folder structure (and design files) for this 4-bit LED Pattern Controller (1 RP) is available here. This circuit has one Reconfigurable Partition (RP).
- **RP output toggling**: The outputs are connected to LEDs, so this is nonissue.
- **Clearing FFs inside the RP after DPR**: Since is a visual application, this is not a problem. In any case, we can always reset the RP manually (via the external *reset* input).

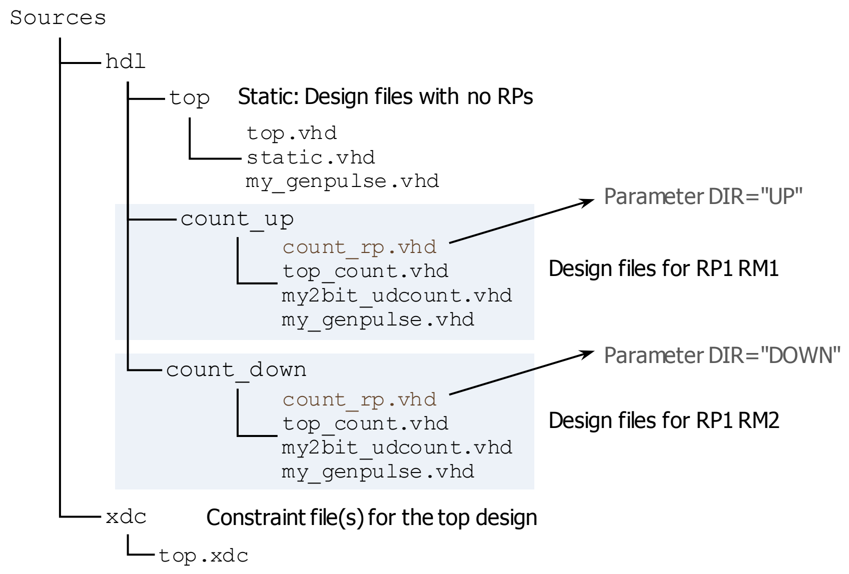


**Hardware Partitioning: Static Region and Reconfigurable Partitions**:

- The figure depicts the organization of the design files when the hardware has been partitioned into the Static Region and the Reconfigurable Partition. This hardware-only project has been thoroughly synthesized and simulated.
  - ✓ Reconfigurable Partition (RP): The file count_rp.vhd is the top file of the Reconfigurable Partition (RP). In this file, the parameters of the RP are indicated, and we can quickly modify them in order to create a RM variant. There are 2 parameters in this design (COUNT, DIR) that allow us to create a large set of RM variants.

