Hardware Acceleration Through FPGAs

1. Basic Concepts and Examples

Giorgio Lopez
CERN TE/EPC/CCE

Summary

- Elements of an FPGA
- FPGA Timing
- Parallel Processing Paradigms and Challenges
- FPGA-based System Architectures
- Hardware / Software Partitioning
- FPGA Design Flow Basics

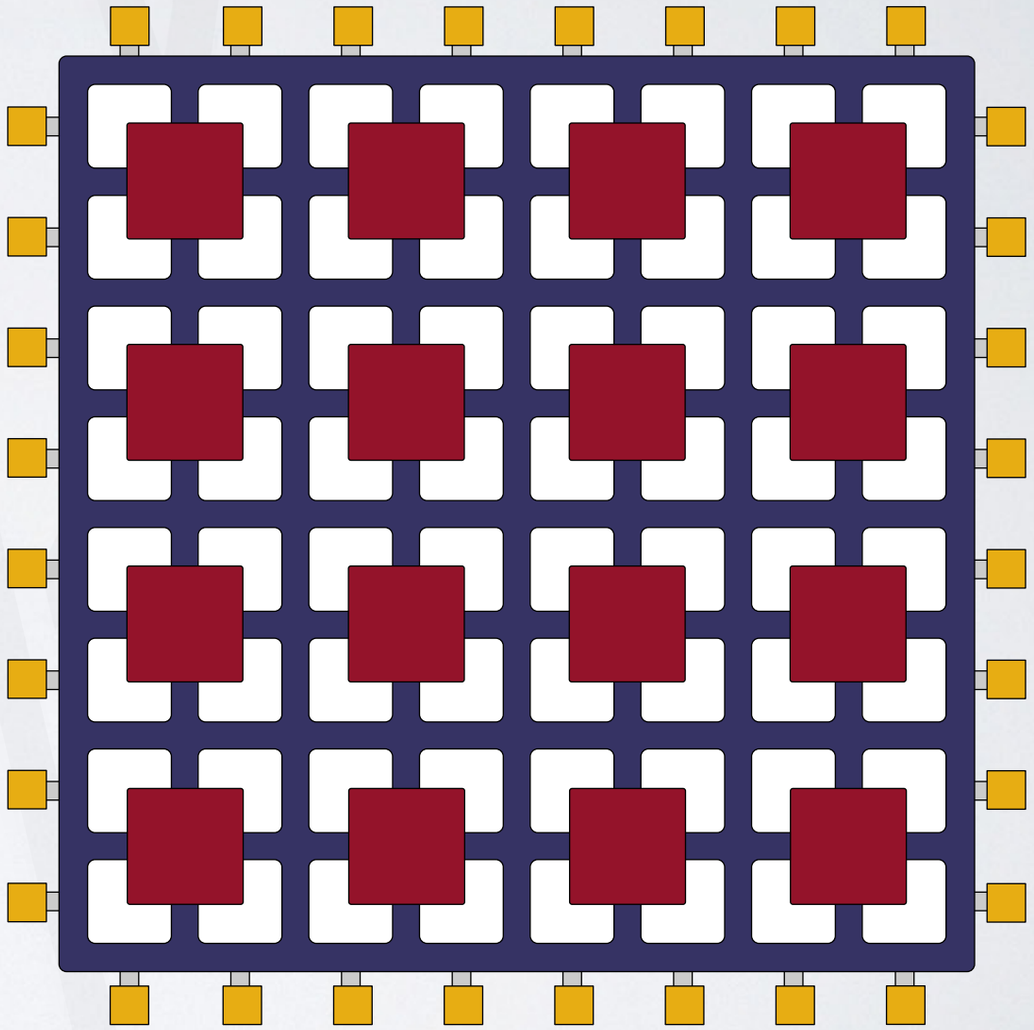
Elements of an FPGA

Basic components of an FPGA include:

- Array of Logic Blocks
- Block RAMs
- Clock Management Circuitry
- I/O Blocks and Interconnect Elements
- DSP Elements (Multipliers, Accumulators, etc.)

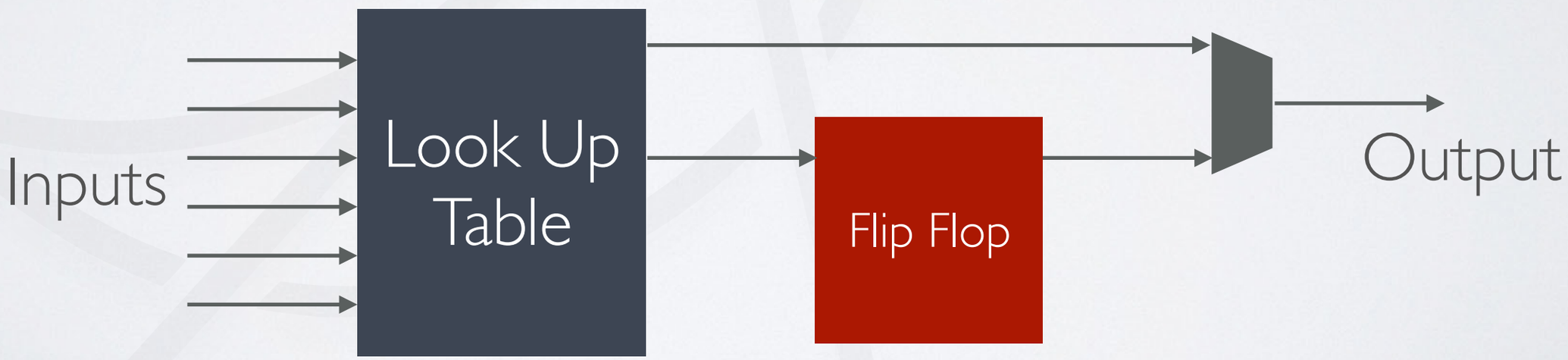
Fancier models may include:

- *Transceivers*
- *Physical CPU cores*
- *External Bus Interfaces (ex. PCIe)*



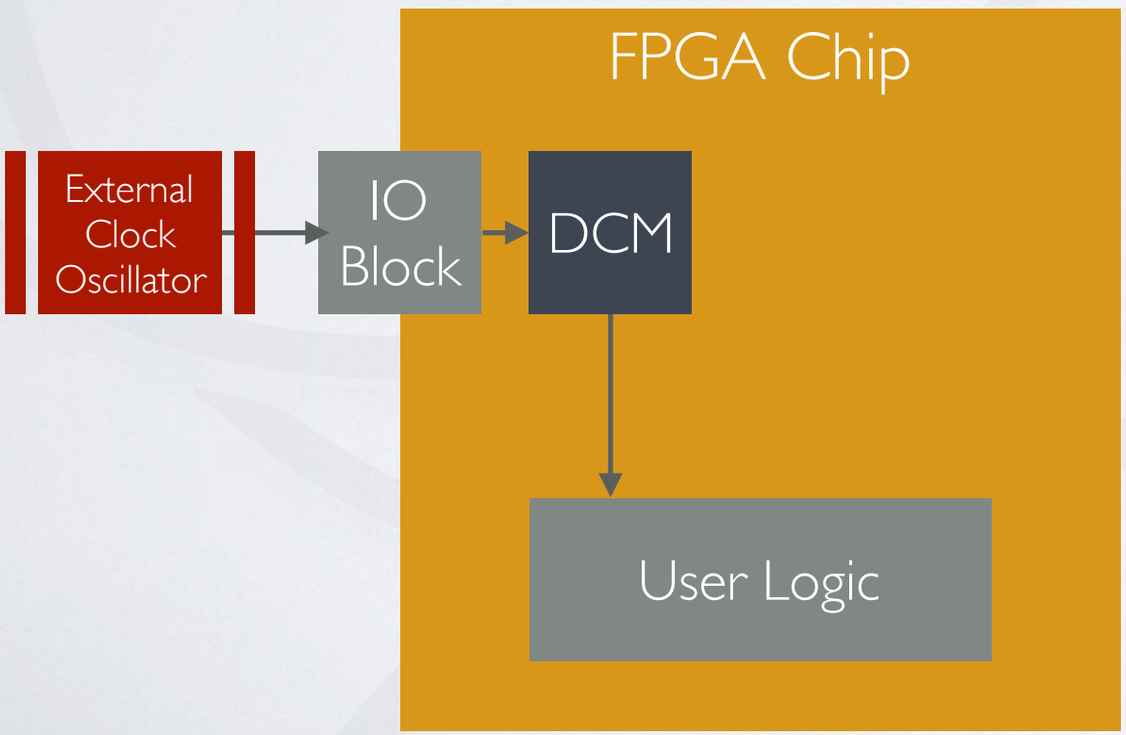
Elements of Logic Blocks

- Look Up Tables (combinatorial): truth-table-like implementation of any N-inputs boolean function
- Flip Flops (sequential): atomic blocks of clocked memory storing a single bit

