VLSI Design Automation
Lecture-3

By

Dr. Swagata Mandal

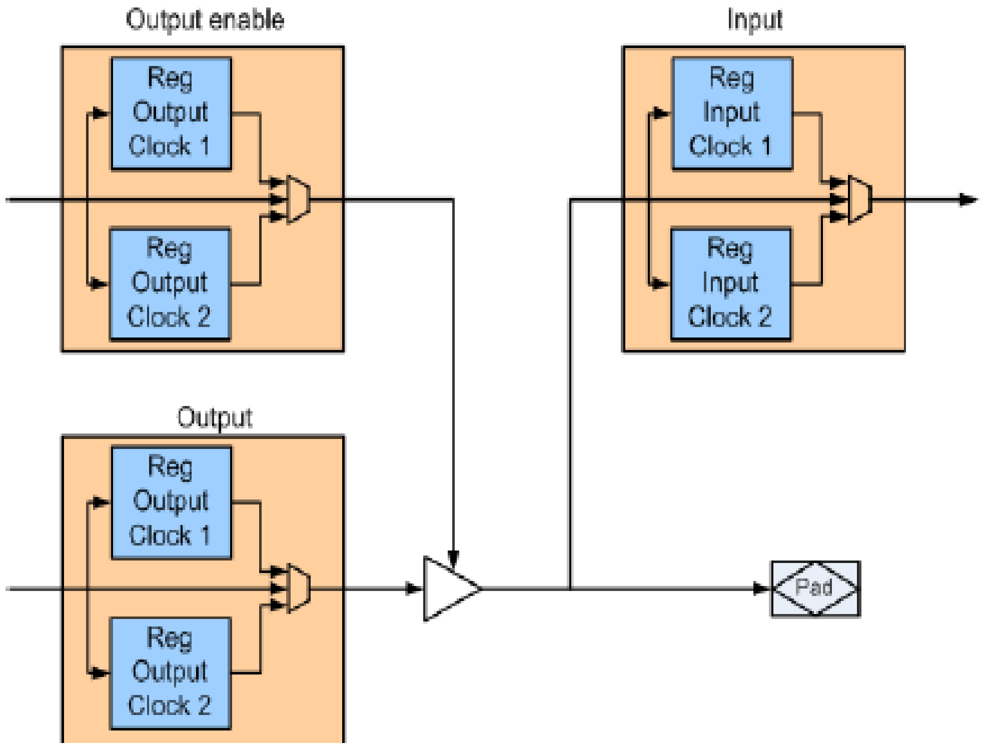
Assistant Professor, Electronics and Communication

Jalpaiguri Government Engineering College

Programmable I/O

- Located around the periphery of the device, I/O components allow for the communication of a design inside the FPGA with off-chip modules.

- Like the logic cells and the interconnections, FPGA I/Os are programmable, which allows designs inside the FPGA to configure a single interface pins either as input, output or bidirectional.
  - The general structure of an I/O component is shown in Fig 1.
  - It consists of an input block, an output block and an output enable block for driving the tri-state buffer.
  - Two registers that are activated either by the falling or by the rising edge of the clock, are available in each block.



Programmable I/O (cont'd)

- The I/Os can be parameterized for a single data rate (SDR) or a double data rate (DDR) operation mode.

- While in the SDR-mode data are copied into the I/O registers on the rising clock edge only, the DDR mode exploits the falling clock edge and the rising clock edge to copy data into the I/O registers.

- On the input, output, tri-state, one of the Double Data Rate (DDR) register can be used.

- The double data rate is directly accomplished by the two registers on each path, clocked by the rising edge (or the falling edge) from different clock nets.

- DDR input can be done using both input registers while DDR output will use both output registers.

- In the Altera Stratix II, the I/O component is called I/O Element (IOE). The Xilinx Virtex I/O components are called I/O Block (IOB) and they are provided in groups of two or four on the device boundary.

Hybrid FPGAs

- The process technology as well as the market demand is pushing manufacturers to include more and more pre-designed and well tested hard macros in their chips.
- It helps the designer to use well tested and efficient modules.
- Moreover, hard macros are more efficiently implemented and are faster than macro implemented on the universal function generators.
- The resources often available on hybrid FPGAs are
  - RAMs,
  - Clock managers (PLL, DCM, MMCM)
  - Arithmetic modules
  - Network interface modules
  - Processors
- The market demand has pushed almost all the manufactures to include hard macros in all their devices.