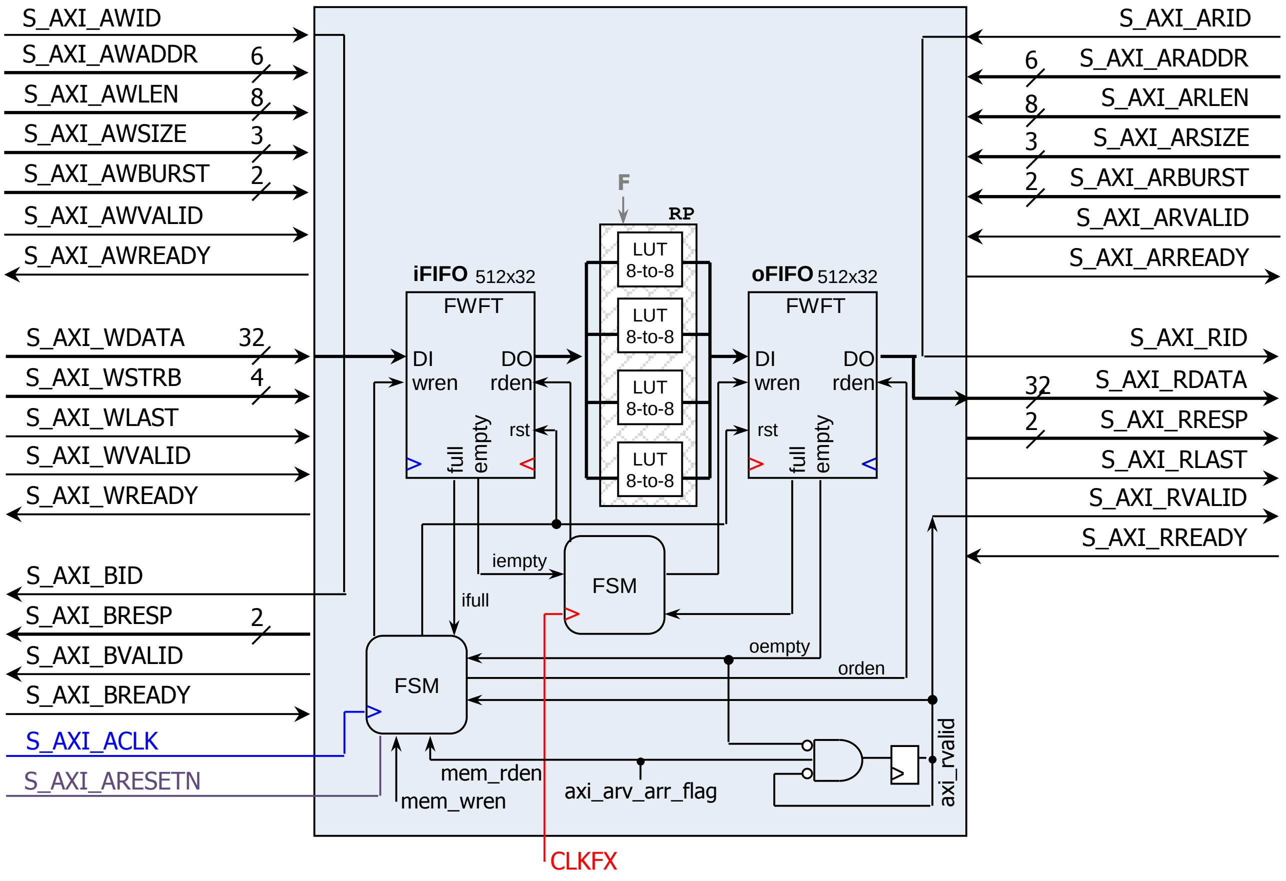


- After we verify the proper functioning of our partitioned design, we create the `/Sources` folder structure for the TCL-based flow for Partial Reconfiguration. In this PR example, we have 1 RP with 2 RM variants that are created by modifying the parameter DIR (RM 1: DIR=UP, RM 2: DIR=DOWN). The figure depicts the `/Sources` directory structure.