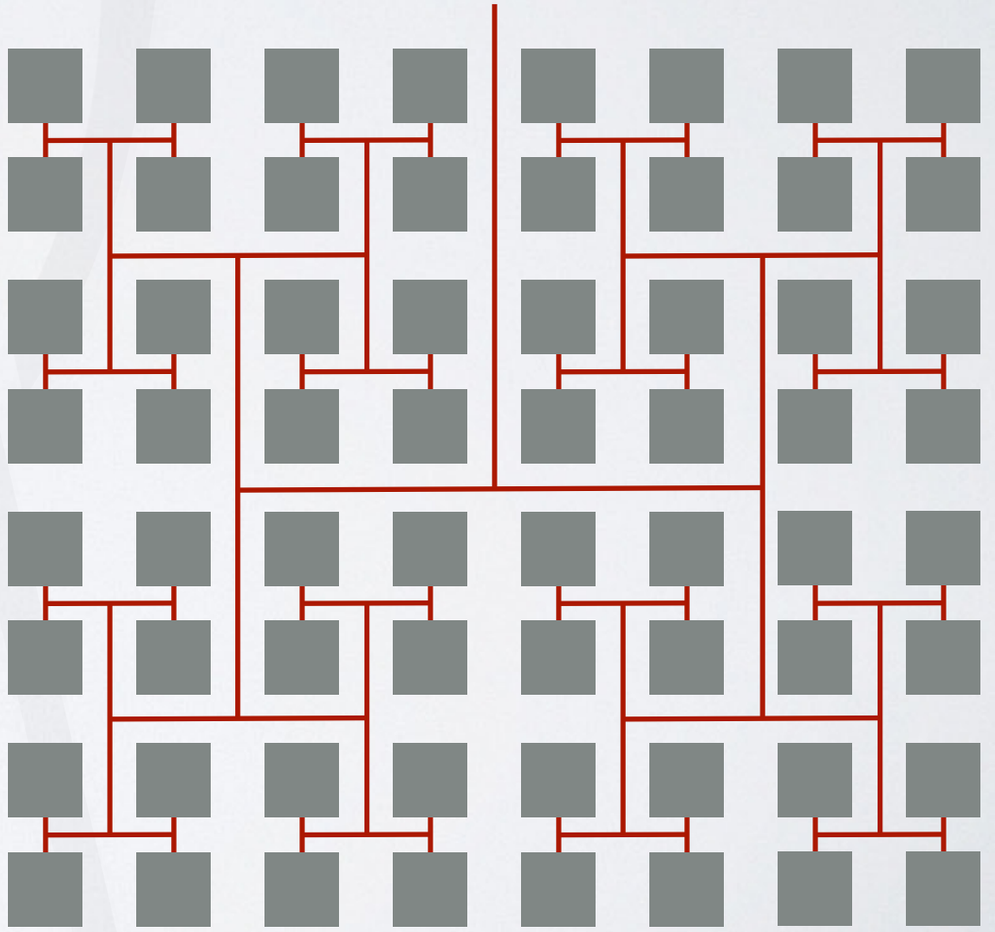


FPGA Clocking

- Clock Management Units (DCMs, PLLs): align internally distributed clock with external oscillator / input. Can also alter clock frequency.



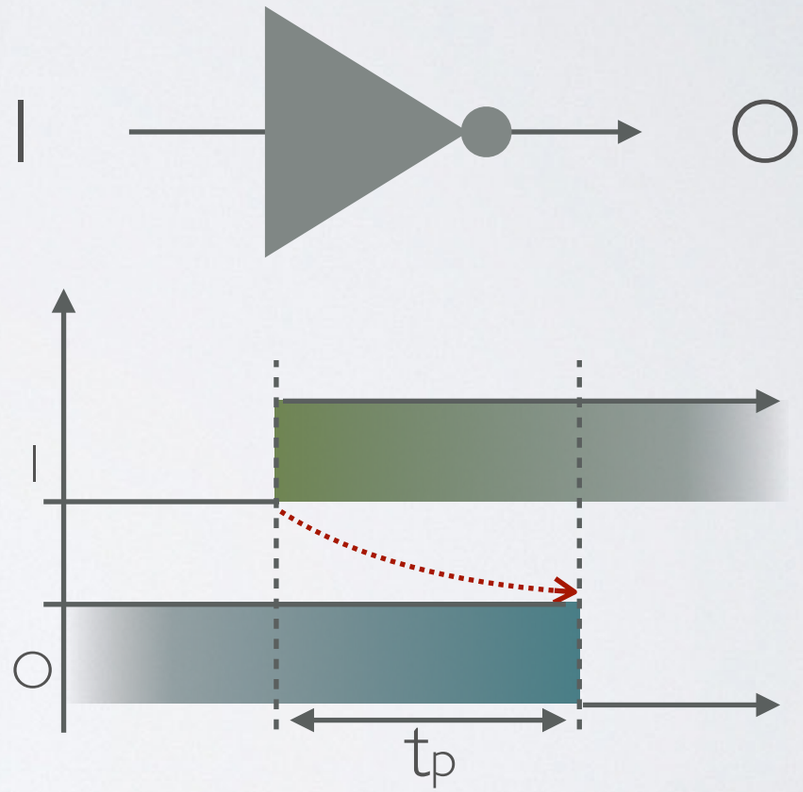
- Clock Trees: distribute the clock signal with minimum skew (equal-length signal paths). Alredy configured by chip manufacturer.



Timing in FPGAs

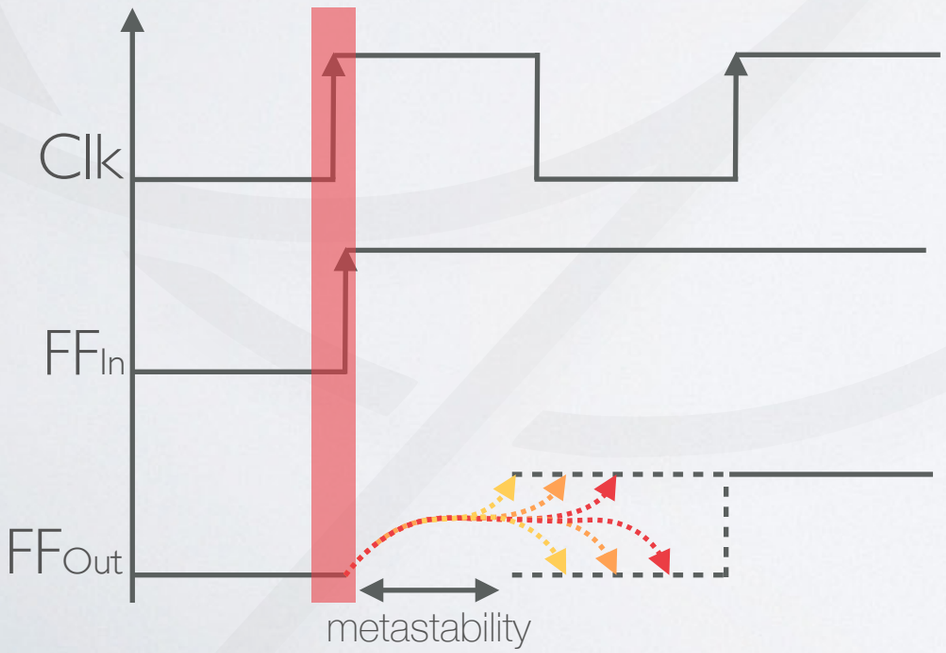
- Propagation of digital signals, most notably on long combinatorial paths, implies delays.
- This is due to the electrical characteristics of logic gates and data transfer lines and can be made worse by:

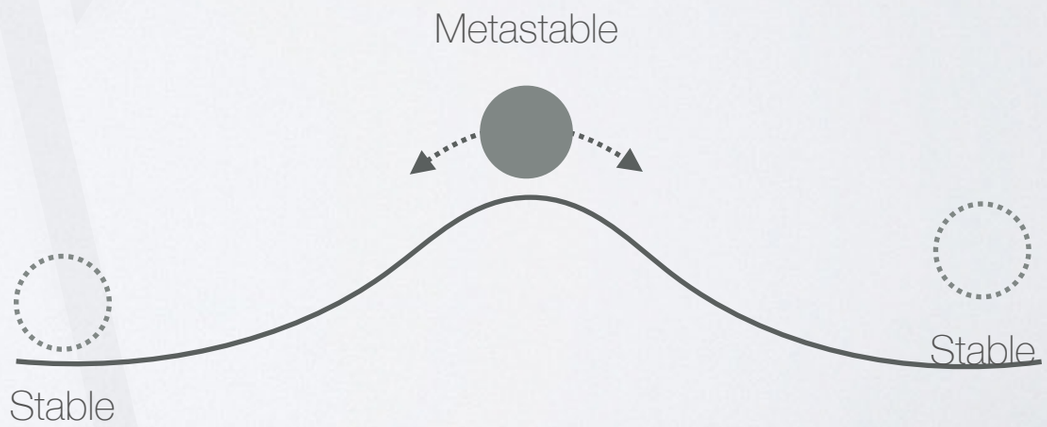
1. Fan-In (# of inputs to the logic gate)
2. Fan-Out (# of circuits fed by logic gate)
3. Length of transfer line (good routing is crucial)



Metastability

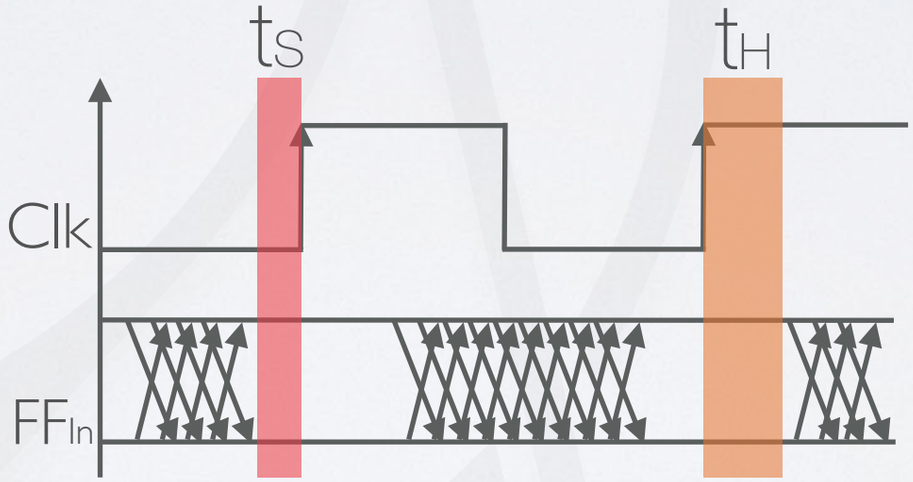
- Flip Flops sample their inputs on the active edge of clock
- Signal must be stable around this edge for correct sampling
- Failure in meeting this condition can lead to “metastability”, a condition where the output of a Flip Flop can get to an intermediate, unstable voltage value, staying in this state for un unpredictable time before finally settling in one of the two stable states.





Timing Constraints

- A “safety window” around the clock edge must be established: this is given by the Hold Time t_H and Setup Time t_S



- Too long/short combinatorial paths between Flip Flops can “challenge” the fulfillment of these constraints

Metastability and Asynchronous Inputs

- Even if we correctly route signals internally, external signals will always be asynchronous

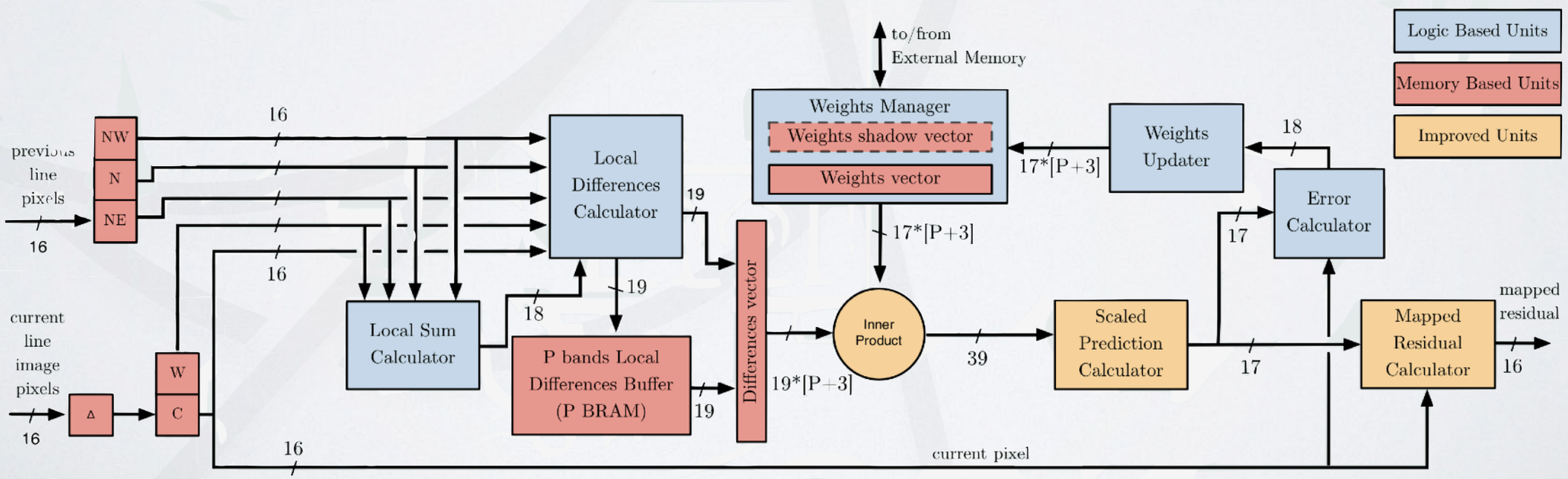
- This is also true for signals coming from different clock domains

Solution:

- Synchronizer chains: cascades of 2 or 3 flip flops on “synchronicity boundaries” which make metastability way less likely *(P_{event}= ~10^{-21})*

Data Intensive Applications: Why Use FPGAs?

- As opposed to the Flow in a CPU which is primarily sequential, digital circuits are intrinsically parallel
- FPGAs are the most accessible and fast-time-to-market solution for digital circuital implementation of applications
- When the data processing flow is predominant on complex control logic an FPGA can greatly accelerate the performance of the application



Example of DSP application: while everything is synchronized by the clock, all parts operate simultaneously on different parts of the input data

Parallel Processing Paradigms: Pipelining

- When implementing a data processing algorithm which implies several phases, each step of the algorithm can be given to a specific unit, with “checkpoints” between stages. At any given time, each unit will be performing its transformation on a different set of data.

