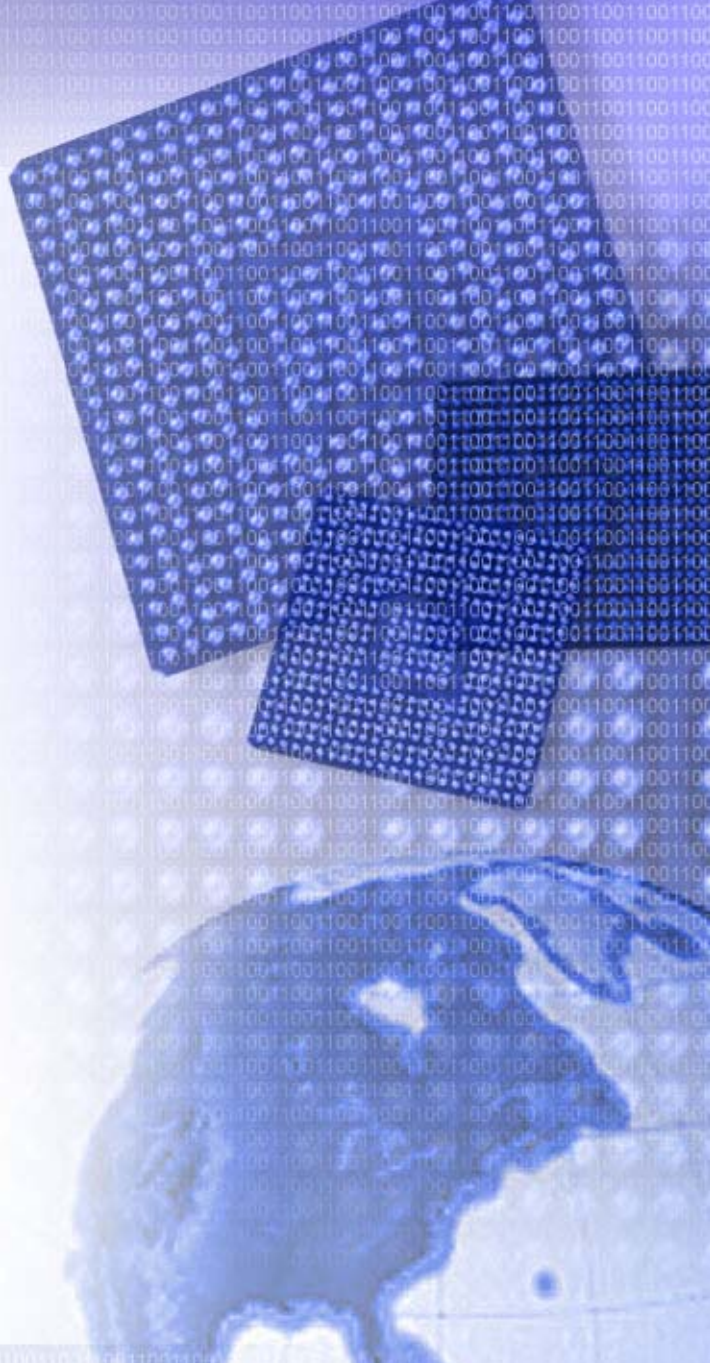
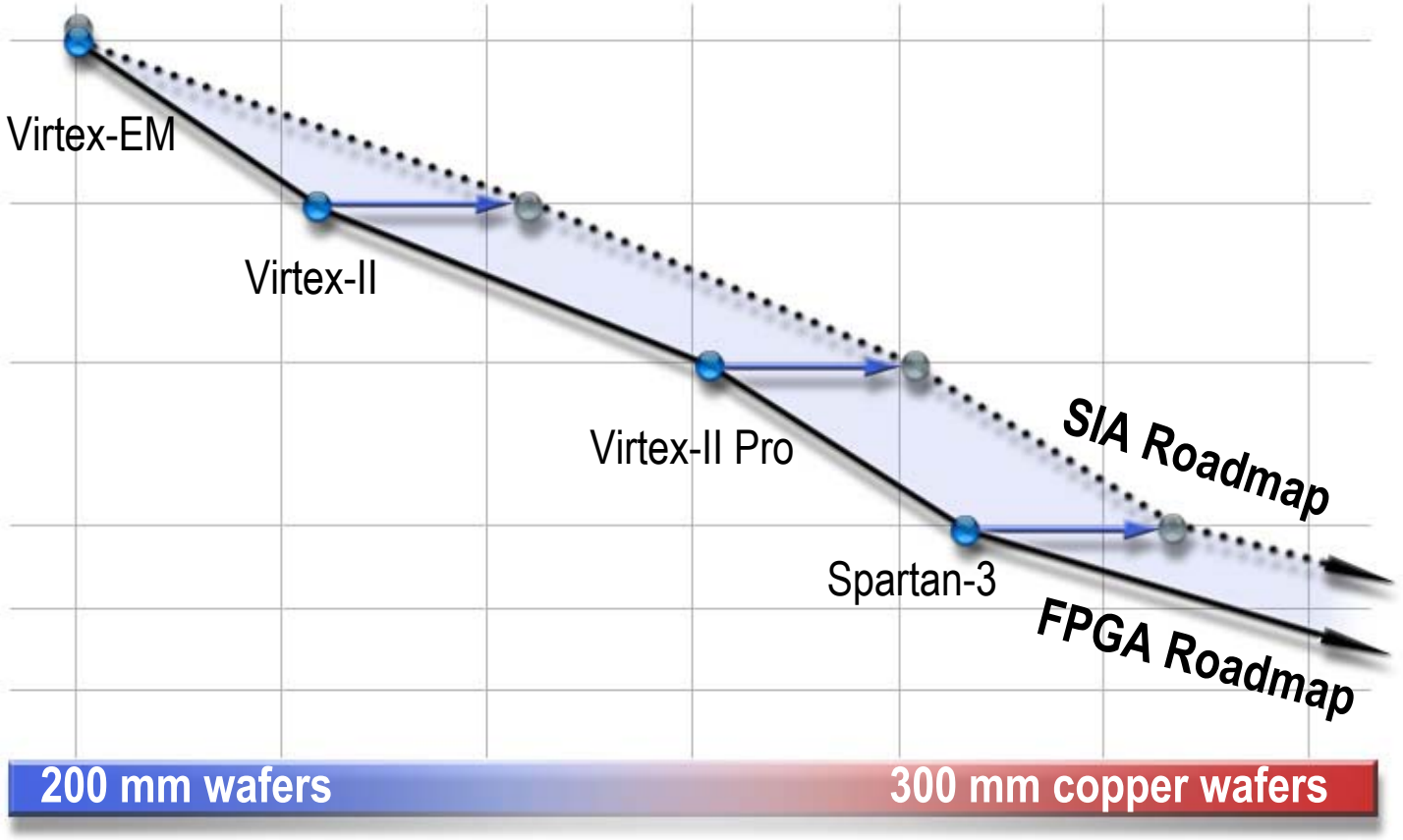
XILINX®