Hybrid FPGAs(cont'd)

- The resources required by the users varies with their application class.
- The most emerging classes as classified by Xilinx are: the System on Chip (SoC), Digital Signal Processing (DSP) and pure logic.
- The system on chip class to which the Virtex 4 FX belongs, is characterized by embedded processors directly available on the chip, memory and dedicated bus resources.
- The DSP class, which contains the Virtex 4 SX is characterized by the abundance of multipliers macros.
- The pure logic class, to which the Virtex 4 LX belongs, is dominated by LUT function generators

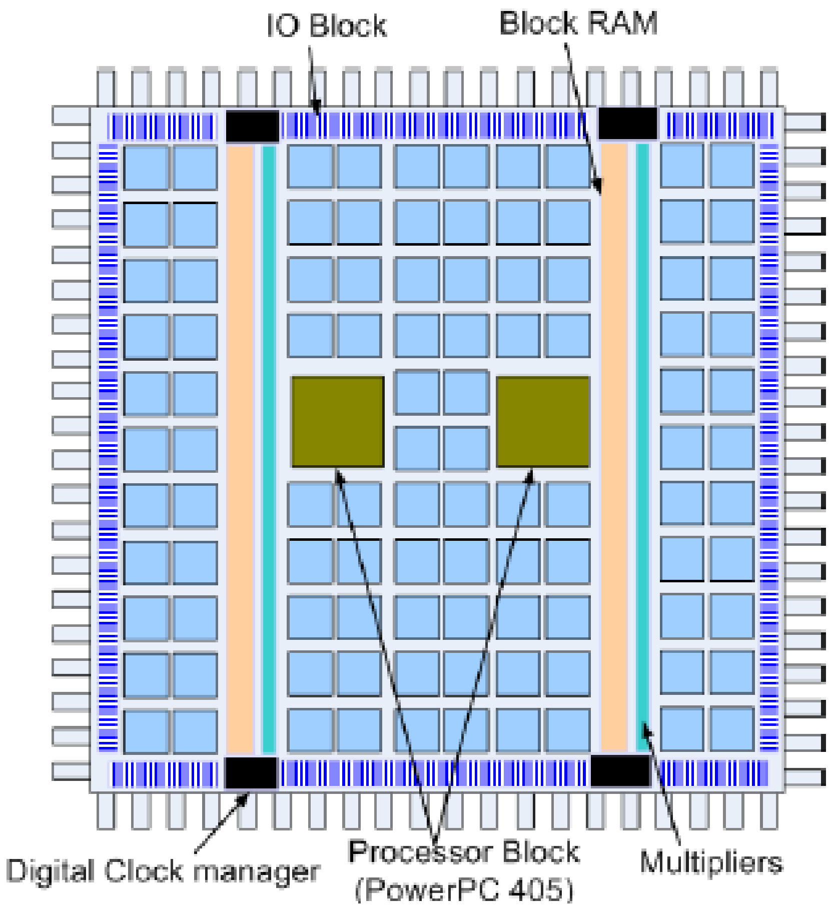


Fig: Structure of a Xilinx Virtex II Pro FPGA with two PowerPC 405 Processor blocks

Coarse-Grained Reconfigurable Devices

- Due to low granularity of the function generators (LUT and MUX) FPGAs can allow to implement function with various complexity.
- However, the programmable interconnections used to connect the logic blocks reduce the performance of FPGAs.
- One way is to use hard macro for frequently use functions. Coarse-grained reconfigurable devices follow this approach.
- In general those devices are made upon a set of hard macros (8-bit, 16-bit or even a 32-bit ALU), usually called Processing Element (PE).
- The PEs are able to carry few operations like addition, subtraction or even multiplication.
- The interconnection is realized either through switching matrices or dedicated busses.

Coarse-Grained Reconfigurable Devices(cont'd)

- Three categories of coarse-grained reconfigurable devices are available:

- ***Dataflow Machines:*** Functions are usually built by connecting some processing elements in order to build a functional unit that is used to compute on a stream of data. Example of dataflow machines are PACTXPP, NEC-DRP, PicoChip.

- ***Network-based Devices:*** Here the connection between the PEs is done using messages instead of wires. Example of network-based device is Adaptive Computing Machine (ACM) developed by Quicksilver Tech.

- ***Embedded FPGA devices:*** It consists of a processor core that cohabit with a programmable logic on the same chip. Example of embedded FPGA devices are Virtex, Kintex or Artix series FPGA of Xilinx, Arrya, Stratix series FPGA developped by Altera etc.

