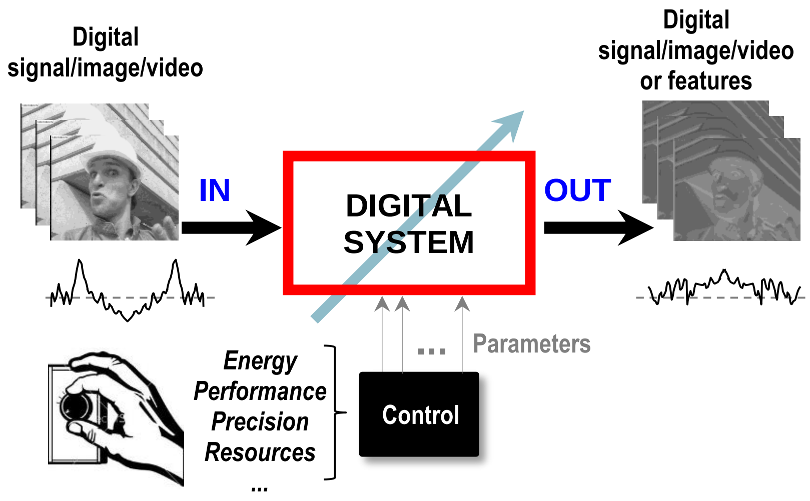
Notes - Unit 6

DYNAMIC PARTIAL RECONFIGURATION

INTRODUCTION TO SELF-RECONFIGURABLE SYSTEMS

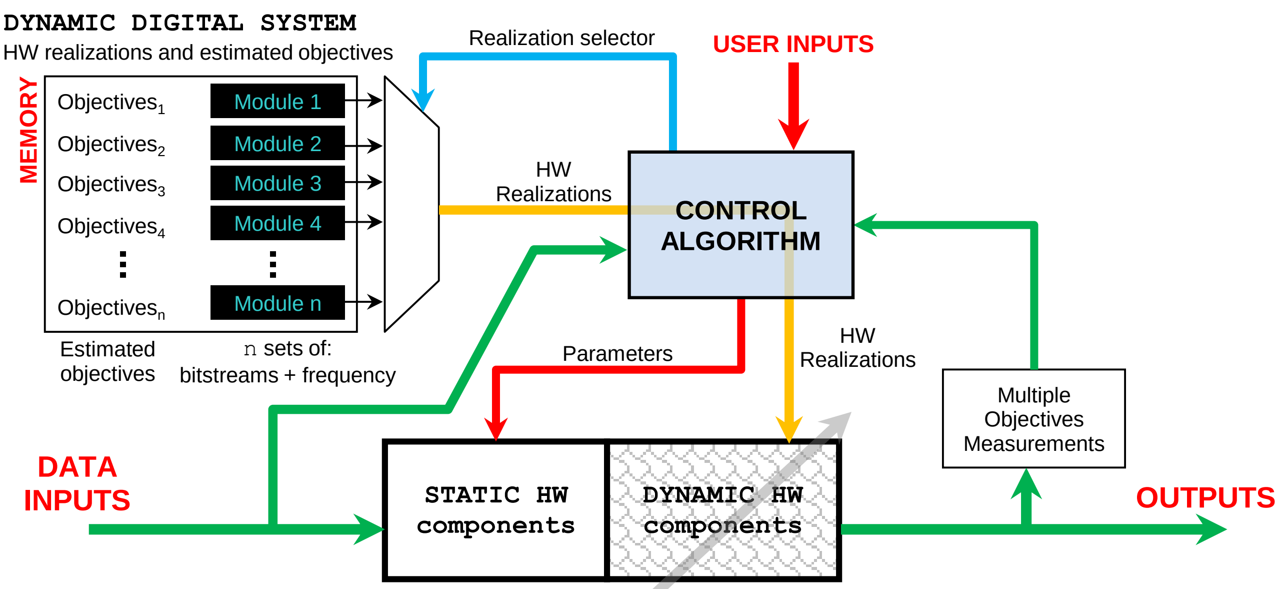
MOTIVATION

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

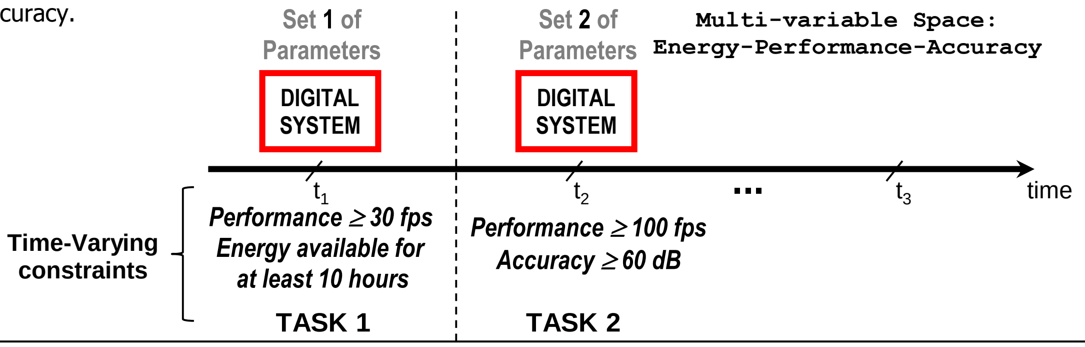


- **Dynamic Reconfigurable Computing Management**: The ability to control the aforementioned properties at run-time. We can deliver a dynamically self-adaptive system (by dynamic allocation of resources and dynamic frequency control) that satisfies time-varying simultaneous requirements.

- **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.

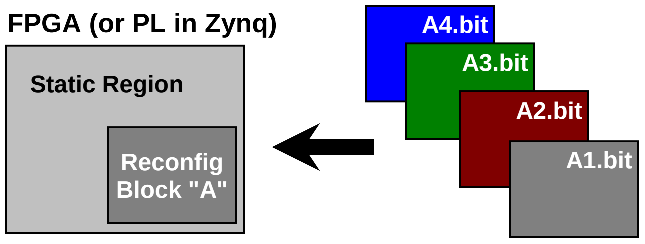


DRP IMPLEMENTATION: RECONFIGURATION CONTROLLER, GENERATION OF PARTIAL BITSTREAMS