Challenges and opportunities for FPGAs

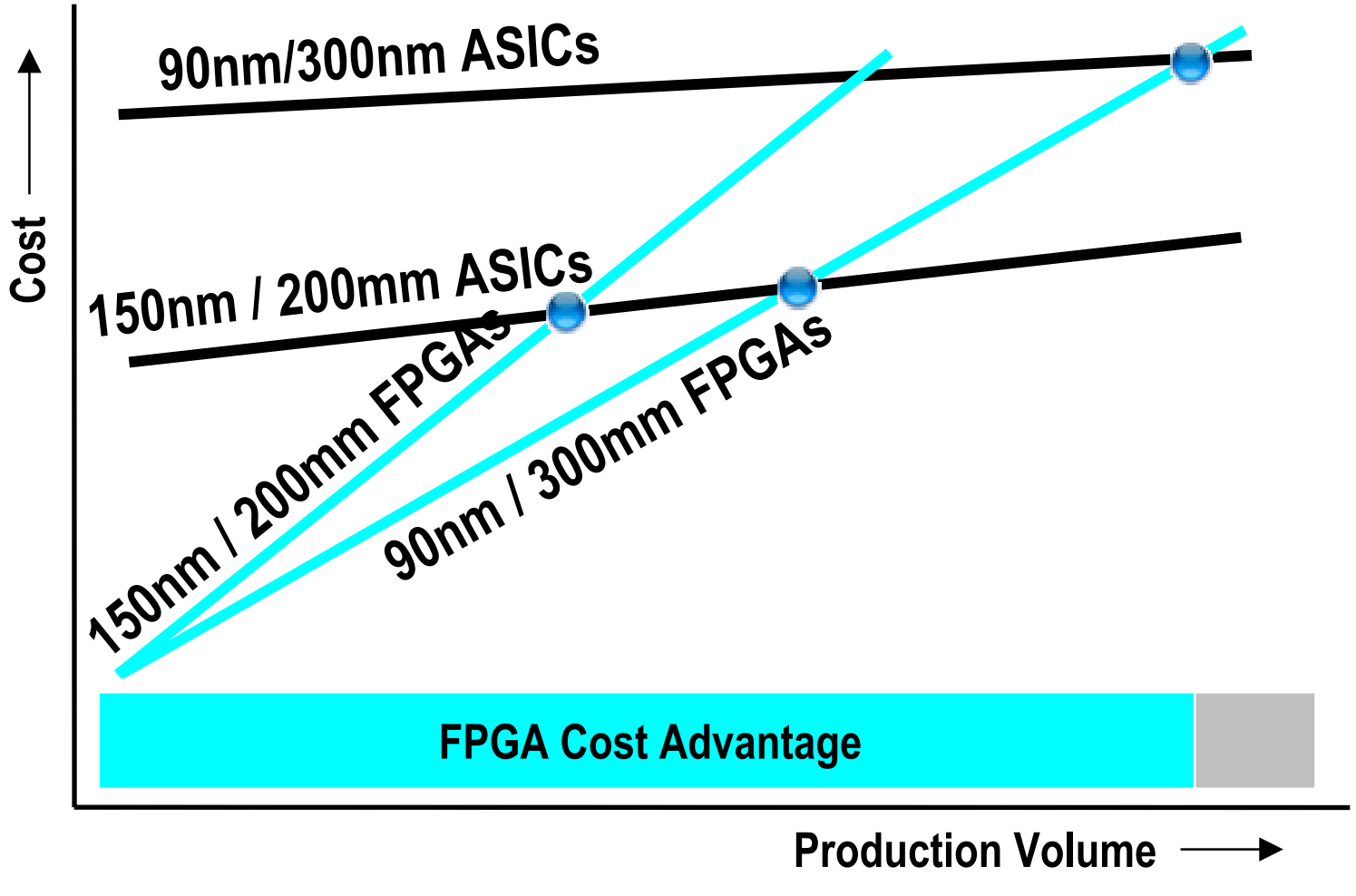
Ivo Bolsens



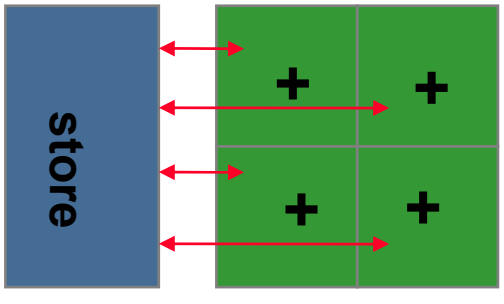
FPGAs ride the tide of Moore's Law



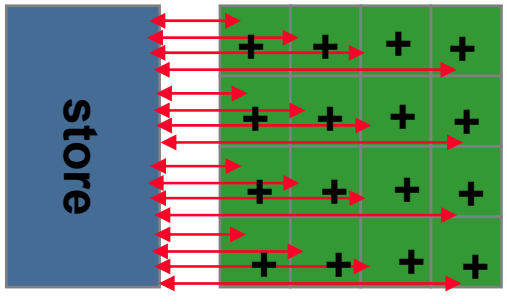
ASICs buck the tide



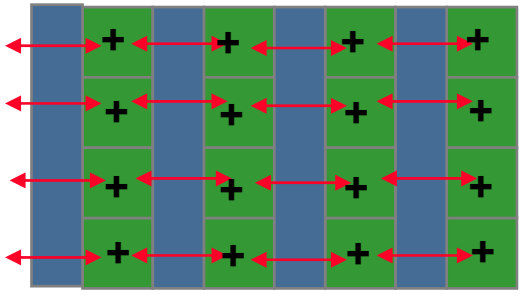
Future proof FPGA architecture



λ



λ/2

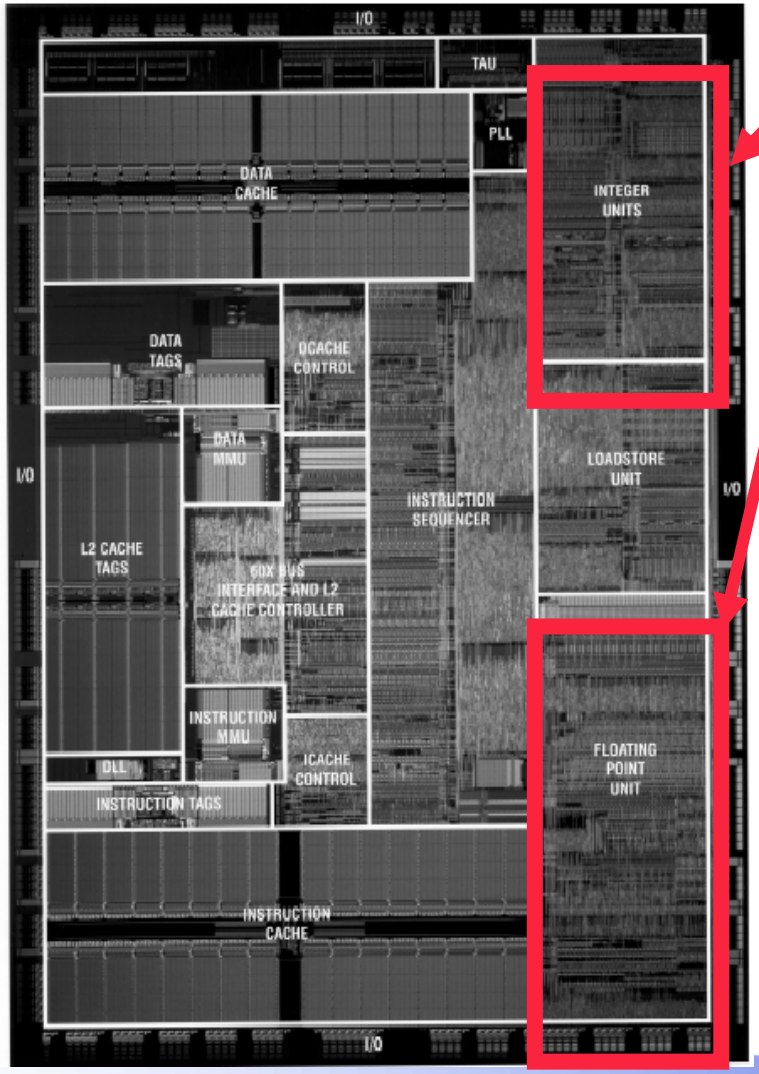


λ/2

$$\text{Tconnect (nsec)} - \rho.l^2/\lambda^2$$

Courtesy :IMEC

Microprocessor



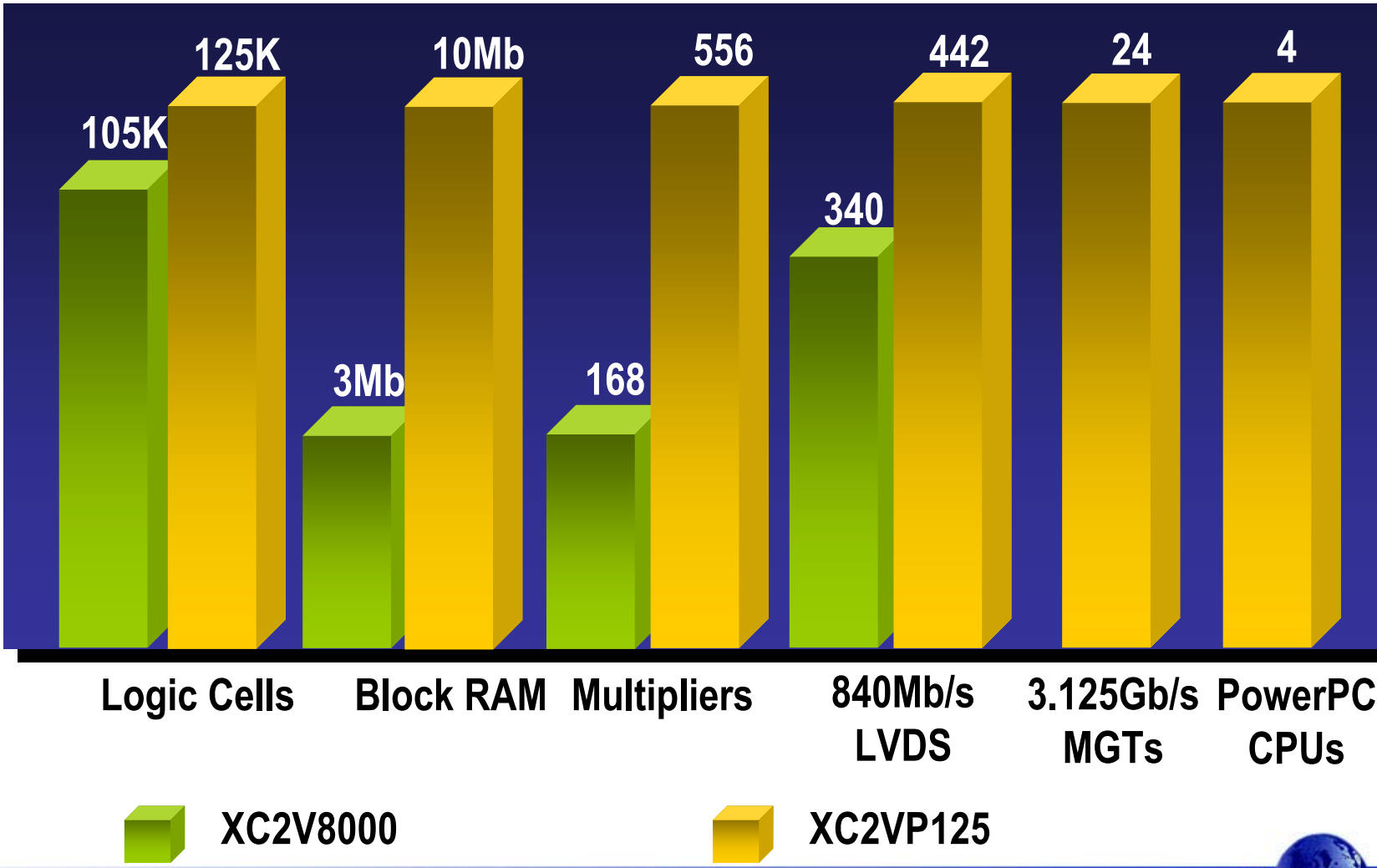
The only circuitry which supports "useful operations"
All the rest is overhead
to support the time multiplexing and data transfer/storage

f_{proc} = 450 MHz
= 900 MOPS

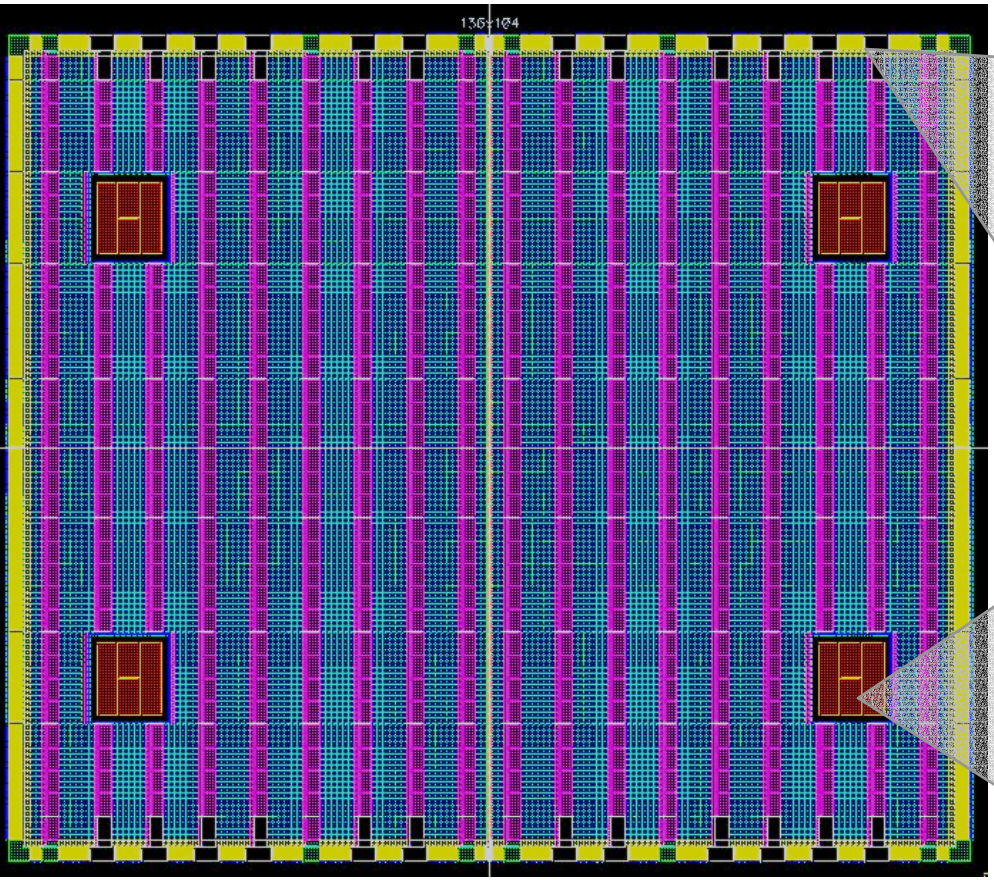
f_{fpga} = 300MHz
= 150 BOPS

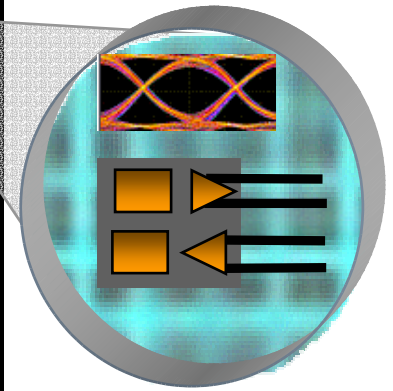
Source : Bob Broderson

Redefining FPGA



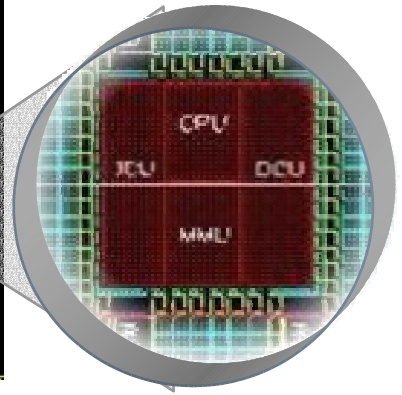
Virtex-II Pro Programmable System Platform





RocketIO High-speed Serial Transceivers

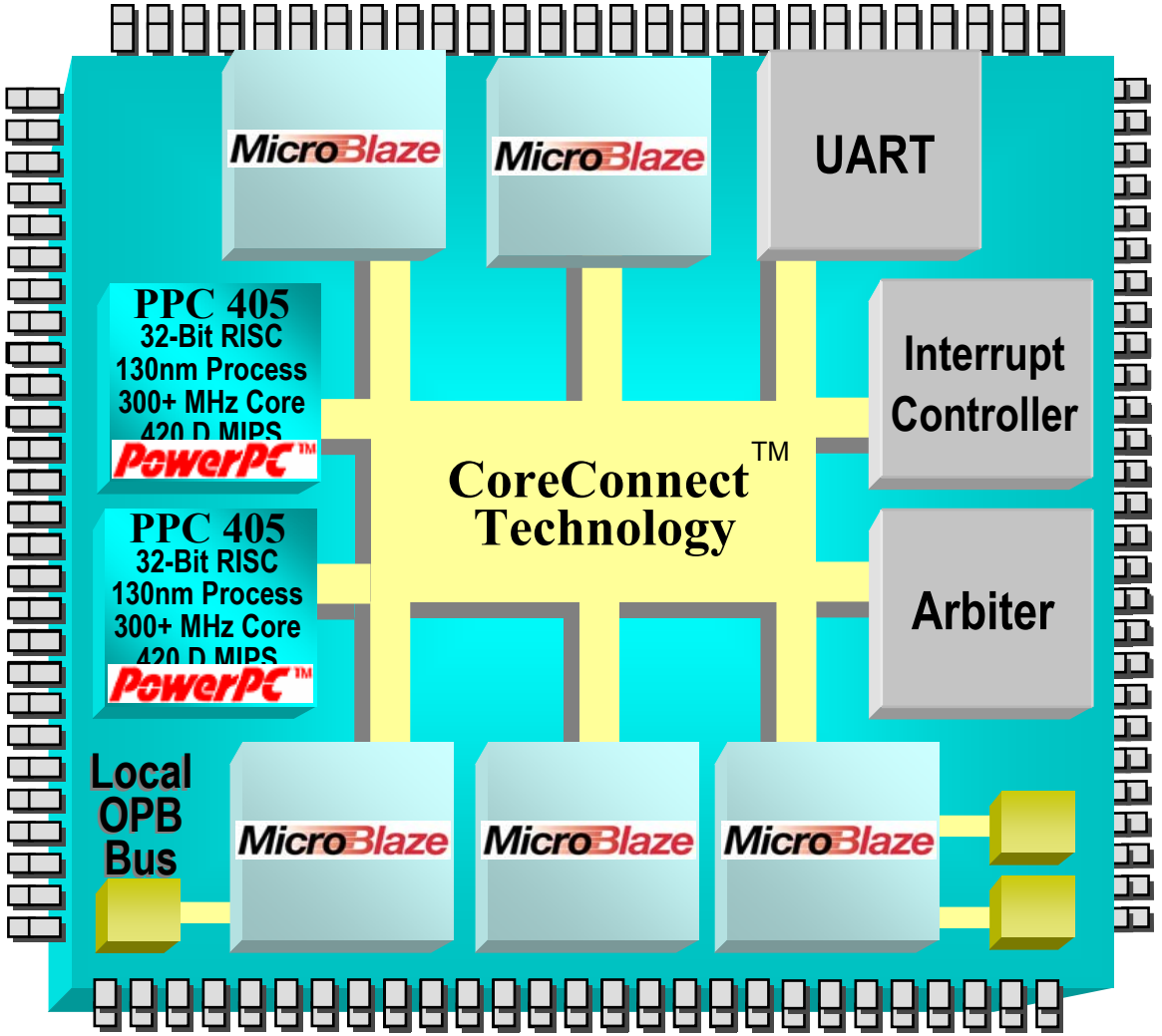
Up to 24 transceivers
622 Mbps to 3.125 Gb