PACT XPP

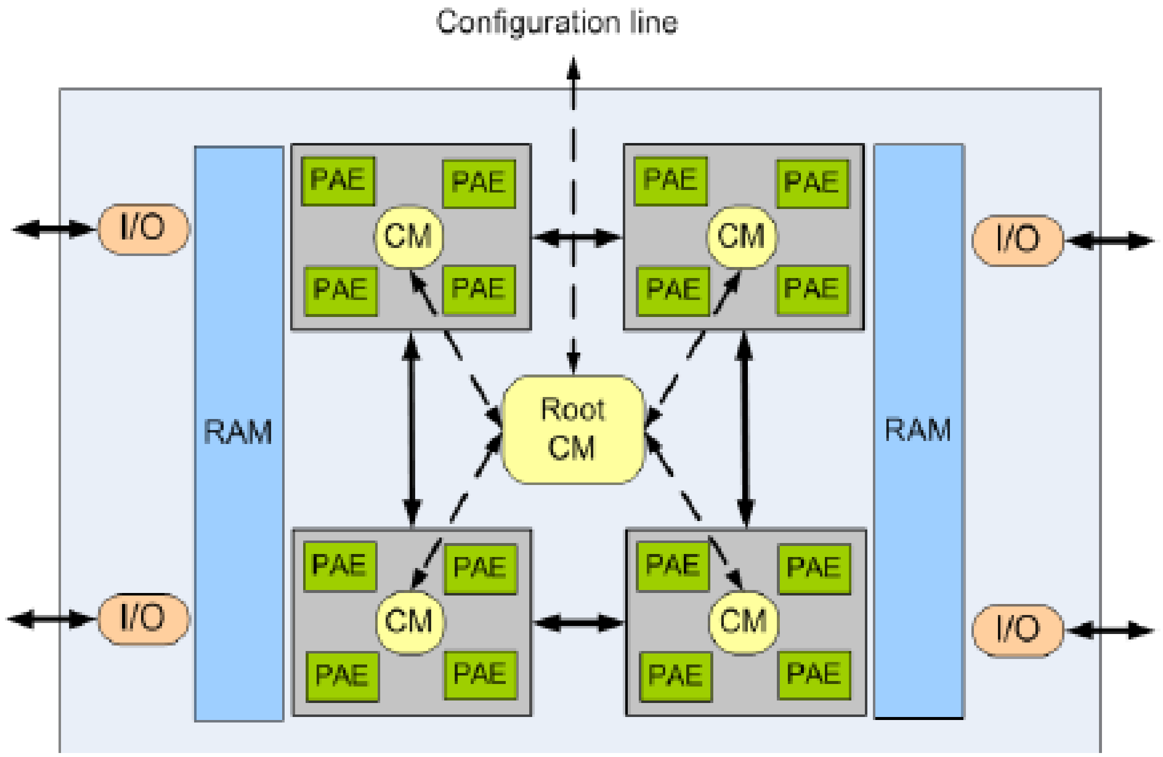
- The idea behind the PACT XPP architecture is to efficiently compute streams of data provided from different source.

- The XPP (eXtreme Processing Platform) architecture of PACT consists of:
  - An array of Processing Array Elements (PAE) grouped in Processing Array(PA)
  - A communication network
  - A hierarchical configuration tree
  - Memory elements aside the PAs
  - A set of I/O elements on each side of the device

- One Configuration Manager (CM) attached to a local memory is responsible for writing configuration onto a PA.

- The configuration manager together with PA builds the Processing Array Cluster (PAC).
- An XPP chip contains many PACs arranged as grid array on the device

PACT XPP(cont'd)

