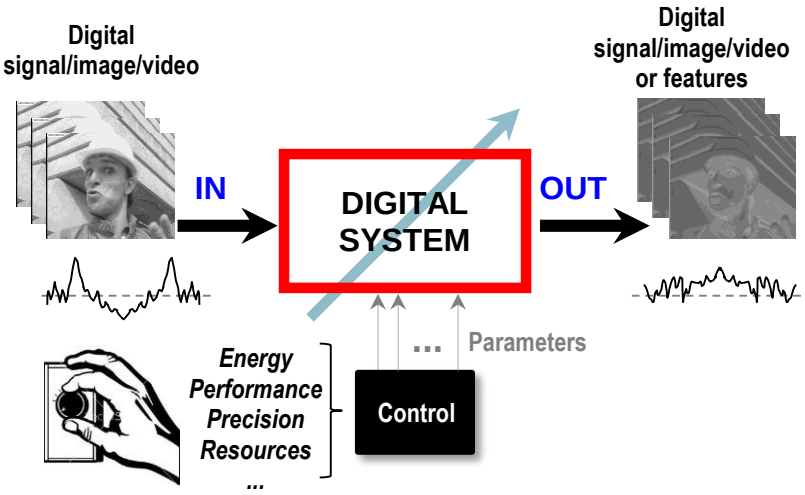
Notes - Unit 6

DYNAMIC PARTIAL RECONFIGURATION

INTRODUCTION TO SELF-RECONFIGURABLE SYSTEMS

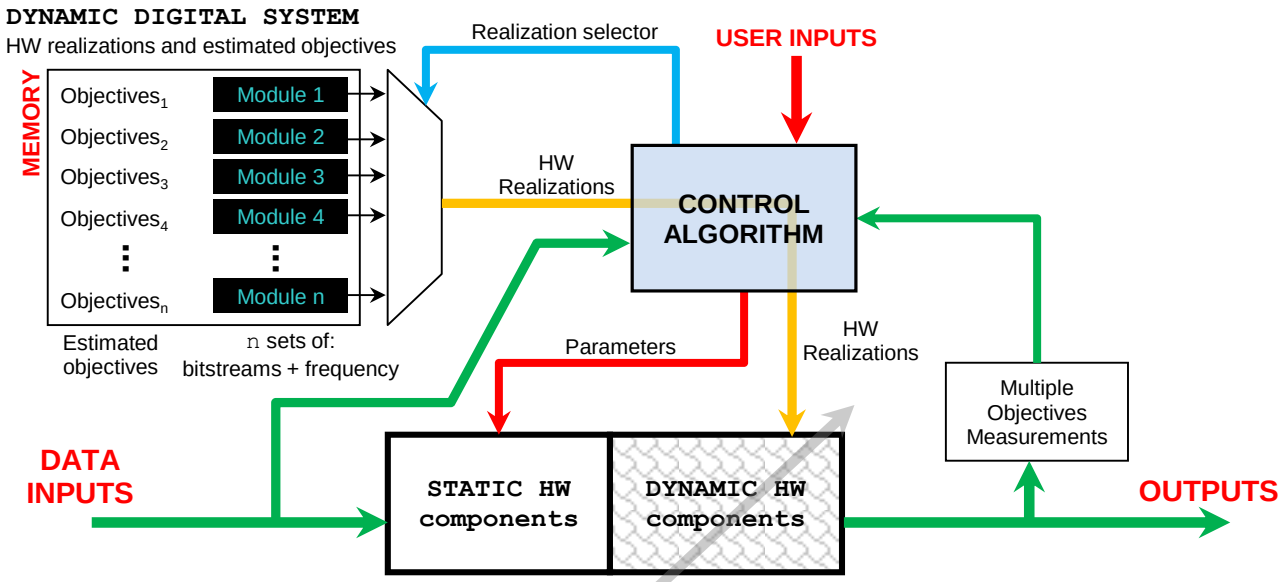
MOTIVATION

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

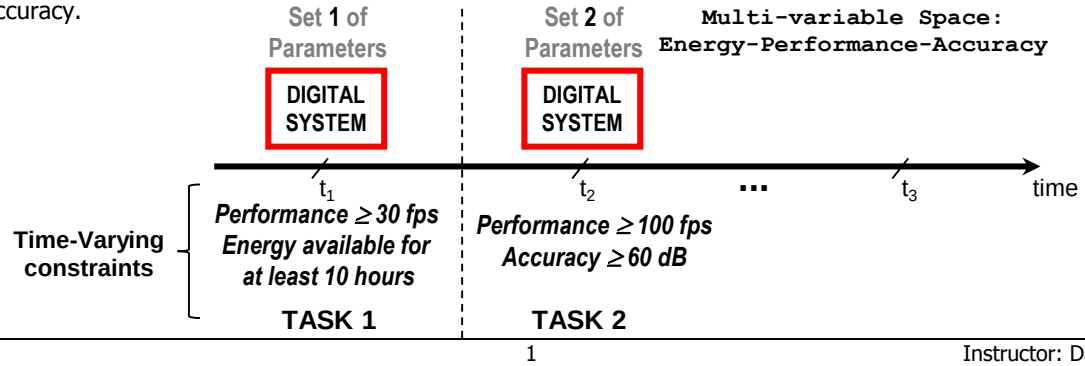


- **Dynamic Reconfigurable Computing Management**: The ability to control the aforementioned properties at run-time. We can deliver a dynamically self-adaptive system (by dynamic allocation of resources and dynamic frequency control) that satisfies time-varying simultaneous requirements.

- **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.

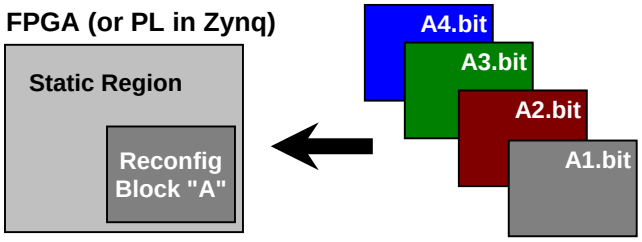


DRP IMPLEMENTATION: RECONFIGURATION CONTROLLER, GENERATION OF PARTIAL BITSTREAMS

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.



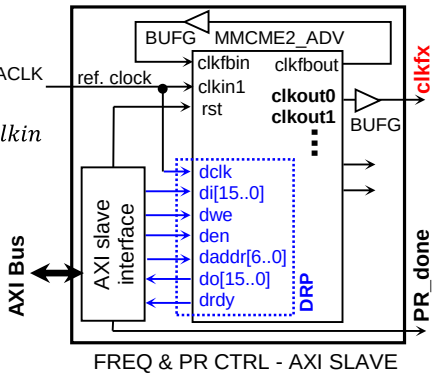
- This technology can dramatically extend the capabilities of FPGAs. In addition to potentially reducing size, weight, power, and cost, Dynamic Partial Reconfiguration enables new types of FPGA designs that