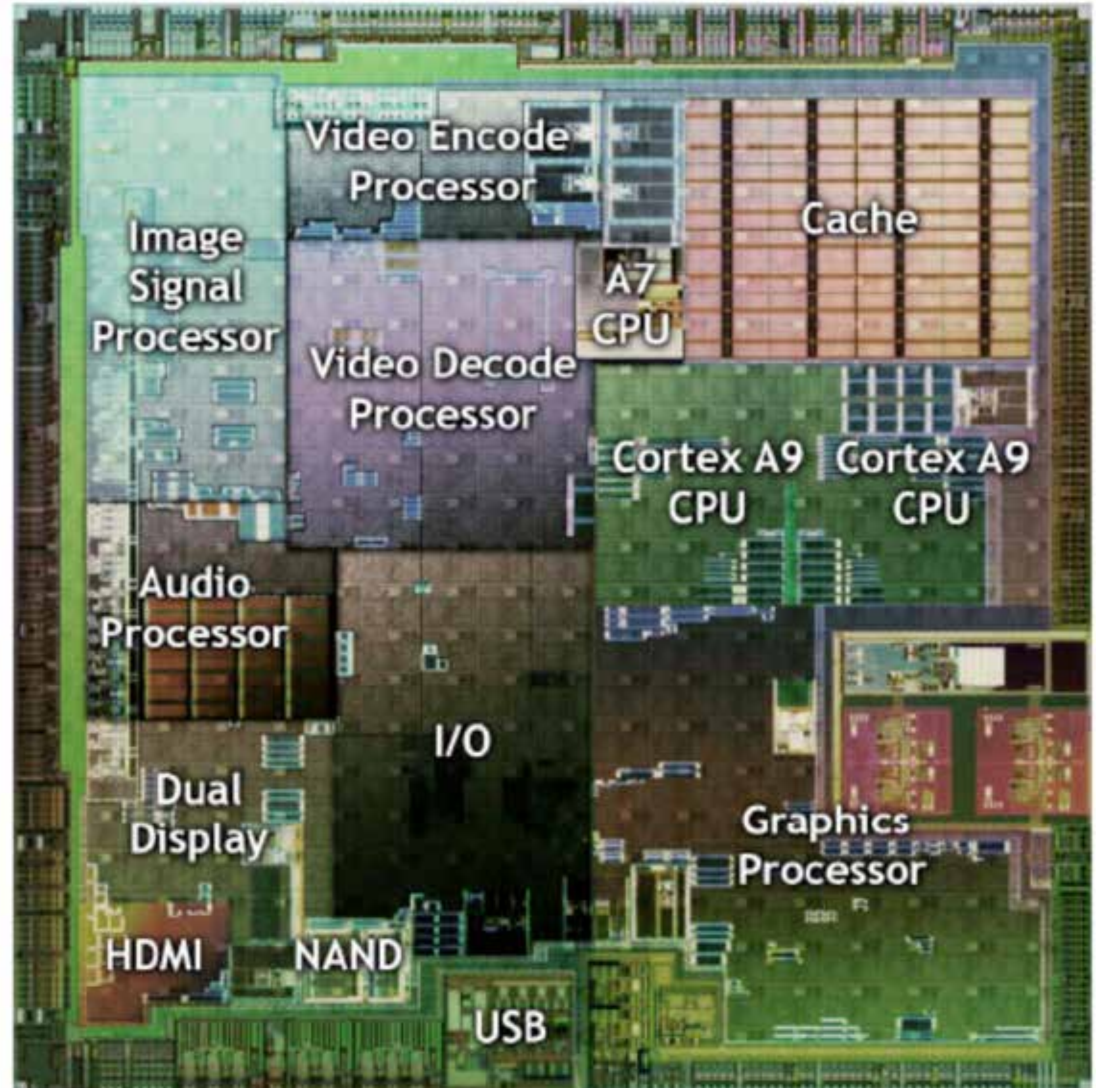
EE382M.20
SOC Design

HW Accelerators and Co-Processors

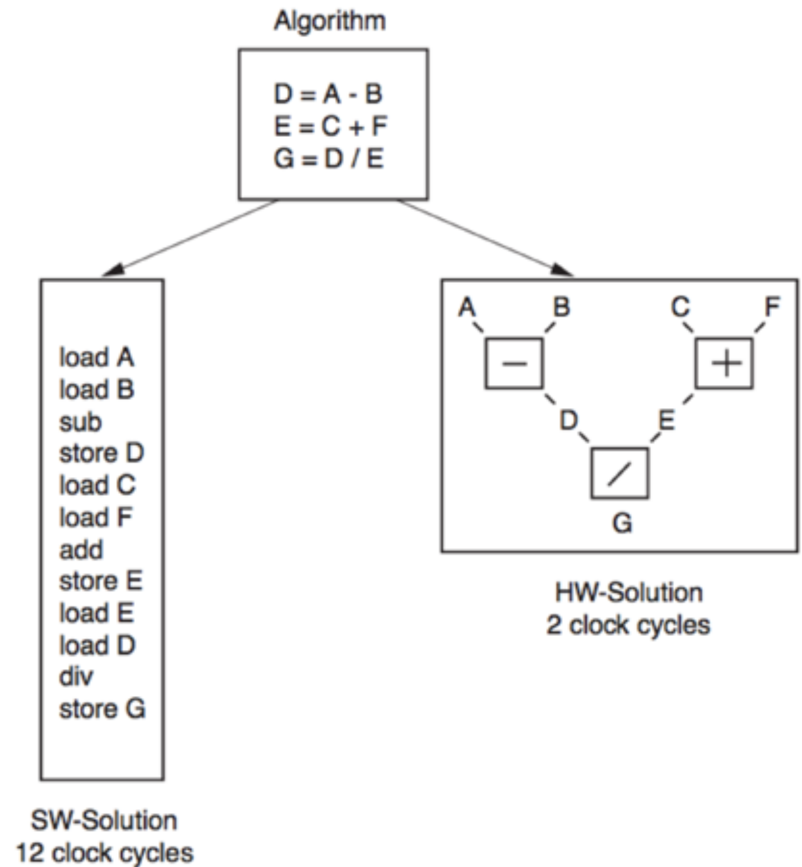
Mark McDermott

Fall 2018



Motivation for HW Acceleration

- **OPs/$ or OPs/Joule**
  - Exploit problem specific parallelism, at thread and instructions level
  - Custom operational units or "instructions" match the set of operations needed for the algorithm (replace multiple instructions with one), custom word width arithmetic, etc.
  - Remove overhead of instruction storage and fetch, ALU multiplexing