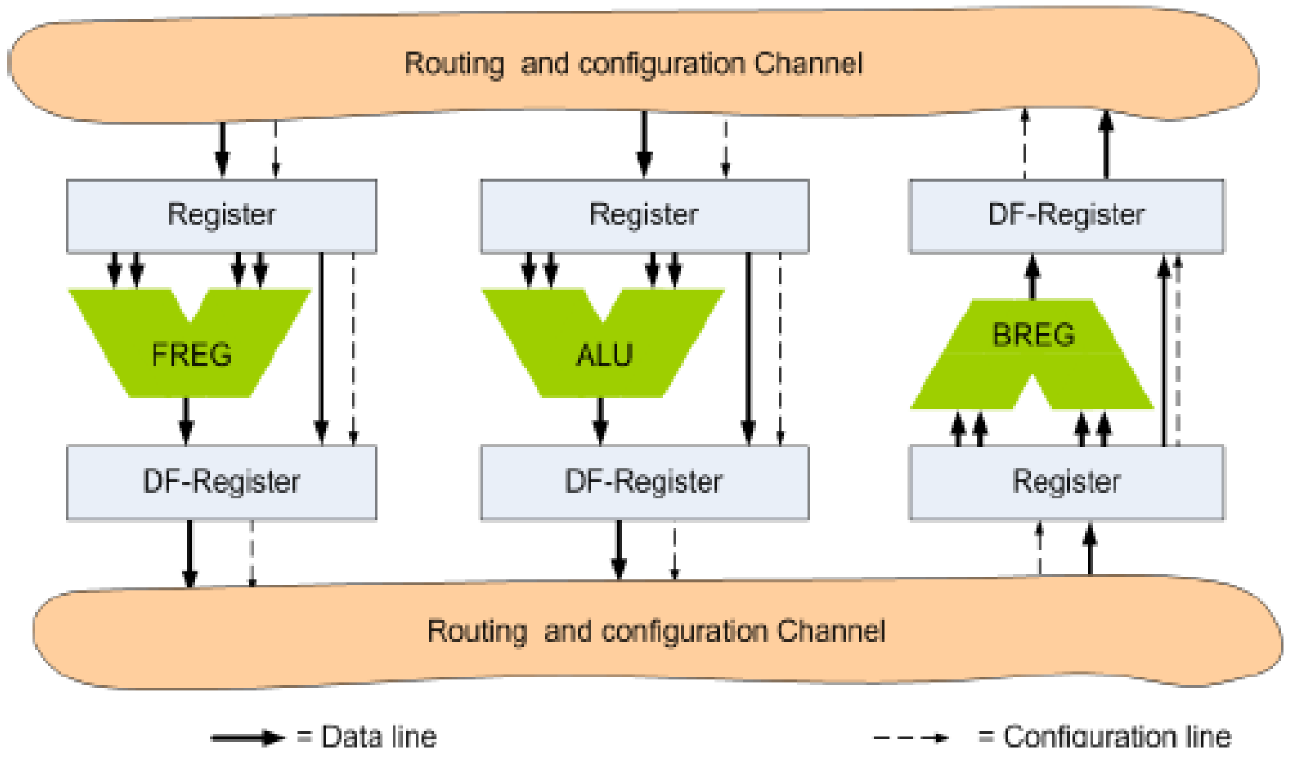


- Figure shows an XPP device with four PACs, each of which contains 4 PAEs and surrounded by memory blocks
- The CMs at a lower level are controlled by a CM at the next higher level.
- The root CM at the highest level is attached to an external configuration memory and supervises the whole device configuration.

Processing Array Element (PAE)

- Two different kinds of PAEs are there : the ALU PAE and the RAM-PAE
- An ALU-PAE contains an ALU that can be configured to perform basic arithmetic operations, while the RAM-PAE is used for storing data.
- The Back-Register (BREG) provides routing channels for data and events from bottom to top, additional arithmetic and register functions while the Forward-Register (FREG) is used for routing the signals from top to bottom and for the control of dataflow using event signals.
- All objects can be connected to horizontal routing channels using switch-objects.
- DataFlow Register (DF-Registers) can be used at the object output for data buffering in case of a pipeline stall.
- Input registers can be pre-loaded by configuration data and always provide single cycle stall.

Processing Array Element(cont'd)



- A RAM-PAE is similar to an ALU-PAE. However, instead of an ALU, a dual ported RAM is used for storing data.
- The RAM generates a data packet after an address was received at the input.
- Writing to the RAM requires two data packet: one for the address and the other for the data to be written.
- Figure shows an ALU-PAE. The structure is the same for a RAM-PAE, however a RAM is used instead of the ALU

Routing & Communication

- The XPP interconnection network consists of two independent networks: One for data transmission and the other for event transmission.

- These two networks consist of horizontal and vertical channels.

- The vertical channels are controlled by the BREG and FREG while connection to horizontal channel is done via switch elements.

- Beside the horizontal and vertical channels a configuration bus exists, which allows the CMs to configure the PAEs.

- Horizontal buses are used to connect a PAE within a row while the vertical buses are used to connect objects to a given horizontal bus.

- Vertical connections are done using configurable switch objects which segment the vertical communication channels.
- The vertical routing is enabled using register-objects integrated into the PAEs.

Interfaces

- XPP devices provide communication interfaces aside the chip.
- The number may vary from device to device.
- The I/O interfaces can operate independently from each other either in RAM, or in streaming mode.
- In streaming mode, each I/O element provides two bidirectional ports for data streaming.
- Handshake signals are used for synchronization of data packets to external ports.
- In RAM mode, each port can access external synchronous SRAMs with 24-bit addresses and 24-bit data.
- Control signals for the SRAM transactions are available such that no extra logic is required.
- The configuration manager interface consists of three subgroups of signals: *code*, *message send* and *message receive*.
- The code group provides channels over which configuration data can be downloaded to the device while the send and receive groups provide communication channels with a host processor.

Network-oriented architectures

- Here reconfigurable devices rely on message passing for data exchange among the PEs. The Adaptive Computing Machine (ACM) is based on this strategy.
- The Quicksilver ACM consists of a set of heterogeneous computing nodes hierarchically arranged on a device.
- At the lowest level, four computing nodes are placed in a cluster and connected locally together.
- Many clusters at a given level are put together to build bigger clusters at the next higher level