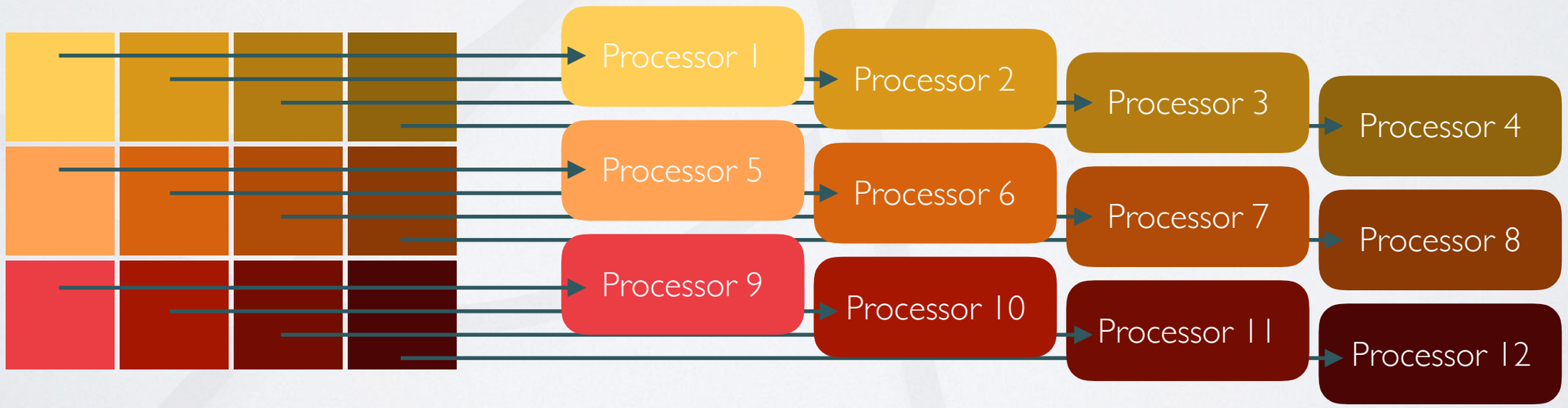
Challenges of Pipelining

- There's always an initial time to "fill the pipeline" where only a subset of steps will be actually operating (can be negligible)
- For pipelining to be effective all stages must take the same time to be executed (to avoid idle times)

(will wait 80ns for DCT)

Parallel Processing Paradigms: Matrix Partitioning

A matrix-like set of data (e.g. an image) can be decomposed in subparts or tiles, each to be independently handled and processed by a specific unit.



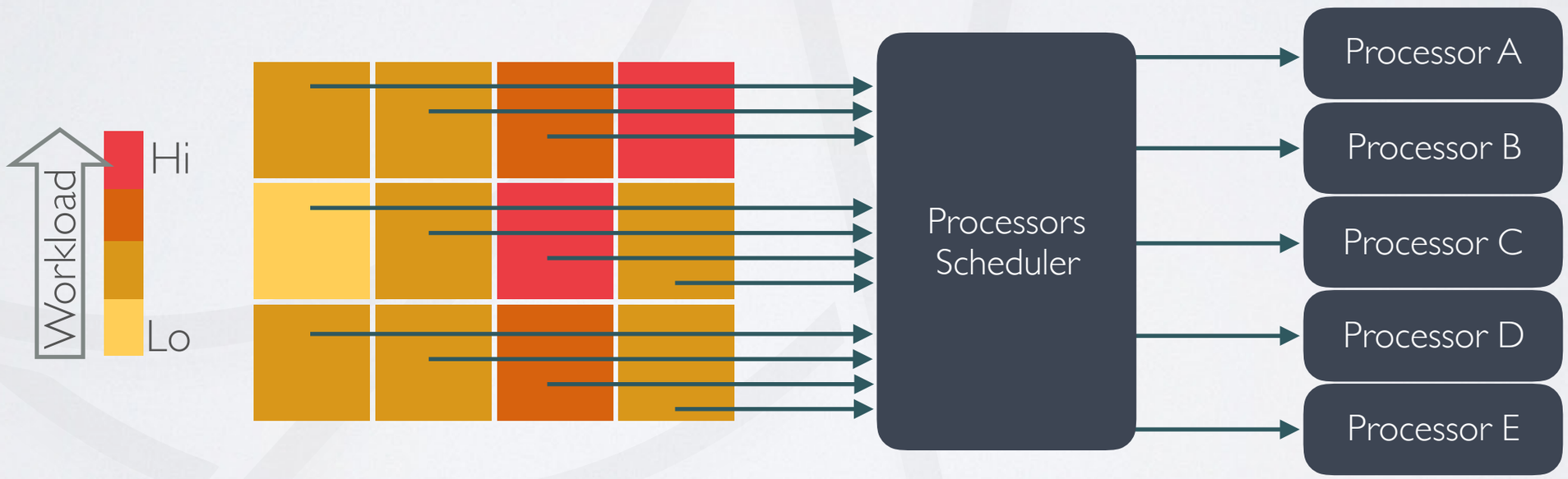
Challenges of Matrix Partitioning

- Since all processing units are identical, the paradigm works well if the workload is also identical for each tile

- If input data size can vary this can complicate the scheme

Parallel Processing Paradigms: Dynamically Scheduled Partitioning

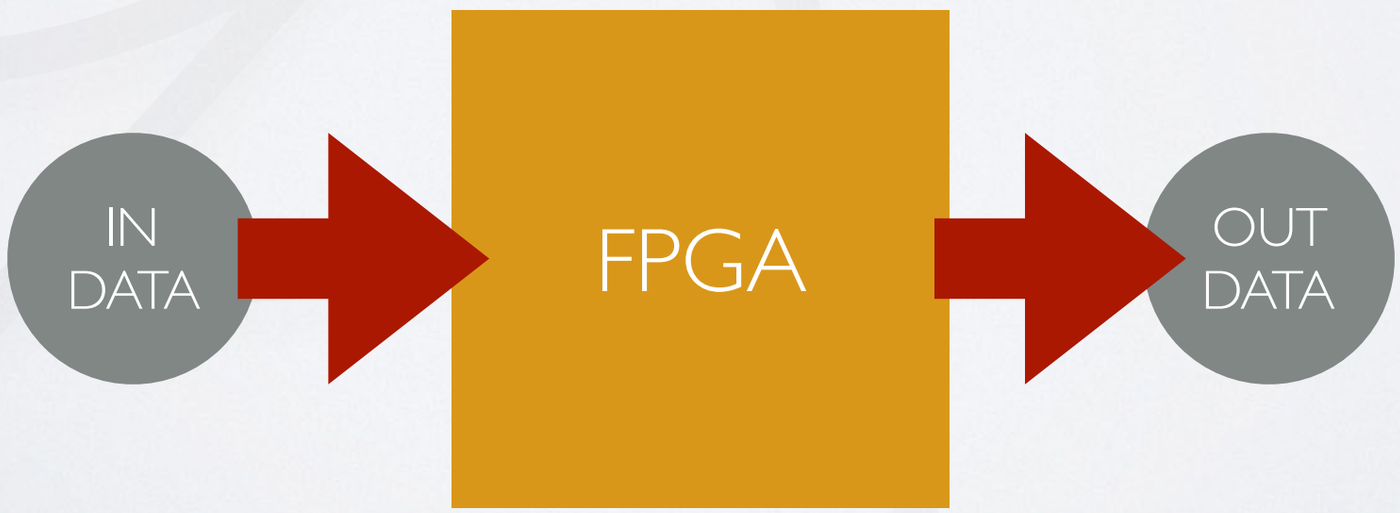
A more sophisticated example: when the workload can be different from tile to tile, a scheduler can allocate resources to different chunks of the input data (also to accomodate changes in the size of the input itself).