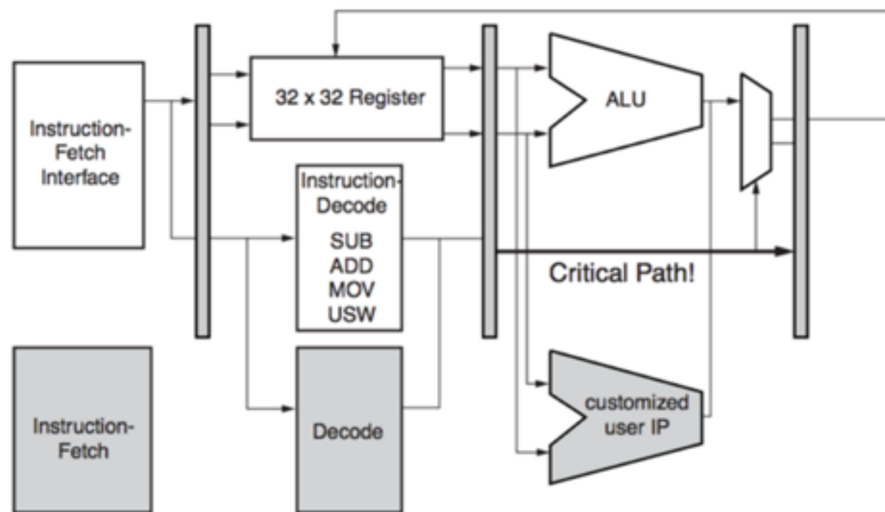


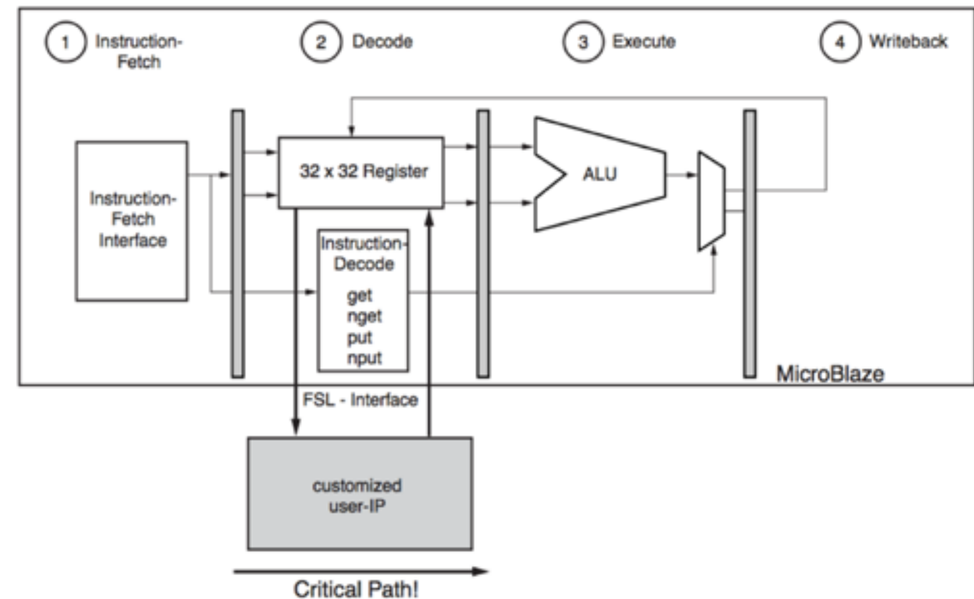
Wawrzynek, 2013

Co-Processors, Reconfigurable Architectures and Custom ISAs

Tightly Coupled Coprocessors

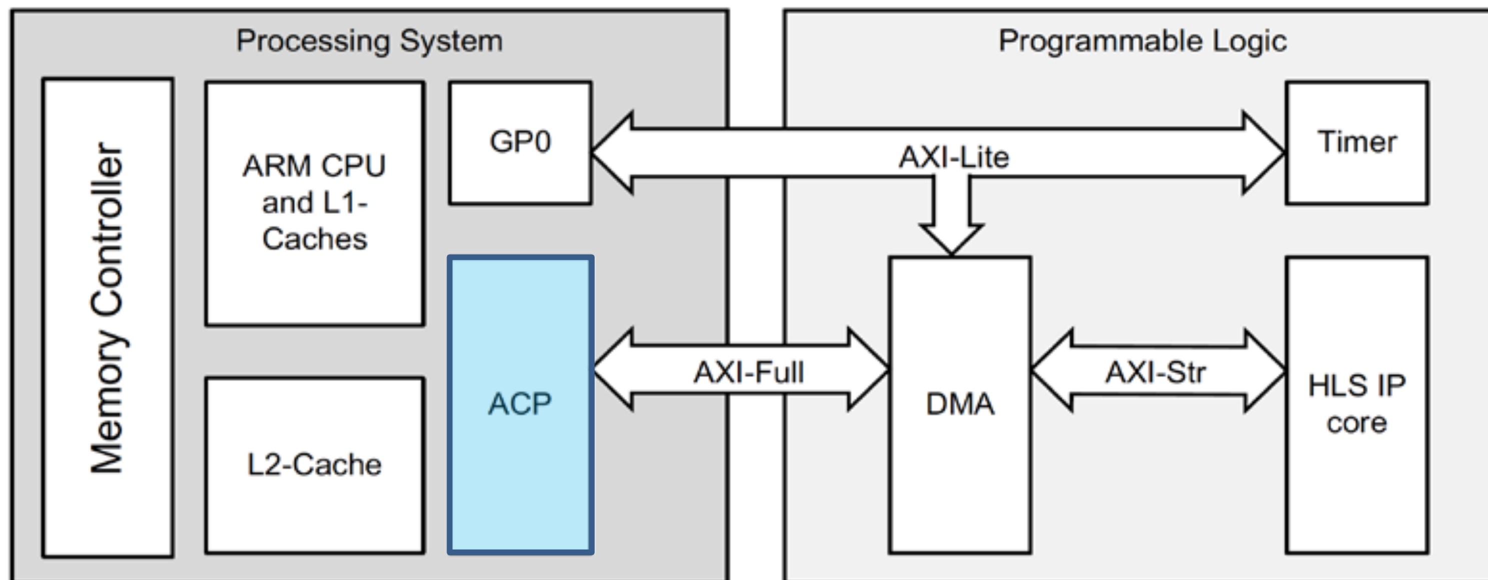
- **Integrated with processor control logic**
  - Task typically completes in a few cycles
  - Small amounts of data
  - Processor stalls waiting for the coprocessor
  - Communication with coprocessor typically via registers and dedicated control signals





Loosely-Coupled Coprocessors

- Loosely-Coupled Coprocessors
  - Used for larger tasks than is the case for tightly-coupled coprocessors
  - Task runs in parallel with main processor
  - May take many cycles per task
  - Large amounts of data that coprocessor may access independent of main processor
  - May or may not use the standard coprocessor interface



https://www.xilinx.com/support/documentation/application_notes/xapp1170-zynq-hls.pdf

Accelerator Coherency Port (ACP)

- Accelerator coherency port (ACP) is a 64-bit AXI slave interface on the SCU that provides an asynchronous cache-coherent access point directly from the PL to the Cortex-A9 MP-Core processor subsystem.

- A range of system PL masters can use this interface to access the caches and the memory subsystem exactly the way the APU processors do to simplify software, increase overall system performance, or improve power consumption.

ACP Usage

- The ACP provides a low latency path between the PS and the accelerators implemented in the PL when compared with a legacy cache flushing and loading scheme. Steps that must take place in an example of a PL-based accelerator are as follows:
  1. The CPU prepares input data for the accelerator within its local cache space.
  2. The CPU sends a message to the accelerator using one of the general purpose AXI master interfaces to the PL.
  3. The accelerator fetches the data through the ACP, processes the data, and returns the result through the ACP.
  4. The accelerator sets a flag by writing to a known location to indicate that the data processing is complete. Status of this flag can be polled by the processor

ACP Caveats

- **NOTE: When compared to a tightly-coupled coprocessor, ACP access latencies are relatively long. Therefore, ACP is not recommended for fine-grained instruction level acceleration.**

- **For coarse-grain acceleration such as video frame-level processing, ACP does not have a clear advantage over traditional memory-mapped PL acceleration because the transaction overhead is small relative to the transaction time, and might potentially cause undesirable cache thrashing.**

- **ACP is therefore optimal for medium-grain acceleration, such as block-level crypto accelerator and video macro-block level processing.**

Performance-Driven ISA Extensions

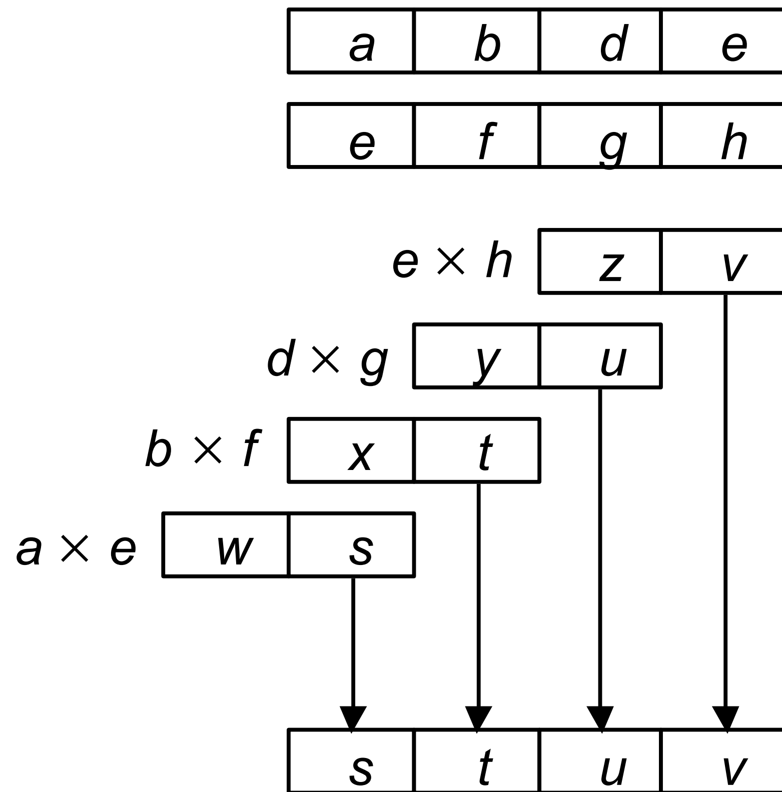
- **Adding instructions that do more work per cycle**
  - Shift-add: replace two instructions with one (e.g., multiply by 5)
  - Multiply-add: replace two instructions with one ($x := c + a \times b$)
  - Multiply-accumulate: reduce round-off error ($s := s + a \times b$)
  - Conditional copy: to avoid some branches (e.g., in if-then-else)

- **Sub-word parallelism (for multimedia applications)**
  - Intel MMX: multimedia extension
  - 64-bit registers can hold multiple integer operands
  - Intel SSE: Streaming SIMD extension
  - 128-bit registers can hold several floating-point operands

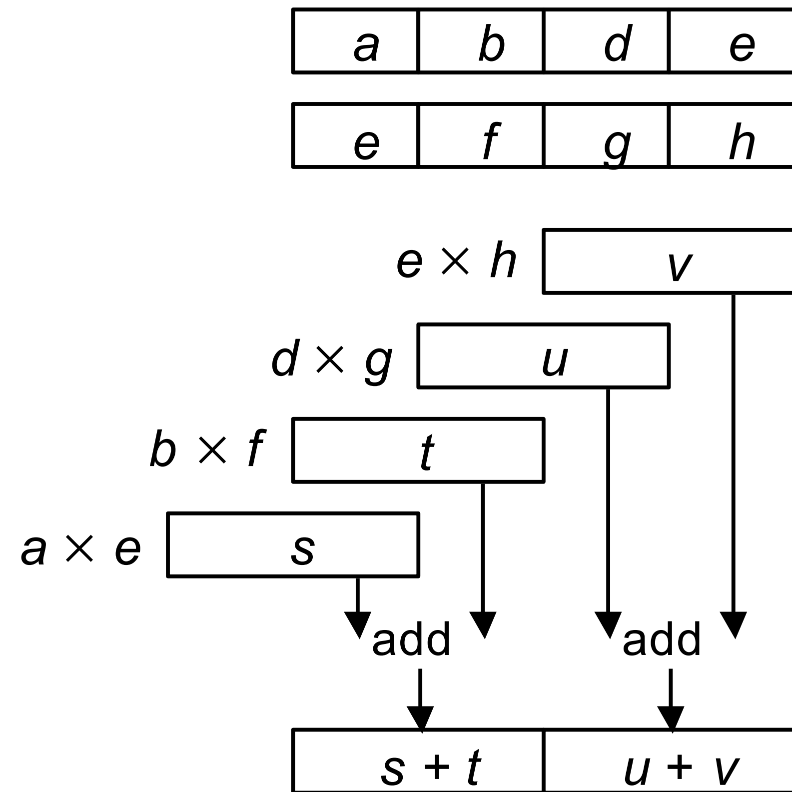
Intel MMX ISA Extension

Class	Instruction	Vector	Op type	Function or results
Copy	Register copy		32 bits	Integer register ↔ MMX register
	Parallel pack	4, 2	Saturate	Convert to narrower elements
	Parallel unpack low	8, 4, 2		Merge lower halves of 2 vectors
	Parallel unpack high	8, 4, 2		Merge upper halves of 2 vectors
Arithmetic	Parallel add	8, 4, 2	Wrap/Saturate#	Add; inhibit carry at boundaries
	Parallel subtract	8, 4, 2	Wrap/Saturate#	Subtract with carry inhibition
	Parallel multiply low	4		Multiply, keep the 4 low halves
	Parallel multiply high	4		Multiply, keep the 4 high halves
	Parallel multiply-add	4		Multiply, add adjacent products*
	Parallel compare equal	8, 4, 2		All 1s where equal, else all 0s
	Parallel compare greater	8, 4, 2		All 1s where greater, else all 0s
Shift	Parallel left shift logical	4, 2, 1		Shift left, respect boundaries
	Parallel right shift logical	4, 2, 1		Shift right, respect boundaries
	Parallel right shift arith	4, 2		Arith shift within each (half)word
Logic	Parallel AND	1	Bitwise	dest ← (src1) ∧ (src2)
	Parallel ANDNOT	1	Bitwise	dest ← (src1) ∧ (src2)′
	Parallel OR	1	Bitwise	dest ← (src1) ∨ (src2)
	Parallel XOR	1	Bitwise	dest ← (src1) ⊕ (src2)
Memory access	Parallel load MMX reg		32 or 64 bits	Address given in integer register
	Parallel store MMX reg		32 or 64 bit	Address given in integer register
Control	Empty FP tag bits			Required for compatibility$