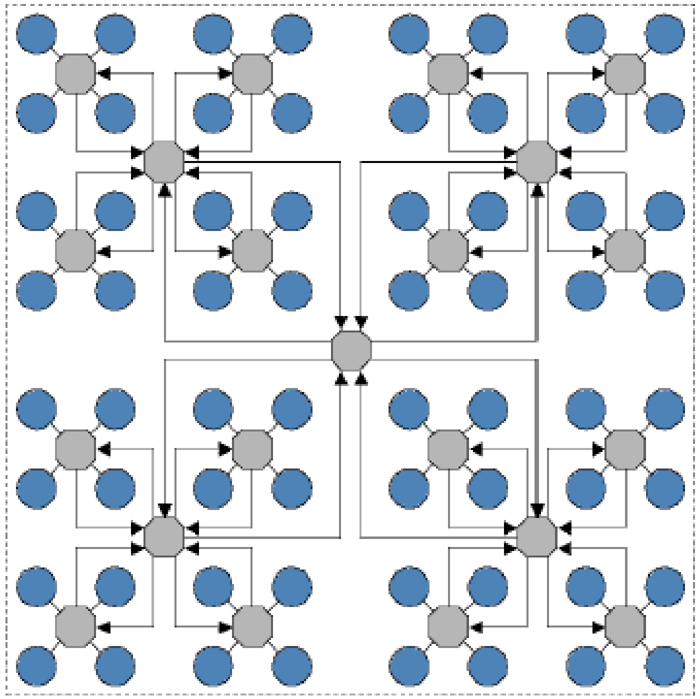


Fig: The Quicksilver ACM hierarchical structure with 64 nodes

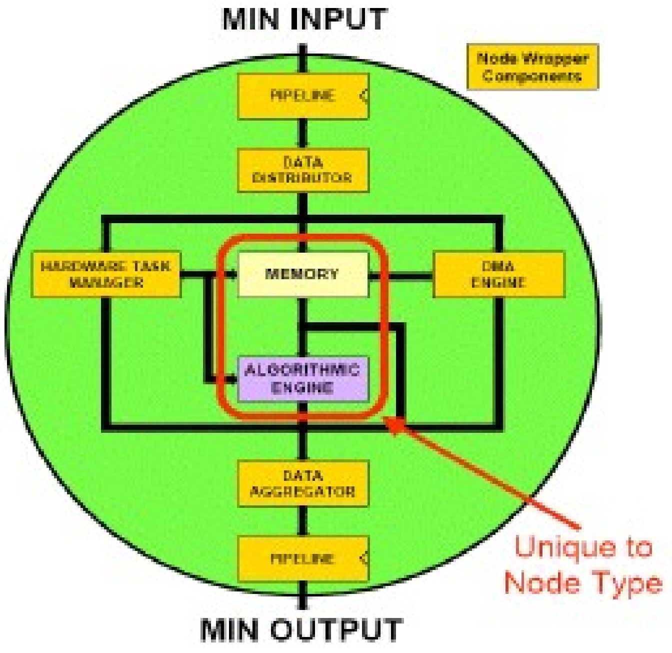
Adaptive Computing Machine

- An ACM chip consists of the following elements:
  - A set of heterogeneous *Processing Nodes (PN)*
  - An homogenous *Matrix Interconnect Network (MIN)*
  - A *system controller*
  - Various I/O interfaces

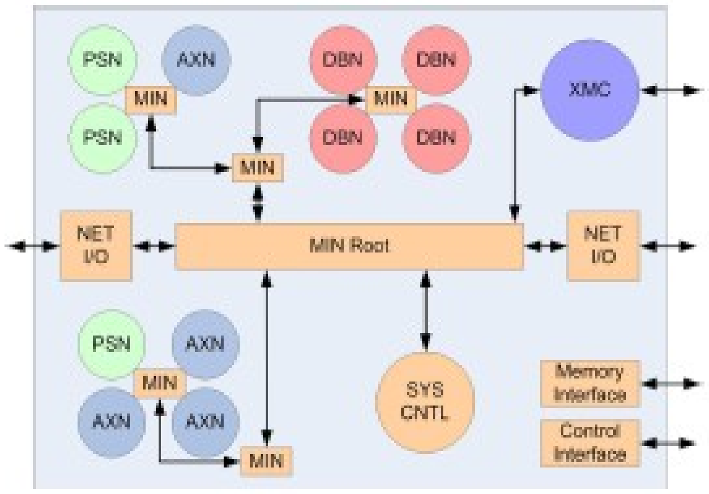
- An ACM processing nodes consist of:
  - An ***algorithmic engine*** that defines the node type. The node type can be customized at compile-time or at run-time by the user to match a given algorithm.
  - Four types of nodes exist
  - The *Programmable Scalar Node (PSN)* provides a standard 32-bit RISC architecture with 32 general purpose registers.
  - *Adaptive Execution Node* (AXN) provides variable word size Multiply Accumulate (MAC) and ALU operations.
  - *Domain Bit Manipulation* (DBN) node provides bit manipulation and byte oriented operations
  - *External Memory Controller* node provides DDRRAM, SRAM, memory random access and DMA control interfaces for off-chip memory access

ACM processing nodes(cont'd)

- ***Node memory*** for data storage at node level.
- ***Node wrapper*** which hide the complexity of the network architecture. It contains a MIN interface to support communication, a hardware task manager for task managements at node level, and a DMA engine.
- It also incorporates dedicated I/O circuitry, memory, memory controllers and data distributors and aggregators



(a) The ACM node structure



(b) The ACM QS2412 Resources

Matrix Interconnect Network

- The communication inside an ACM chip is done via a MIN, which is organized hierarchically.
- At a given level, the MIN connects many lower level MINs. The top level MIN, the MIN root, is used to access the nodes from outside and to control the configuration of the nodes.
- The communication among nodes is done via the MIN with the help of the node wrapper.
- The ACM chip also contains various I/O interfaces accessible via the MIN for testing (JTAG) and communication with off-chip devices.
- The system management is done via an embedded system controller.
- The system controller loads tasks into the node’s ready-to-run queue for execution, statically or dynamically sets the communication channels between the processing nodes.