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 Programmable Logic 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.



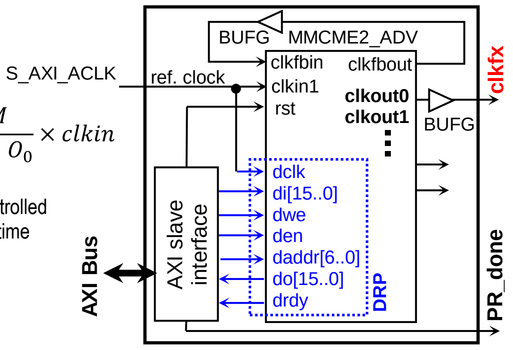
- This technology can dramatically extend the capabilities of FPGAs. In addition to potentially reducing size, weight, power, and cost, Dynamic Partial Reconfiguration enables new types of FPGA designs that provide efficiencies not attainable with conventional design techniques. The main FPGA vendors, 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.

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*)
- 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 clkfx.

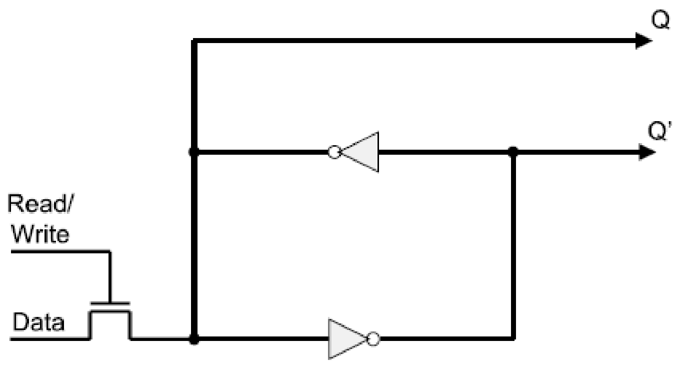
$$clkfx = \frac{M}{D \times O_0} \times clkin$$

O_0 is controlled at run-time



FREQ & PR CTRL - AXI SLAVE

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.