MMX Multiplication and Multiply-Add

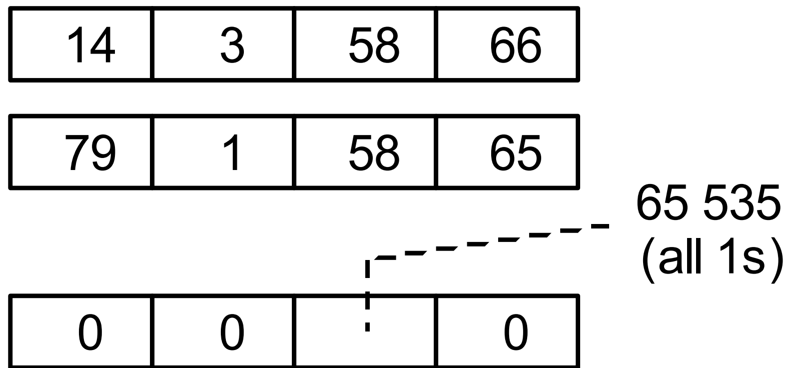


(a) Parallel multiply low

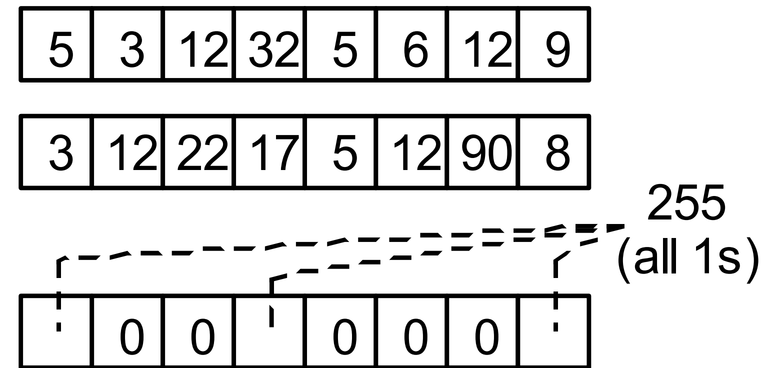


(b) Parallel multiply-add

MMX Parallel Comparisons



(a) Parallel compare equal

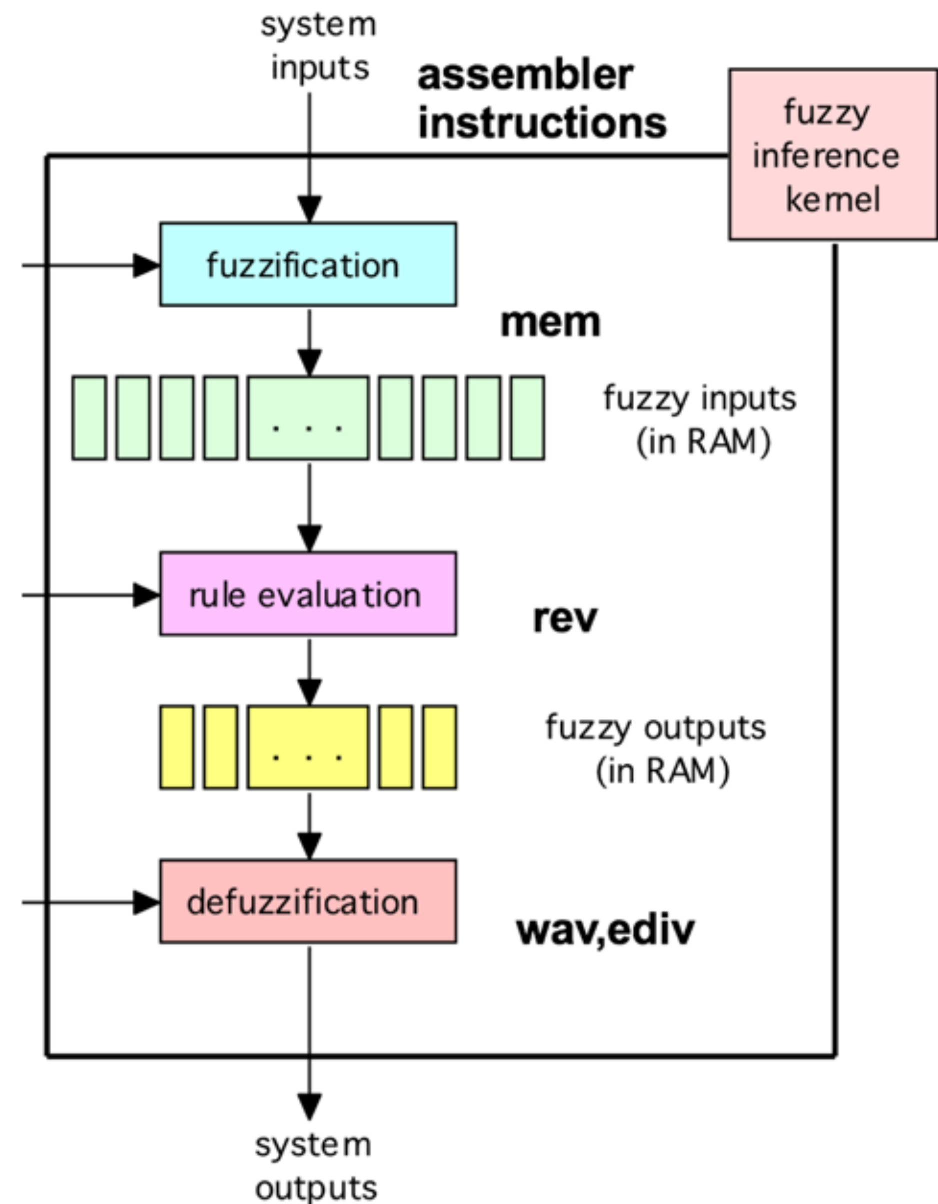


(b) Parallel compare greater

Custom ISA: HC12 Fuzzy Logic Acceleration

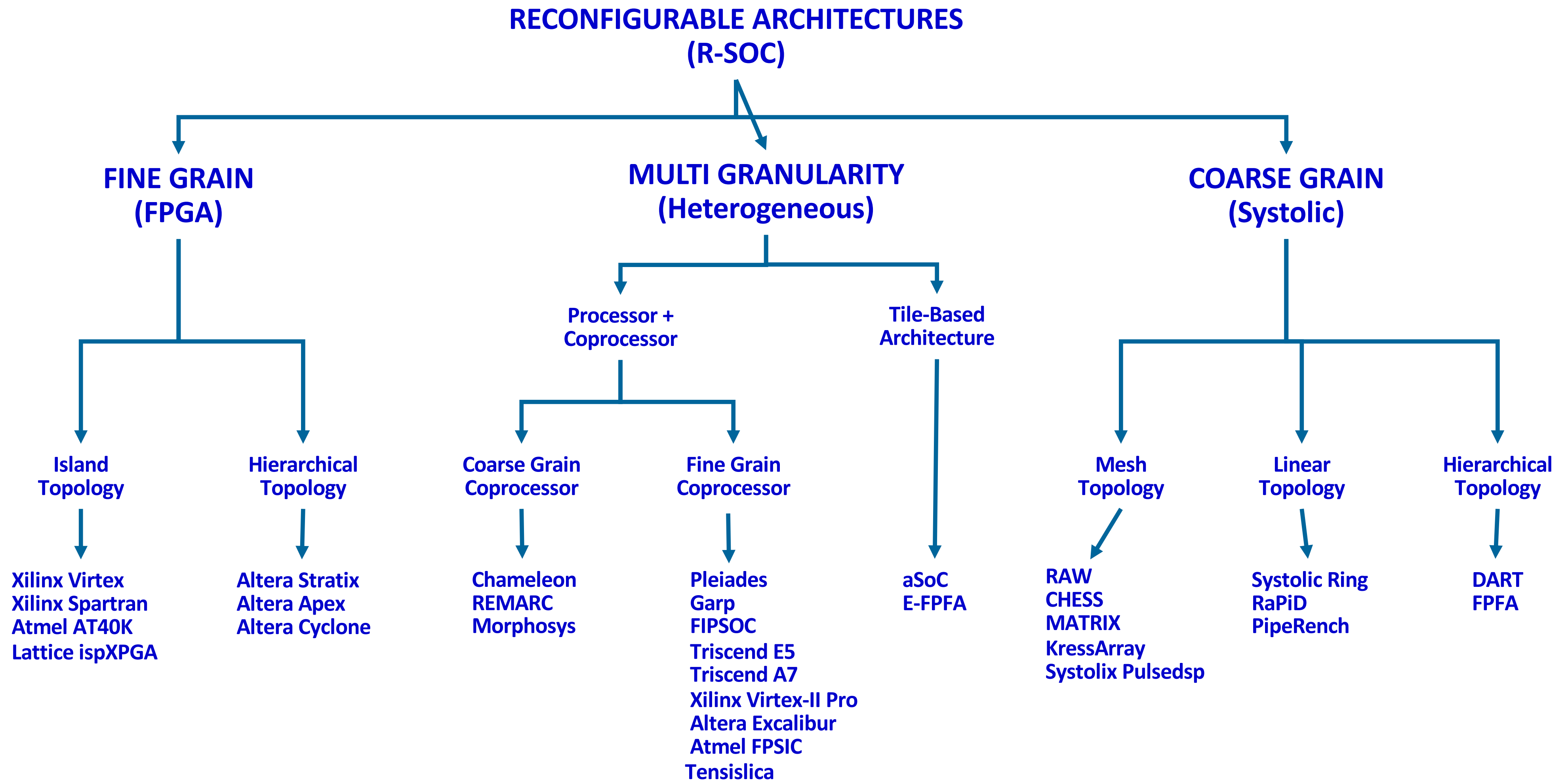
- **Native fuzzy instructions:**
  - MEM; evaluate membership functions
  - REV; rule evaluation: IF a is x THEN b is y
  - WAV; weighted averaging
- **Additional related instructions**
  - MINA (place smaller of two unsigned 8-bit values in accumulator A)
  - EMIND (place smaller of two unsigned 16-bit values in accumulator D)
  - MAXM (place larger of two unsigned 8-bit values in memory)
  - EMAXM (place larger of two unsigned 16-bit values in memory)
  - TBL (table lookup and interpolate)
  - ETBL (extended table lookup and interpolate)
  - EMACS (extended multiply and accumulate signed 16-bit by 16-bit to 32-bit)
  - EDIV (extended divide)