The scheduler could be implemented in a processor core (even on board of the FPGA)

Typical System Architectures: Standalone FPGA Systems

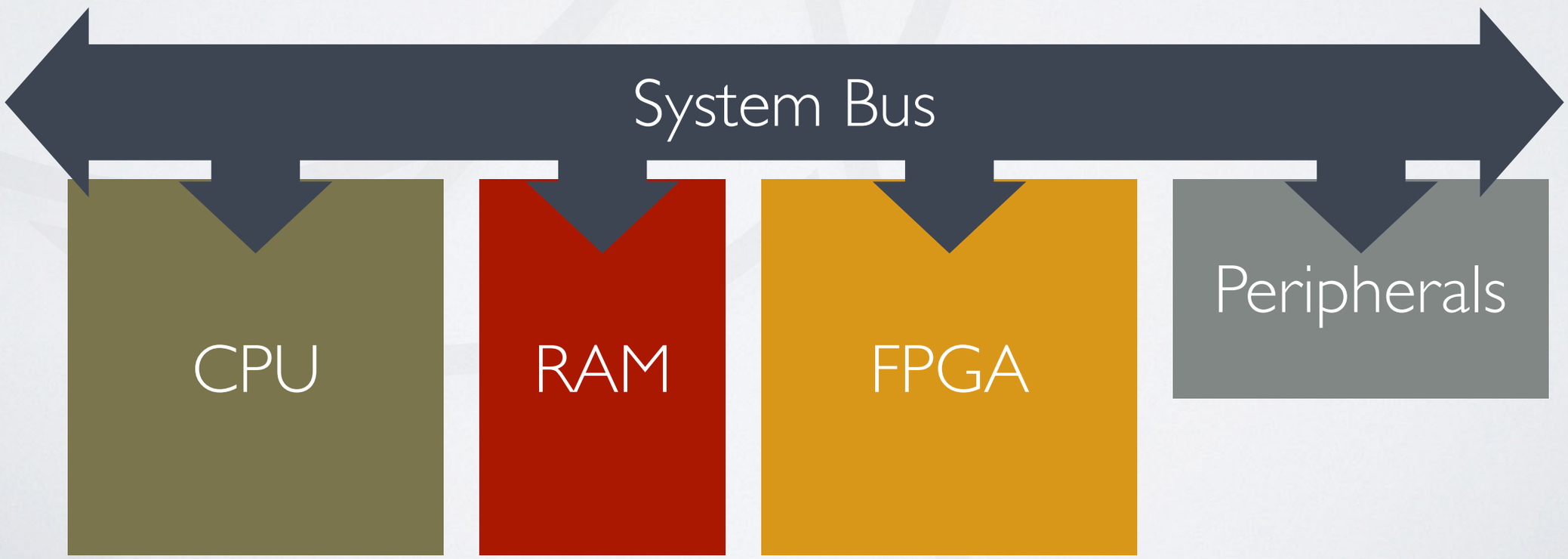
- The FPGA operates without the aid of a CPU: an external (DRAM) memory may or may not be present.
- Most indicated when the algorithm is data-intensive and of relatively low control complexity.



Typical System Architectures: Multiprocessor Arrangement

- The FPGA behaves as an additional processor, residing on the system bus and receiving input data by the CPU via DMA on a local memory, transferring back the results either through DMA or through mapped registers,

- Control flow managed by the CPU / computationally dense portions of the algorithm accelerated via hardware implementation.

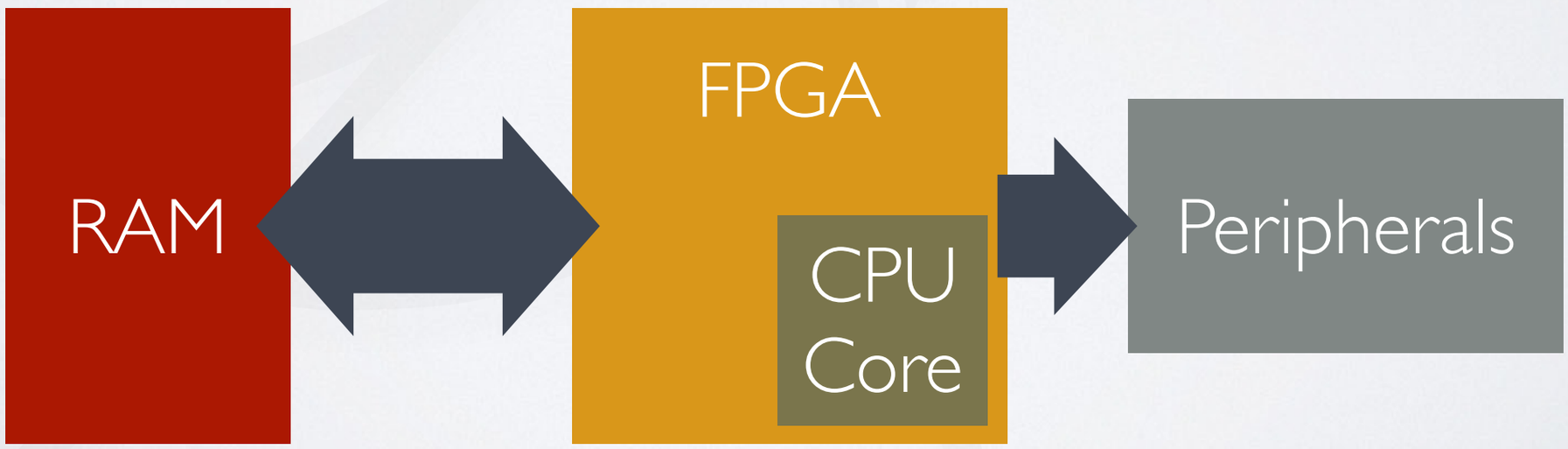


Typical System Architectures: System-on-Chip Architecture

- An evolution of the previous scheme, where the whole System is on the FPGA chip itself: the CPU can be either a physical CPU core embedded on the Silicon die or a “soft-CPU” IP block (or even more than one…)

- Can be an easy access solution for complex algorithms

- Typically some RAM (internal or external w.r.t. the FPGA) will be needed



Hardware / Software Partitioning: Why?

- Typically, the engineering cost required to translate a whole algorithm into a digital circuit implementation is not reasonable.

- As a rule of thumb, 90% of the time is spent in execution of 10% of the code

- Complex applications are composed by “sequential” or “control intensive” tasks and “data intensive” tasks.

- While “data intensive” tasks would crucially benefit from parallel implementations on FPGA technology, the same doesn’t hold for the control part of the algorithm, which is more aptly carried out by a processor.

Hardware / Software Partitioning: How?