Industry standard PowerPC Processor

Up to 4 processors
600 DMIPs at 400 MH

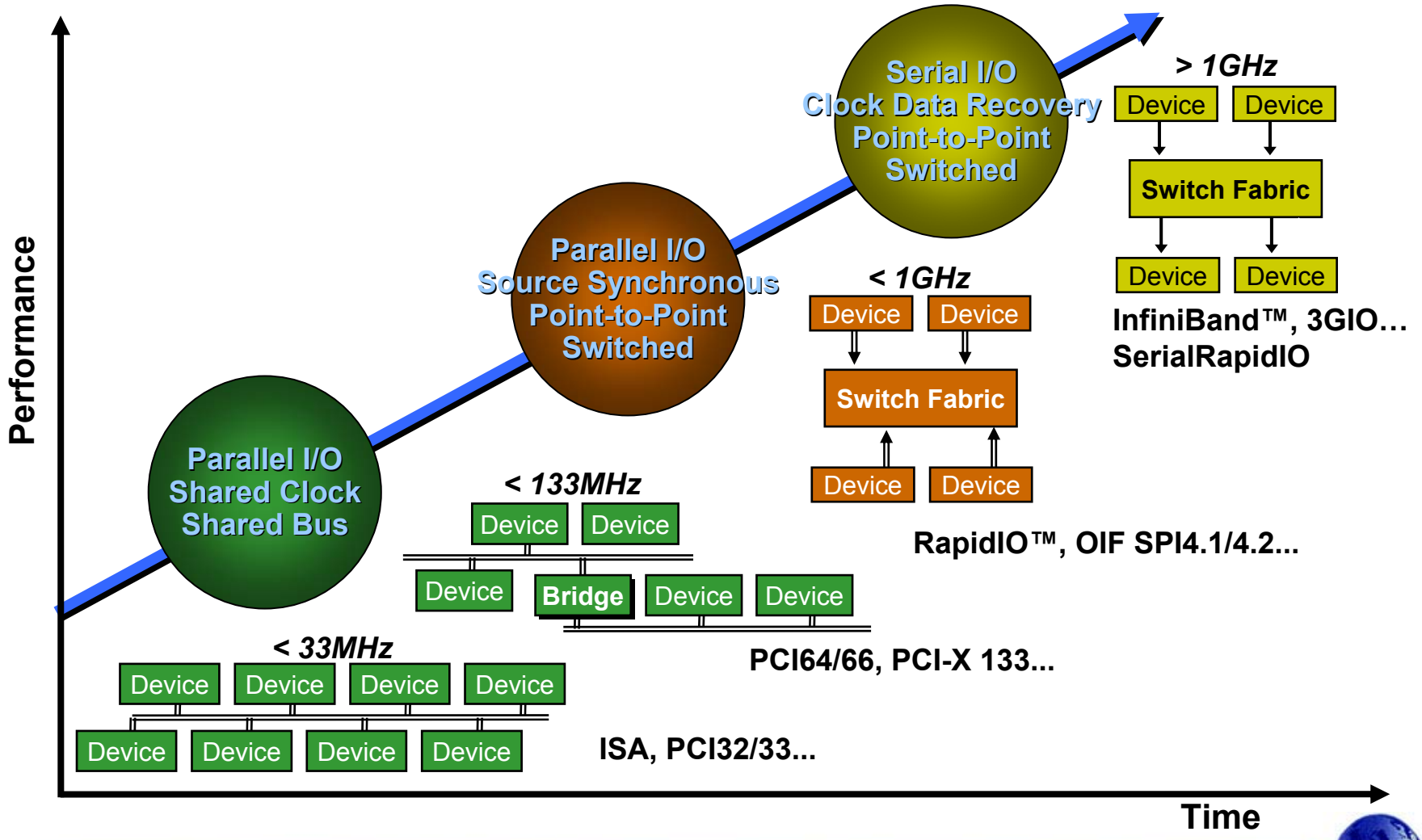
Platform Design Concept



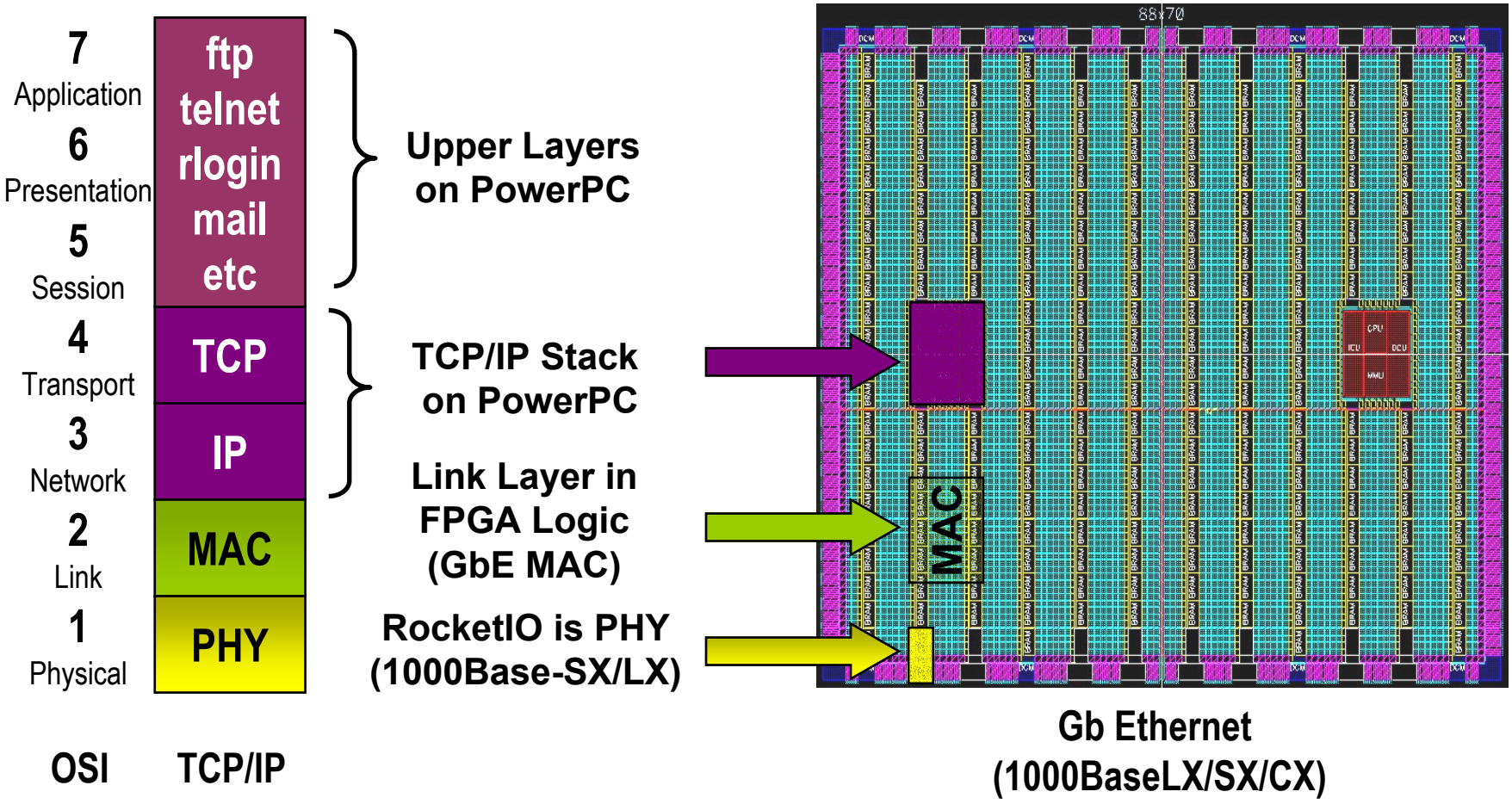
The System Components

- Hardwired CPUs
- Softwired CPUs
- Configurable CPUs
- Softwired Logic
- Hardwired Logic
- Reconfigurable Logic
- RAM
- Busses
- Networks
- Reconfigurable interconnect
- Input / Output

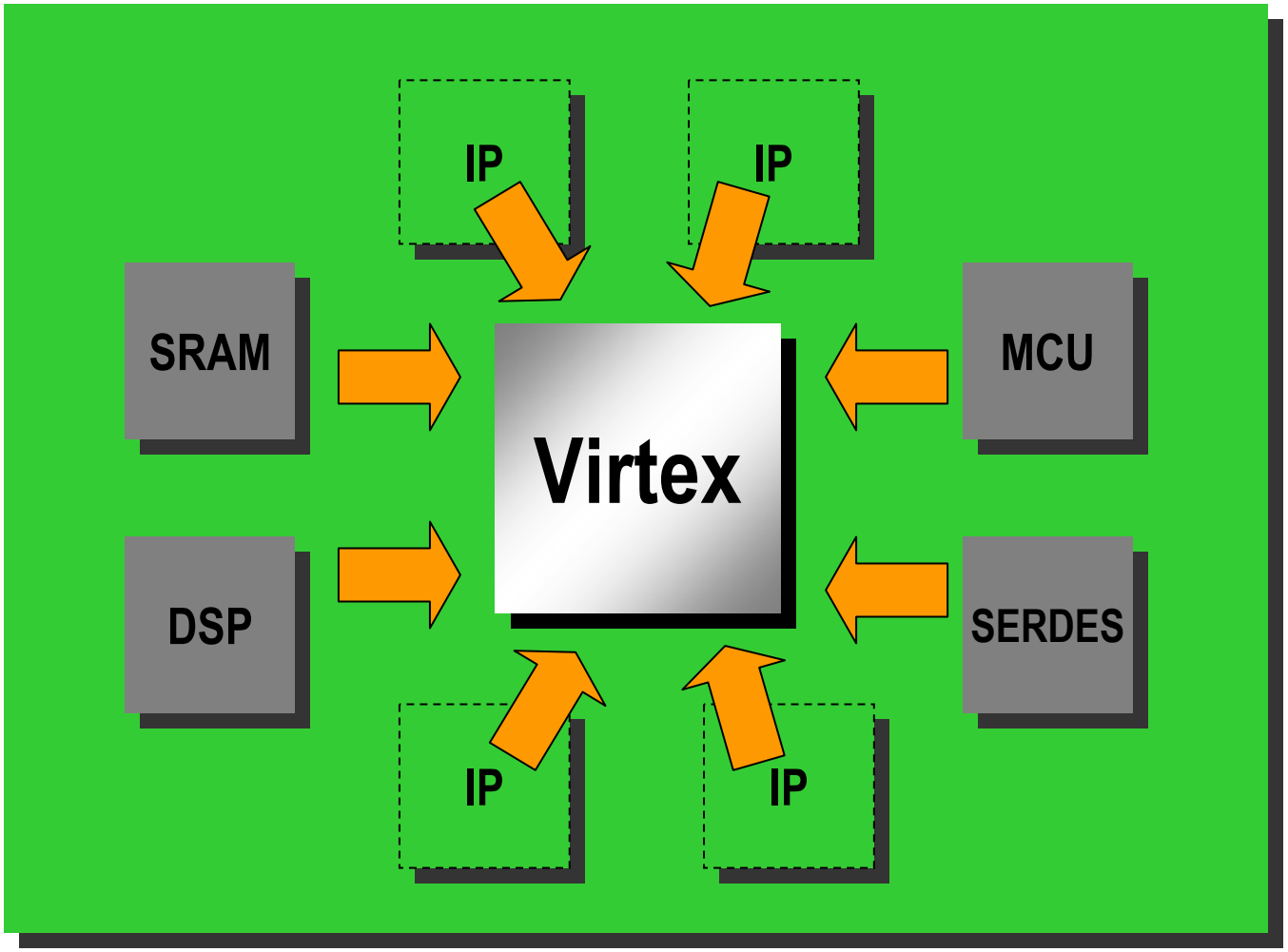
Serial Connectivity Trend



Creating Complete System Solutions

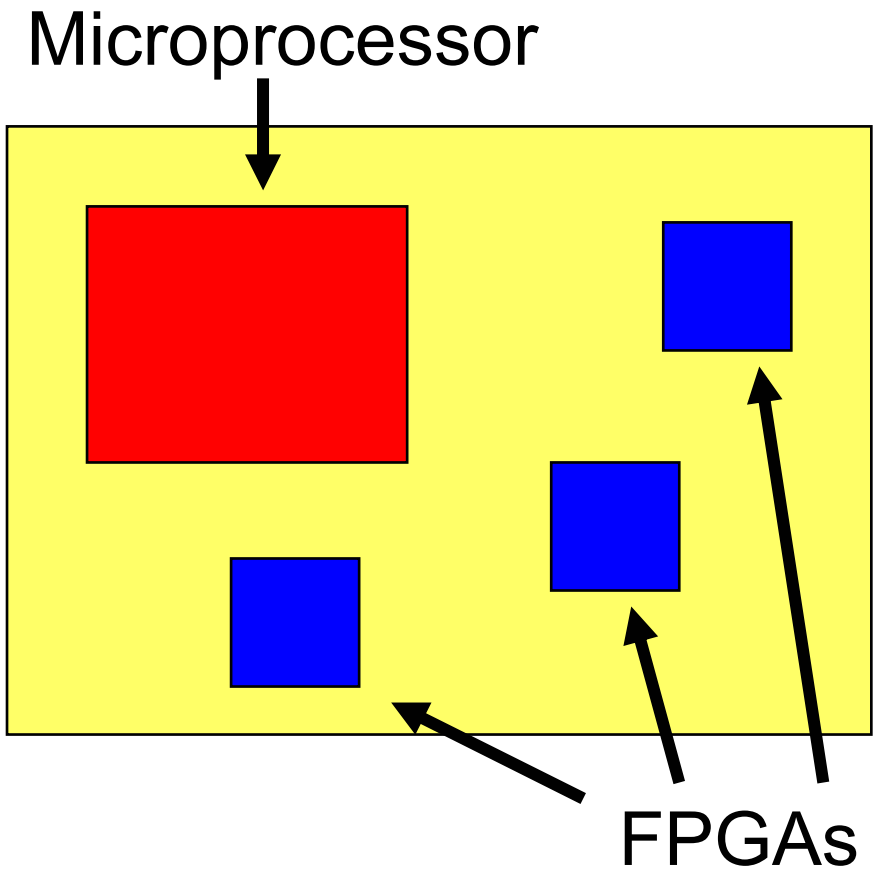


The “Virtex Vortex”