15,000 times faster than HC11



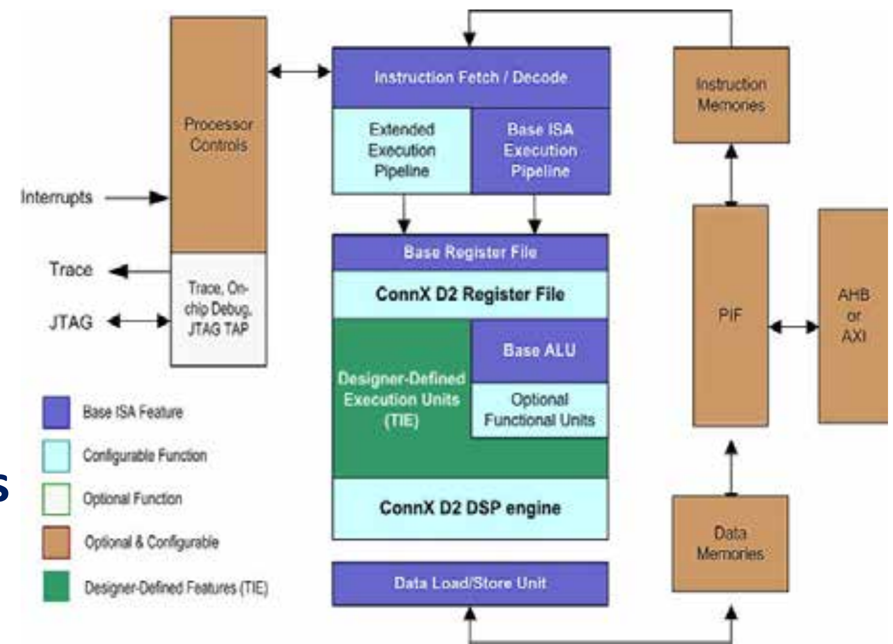
Reconfigurable Architectures

Taxonomy of Reconfigurable Architectures



Customizable ISA: Cadence/Tensilica Xtensa

- **32-bit ALU**
- **1 or 2 Load/Store Model**
- **Registers**
  - 32-bit general purpose register file
  - 32-bit program counter
  - 16 optional 1-bit Boolean registers
  - 16 optional 32-bit floating point registers
  - 4 optional 32-bit MAC16 data registers
  - Optional Vectra LX DSP registers



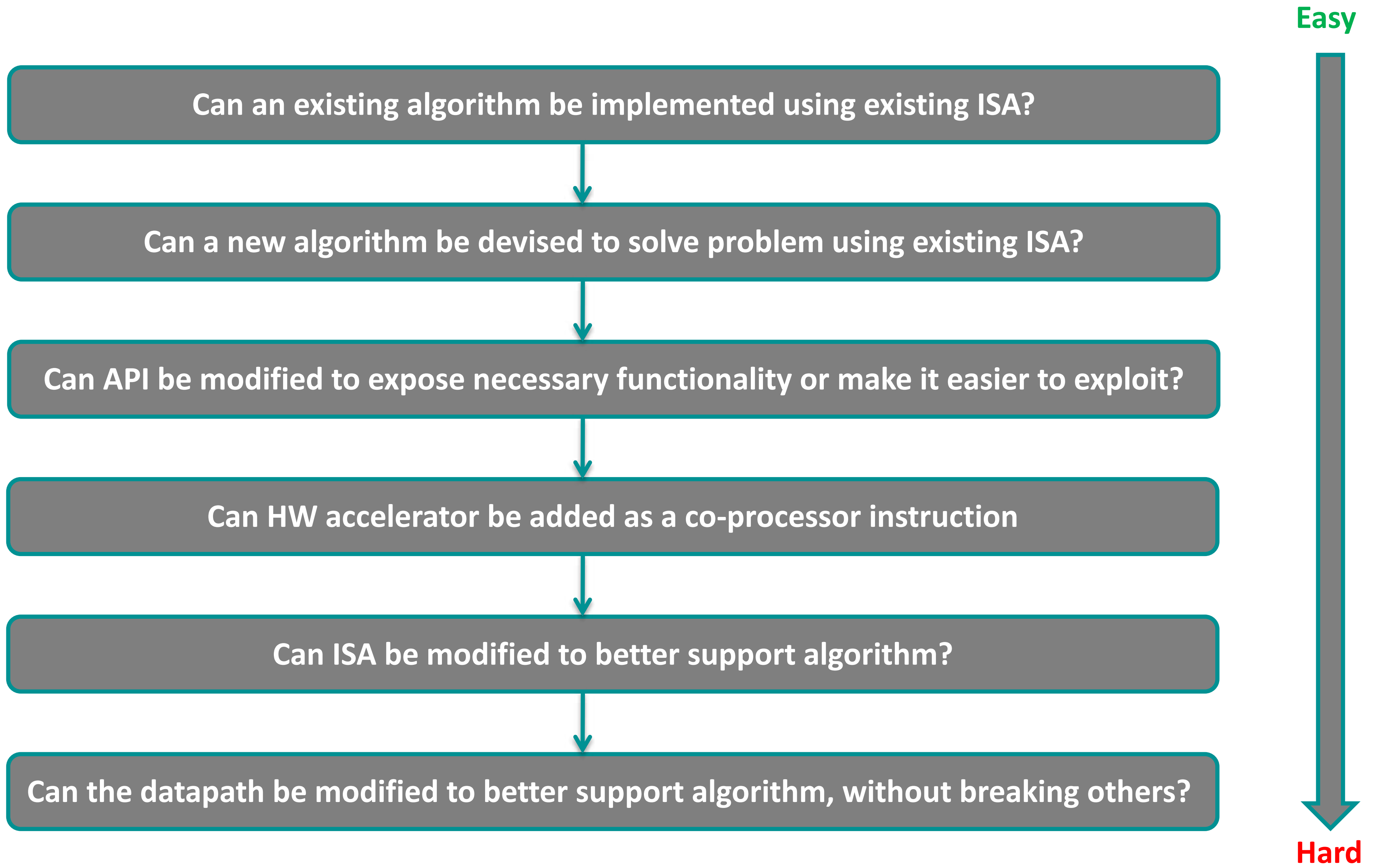
- **General Purpose AR Register File**
  - 32 or 64 registers
  - Instructions have access through "sliding window" of 16 registers. Window can rotate by 4, 8, or 12 registers
  - Register window reduces code size by limiting number of bits for the address and eliminated the need to save and restore register files

Hardware Acceleration

Common HW Acceleration Applications

- Graphics
- Data Compression/Decompression
- Data Streaming: Audio/Video Encoding/Decoding, Network, I/O
- Image sensing and processing
- Logic Simulation
- Data Encryption: RSA, DES, AES
- FFT, DCT, EXP, LOG, …
- Neuronal Networks
- Neuromorphic

Decision Tree: When do you use a hardware accelerator?