Embedded FPGA

- It integrates a processor core, a programmable logic or FPGA and memory on the same chip.
- Two examples of such devices are the *DAP/DNA* from IPflex and the S500 series from Stretch.
- The DAP/DNA consists of an integrated DAP RISC processor, a *Distributed Network Architecture* (DNA) matrix and some interfaces as shown in Fig 1.

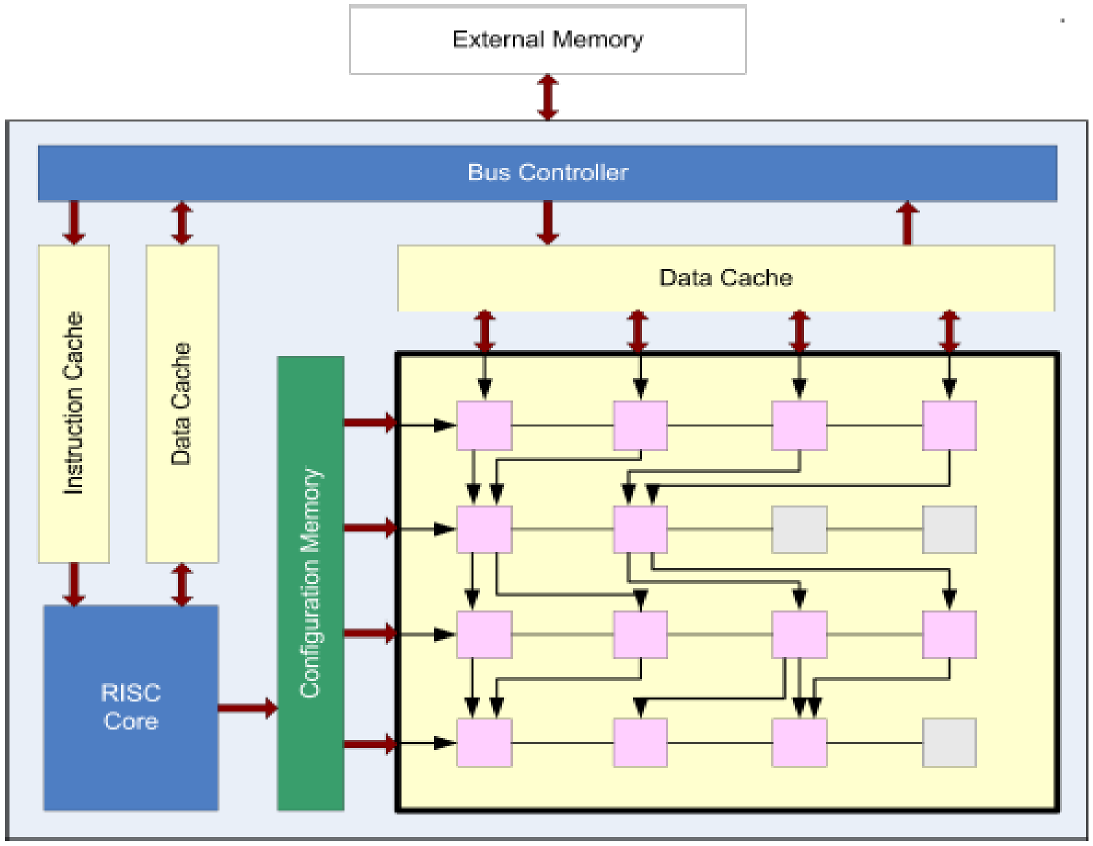


Fig 1: IPflex DAP/DNA reconfigurable processor

Embedded FPGA(cont'd)

- The processor controls the system, configures the DNA, performs computations in parallel to the DNA and manages the data exchange on the device.
- The DNA Matrix is a dataflow accelerator with more than hundred dynamic reconfigurable processing elements.
- The wiring among elements can be changed dynamically, therefore, providing the possibility to build and quickly change parallel/pipelined processing system tailored to each application.
- The DNA configuration data are stored in the configuration memory from where it can be downloaded to the DNA on a clock-by-clock basis.
- Like in other reconfigurable devices, several interfaces exist for connecting the chip to external devices.
- While the processor controls the whole execution process, the DNA executes critical parts of the application.

Implementation

- Reconfigurable devices are usually used in three different ways:
  - ***Rapid Prototyping***
  - ***Non-frequently Reconfigurable Systems***
  - ***Frequently reconfigurable systems***

- With the increasing size and speed of reconfigurable processors it is possible to implement many large modules on a reconfigurable device at the same time.