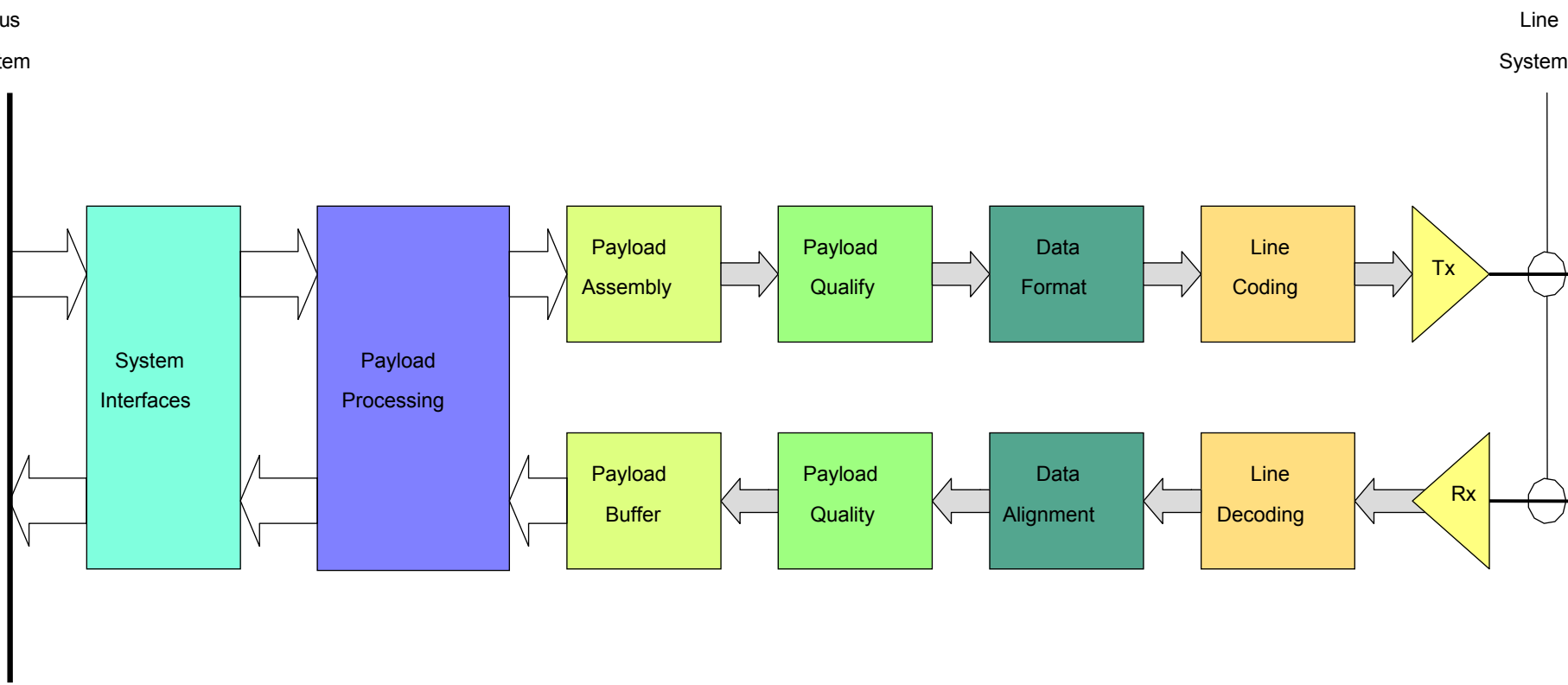


Board-level ancestry

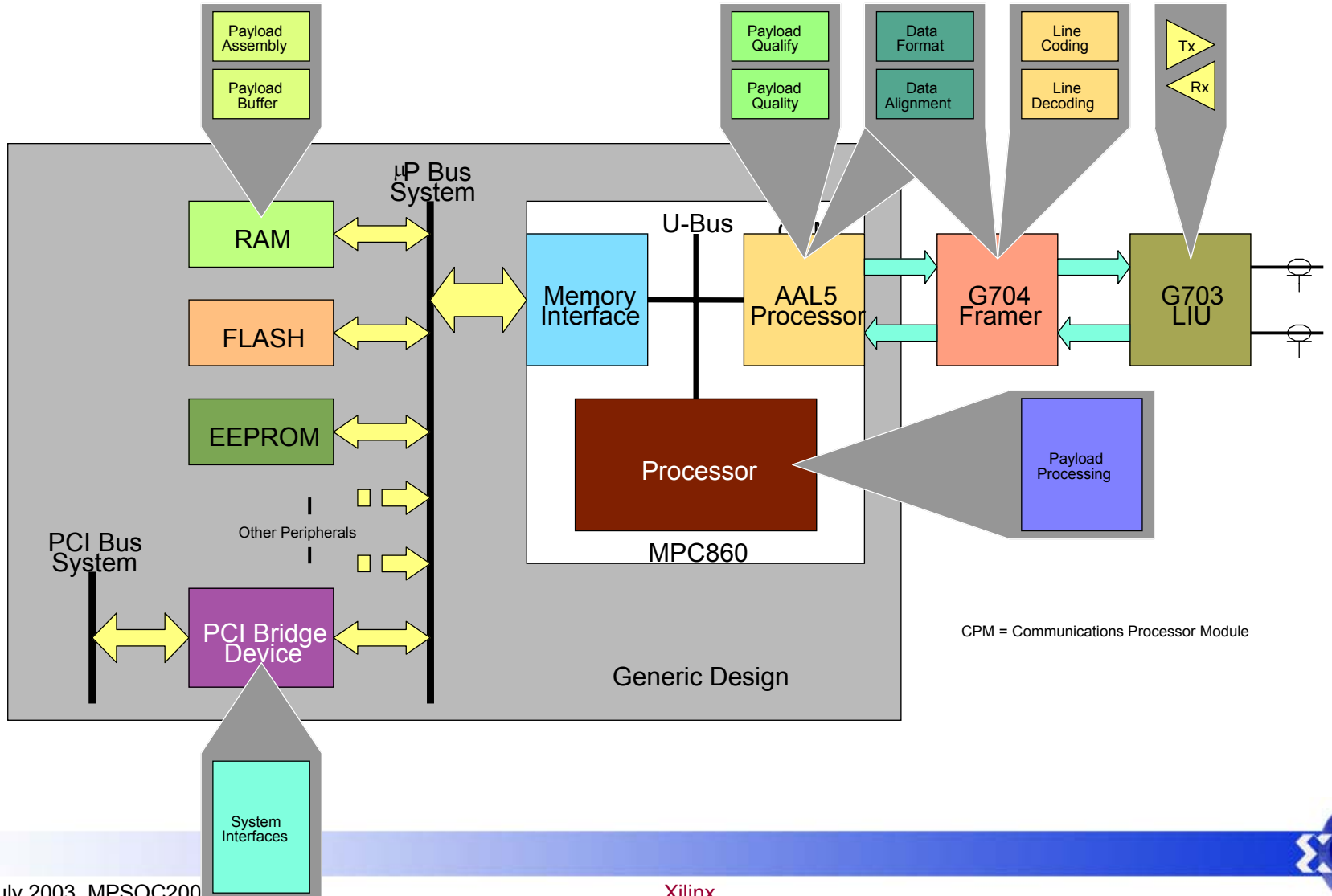
- Microprocessor is primary functional component on board
- FPGAs may be
  - secondary functional components (e.g. for accelerators, interfacing)
  - or just tertiary functional components (e.g. for glue logic, prototyping)