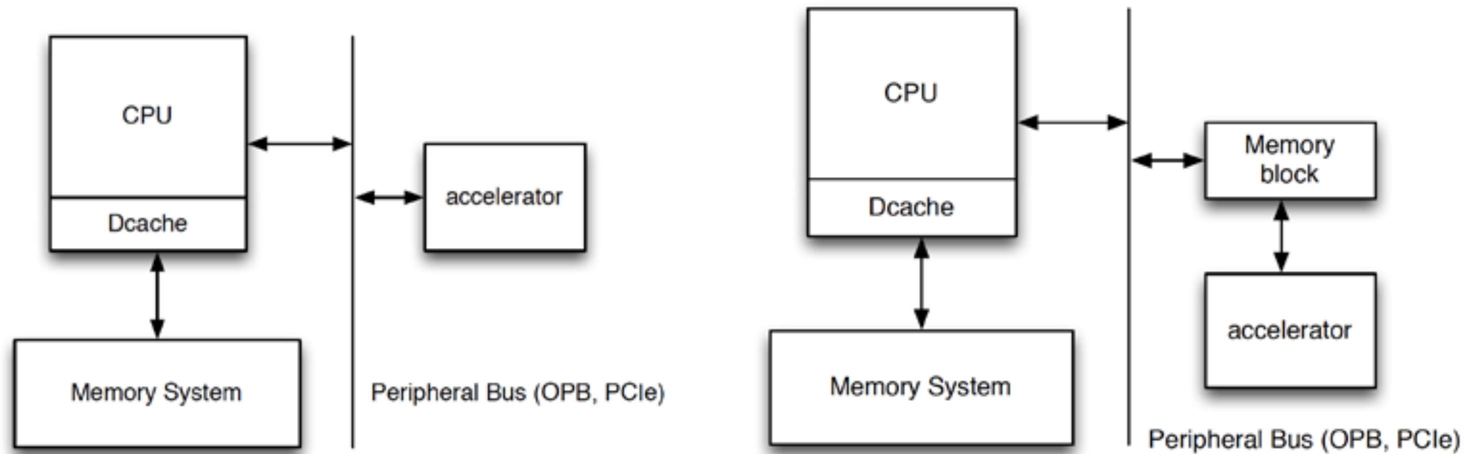


Hardware Acceleration

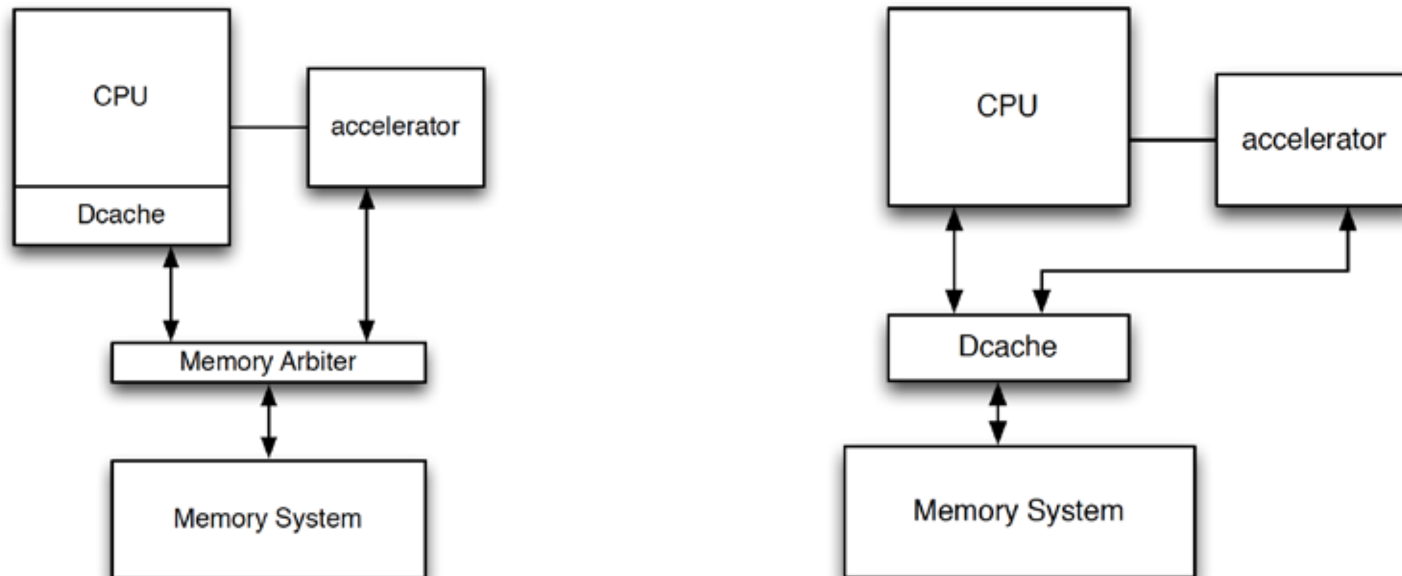
- Ad hoc interface to controlling processor
  - Accelerator registers are memory-mapped
  - Bus-based, FIFO, or register data interfaces
  - Uses DMA for high speed transfers
- Typically, the processor transfers data to the accelerator, issues a "go" command, and then collects result data later.
  - Polled or interrupt-based interface
- Accelerator may have its own path to/from memory
- Often fixed function but can be microcoded for programmability

Hardware Accelerator Topologies

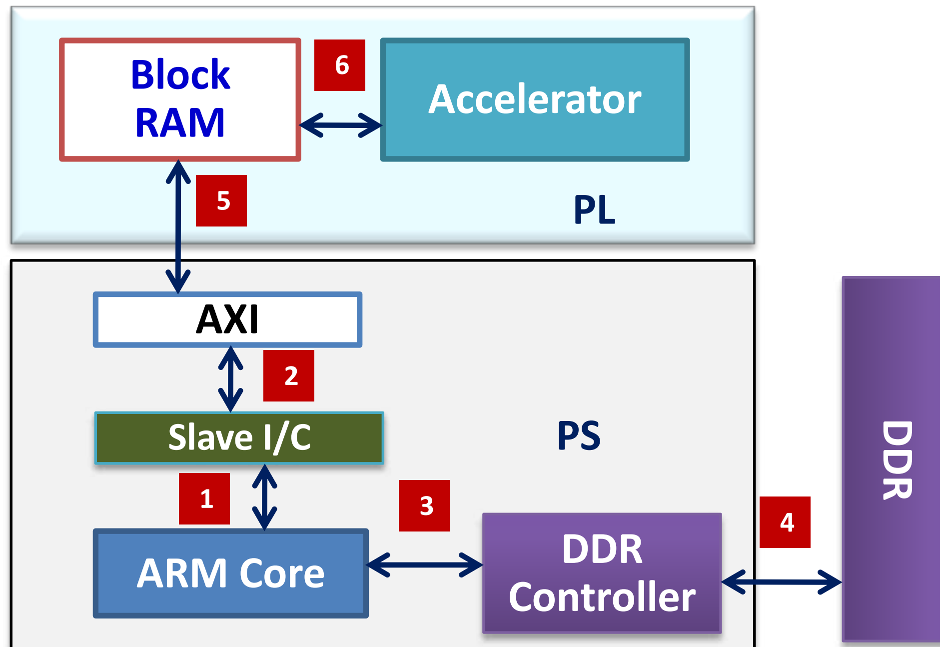
Accelerator appears as a device on a bus



Accelerator is tightly coupled into the processor memory system



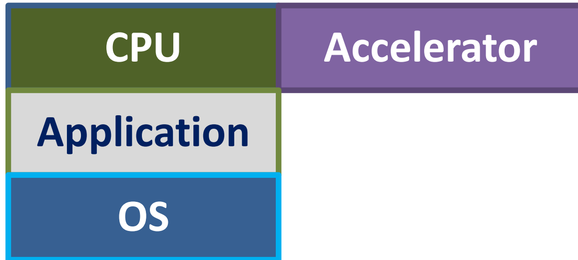
CPU-Accelerator Interface Example



Bus		First Access		Pipelined Access		Arbitration
		Read	Write	Read	Write	
1	ARM → I/C	2	2	2	2	
2	I/C→ AXI	8	8	3	3	5
3	AHB → DDRC	4	4	4	4	
4	DDRC → DRAM	8	9	3	3	5
5	AXI ↔ BRAM	20	20	8	8	12
6	BRAM ↔ ACC	2	2	2	2	

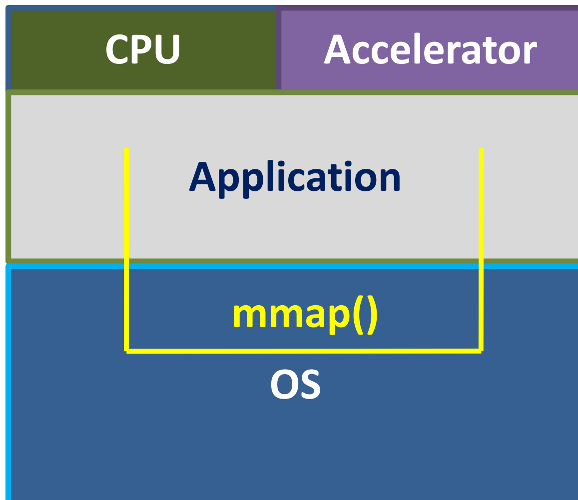
- **AXI**
  - 32 bit Bus
  - Access to DRAM data & programmable logic fabric
  - 1/2 CPU frequency
  - Big penalty if bus is busy during first attempt to access bus
- **AHB (AMBA High Speed Bus)**
  - 64 bit bus
  - Runs at CPU clock frequency
  - Access to DDR Controller to provide addresses to SDRAM

Four Programmers Models of Accelerator Design

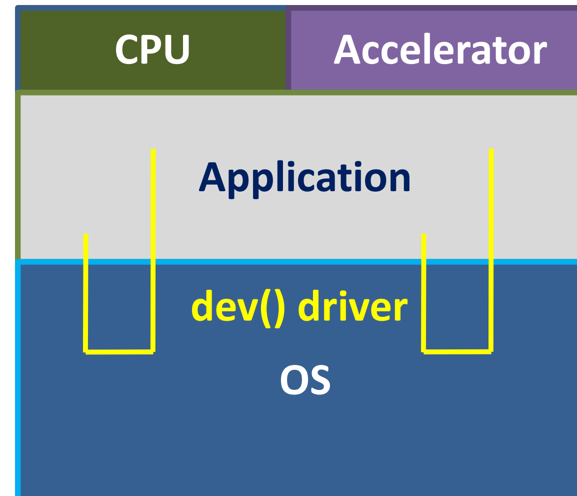


Base - HW I/F only

No OS Service (in simple embedded systems)

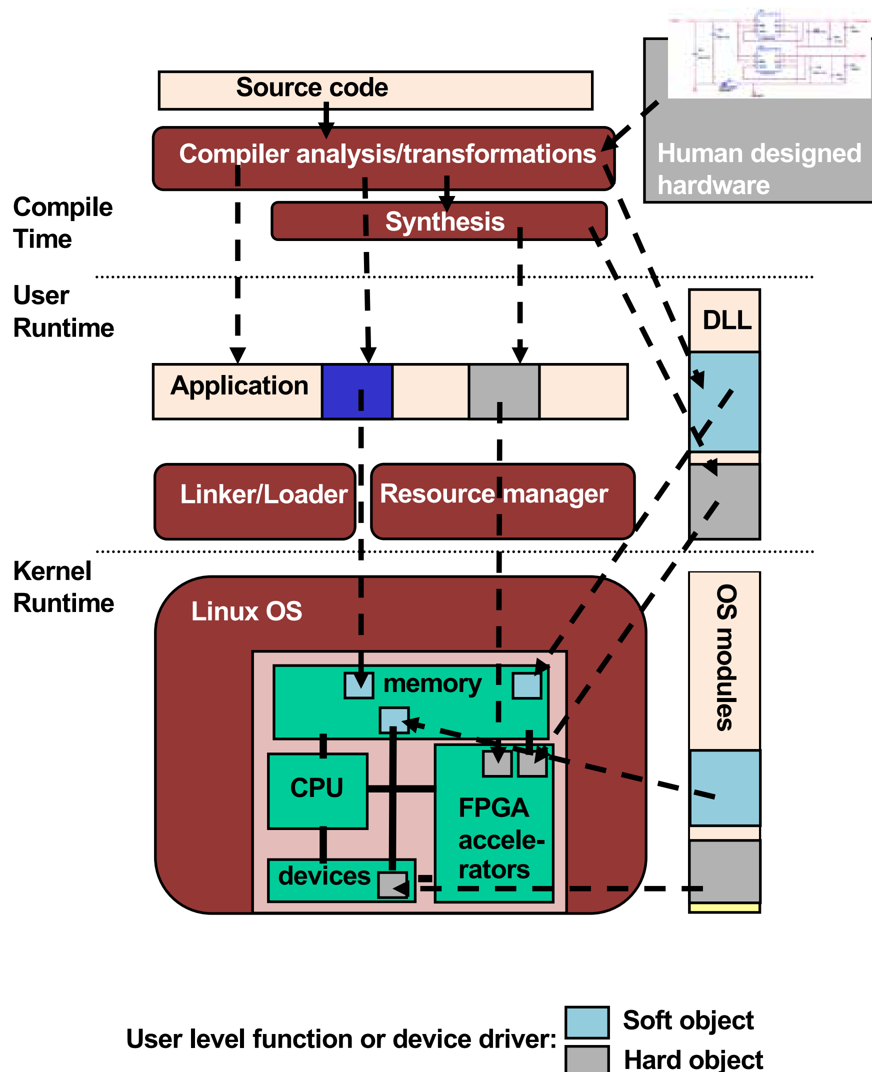


OS service – Accelerator accessed as a user space memory mapped I/O device



Virtualized Device with OS scheduling support

Hybrid Hardware/Software Execution Model



- **Hardware Accelerator as a Kernel Module**
  - Seamless integration of hardware accelerators into the Linux software stack for use by mainstream applications
  - The KM approach enables transparent interchange of software and hardware components
- **Application level execution model**
  - Compiler deep analysis and transformations generate CPU code, hardware library stubs and synthesized components
  - FPGA bitmaps as hardware counterpart to existing software modules.
  - Same dynamic linking library interfaces and stubs apply to both software and hardware implementation
- **OS resource management**
  - Services (API) for allocation, partial reconfiguration, saving and restoring the status, and monitoring
  - Multiprogramming scheduler can pre-fetch hardware accelerators in time for next use
  - Control the access to the new hardware to ensure trust under private or shared use

Hardware Accelerator Interface: Interrupts or Polling?