- Moreover, for some reconfigurable devices, only a part of the device can be configured while the rest continues to operate which is known as *partial reconfiguration*.

- This partial reconfiguration capability enables many functions to be temporally implemented on the device.

- Depending on the time at which the reconfiguration sequence are defined, the computation and configuration flow on a reconfigurable devices can be classified in two categories:

Implementation(cont’d)

- **Compile-time reconfiguration:** In this case, the computation and configuration sequences as well as the data exchange are defined at compile-time and never change during a computation.
- This approach is more interesting for devices, which can only be full reconfigured.
- However it can be applied to partial reconfigurable devices that are logically or physically partitioned in a set of reconfigurable regions.
- **Run-time reconfiguration :** The computation and configuration sequences are not known at compile-time.
- Request to implement a given task is known at run-time and should be handled dynamically.
- The reconfiguration process exchange part of the device to accommodate the system based on the operational and environmental conditions.

Run-Time Reconfiguration

- Run-time reconfiguration is a difficult process that must handle side effect factors like defragmentation of the device and communication between newly placed modules.

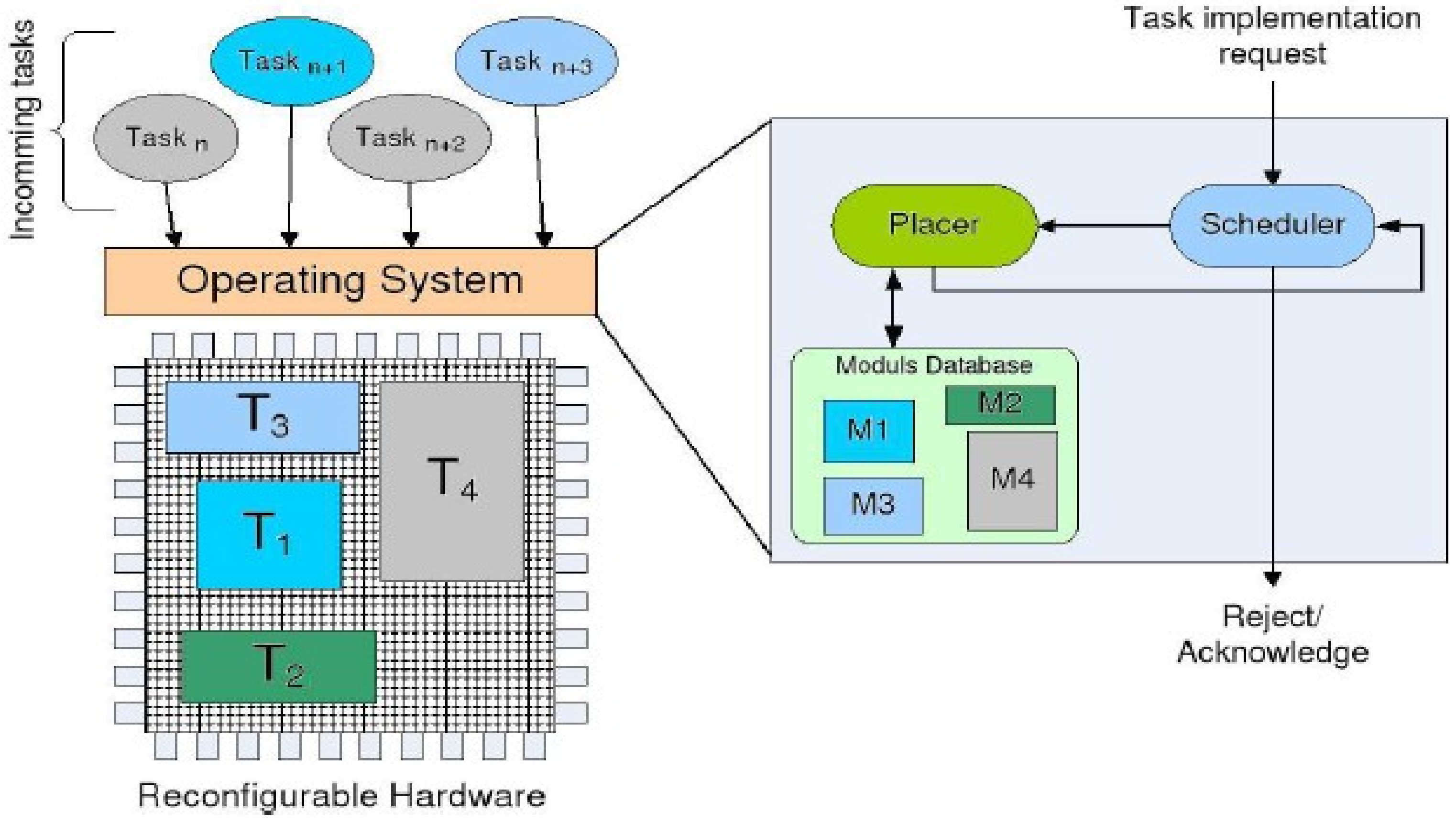


Fig 1: Architecture of a run-time reconfigurable system

Run-Time Reconfiguration(cont’d)

- The management of the reconfigurable device is usually done by a scheduler and a placer that can be implemented as part of an operating system running on a processor .
- The processor can either resides inside or outside the reconfigurable chip.
- The scheduler manages the tasks and decides when a task should be executed.
- The tasks, which are available as configuration data in a database are characterized through their bounding box and their run-time.
- The bounding box defines the area that a task occupies on the device.
- The management of task execution at run-time is therefore a ***temporal placement problem***.
- The scheduler determines which task should be executed on the RPU and then gives the task to the placer which will try to place it on the device, i.e. allocate a set of resources for the implementation of that task.