System Exploration in Platform FPGA

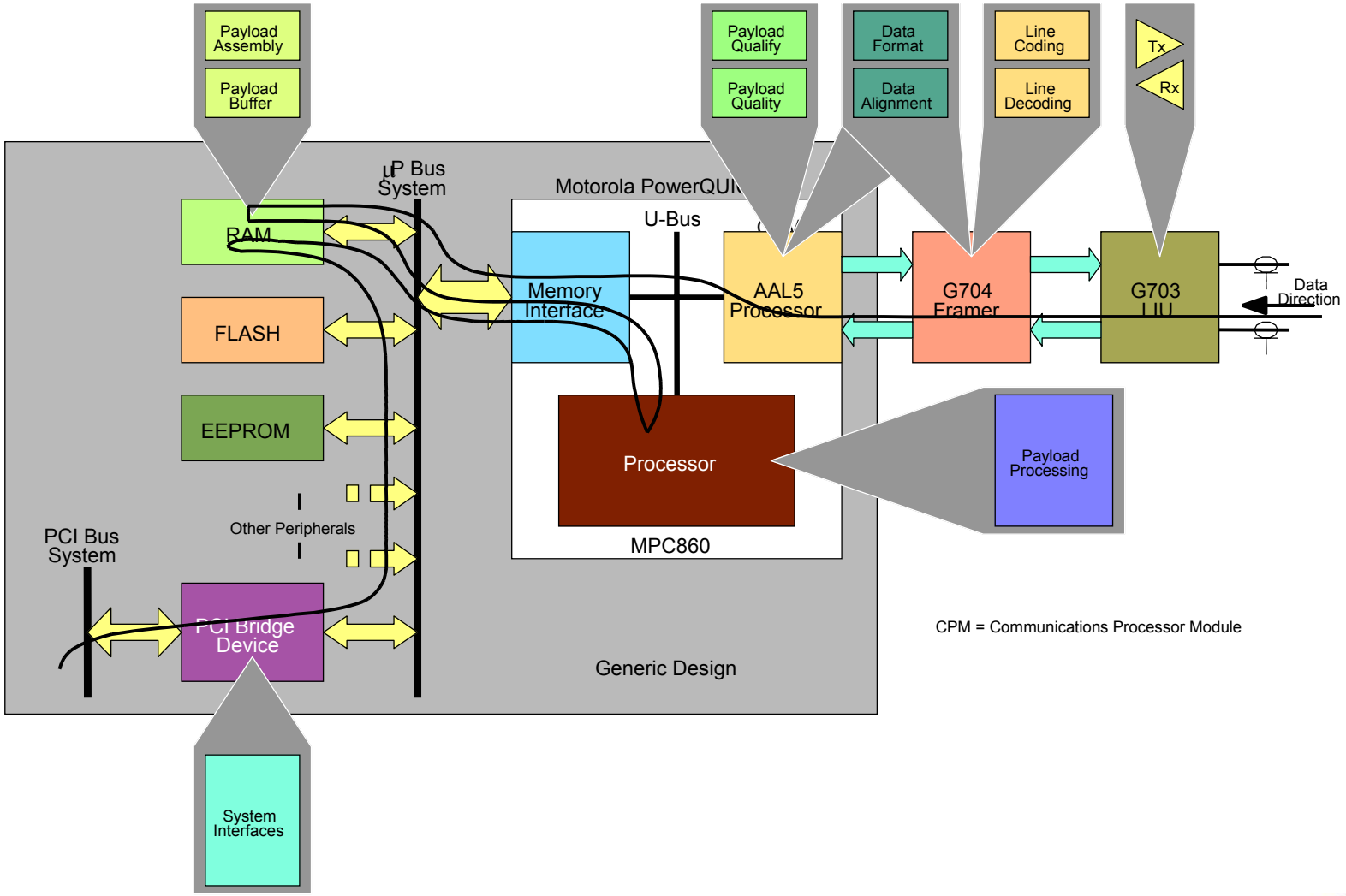


Board Architecture

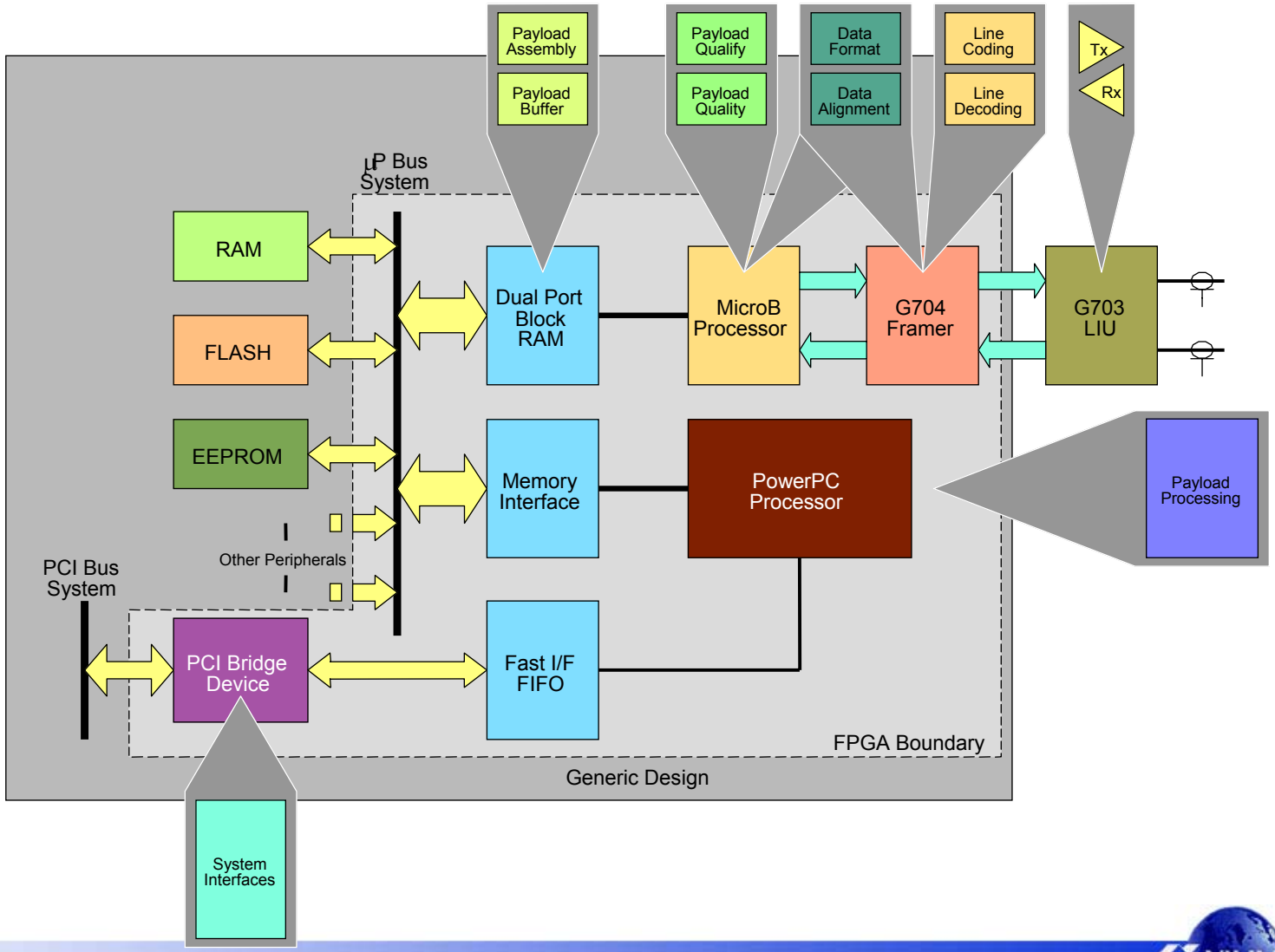


CPM = Communications Processor Module

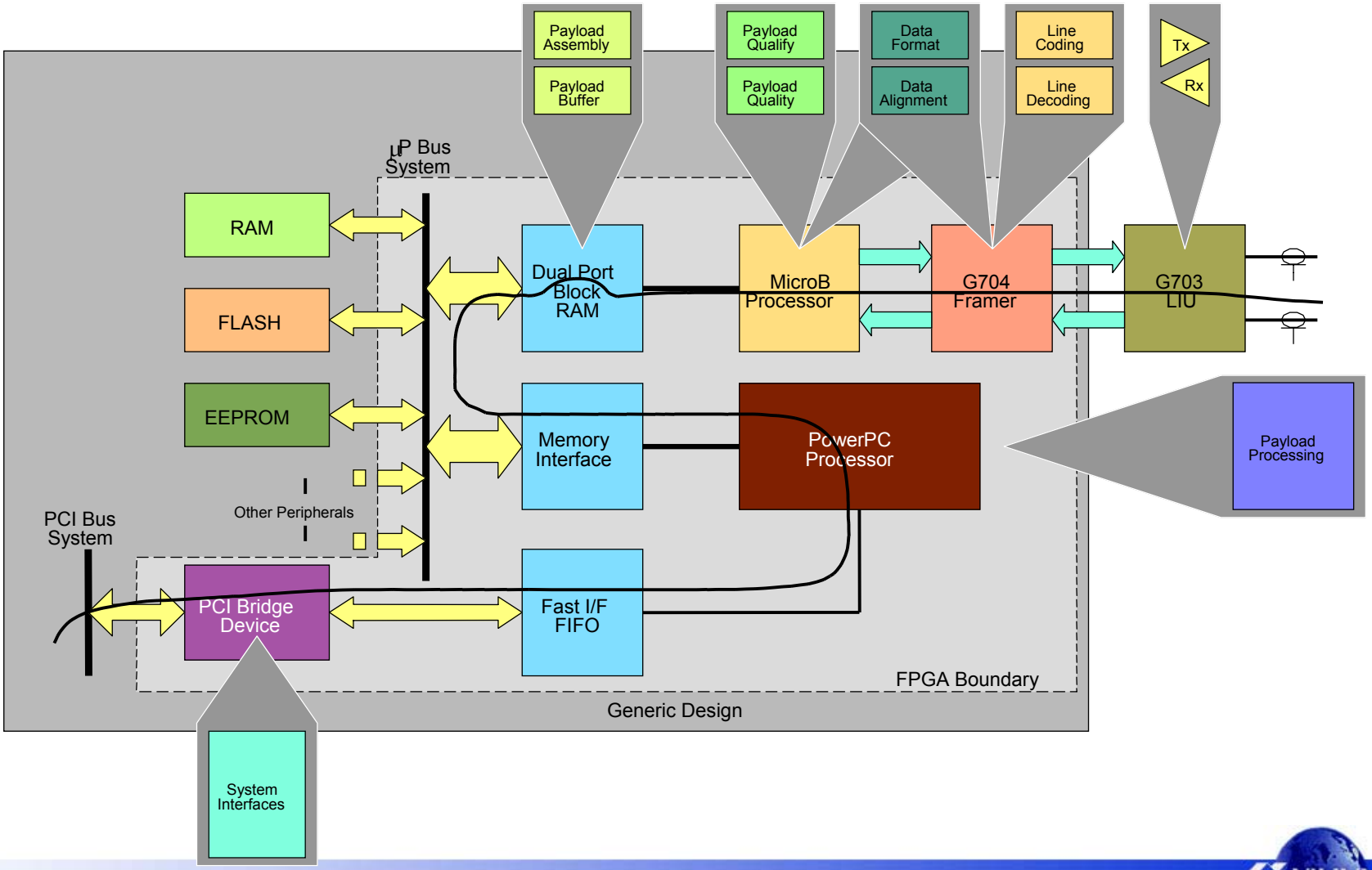
Communication Bottleneck



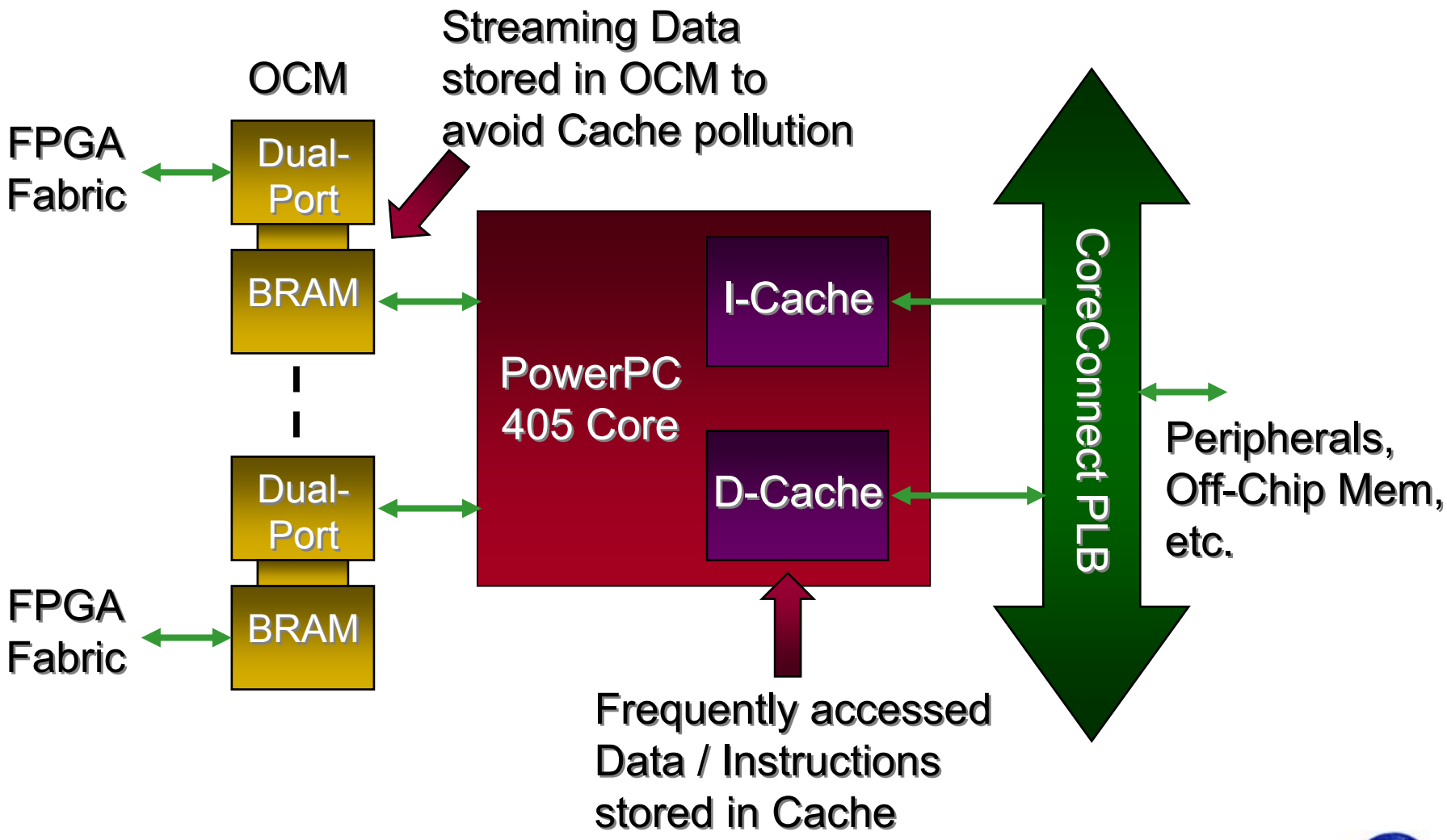
Optimized Architecture



Eases Bus Bandwidth



Data Processing Efficiency



Shrink to single chip

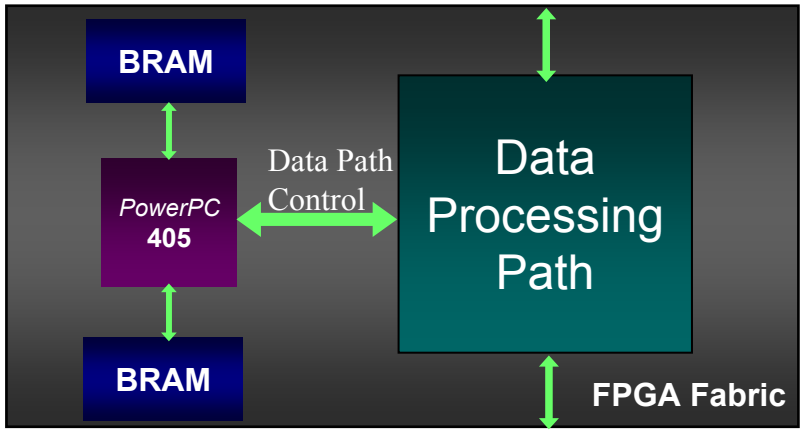
- Two ways of interpreting board to chip mapping:
  - Microprocessor with embedded programmable logic
    - Natural shrinkage of board-level model
    - Example: possible use of embedded FPGA in SoC
  - Programmable logic with embedded microprocessor(s)
    - Platform FPGA: inversion of board-level model
    - Example: PowerPC(s) in Xilinx Virtex-II Pro
    - However, still open to processor-centric interpretation

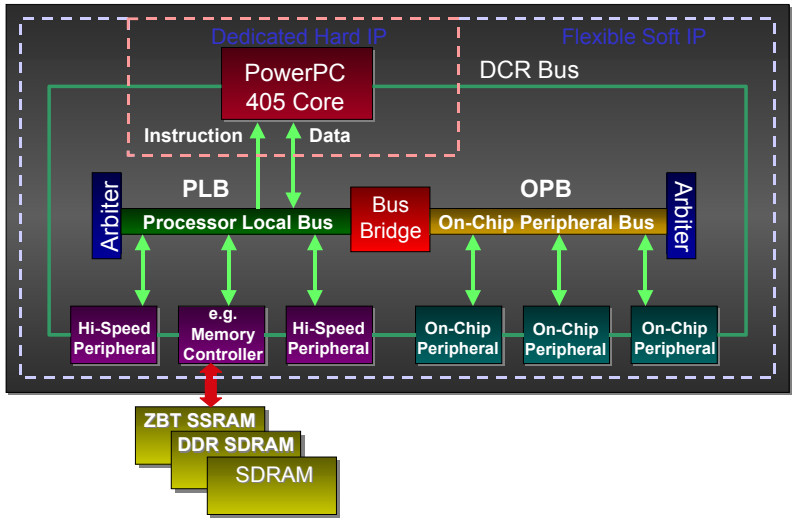
Question

- Other creative ways to use
  - Concurrency
  - Programmability
  - Interconnect topology

- YES : Interface centric

Processor Use Models





Buried Processor

- Processor runs from BRAM only
- No external pins, no RTOS, no peripherals
- Typical use: packet processing, control functions

Embedded Computing

- Processor runs from large external memory
- CoreConnect bus structure, peripherals
- Typical use: running embedded software applications on RTOS