## CASE EXAMPLE: PIXEL PROCESSOR (PS+PL)

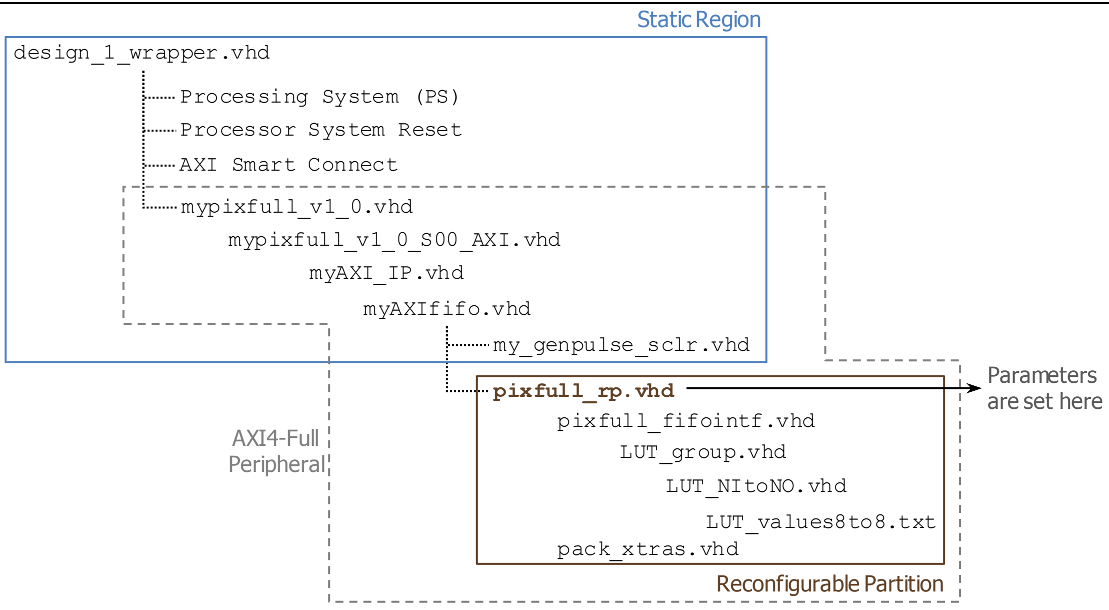
- Step-by-step instructions on how to implement this project are available at Embedded System Design for Zynq PSoC Tutorial → Unit 7. The folder structure (and design files) for this project is available here. The design includes one Reconfigurable Partition (RP).
  - ✓ **Reconfigurable Partition (RP)**: It consists of 4 LUTs 8to8. We can create different hardware configurations by modifying the parameter F (1..5) of the Pixel Processor design. We fix NC=4, NI=NO=8.
  - ✓ **Static Region**: It consists of all the hardware outside the 4 LUTs 8to8. The portion in light blue is the static portion in the AXI4-Full Peripheral. However, note that the Processing System (PS) and any extra hardware (e.g.: Processor System Reset, AXI Bus) is also considered part of the static region. This poses a challenge to the Tcl-based non-project flow.
- **RP output toggling**: The FIFO structure avoids PR toggling as the PR outputs are only connected to the FIFO data input.
- **Clearing FFs inside the RP after DPR**: The RP does not have FFs, so we do not face this issue here.



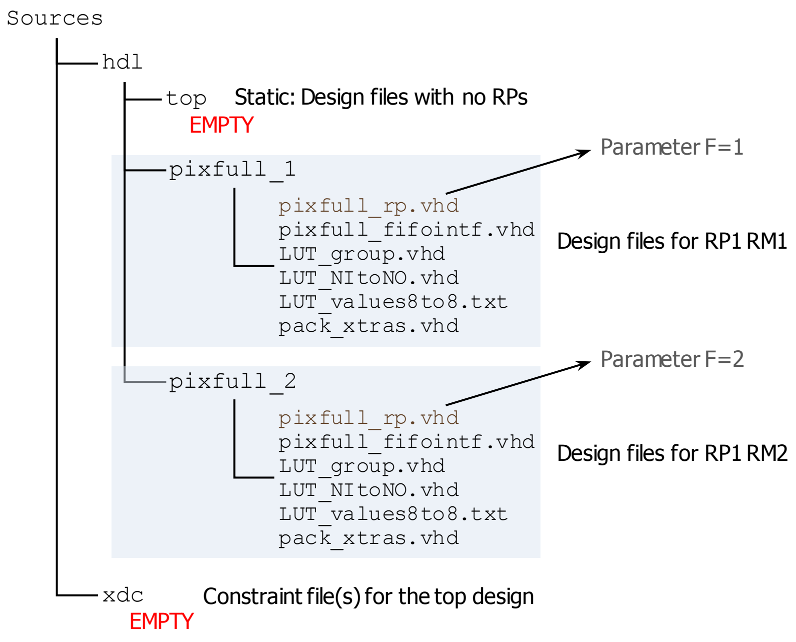
**Hardware Partitioning: Static Region and Reconfigurable Partitions**:

- We depict the design structure of the Embedded System that has been partitioned into the Static Region and the Reconfigurable Partition. This hardware/software project has been tested (via SDK) as a non-PR project. Once we complete the PR implementation, we will use this hardware/software project to test the PR Project.
- The AXI4-Full Pixel Processor peripheral is depicted in this design structure. Note that it contains the Reconfigurable Partition (RP) and some Static logic. Before we tested the hardware/software system, this hardware-only project was thoroughly synthesized and simulated.
  - ✓ AXI4-Full Pixel Processor Peripheral: Note that the file arrangement has been modified (with respect to the AXI4-Full Pixel Processor peripheral in Embedded System Design for Zynq PSoC Tutorial → Unit 4) in order to expose the Reconfigurable Partition (RP).
  - ✓ Reconfigurable Partition (RP): The file pixfull_rp.vhd is the top file of the RP. In this file, the parameters of the RP are indicated, and we can quickly modify them in order to create a RM variant. There is only one modifiable parameter (to keep the hardware interfacing as depicted here): F (1, 2, 3, 4, 5) that allow us to create up to 5 RM variants.

- Static Region: Since the PS (and extra designs) is included here, the VHDL files of the static portion of the AXI4-Full Peripheral are not sufficient.
  - ✓ We will create instead an embedded system for the AXI4-Full Pixel Processor Peripheral that does not include the Reconfigurable Partition (RP). We will synthesize this project (the RP will be a black box) and extract the synthesized checkpoint (`.dcp` file) from the embedded synthesis. This file will be placed in the PR Project folder `\Synth\Static`.



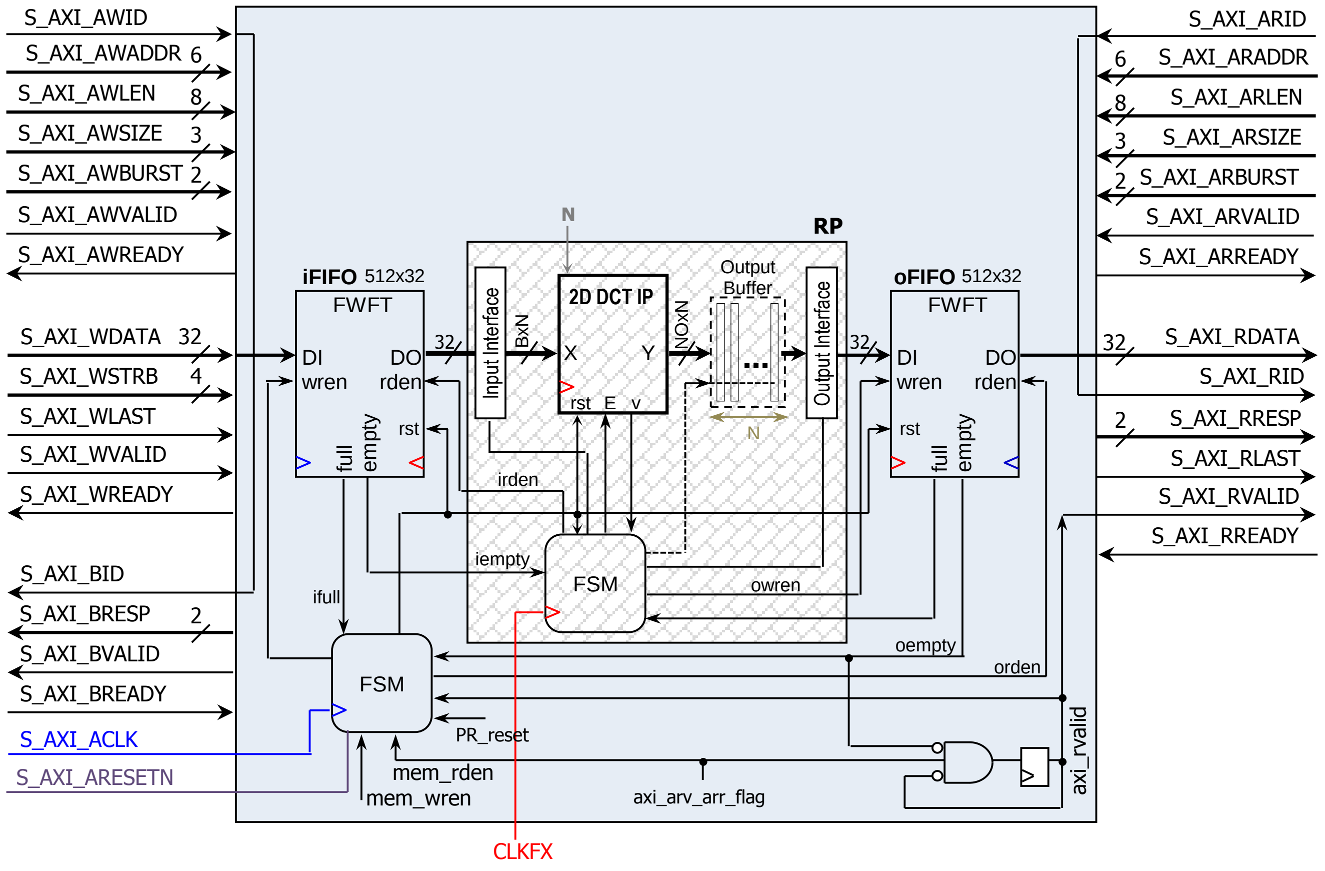
- After we verify the proper functioning of our partitioned design (as a non-PR project), we create the `/Sources` folder structure for the TCL-based flow for Partial Reconfiguration. In this PR example, we have 1 RP and we use only 2 RM variants that are created by modifying the parameter F (RM 1: F= 1, RM 2: F=2). The figure depicts the `/Sources` directory structure.
  - ✓ Note that the `\Sources\hdl\top` folder is empty. The Master Tcl scripts will not compile this folder as the Synthesized file (`.dcp`) is already available in `\Synth\Static`.
  - ✓ The `\Sources\xdc` folder is also empty. There are no I/Os from the PL. However, we still need to indicate the RP constraints (this will be done during the PR implementation of this design).