Run-Time Reconfiguration(cont'd)

- If the placer is not able to find a site for the new task, then it will be sent back to the scheduler which can then decide to send it later and to send another task to the placer.
- Both for compiled time and run time reconfigurable system the computation and reconfiguration flow is usually the one shown on Figure.

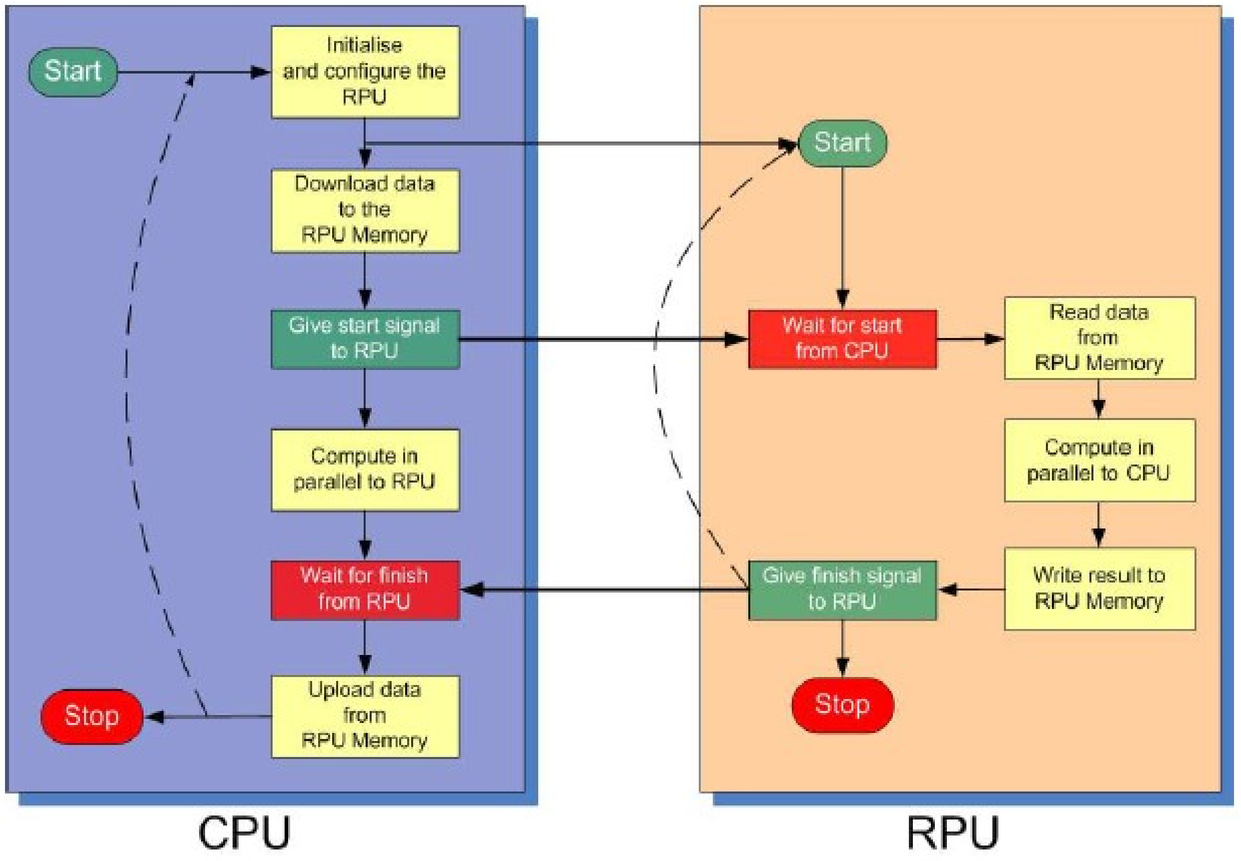

Start
Initialise and configure the RPU
Download data to the RPU Memory
Give start signal to RPU
Compute in parallel to RPU
Wait for finish from RPU
Upload data from RPU Memory
Stop
Start
Wait for start from CPU
Read data from RPU Memory
Compute in parallel to CPU
Write result to RPU Memory
Give finish signal to RPU
Stop
CPU
RPU

Figure: A CPU-RPU configuration and computation step

Thank You