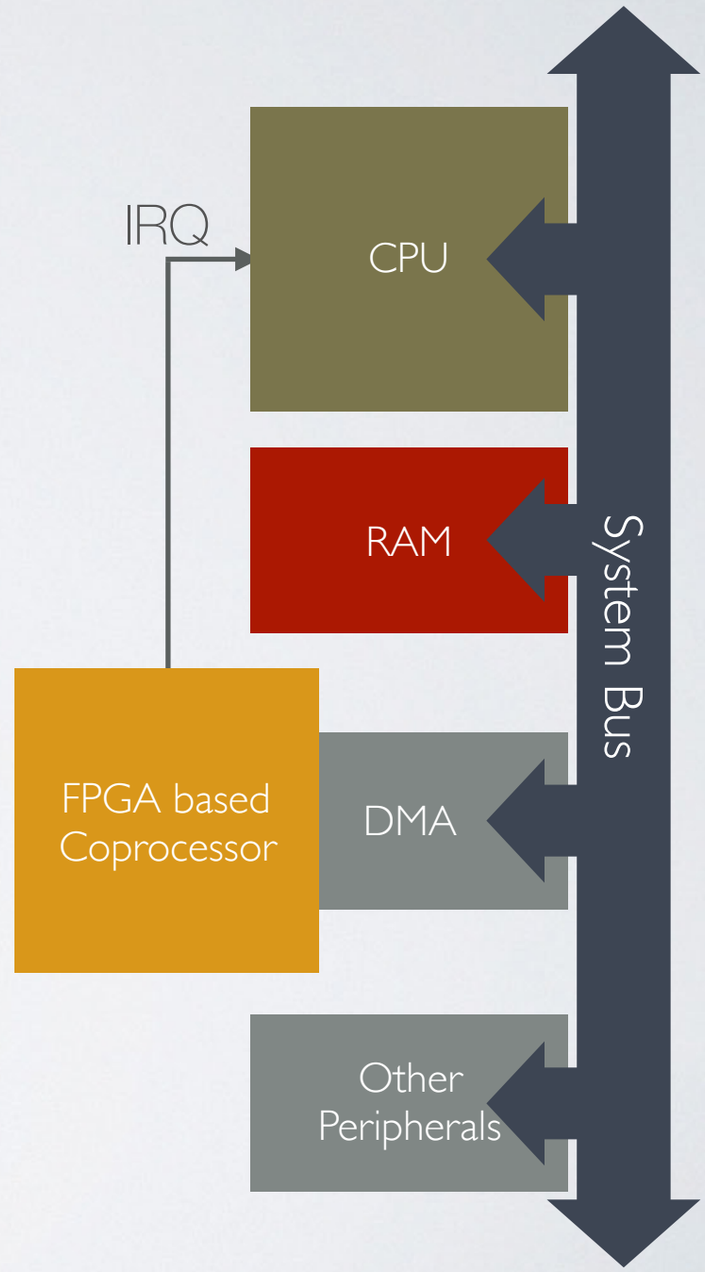
- A crucial step is understanding which parts of the SW algorithm are taking up most of the processing time

- Profiling tools are used for this purpose (they must be run on the target platform/architecture)

- In addition, it is generally more beneficial to keep complex control logic in the SW domain, whereas data-intensive tasks are more easily translated into a digital circuit

Connections Scheme: Bus-Connected FPGA Coprocessor

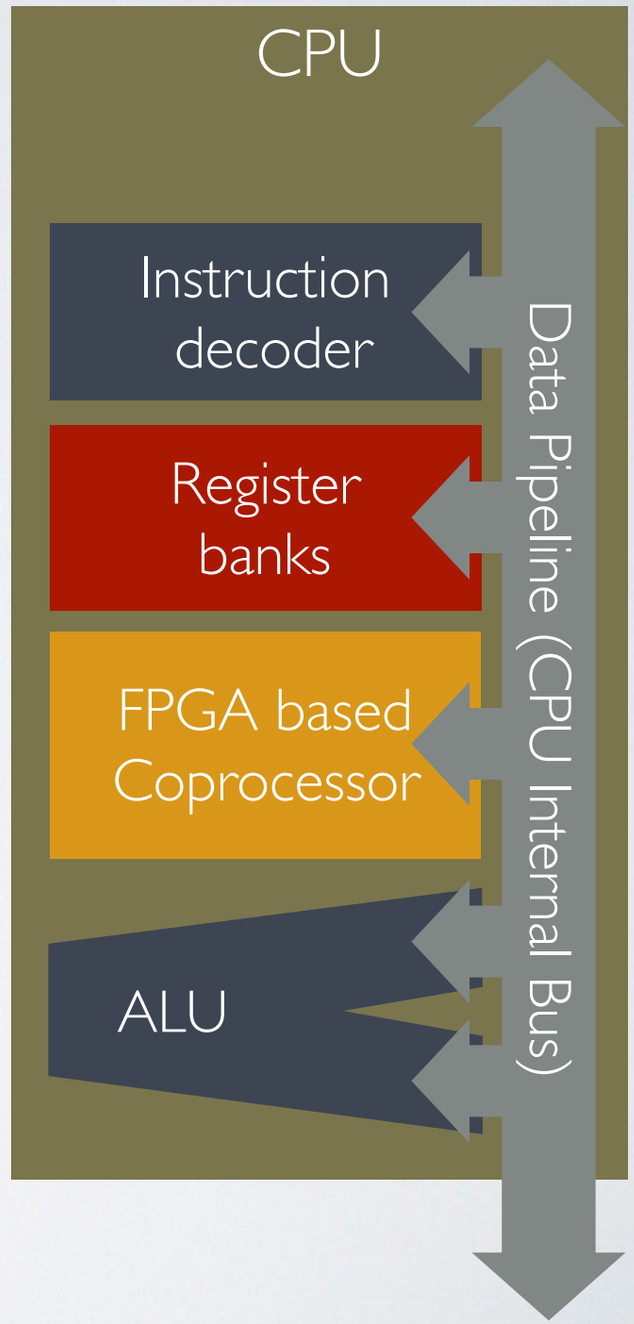
- In HW/SW partitioning the FPGA acts as a coprocessor to the CPU (like a FPU would do)
- Several schemes are possible: for instance one where the FPGA exchanges data with RAM through a DMA and “wakes” the CPU with an IRQ line (e.g: for longer processing and/or larger in/out data transfers)



Connections Scheme: Instruction Pipeline Connection

- A more tightly coupled scheme, useful for operation with smaller operands and faster execution times (otherwise the CPU would get stalled)

- Special instructions, not recognized by the CPU, are processed by the FPGA coprocessor directly, which takes inputs and returns outputs/statuses on the CPU data pipeline (the internal bus)



Hardware / Software Partitioning: Advanced Strategies

- In general, HW/SW partitioning is not a trivial task and several approaches exist in literature to explore the space of solutions.

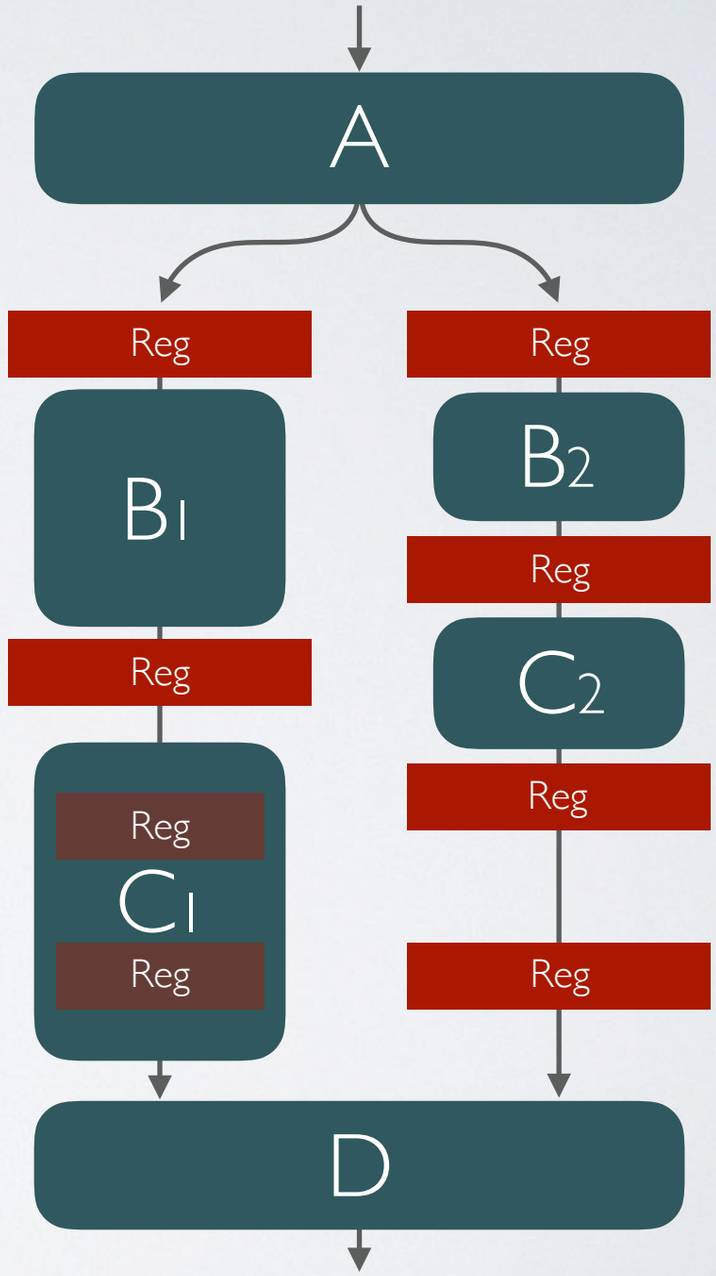
- Examples include algorithms based on simulated annealing, branch and bound and so on…