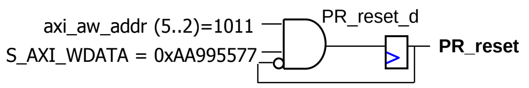
## CASE EXAMPLE: 2D DCT (PS+PL)

- The VHDL code of this IP is available at Tutorial: Embedded System Design for Zynq SoC - Unit 7.
  - ✓ **Reconfigurable Partition (RP)**: In this project, we allow the RP to vary one parameter (N: DCT Transform Size: 4, 8, 16), while we fix the parameters B=8, NO=16, NH=16. If the Transform size changes, so do the input interface, the output buffer, the output interface and the FSM @ CLKFX. This is why all these components are part of the RP (including the 2D DCT IP).
  - ✓ **Static Region**: It consists of all the hardware outside the RP. The portion in light blue is the static portion in the AXI4-Full Peripheral (here, this is the circuits working @ CLKFX). Any extra hardware (e.g.: Processor System Reset, Processing System) is considered part of the static region.

- **RP output toggling**: The signal *owren* can modify the oFIFO contents. So, we need to reset the FIFOs after DPR.
- **Clearing FFs inside the RP after DPR**: The RP includes FFs, including those of the FSM @ CLKFX. We need to clear all the FFs after DPR, especially to place the FSM @CLK_FX into the initial state after DPR.



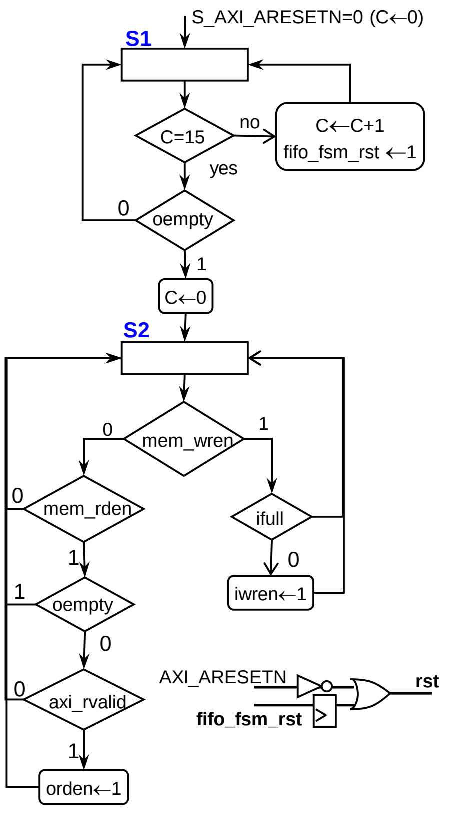
- *PR_reset*: This signal resets both the RP FFs and the FIFOs via a simple software command (we write the word 0xAA995577 onto address 101100). Make sure than when writing to the peripheral, we avoid the address 101100.
  - ✓ *PR_reset*: This is the output of a flip flop. This signal is a pulse of one clock cycle. Every time axi_awaddr (latched S_AXI_AWADDR) and S_AXI_WDATA match what we want, we generate a pulse.