Interface-centric architecture

- Is appropriate for reactive systems - highly relevant for future ambient intelligence/ubiquitous computing
  - 'Dissapearing computer'
- Processors have no special status in systems, and play only a secondary role as 'function units'
- Explicit 'hardware-software co-design' becomes lesser issue - certainly no top-level partitioning
- Hardware accelerators of processor-centric model are inverted and replaced by 'software decelerators'

Software decelerators

- Processor executes software to perform one or more services for programmable logic
- Termed a ‘decelerator’ because execution is likely to be slower than for a logic-based implementation
- Rationale for use includes:
  - speed only has to be adequate to meet system deadlines
  - may save chip real estate and/or energy overall
  - potential simplification of implementation

Network processing

= DSP

- Dataflow
- Interface driven

= Dataprocessing

- complex data structures
- irregular processing

<> Lightweight processing

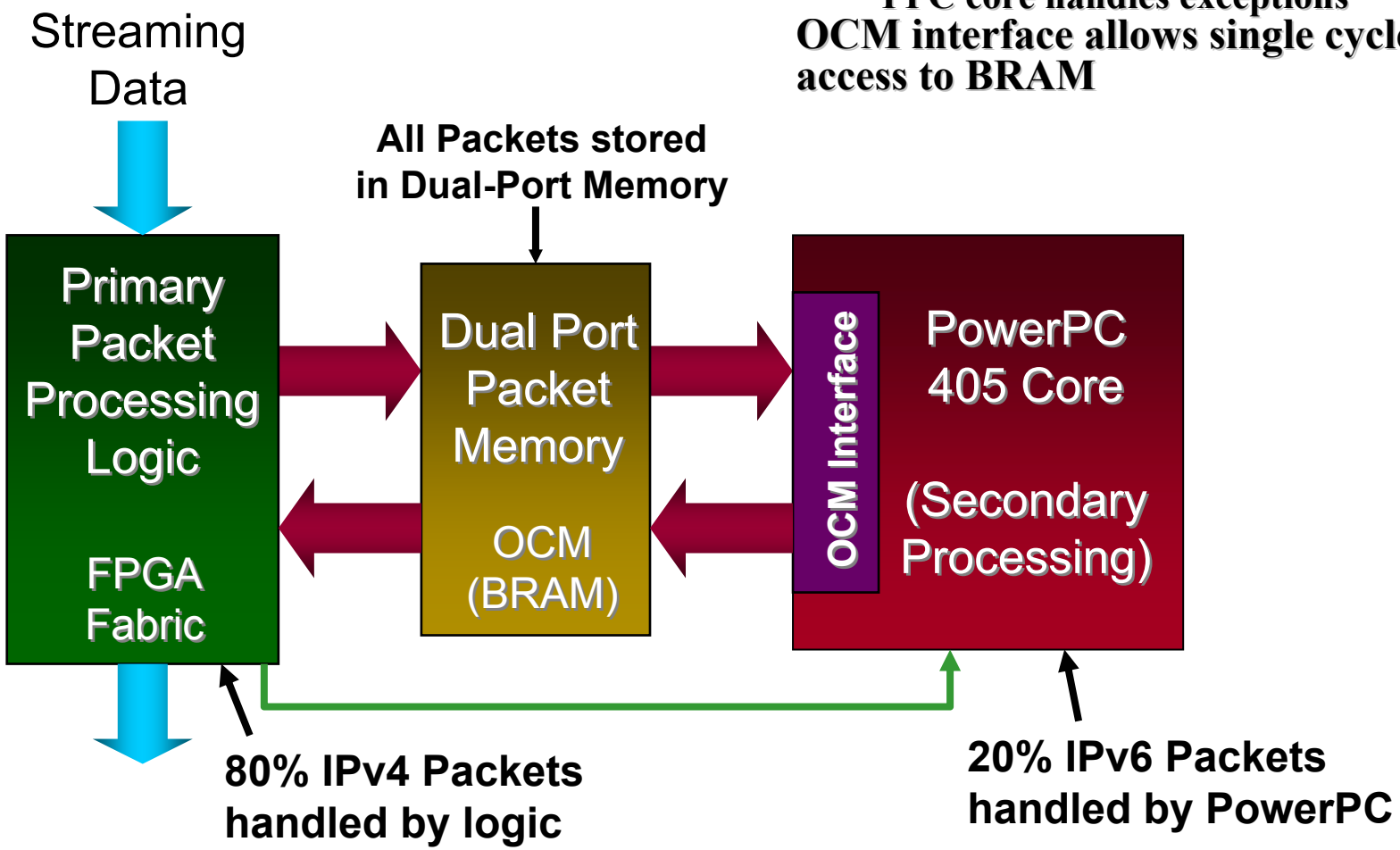
Example:Gigabit packet processor

- Four-port mixed-version IPv4 and IPv6 packet router implemented on single Virtex-II Pro chip
- Goal: 80% of packets handled entirely in the programmable logic, with zero latency
- PowerPC just used as an *assistant* to handle infrequent or unexpected types of packets
- Could dynamically change assignment of functions to logic or to program during system operation to reflect changes in packet traffic profile

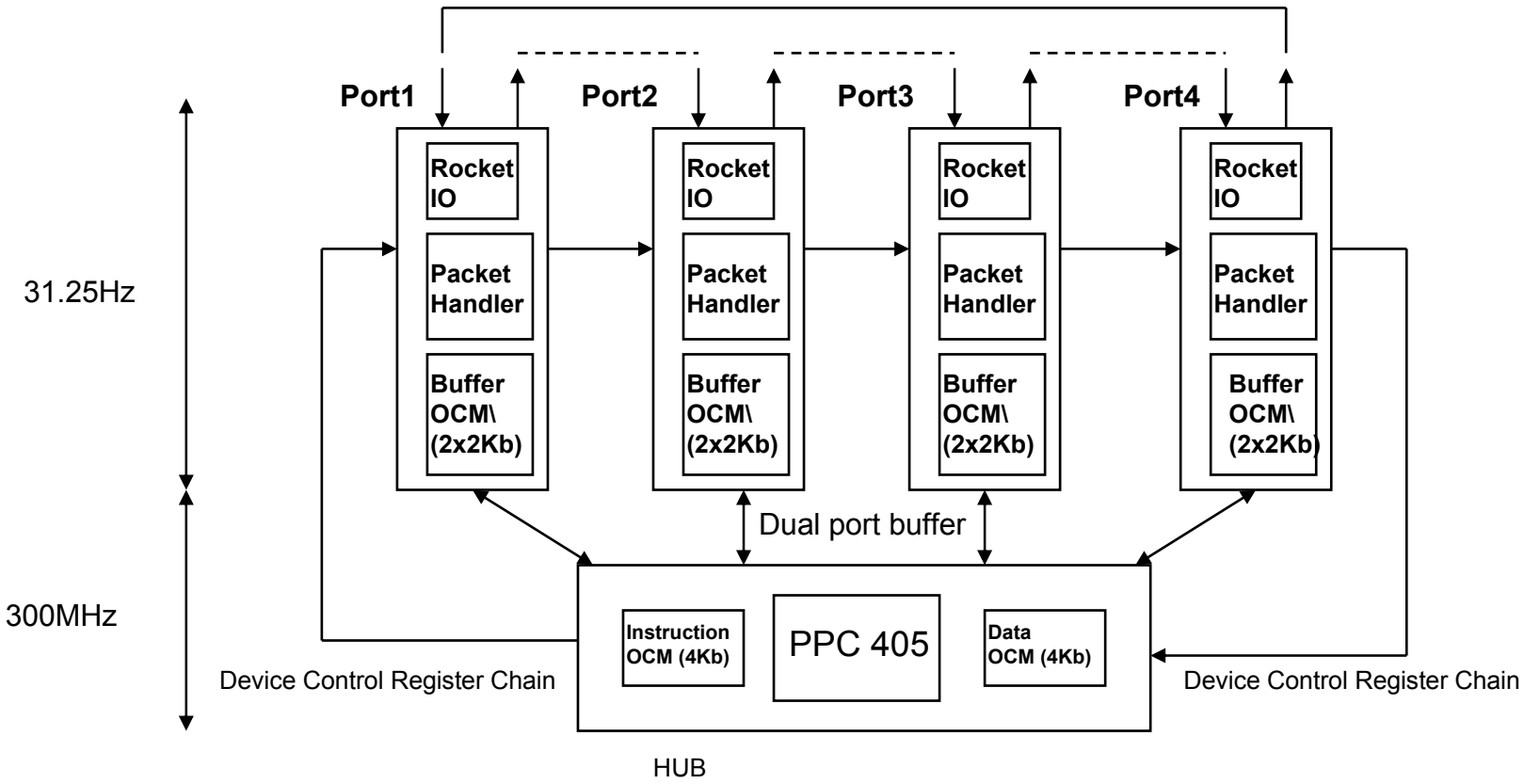
Application Specific Memory

Optimal HW / SW partitioning
- **FPGA logic handles common case**
- **PPC core handles exceptions**

OCM interface allows single cycle access to BRAM



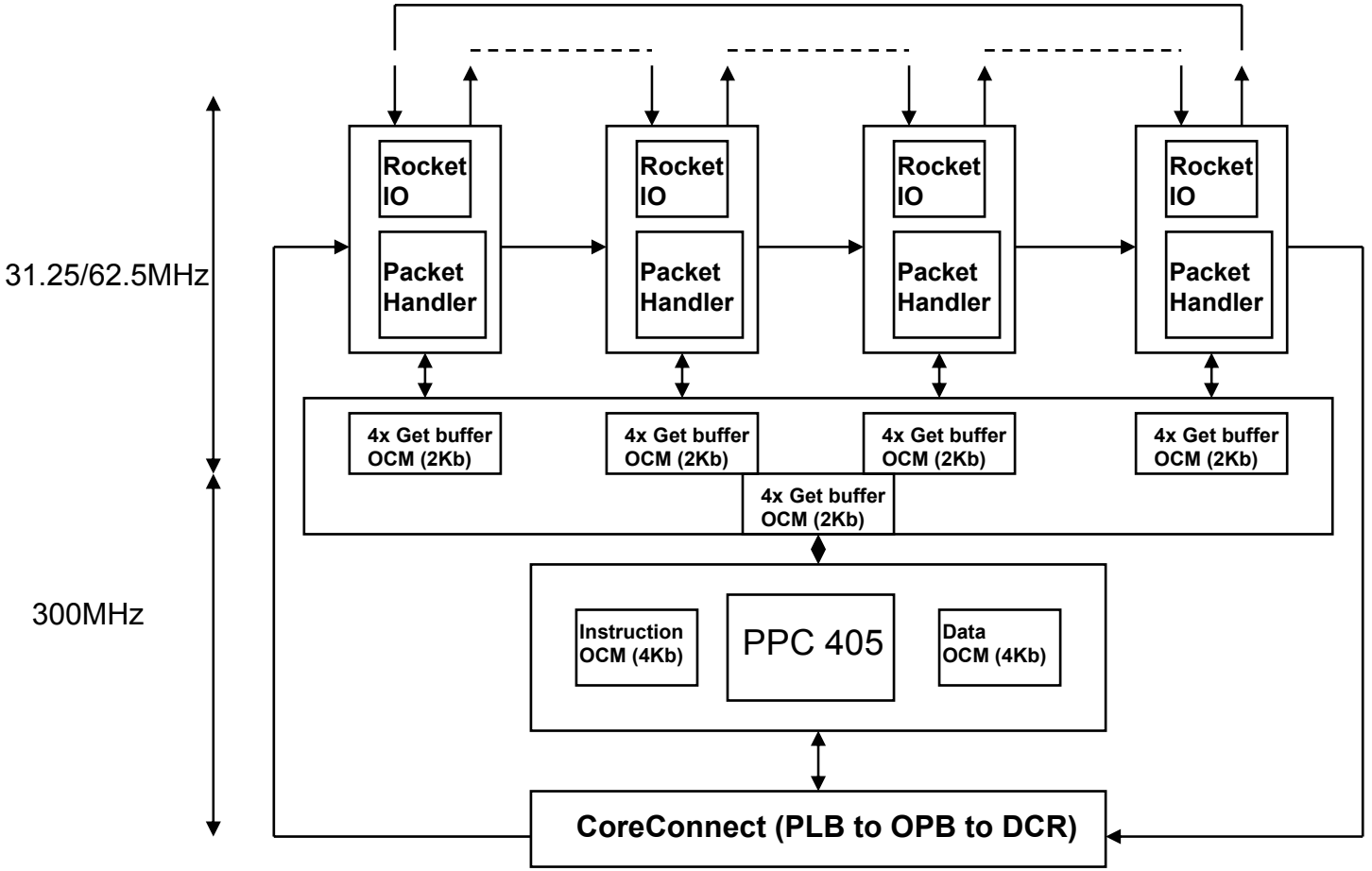
Processor Centric



200% overhead for moving packets between ports and hub

Source : Gordon Brebner, FCCM 2002

Interface Centric



ource : Gordon Brebner, FCCM 2002

Example: address lookup

- Need IP packet address lookup within schedule imposed by zero-latency logic-based packet handling
- Direct logic implementation, or indirect logic implementation using CAM, is possible but expensive in terms of resources used
- Software decelerator option is to use more sophisticated algorithms and data structures implemented on a processor, while also minimizing logic/processor interface overheads

Lookup results

- In the case study, have 240 ns to perform lookup
- Using a hashing algorithm programmed in C on an embedded PowerPC, can perform a lookup within 164 ns for 99.5% of cases
- With more subtle memory management for the hash table, and careful implementation, expect average lookup in about 30 ns, with less than 0.0001% chance of not completing within 240 ns