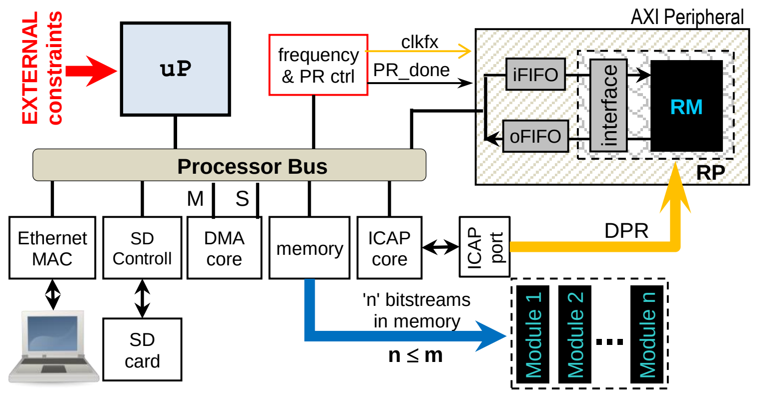
IMPLEMENTATION DETAILS

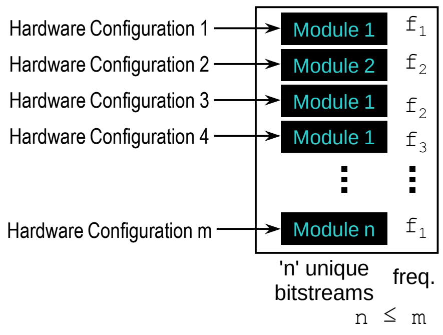
- Reconfigurable Partition (RP). Region in the FPGA fabric (or PL fabric) that can be modified at run-time. It used to be called Partial Reconfigurable Region (PRR). There can be several RPs in a design.

- The figure depicts an embedded All-Programmable SoC system that supports Dynamic Partial Reconfiguration (DPR) and Dynamic Frequency Control (DFC). The system contains one AXI custom-built 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.

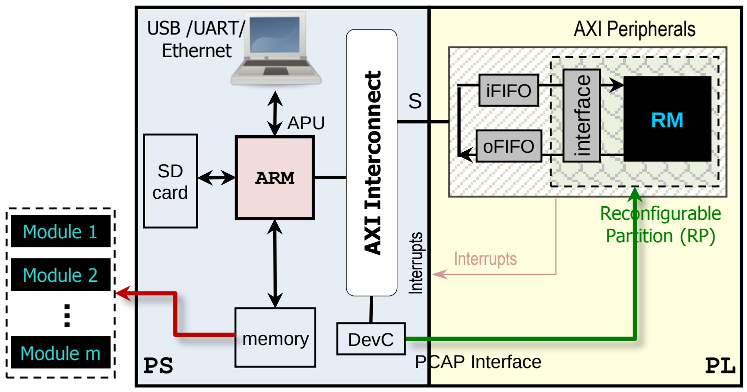
- The following figure depicts a generic dynamically reconfigurable embedded system where we can reconfigure using the ICAP. Also, many peripherals (memory controller, DMA controller, Ethernet, SD controller) are part of the FPGA fabric. Here, we would have to instantiate every peripheral into the FPGA fabric. The microprocessor (uP) can be hard-wired (ARM, PowerPC) or soft-core (MicroBlaze).

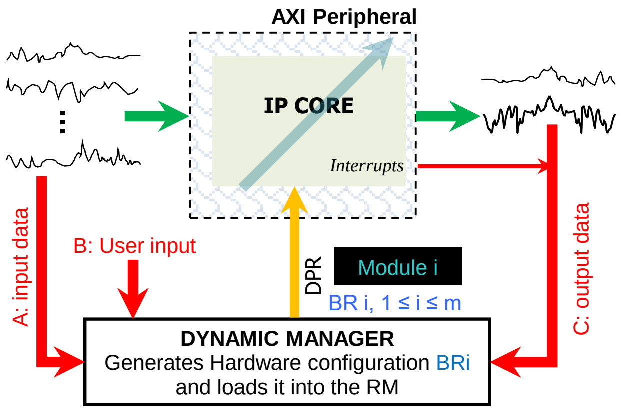




*RP: Reconfigurable Partition

- 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 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), or we reset the
  - ✓ The RP flip flops are not automatically reset after DPR (unlike in full reconfiguration). Depending on the circuitry, this is not usually 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.