provide efficiencies not attainable with conventional design techniques. The main FPGA vendors, ALTERA and Xilinx provide commercial support for this technology.
- **Xilinx devices**: The PRR can be dynamically reconfigured by writing on the internal configuration access port (ICAP). 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.

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

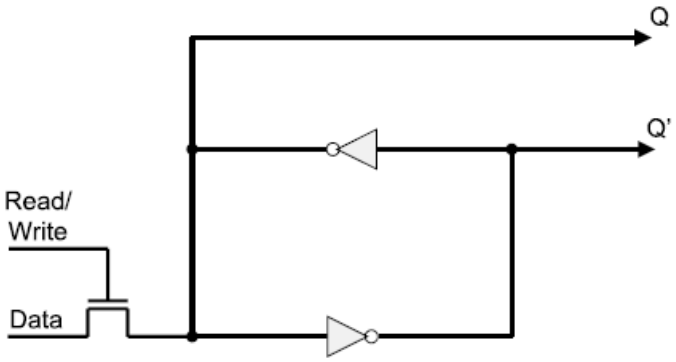


FREQ & PR CTRL - AXI SLAVE

DPR ISSUES

- For proper DPR operation, we need to address two issues that arise during DPR (especially when the interface to the FIFOs is inside the PRR):
  - ✓ The PRR 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). Thus, they need to be disabled (usually they are AND'ed with 0) during Partial Reconfiguration.
  - ✓ The PRR flip flops are not automatically reset after DPR (unlike in full reconfiguration). Depending on the circuitry, this is not usually an issue; however, it is a good practice to 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 after the DPR process is completed.