- **Polling interfaces usually require the processor to read a memory-mapped register to determine the state of the accelerator.**
  - Can the accelerator accept new input data?
  - Is the accelerator done with its current task?
  - Has the accelerator generated an error condition?

- **Polling interfaces offer minimal latency between the setting of a condition on the accelerator and its discovery by the controlling processor.**
  - But processor isn't doing useful work while it polls…

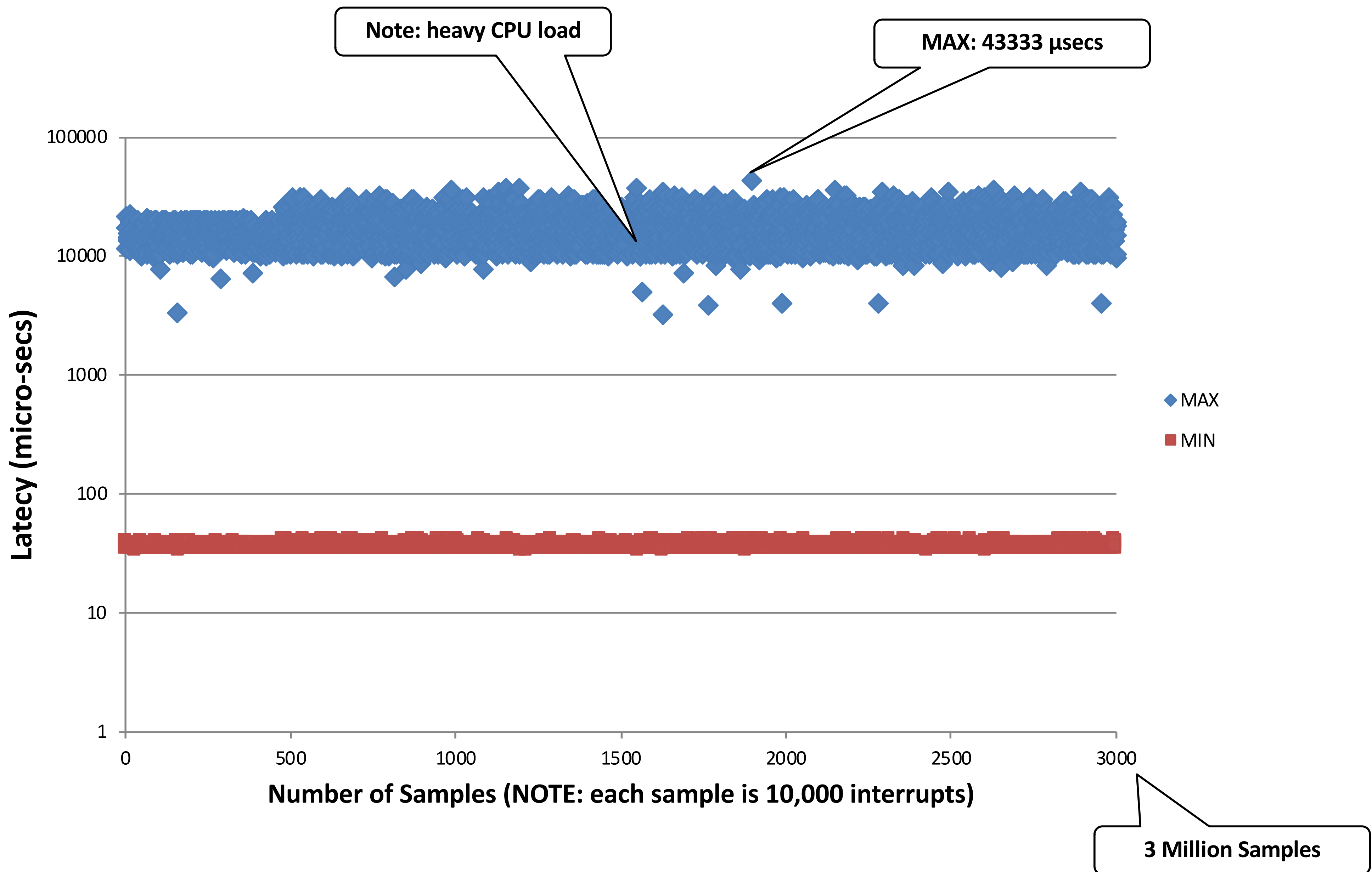
Hardware Accelerator Interface: Interrupts or Polling?

- **Interrupt-based interfaces allow the accelerator to signal conditions to the controlling processor.**
  - Interrupt latency is longer than is achievable via the polling method.
  - But the processor can more easily proceed with other work while the accelerator is busy with a task.

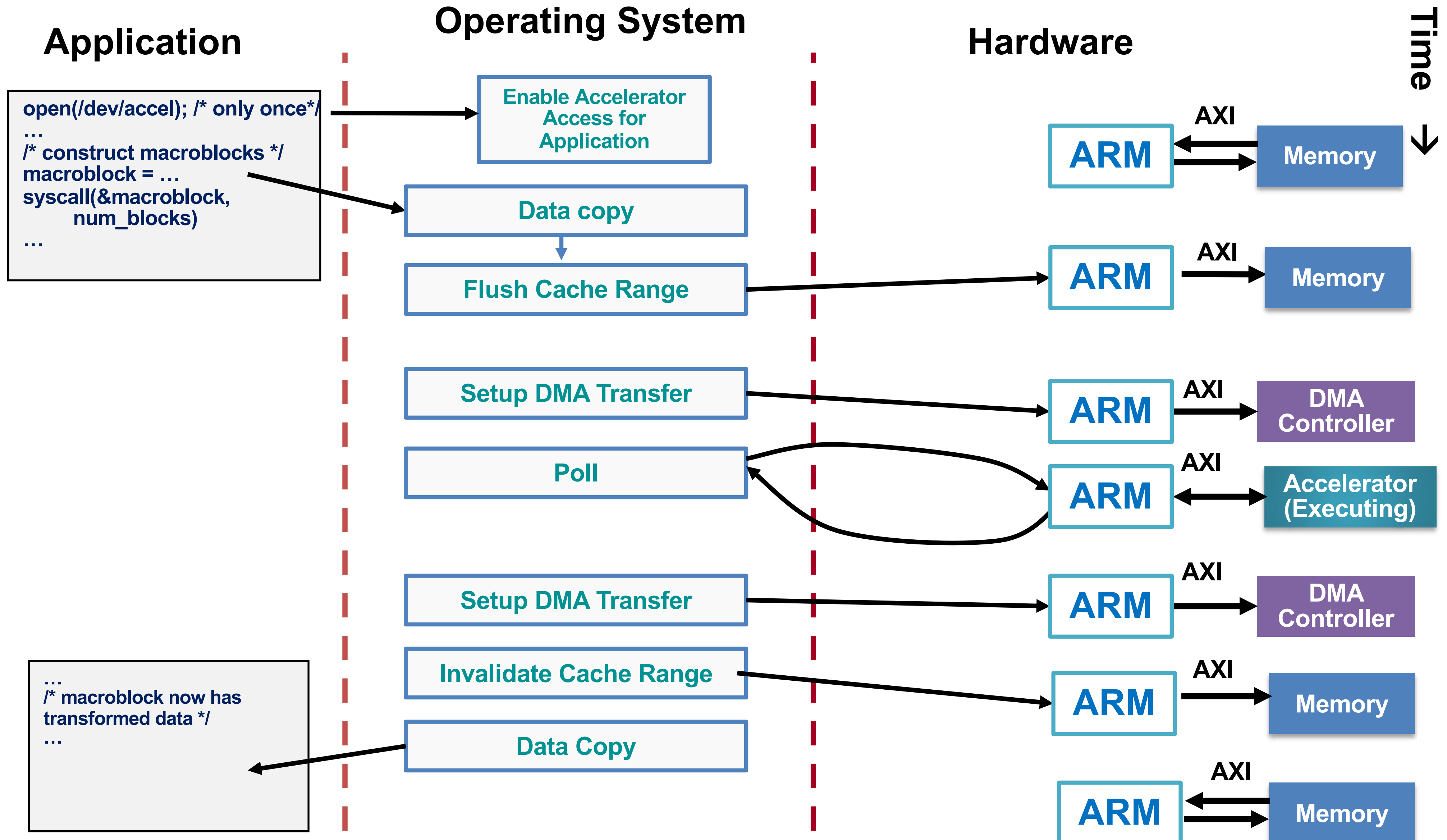
- **Interrupts more efficient for coarse grained parallelism (i.e., larger tasks with looser and less frequent synchronization requirements)**