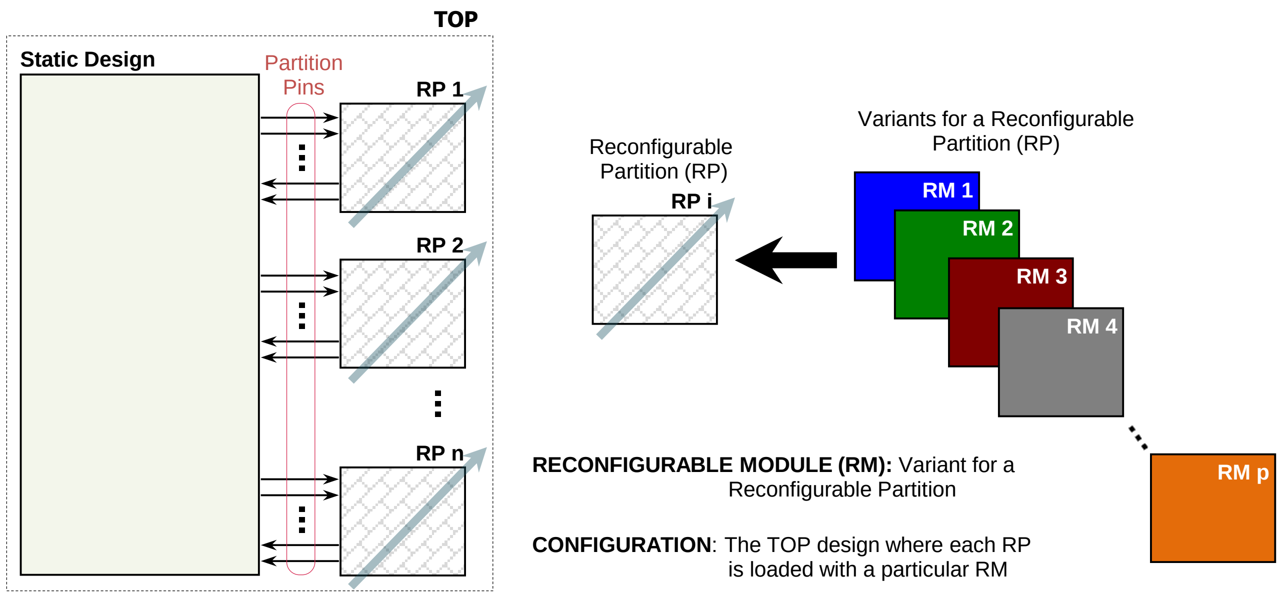
TIME AND MEMORY OVERHEAD
- Reconfiguration Time Overhead: This depends on the bitstream size, the design of the AXI interface design around the ICAP core, and the speed with which we can move data from memory to the ICAP core. Depending on the application, this overhead can be negligible or significant (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.