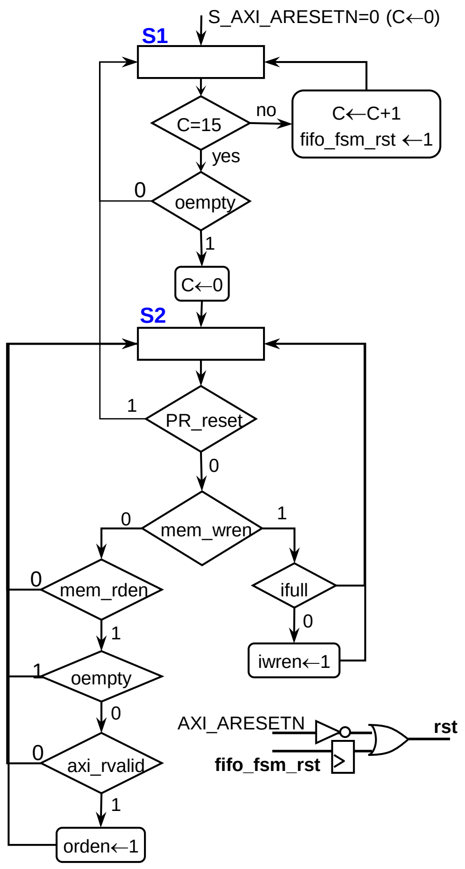
  - ✓ Notice that once S_AXI_WDATA is captured by AXI, the latched address axi_awaddr might increase its value by 4 (or changes). Or S_AXI_WDATA will not be the valid value anymore. This usually makes sure that *PR_reset* is only one pulse (and not a sequence of pulses generated in case that axi_awaddr and S_AXI_WDATA hold their values). To be absolute sure, include the condition axi_wready=axi_wvalid=1 when asserting *PR_reset*.

- Note that in Vivado 2016.2, we can use the RESET_AFTER_RECONFIGURATION property to reset all the flip flops inside the PR (at the expense of extra constraints on the RP shape). However, this will not reset the FIFOs, as they are not part of the RP. If we do not reset FIFO, the circuit might not work after DPR.

- The original FSM @ S_AXI_ACLK was used for the pixel processor and the 2D DCT. This FSM can be used for any circuit that uses the iFIFO/oFIFO structure (as mentioned in Notes – Unit 5). This circuit is shown below (on the left).
- However, if we want to carry out Dynamic Partial Reconfiguration, we need to reset both the FIFOs and the RP using the signal $PR\_reset$. To reset the FIFOs, we need to assert the signal 'rst' again. The new FSM @ S_AXI_ACLK account for this and it is shown below (on the right).



**FSM at S_AXI_ACLK**



**FSM at S_AXI_ACLK**

- Note that you can use this circuit as a template to build any AXI4-Full Peripheral that supports DPR. The AXI4-Full Peripheral should like 2D DCT, where the RP includes: input and output interface to FIFOs, FSM @ CLKFX (to interface to FIFOs and the IP core), and the IP core (in this case the 2D DCT). These are the only components that we need to modify. The components running @ S_AXI_ACLK do not need to be modified.
  - ✓ Output buffer of the 2D DCT: This is considered part of Output Interface to oFIFO. In the figure, we chose to display it independently.