- **Interrupts may not work for real-time control tasks with tight schedules**

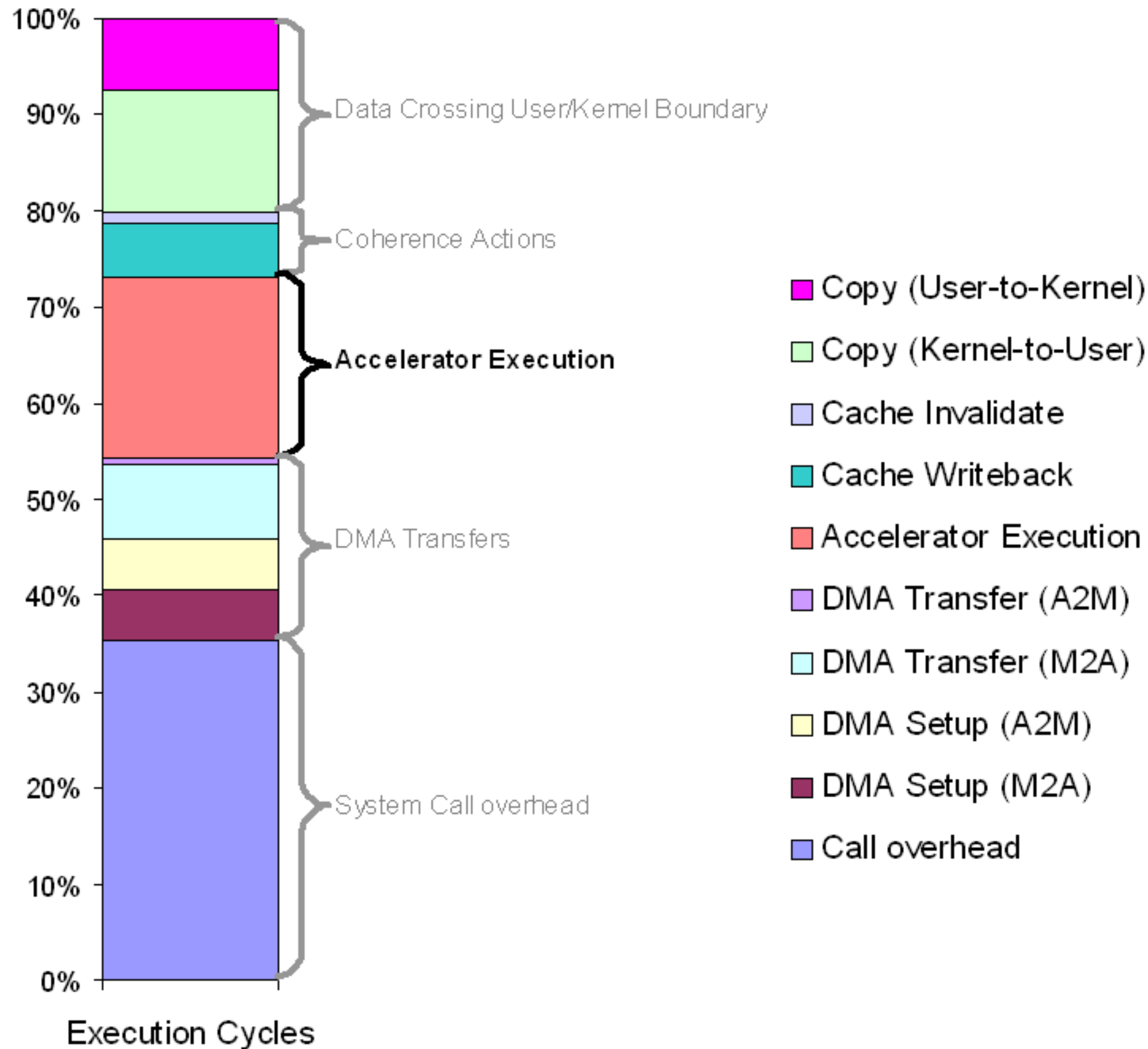
Zedboard Interrupt latency measurement results



Typical CPU → Accelerator Transaction



Device Driver Access Cost

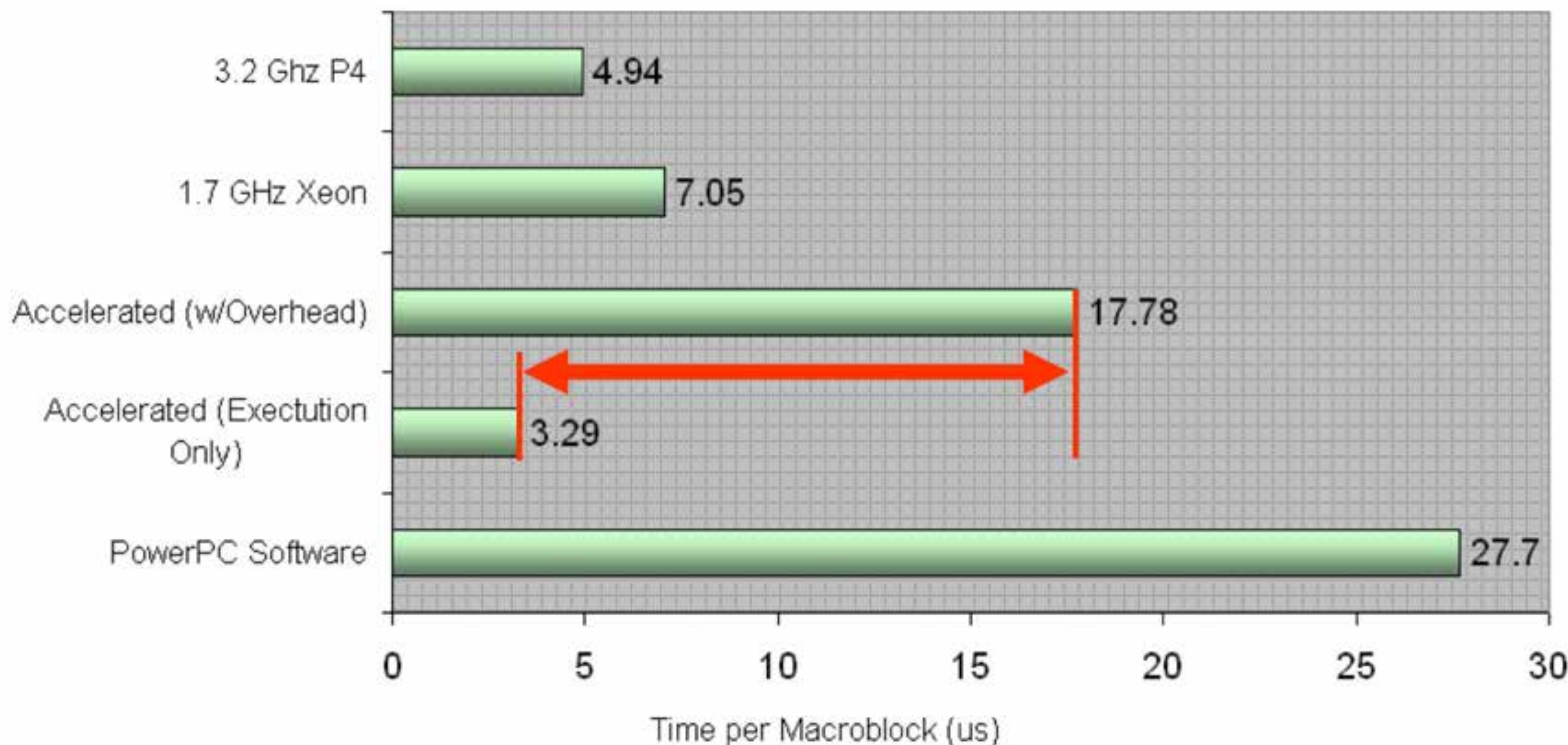


Accelerator Speedup

- **Assume loop is executed n times.**

$$\text{Speedup} = n(t_{CPU} - t_{accel})$$
$$= n(t_{CPU} - (t_{in} + t_{exec} + t_{out}))$$

- **Compare accelerated system to non-accelerated system:**



Single-threaded vs Multi-threading

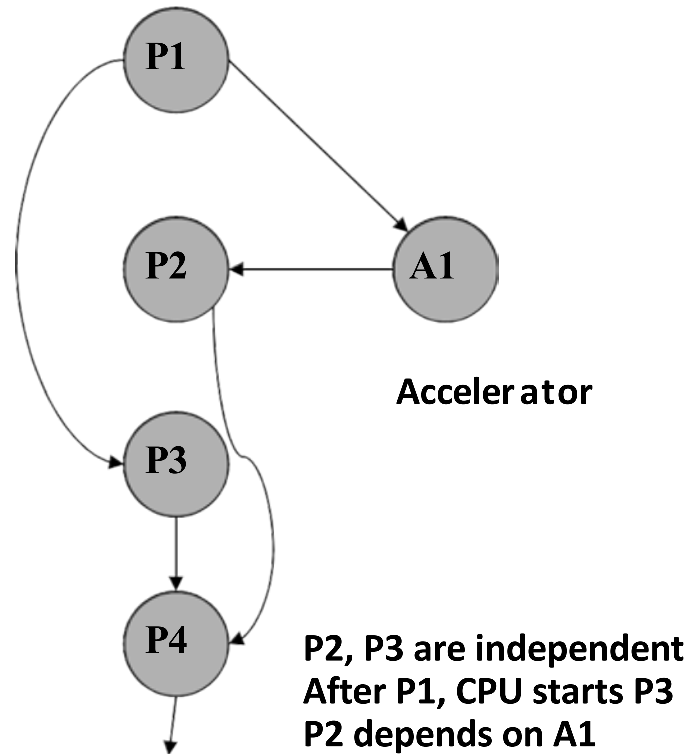
- **One critical factor is the available parallelism in the application:**
  - Single-threaded/blocking: CPU waits for accelerator;
  - Multithreaded/non-blocking: CPU continues to execute along with accelerator.
- **For multithread, CPU must have some useful work to do while accelerators perform some tasks.**
  - Software environment must also support multi- threading.

- **Blocking: CPU waits for the accelerator call to complete.**

Determining total execution time

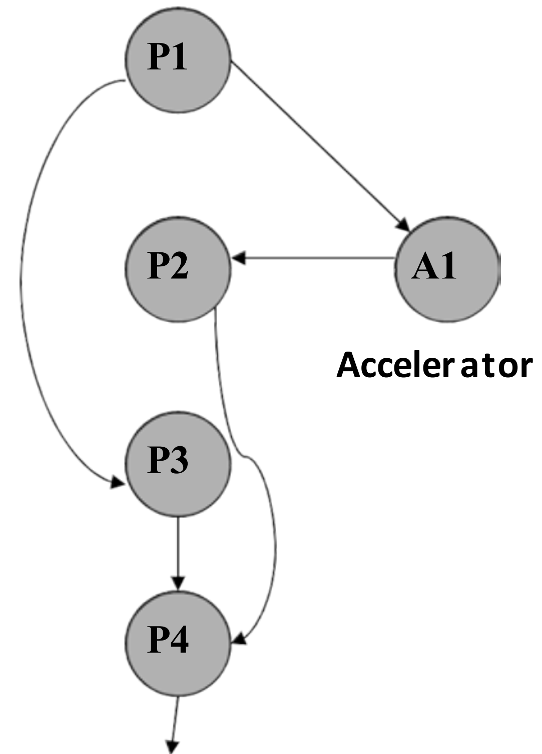
Single Threaded CPU:

Count execution time of component processes



Multi-Threaded CPU:

Find longest path execution time of component processes



Caching Issues with Accelerators

- **Main memory provides the primary data transfer mechanism to the accelerator.**
- **Programs must ensure that caching does not invalidate main memory data.**
  - CPU reads location S.
  - Accelerator writes location S.
  - CPU writes location S.
    - BAD – Program will not see proper value of S stored in the cache

The bus interface may provide mechanisms for accelerators to tell the CPU of required cache changes…