XILINX VIVADO™ SOFTWARE FLOW

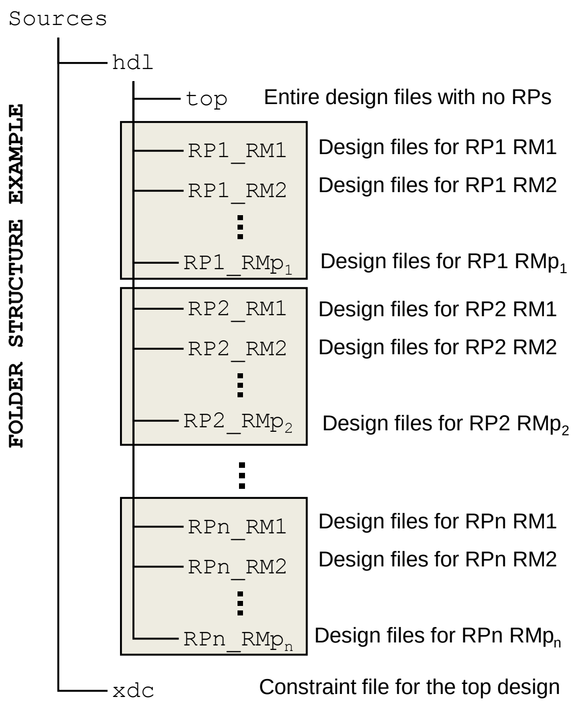

TOP
Static Design
Partition Pins
RP 1
RP 2
RP n
Reconfigurable Partition (RP)
RP i
Variants for a Reconfigurable Partition (RP)
RM 1
RM 2
RM 3
RM 4
RM p

RECONFIGURABLE MODULE (RM): Variant for a Reconfigurable Partition

CONFIGURATION: The TOP design where each RP is loaded with a particular RM

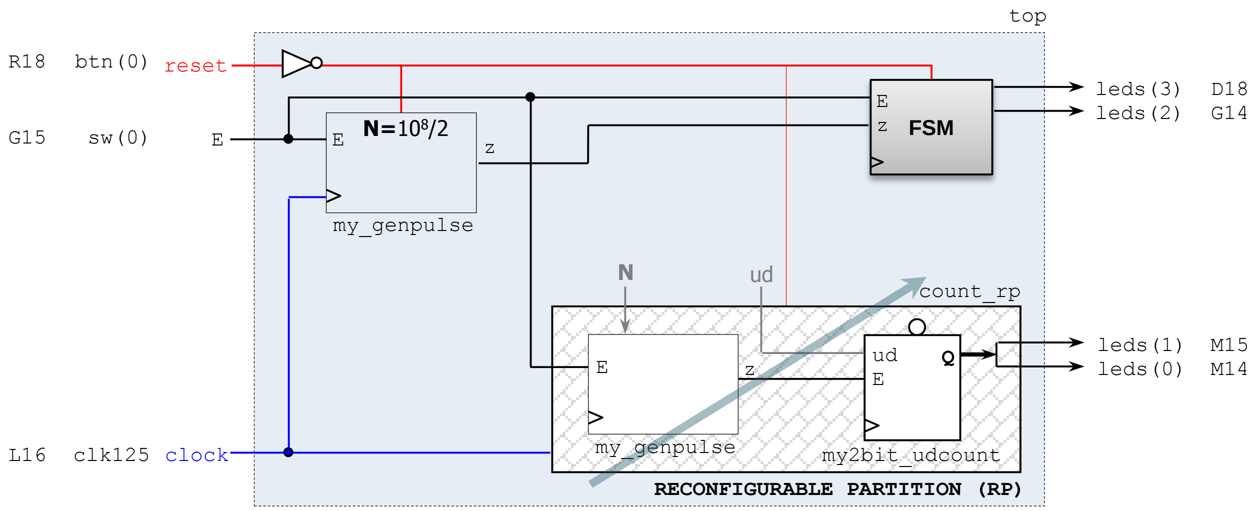
These steps summarize processing a Partial Reconfigurable Design:

- Follow Bottom-Up Synthesis for your VHDL design: Place top-level logic without the RPs on a specific folder, so that the top-level logic is synthesized with black boxes for Partitions (this is the Static Design). Have a separate folder for each Reconfigurable Partition (RP) and for each Reconfigurable Module inside a RP. Also, include the top level constraint file (.xdc). The figure shows an example:
- Synthesize the static and Reconfigurable Modules separately.
- Create physical constraints (Pblocks) to define the RPs.
- Set the HD.RECONFIGURABLE property on each RP.
- Implement a complete design (static and one Reconfigurable Module per Reconfigurable Partition). This is a *Configuration*.
- Save a design checkpoint for the full routed design.
- Remove Reconfigurable Modules from this *Configuration* and save a static-only design checkpoint.
- Lock the static placement and routing.
- Add new RMs to the static design and implement this new configuration, saving a checkpoint for the full routed design.
- Repeat the previous step until all Reconfigurable Modules per each RP are implemented as unique *Configurations*.
- 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).