Example: finite state machines

- Use of software decelerators to implement a general class of sequentialized functions that are ubiquitous and recognizable in digital designs
- Processor has to determine next state and state outputs to meet schedule determined by logic-based system including the state machine
- As long as overall schedule is met, decelerator might support multiple state machines

Processor implications

- Maximizing clock rates may not be crucial
- Internal brute-force concurrency might also be eliminated, giving simpler instruction execution
- Closer integration of i/o channels with internal datapath can reflect interface-centricity of system
- No need for *continuous* fetch-execute cycles
- Support for polling instead of interrupt handling
- In short: more like a programmable function unit
- Soft processors offer scope for specialized tailoring

Multiple processor implications

- With conventional processor-centric view:
  - the multiple centers of control lead towards all the complications of parallel supercomputers
  - multi-processor programming models needed
- However, with logic-centric view:
  - processors do not interfere with each other directly, so no extra synchronization or programming issues
  - essentially just replicated function units, which is already well-understood for programmable logic

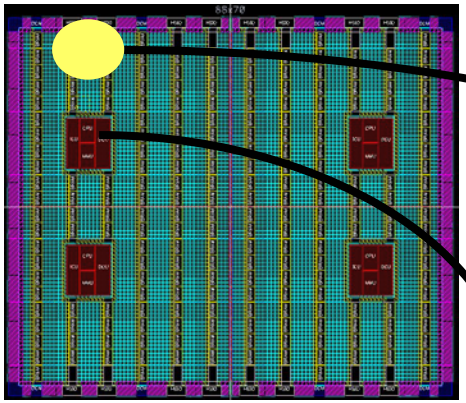
System architecture implications

- Interface-centric view removes processor behavior and needs as a main architectural driver
- In particular:
  - shared buses are less likely to feature, since serialized processor behavior is dominated by potentially highly-concurrent behavior
  - control is decentralized with no inherent hot spots
  - programmable logic can be sized, organized and located to match best the interfaces, not the processor(s)

System design options

- Option 1: follow conventional hardware design process - problem is how to identify good functions to be executed by a processor instead of logic
- Option 2: follow conventional hardware/software co-design process - problem is that processors / software are given too high status in the system
- Option 3: devise a revised design process, aimed at the unique *overall* capabilities of new platform FPGAs - this is currently an active research area

Self-reconfiguring platform (SRP)



FPGA Configuration Memory

ICAP

Control Logic

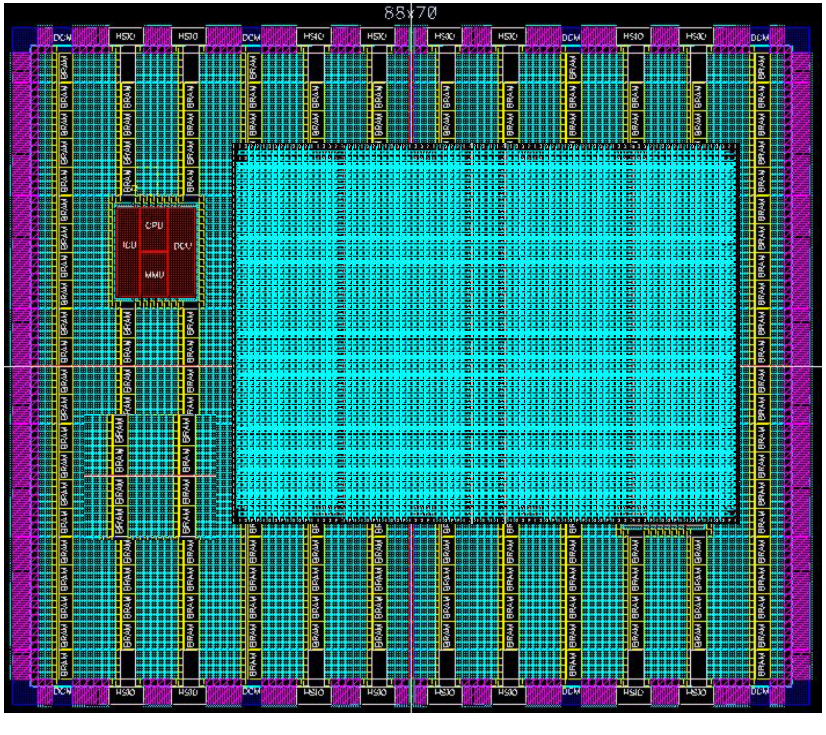
Dual-port Block RAM

PowerPC

CoreConnect OPB

Source : Brandon Blodget et al, DATE 2003

Self-reconfiguring integrated switch



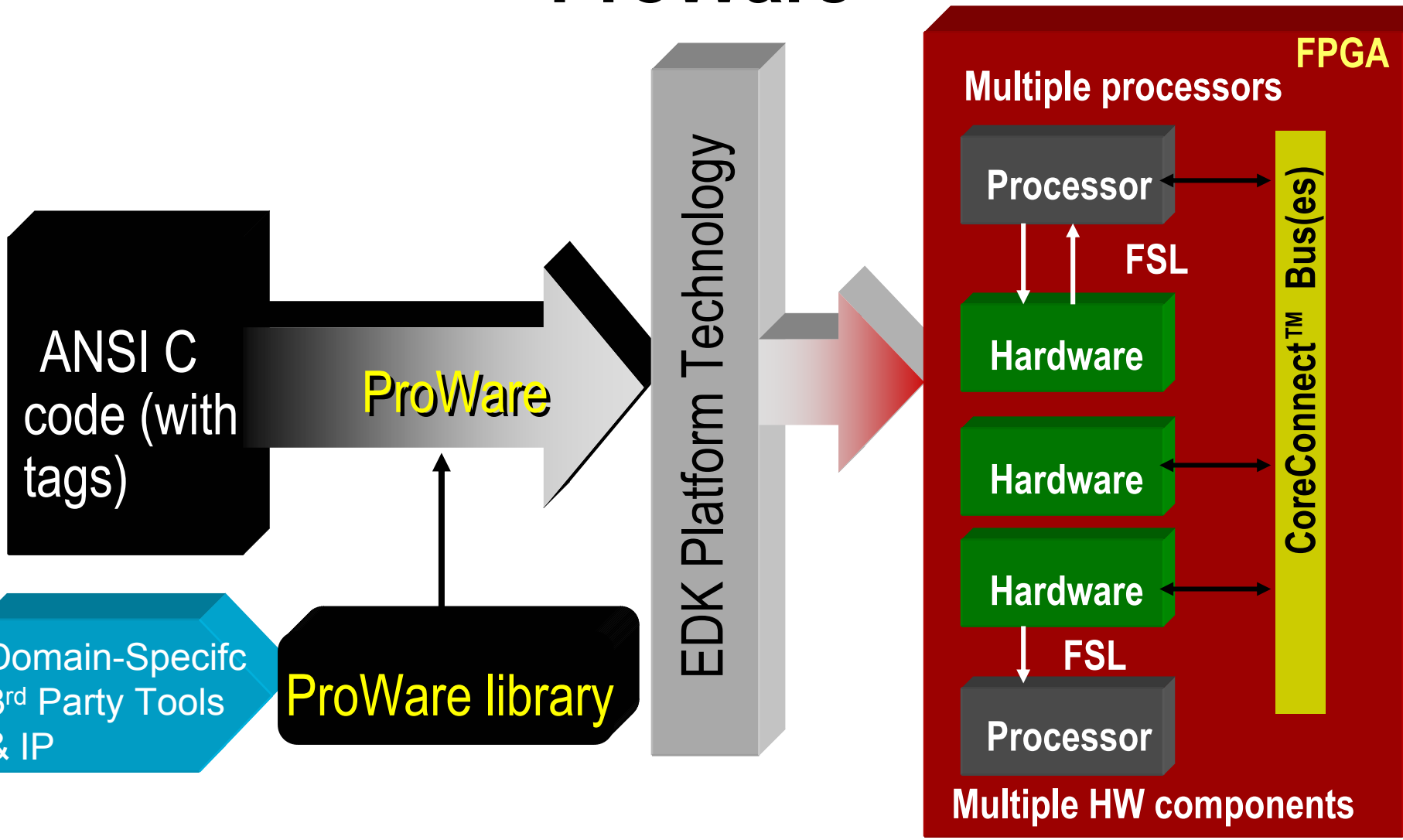
- 928x928 switch
- Fully integrated solution using ICAP
- One PowerPC dedicated to reconfiguration control via ICAP
- >155Mbps/channel
- 144Gbps througput

Potential Applications

- Ethernet configuration
- Relocate, swap, cut and paste hardware modules
- “Intelligent” relocatable modules
- Memory mapped access to reconfigurable elements
- Testing (design, built in self test, etc)


BOZBURUN YACHTING

“ProWare”

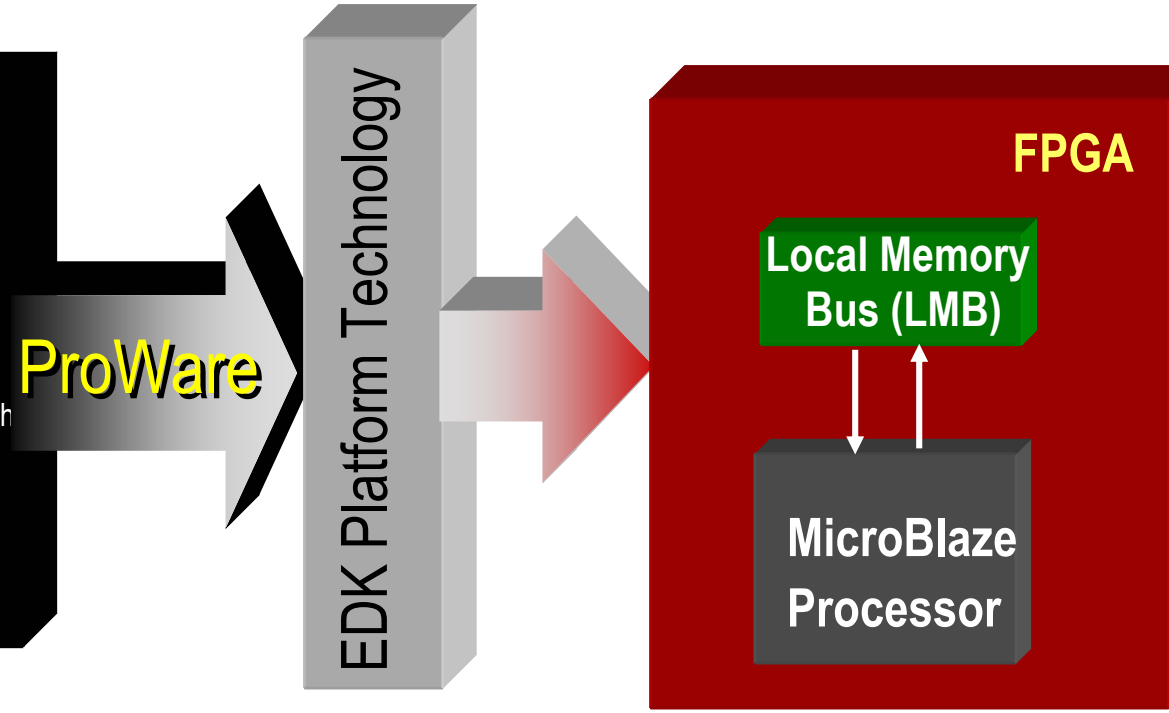

ANSI C code (with tags)
ProWare
Domain-Specifc
3rd Party Tools
& IP
ProWare library
EDK Platform Technology
FPGA
Multiple processors
Processor
FSL
Hardware
Hardware
Hardware
FSL
Processor
CoreConnect™ Bus(es)
Multiple HW components

FSL & Bus: Complementary

- FSL provides:
  - Unidirectional point to point communication
  - Unshared non-arbitrated communication mechanism
  - Best for pipelining / data plane processing
- Bus provides:
  - Shared resource usage
    - Memory
    - Slow peripherals
  - Addressed-mapped devices
- Use of both/either provides:
  - Support for any imaginable data flow requirements
  - Match interconnect architecture to data flow requirements of user program
  - Supports any arbitrary connection topology
    - Star topology
    - Pipelined unidirectional flow network
    - Bi-directional flow
    - Ring topology etc.