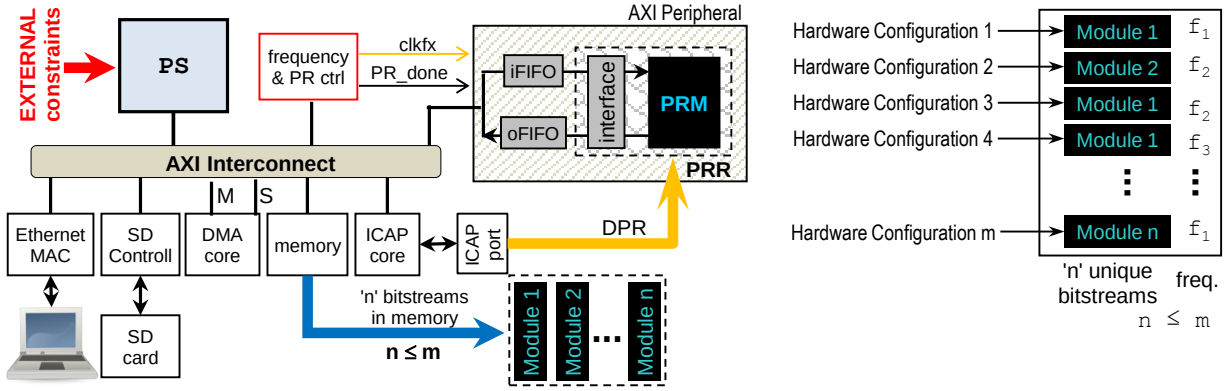
Technology that enables reconfiguration (full/partial) of FPGAs



- 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

- Partial Reconfigurable Region (PRR): Region in the FPGA fabric (or PL fabric) that can be modified at run-time. There can be several PRRs 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 Partial Reconfigurable Region (PRR).
- In general, we can have several AXI custom-built peripherals, where each peripheral can have its own Partial Reconfigurable Region (PRR). In addition, within each peripheral, there can be several PRRs.



*PRR: Partial Reconfigurable Region

- 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 Partial Reconfiguration Region, 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.

TIME AND MEMORY OVERHEAD

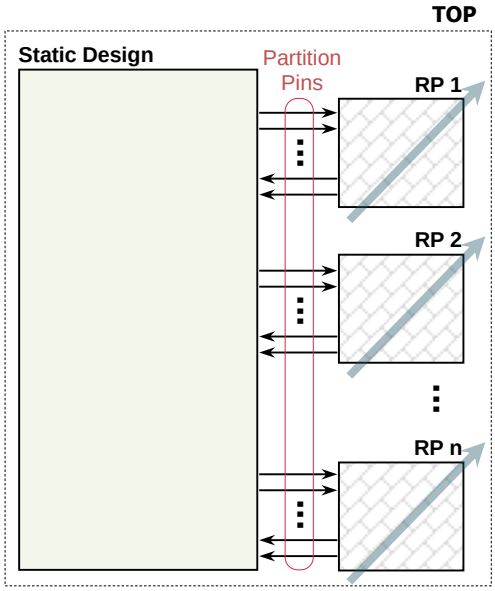
- Reconfiguration Time Overhead: This depends on the bitstream size, the design of the AXI interface design around the ICAP core, and the speed with which we can move data from memory to the ICAP core. Depending on the application, this overhead can be negligible or significant.

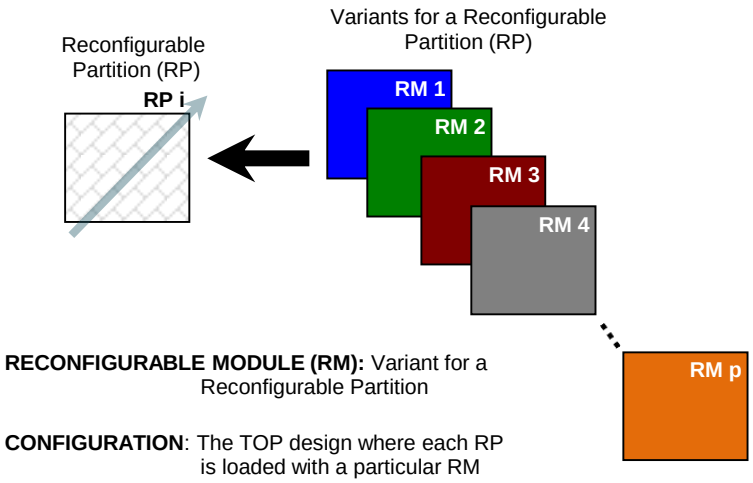
- Memory overhead: The partial bitsream files are stored in memory. Depending on the application, the number of combinations can range from the MBs to the GBs and it can pose a significant challenge to the system design.

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





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).