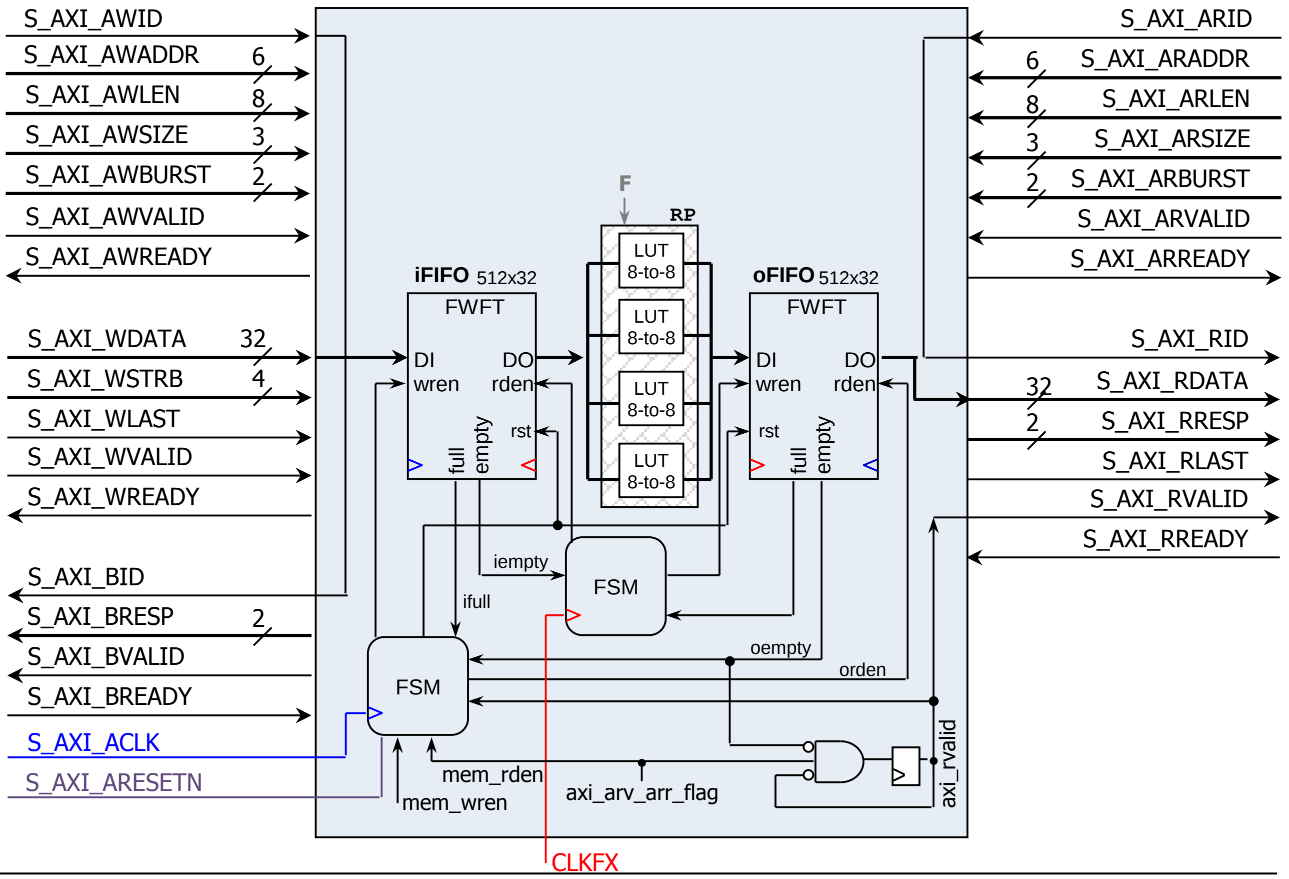
CASE EXAMPLE: SIMPLE LEDS (ONLY PL)

- The description (and code) of this circuit is available in Tutorial: Embedded System Design for Zynq SoC - Unit 6. It includes 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).