ProWare Use Case:
Single Processor

```
#include “xilprowarelib.h” // ProWare Library
extern int input[64];
extern void process_2dct(int *);
int output[64], temp[64];
int main () {
  int temp1[64], temp2[64];
  idct1d(input, temp1); // from xilprowarelib.h
  matrix_transpose(temp1, temp2); // from xilprowarelib.h
  idct1d(temp2, output); // from xiltags.h
  process_2dct(output); // user’s function
}
```

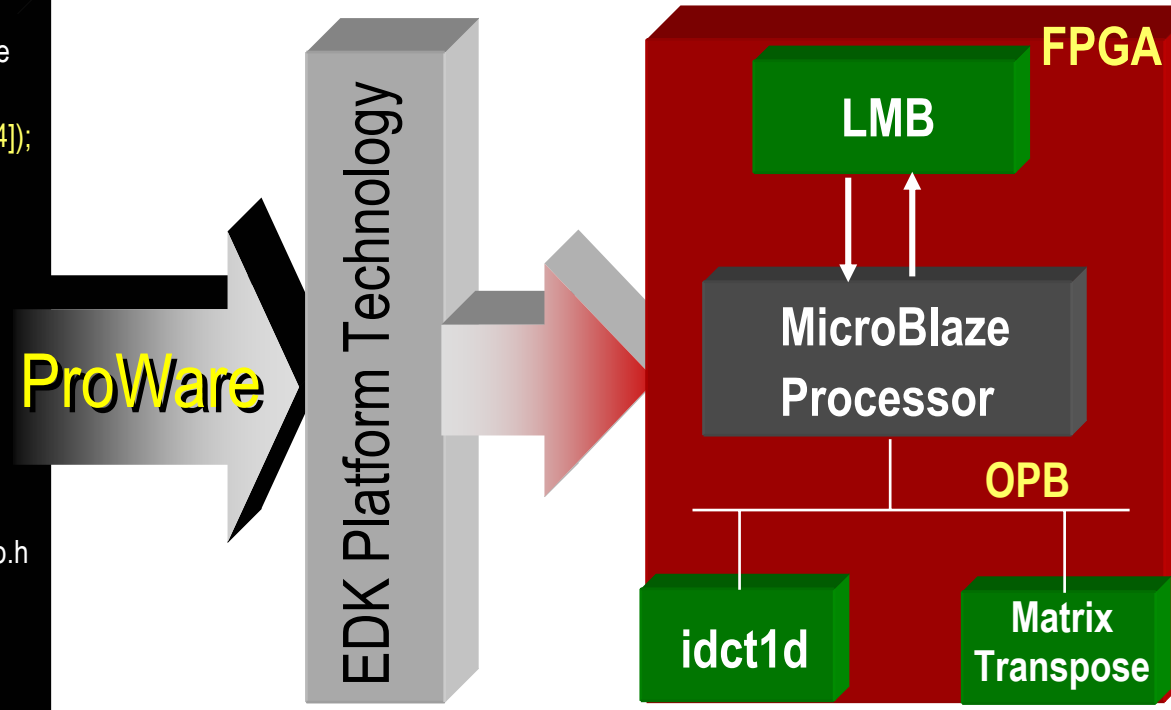


- Use software functions from ProWare Library
- Profile/simulate to find hot spots in code

ProWare Use Case:

Co-Design Only

```
#include “xilprowarelib.h” // ProWare Library header file
// Defines idct1d to be a PE and define port widths
XIL_PE idct1d(XIL_INPUT in[64], XIL_OUTPUT out[64]);
// Define matrix_transpose to be a PE and port widths
XIL_PE matrix_transpose (XIL_INPUT[64],
XIL_OUTPUT[64]);
extern int input[64];
extern void process_2dct(int *);
int output[64], temp[64];
int main () {
  int temp1[64], temp2[64];
  idct1d(input, temp1); // from xilprowarelib.h
  matrix_transpose(temp1, temp2); // from xilprowarelib.h
  idct1d(temp2, output); // from xilprowarelib.h
  process_2dct(output); // user’s function
}
```

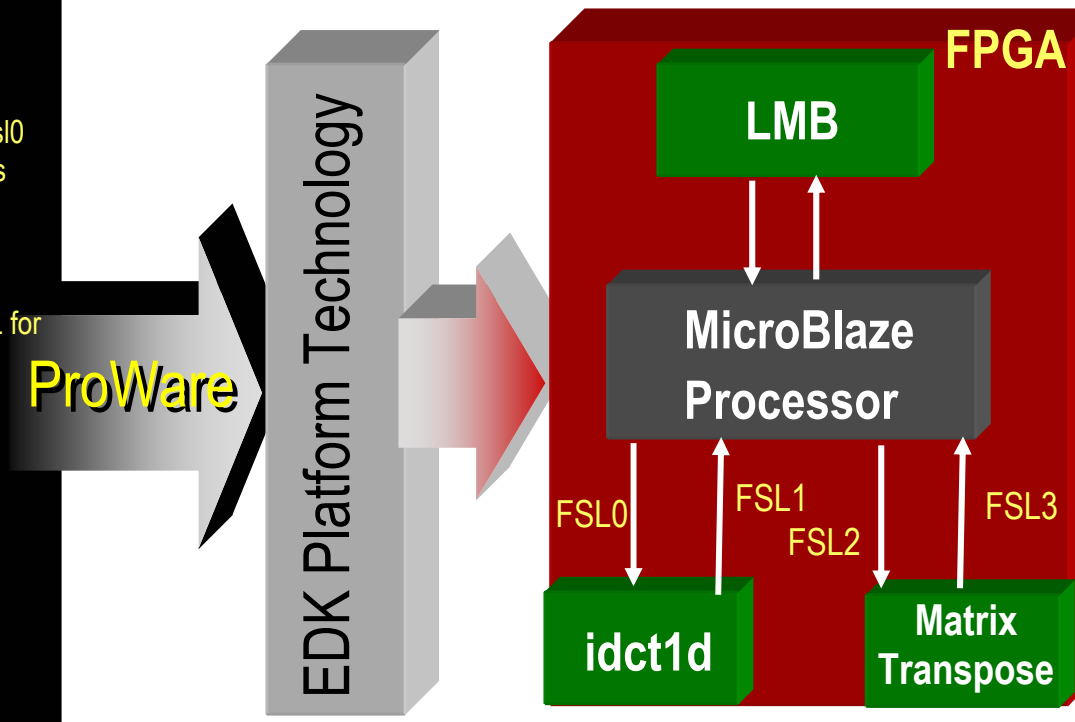


- Calls to idct1d and matrix_transpose become calls to driver code that writes parameters to HW and reads results from HW
- Tags are macros: compile with no side effects on any ANSI C compiler

ProWare Use Case:

Change Connectivity

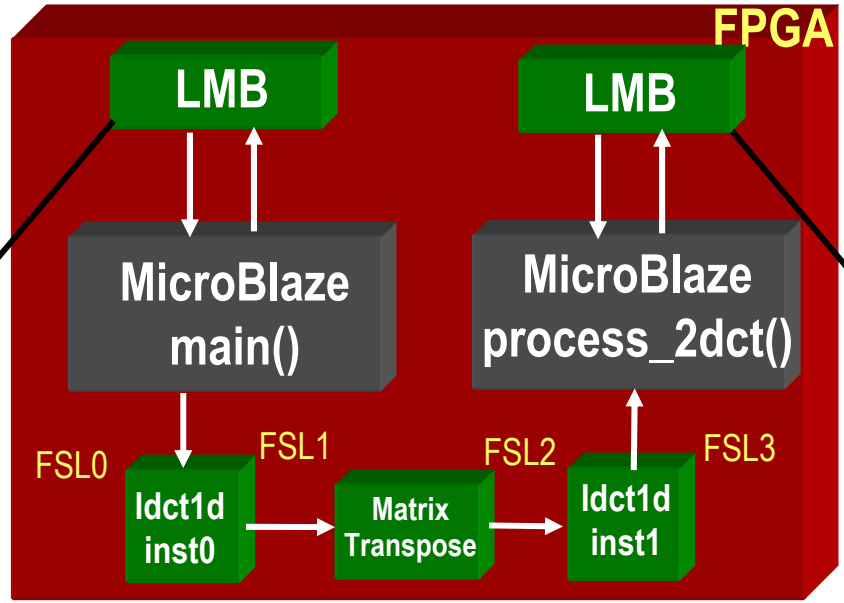
```
#include “xilprowarelib.h” // ProWare Library header file
// Defines idct1d to be a PEand defines the port widths
XIL_PE idct1d (XIL_INPUT in[64], XIL_OUTPUT out[64]);
// Define matrix_transpose to be a PE and define port widths
XIL_PE matrix_transpose (XIL_INPUT in[64], XIL_OUTPUT out[64]);
// Defines idct1d inst0 to be a HWPE with its ports mapped to fsl0
// of type FSL for  the inputs and fsl1 of type FSL for the outputs
XIL_PEINST(idct1d, inst0, XIL_HWPE, XIL_PORTMAP(
XIL_FSLIN(fsl0), XIL_FSLOUT(fsl1)))
// Defines matrix_transpose inst0 to be a HWPE with its ports
// mapped to fsl2 of type FSL for the inputs and fsl3 of type FSL for
// the outputs
XIL_PEINST(matrix_transpose, inst0, XIL_HWPE,
XIL_PORTMAP(XIL_FSLIN(fsl2), XIL_FSLOUT(fsl3)))
extern int input[64];
extern void process_2dct(int *);
int output[64], temp[64];
int main () {
  int temp1[64], temp2[64];
  idct1d(input, temp1); // from xilprowarelib.h
  matrix_transpose(temp1, temp2); // from xilprowarelib.h
  idct1d(temp2, output); // from xilprowarelib.h
  process_2dct(output); // user’s function
```



- Calls to idct1d and matrix_transpose become calls to driver code that writes parameters to HW and reads results from HW
- Tags are macros: compile with no side effects on any ANSI C compiler

ProWare Use Case:

Co-Design + Multiprocessing




```
void idct1d_inst0(int *in, int *out) {
  int i;
  for (i = 0; i < 64; i++) {
    XilWriteFsl(in[i]);
}}
int main () {
  int temp1[64], temp2[64];
  idct1d_inst0( input, temp1);
  matrix_transpose(temp1, temp2);
  idct1d_inst1( temp2, output);
  process_2dct(output);
}
```

Write data to FSL0

Driver call

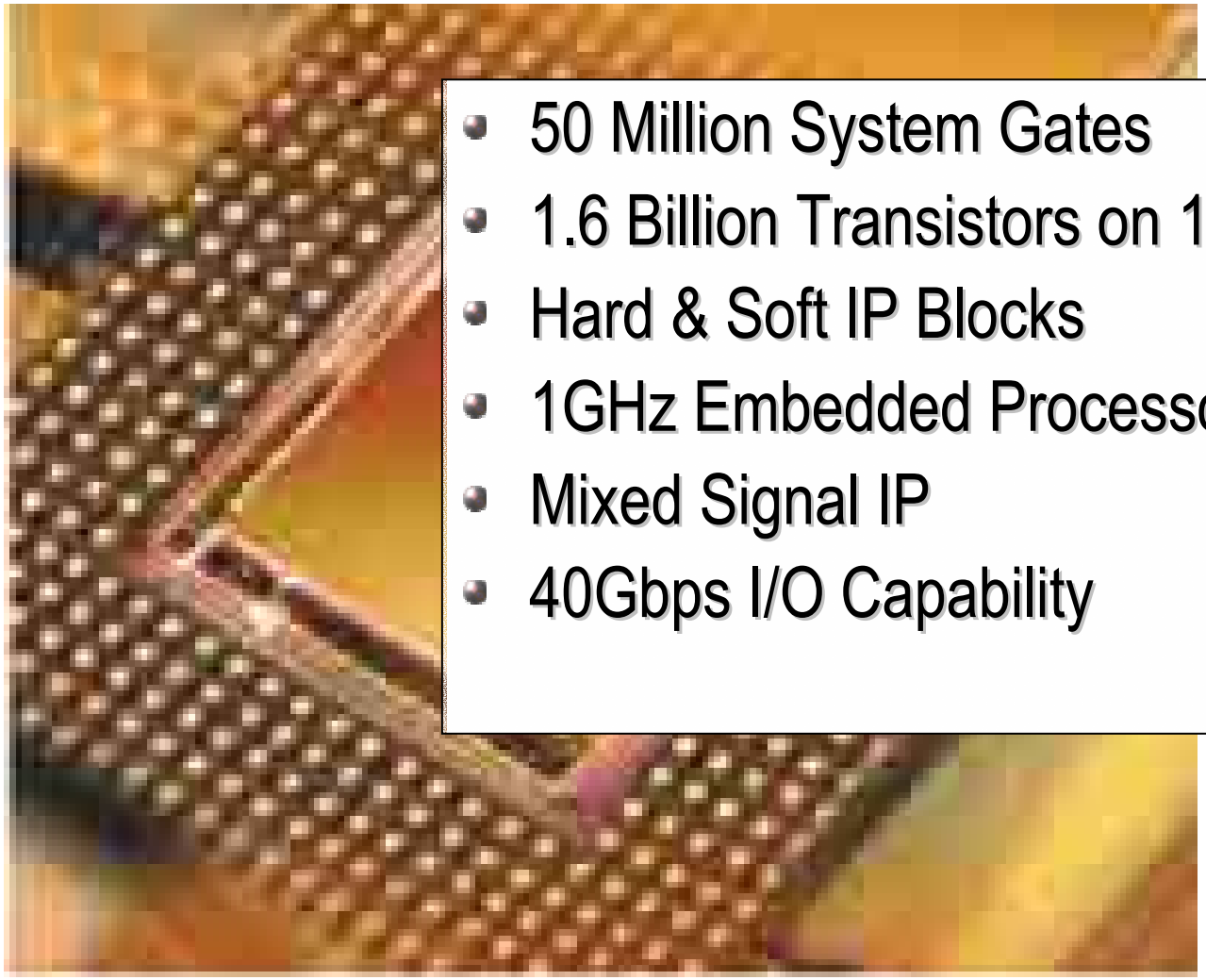
Driver calls
Do nothing

User code

Main automatically generated. Read input from FSL 2, and call process_2dct

```
void process_2dct(int *in) {
:
}
int main () {
  int in[64],  i;
  while (1) {
    for (i = 0; i < 64; i++) {
      XilReadFsl(in[i]);
    }
    process_2dct(in);
}
```

Platform FPGAs circa 2005



- 50 Million System Gates
- 1.6 Billion Transistors on 1 Chip!!
- Hard & Soft IP Blocks
- 1GHz Embedded Processor
- Mixed Signal IP
- 40Gbps I/O Capability

Conclusions

- FPGA is a programmable system platform
- Focus on overall system cost reduction
- New use models for HW and SW : the Interface Centric model
- Software deceleration versus Hardware acceleration
- Runtime reconfigurable architecture through Internal Configuration Access Port (ICAP)
- Challenge is Design Environment