Communication Bottlenecks

- FPGA processing can be fast, but when HW/SW partitioning is used we also have to keep in mind the bandwidth of communication between CPU/RAM and FPGA.
- Example: a circuit to obtain the mean of arrays of 1024 64-bits values
- Input: 1024 values array from external RAM (size = 65Kb)
- Output: average value to external RAM (size = 64 bit)
- Mean in $\log_2(1024) = 10$ clock cycles @ (let's say) 100MHz = 100ns
- RAM bandwidth (DDR2): 3200MB/s = transfer of 65Kb takes 2.56us
- Transfer time of the array from RAM to FPGA is dominant!

Synchronization Challenges

- Another problem that arises when dealing with complex digital circuits is synchronization: all parts which operate on the same piece of data must be timed coherently.
- For instance, in a pipeline, where stages which operate at different times on the same (or correlated) pieces of data.
- In addition, some stages may have internal registers to shorten combinatorial delays. This also needs to be considered.



Basics of the Design Flow for FPGAs

- Main Steps of the FPGA design flow:
- Hardware Description (HDL/Schematics/etc.)
- Synthesis
- Mapping
- Place and Route
- Bitstream Generation and Deployment

Hardware Description

- The first step in the design flow is the description of the user logic. This can be specified in a broad variety of abstraction levels, including:
  - High level programming languages
  - Behavioral descriptions
  - Register Transfer Level description
  - Schematic Capture

Synthesis

- The behavior of the user logic, as described by the HDL, is translated into a “netlist”, or a low level description of the circuit where the components are basic “primitives”, close to what is physically available in the FPGA fabric (e.g. flip flops, multiplexers, arithmetic circuits, etc…).

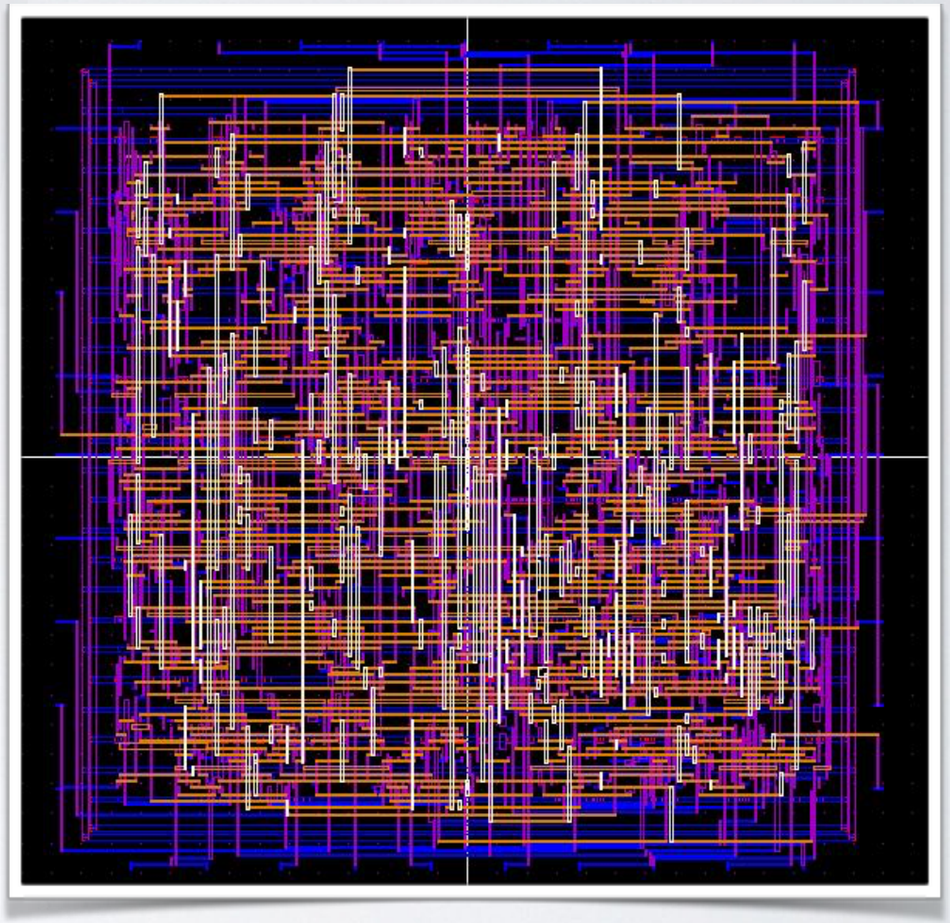
Translate and Map

- The “netlist” generated by the synthesizer is translated on the physical logic cells that compose the programmable fabric of the FPGA.

- The mapping then fits the obtained design into the resources which are physically available on the specifically targeted FPGA chip. This includes logic cells, block RAM but also clocking resources, DSP primitives and so on…

Place and Route

- The logic blocks are then assigned to physical locations on the device based on the given design constraints (which specify timing requirements and the position on the FPGA chip of the I/O ports needed by the user logic design).



Bitstream Generation and Deployment

- The bitstream is a binary file generated from the routed design, ready to be downloaded to the physical FPGA device.
- Deployment on the FPGA is carried out through standard interfaces like JTAG ports and can be volatile or non volatile (e.g: through on board or external SPI Flash memories)

Simulating the Design

- Simulation is a very important step in the design of digital circuits
- A gradual approach to simulation (e.g: bottom up) is recommended to help bugs emerge earlier
- Simulation can be either behavioral (doesn’t take into account delays in the implementation) or time-accurate (either post synthesis or post place and route, requires more resources and can change if the design is re-synthesized)
- In the majority of cases, behavioral simulation already can make any flaw in the design evident.