CASE EXAMPLE: PIXEL PROCESSOR (PS+PL)

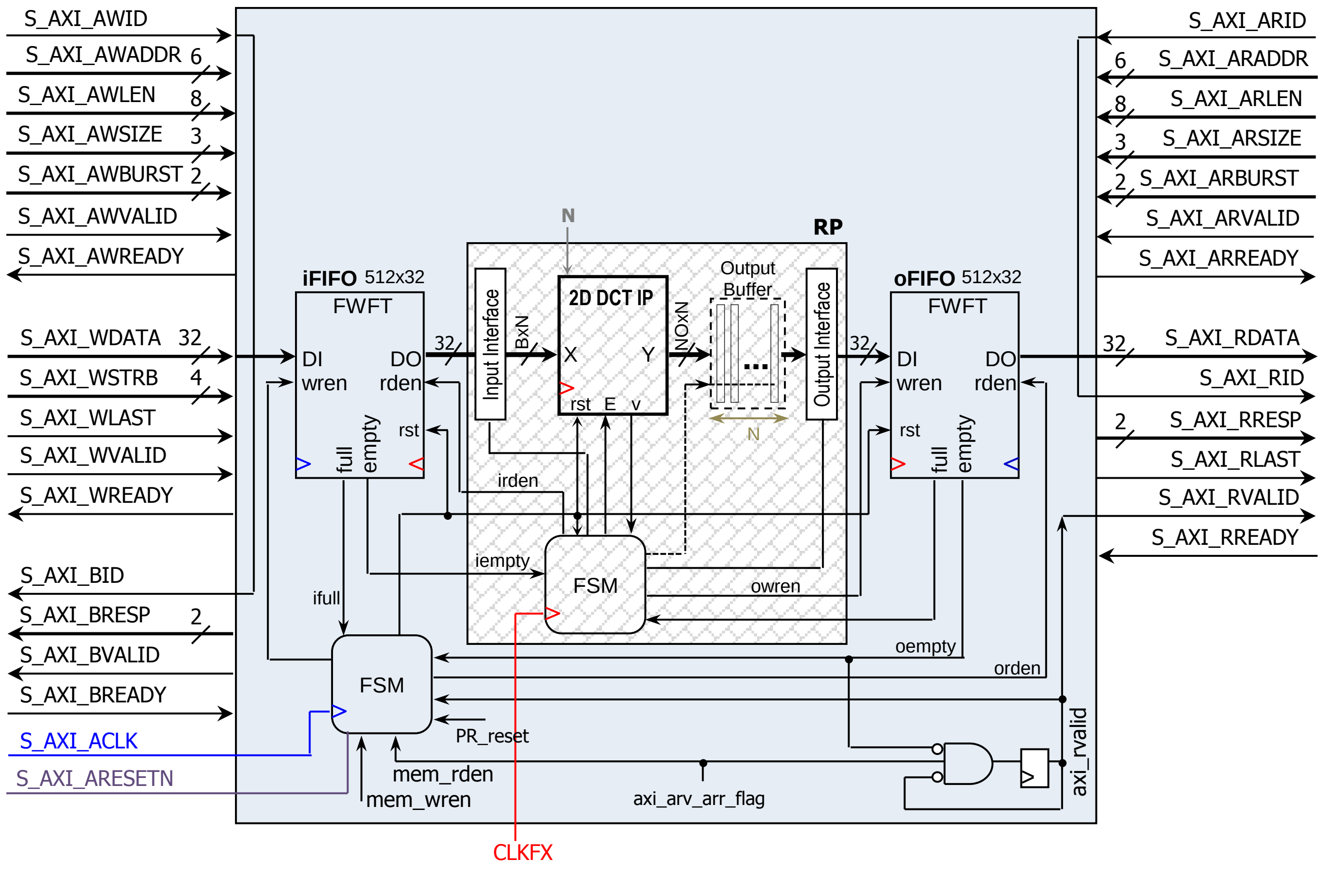
- The VHDL code of this IP is available at Tutorial: Embedded System Design for Zynq SoC - Unit 7.
  - ✓ **Reconfigurable Partition (RP)**: It consists of 4 LUTs 8to8. We can create different hardware configurations by modifying the parameter F (1..5). We fix NC=4, NI=NO=8.
  - ✓ **Static Region**: It consists of all the hardware outside the 4 LUTs 8to8. The portion in light blue is the static portion in the AXI4-Full Peripheral. Any extra hardware (e.g.: Processor System Reset, Processing System) is considered part of the static region.
- **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.