Synchronization and Memory

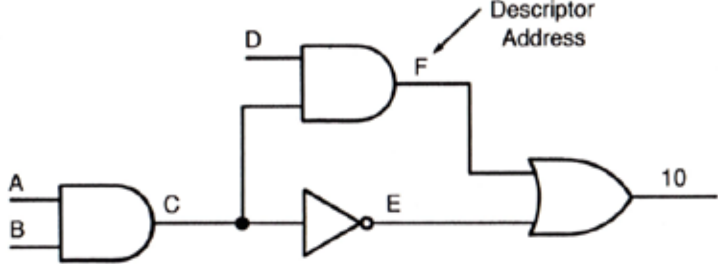
- **As with cache, main memory writes to shared memory may cause invalidation (memory incoherence).**
  - CPU reads location S
  - Accelerator writes S
  - CPU writes S
- **Many CPU buses implement test-and-set atomic operations that the accelerator can use to implement a semaphore. This can serve as a highly efficient means of synchronizing inter-process Communications (IPC)**

Logic Simulation Acceleration

Metrics

- Performance:
  - 400 – 600X faster than SW simulator
  - 400K Evaluations/Sec
  - I/O speed: 100 MB/sec
- Simulation algorithm
  - 2-Pass event driven, selective trace
    - Evaluation pass
    - Update pass
- Supported functions:
  - Logic Verification
    - Delay assigned per element
    - Delay assigned per pin type
    - 4 value logic
    - 16 value logic
  - Rise and Fall delays
  - Setup and Hold time analysis
  - Minimum pulse width detection
  - Worst case analysis
  - Wire delay
  - Transmission gates
  - Fault simulation
  - Behavioral simulation

Simulation Processing Memory



768 Bits wide

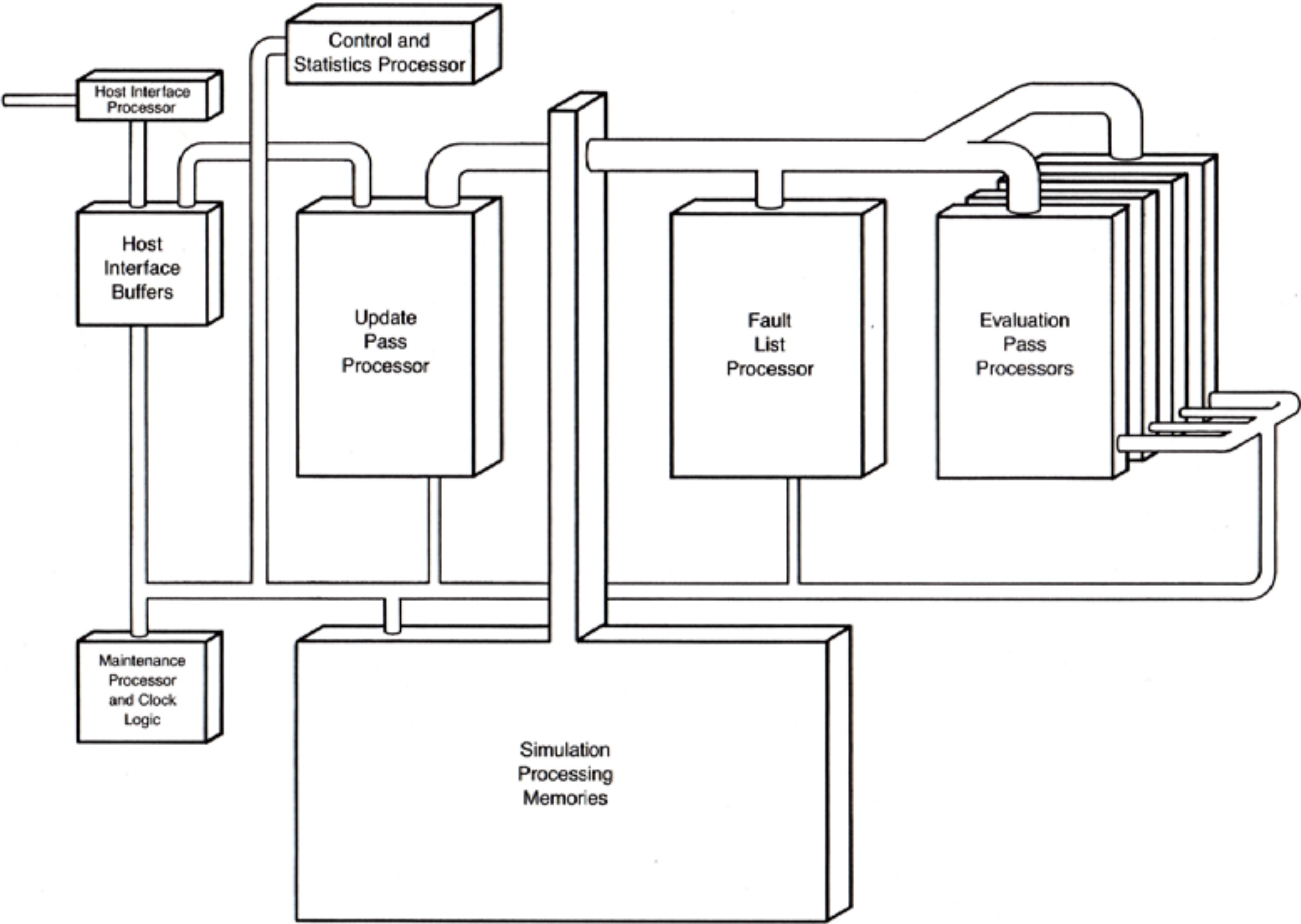
Descriptor Address	Gate Type	Previous Value	Value	Pending Value	Delay Count	Gate I/O Counts		Output Delays		Time of Change Fields	Event Chained Fields	Fault Fields	Behavioral Fields	Misc. Fields	Fan-out Fields			Fan-in Fields	
C	AND	0	1	0	15	2	2	25	20	X	X	X	X	X	F	E		A	B
E	INV	1	0	0	0	1	1	10	10	X	X	X	X	X	10			C	
F	AND	0	1	1	0	1	2	20	20	X	X	X	X	X	10			D	C
10	OR	0	1	1	0	1	2	10	15	X	X	X	X	X	X	X	X	F	E

Status and Data Fields — Fan-out Fields — Fan-in Fields

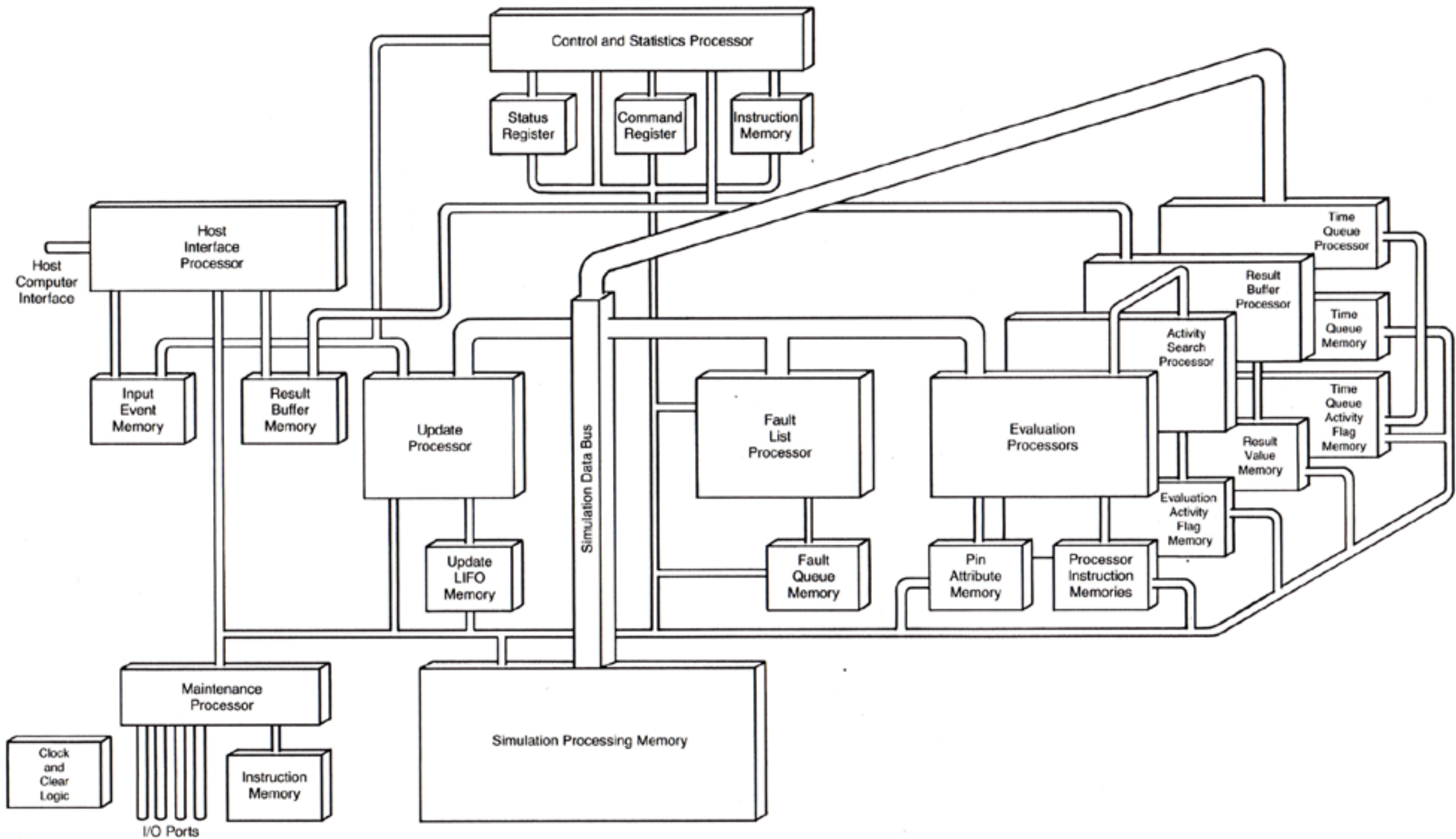
Descriptor Word

Simulation Processing Memories

High Level Block Diagram



Detailed Block Diagram



Final observations

- 2 hours to compile 64K gate design
  - No incremental compile
- 75 I/O pins
- 500+ observation points
- 30 minutes to download compiled descriptors to accelerator
- 11 seconds to simulate 2000 µSec of sim-time
- 3-4 hours to unload accelerator data
  - Pins & observation points
- Only marginally faster than SW simulation
  - Amdahl's Law at work....