Take Home Messages

- FPGAs can be powerful allies in intensive data processing applications
- It is possible (and advisable) to focus on specific parts of the algorithm to be implemented in hardware, leaving the rest in software
- Several processing paradigms and system architectures are possible: it is important to choose wisely, according to the specific application/algorithm

Backup Slides

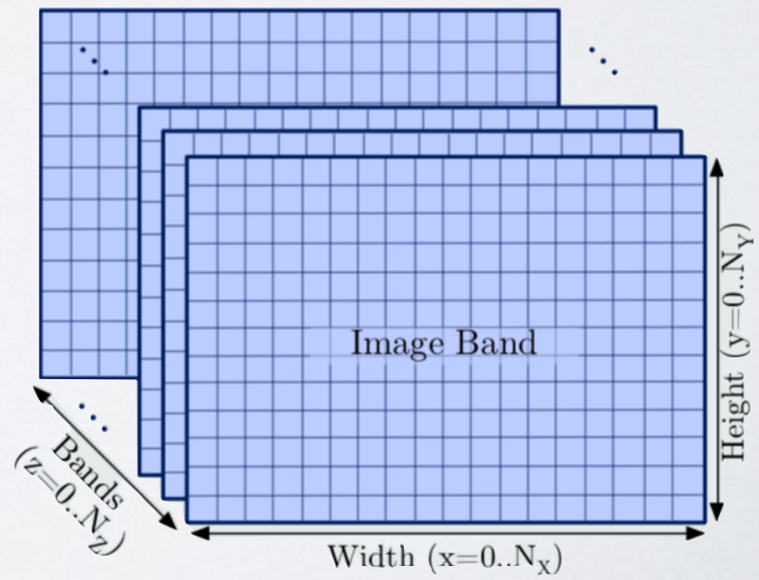
A couple of real life examples

A Standard Compliant Compressor for Hyperspectral Images

Hyperspectral Images are three-dimensional arrays of pixels. Each layer represents a spectral band. Typical image size: 1 billion pixels and over

Host Systems such as satellites and aircrafts: low memory, low power, communication links towards base stations are limited in bandwidth

An efficient compression scheme is crucial

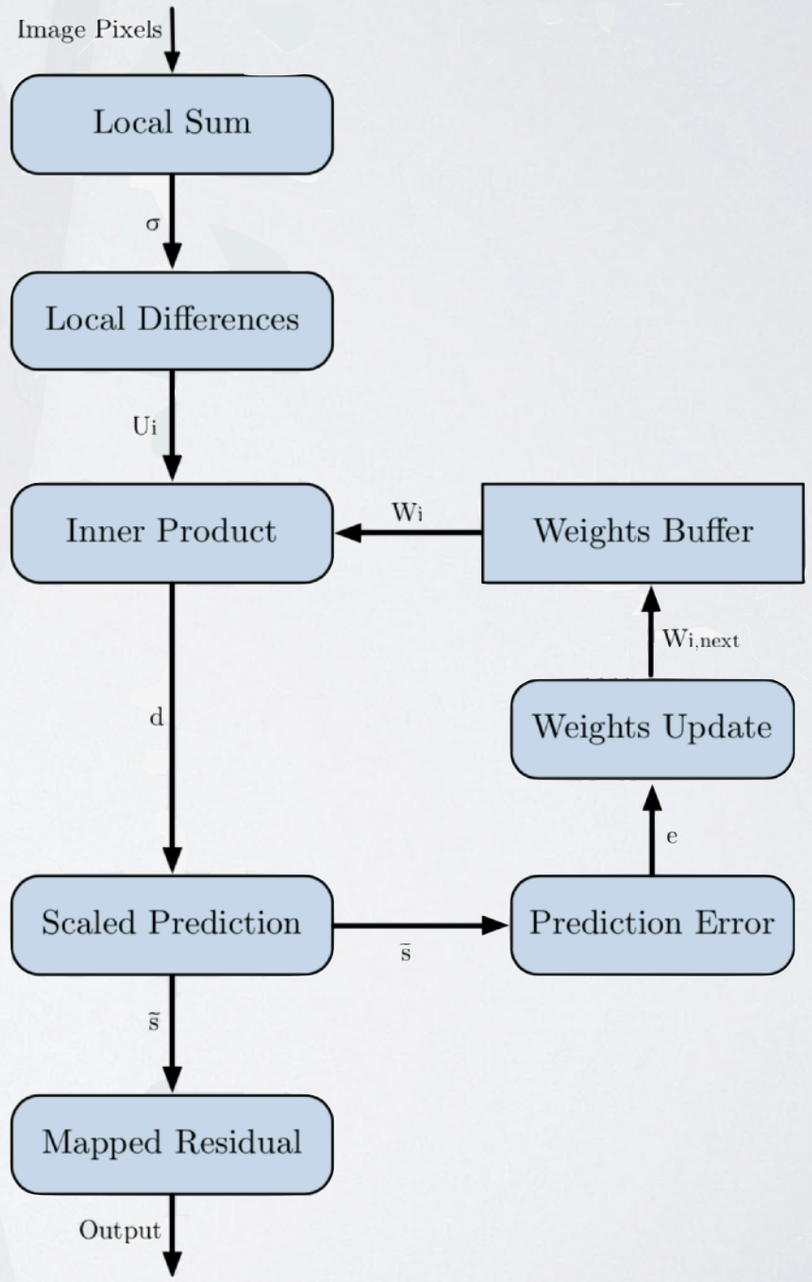


CCSDS 123.0 Standard Algorithm

CCSDS-123.0 is a lossless, causal algorithm. It is linear and follows an adaptive approach: weights are updated at each pixel on the basis of the error in the previous pixel prediction.

Number of arithmetic operations per single pixel = 100 ca. (rough estimate)

…and this is NOT the number of instructions per pixel!



Constraints and Strategies

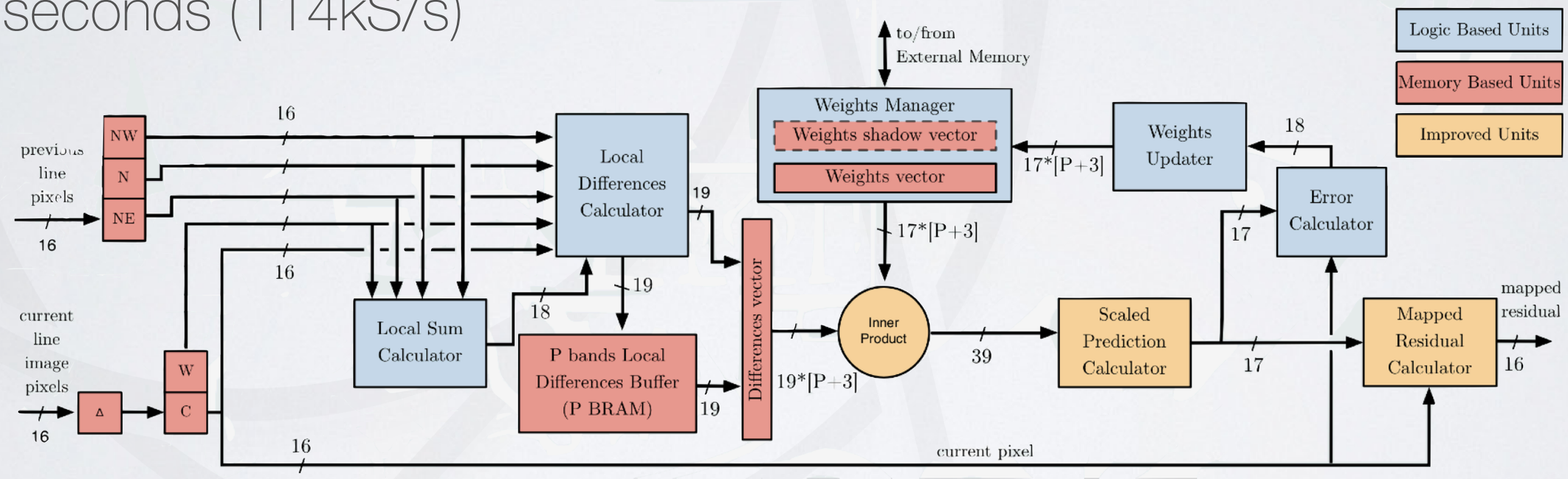
- The algorithm needed to be implemented on a space graded platform. As a consequence no hard-CPU core adoption was possible

- High performance was a crucial requirement (throughput over 50 MSamples/sec)

- The full algorithm was translated into a digital circuit with a pipeline approach: different pixels enter the pipeline in different clock cycles and, at full speed, one sample per clock cycle is processed

Implementation Results

- The time for execution of the standard software algorithm on an image of 1024x1024x6 pixels (which is quite small compared to the average dataset) on an intel i7 workstation with 32GB of RAM is about 55 seconds (114kS/s)



- Throughput of the circuit: 1 sample/clock @ 55MHz : 55MS/s.

- Acceleration factor = 482.5x ca.