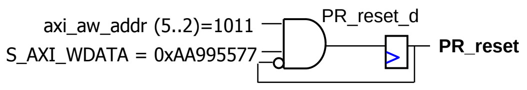
CASE EXAMPLE: 2D DCT (PS+PL)

- The VHDL code of this IP is available at Tutorial: Embedded System Design for Zynq SoC - Unit 7.
  - ✓ **Reconfigurable Partition (RP)**: We can create different hardware configurations by varying the parameter N (DCT Transform Size: 4, 8, 16). We fix 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.



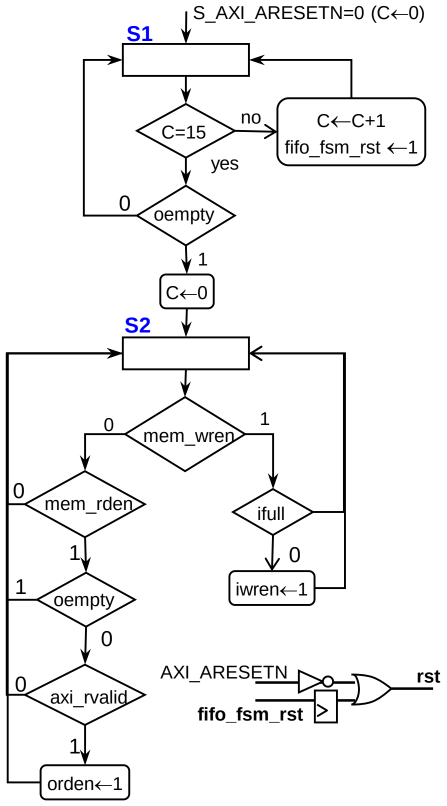
- PR_reset: This signal resets both the RP FFs and the FIFOs via a simple software command (we write the word 0xAA995577 onto address 101100). Make sure than when writing to the peripheral, we avoid the address 101100.
  - ✓ PR_reset: This is the output of a flip flop. This signal is a pulse of one clock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



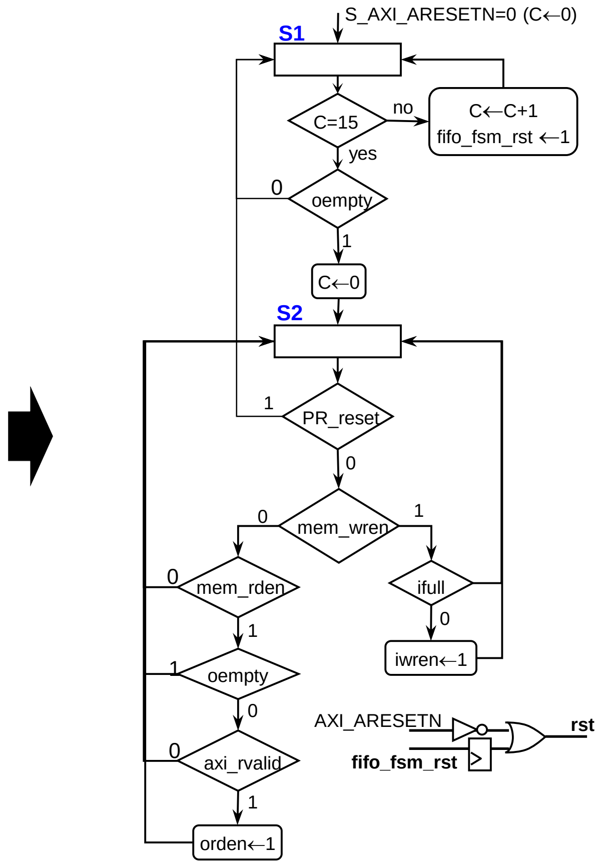
  - ✓ Notice that once S_AXI_WDATA is captured by AXI, the latched address axi_awaddr might increase its value by 4 (or changes). Or S_AXI_WDATA will not be the valid value anymore. This usually makes sure that PR_reset is only one pulse (and not a sequence of pulses generated in case that axi_awaddr and 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.