Standardized APIs Facilitate Software and Hardware Development of Embedded Multicore Designs

Markus Levy

THE Multicore ASSOCIATION™

EEMBC
An Industry Standard Benchmark Consortium

The Multicore Challenges

- Cores/chip will grow exponentially with each processor generation.
- Higher degrees of heterogeneity
  - Cores
  - Interconnect
  - Hardware acceleration
  - Memory hierarchies
- Shared memory scales linearly with boundaries
- Industry must learn how to efficiently harness this processing capability
- Measuring and comparing performance

Driving Changes

- Development tools, run-time software, languages
- Programming such systems effectively requires new approaches.
- This session covers a variety of multicore development topics that will help address these barriers
  - Standardized APIs for communication and debug.
  - Current approaches being adopted by the Multicore Association.

Multicore Issues to Solve

- Communications, synchronization, resource management
- Debugging
- Distributed power management
- Concurrent programming
- OS virtualization
- Modeling and simulation
- Automatic load balancing
- Algorithm partitioning
- And more.....

How?

Solving Multicore Issues

- Admitting there is an issue
  - The first step in fixing the issue
- Proprietary solutions
  - Temporary, limited, commercially practical
- Research and academic communities
  - Far reaching, big issues
- Industry standard
  - Which one?
  - Many beneficiaries

Formulating the Multicore Association

- Envisaged in February 2005
- Initial engagement began in May 2005
- Formal beginnings assembled in January
  - Legal aspects
- Membership growing
  - *List to be announced in September 2006*

Multicore Association Objectives & Activities

- Multicore ecosystem enablement
- Avoid ‘re-inventing the wheel’
- Industry-wide participation
  - Including system developers, processor vendors, infrastructure developers, device manufacturers, and software and application developers.
- Current efforts
  - Communications & resource management APIs
  - Debug API

Firefighting and Multicore Similarities

- Firefighting is a hot job, multicore is hot technology.
- Firefighting and multicore both have the need to get the job done as quickly and effectively as possible.
- Both require reliable and standardized tools.
- Firefighters always act as a team of two or more, the same for multicore.
- But by far the most important commonality is that they both require excellent communication capabilities.

- Without communication, firefighters don't survive and the cores in a multicore implementation may as well be operating alone.

MPI for Widely Distributed Computing

- Today MPI is the most popular form of message passing APIs for “widely” distributed computing
  - Portable because MPI has been implemented for many distributed memory architectures
  - Fast because each implementation is optimized for the hardware on which it runs)
- MPI is quite complex and will likely in its full form consume memory resources beyond what’s acceptable in a multicore chip’s available memory and introduce too much computational overhead (latency).

OpenMP for Shared Memory Architectures

- Language pragmas for shared memory architectures
  - Supporting multiprocessing programming in C/C++ and FORTRAN on many architectures
- Portable and simplifies expression of threaded code
  - Gives programmers a simple and flexible interface for representing data level parallellism
  - Platforms ranging from the desktop to the supercomputer
- Focused on representing one specific type of task decomposition

Target Domain for Communication API ('CAPI')

- Embedded Multi-processing
- Task to task communication
- Multiple cores on a chip & multiple chips on a board
- Heterogeneous & homogeneous systems
  - Cores, interconnects, memory architectures, OSes
- Scalable: 2 – 1000's of cores
  - Assuming that the 'cost' of messaging/routing isn't greater than the computation of a node
- Allows implementations with significantly lower latency, overhead & memory footprint than MPI and TIPC

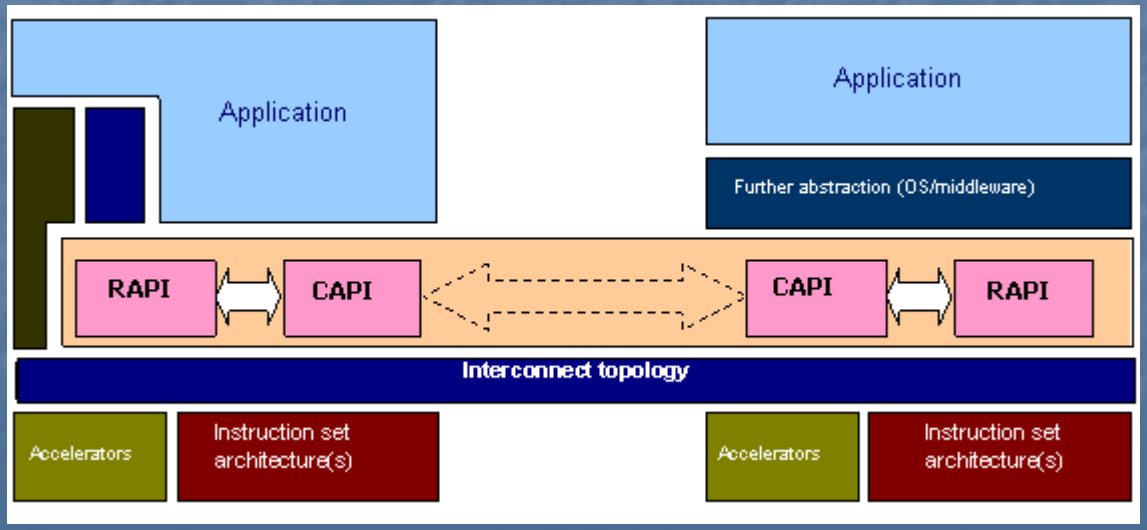
Not What it is, But What it isn't

- OSI
- TCP/IP (or sockets)
- UDP/IP
- TIPC
- RIO
- Sun RPC
- CORBA
- OpenMP
- MPI
- MPIRT
- Embedded MPI

Standards Evaluated, 'Borrowed From', or Disregarded

An Agnostic CAPI

- Unified API designed specifically for communication in a **closely distributed** embedded system (multiple cores on a chip and/or multiple chips on a board)
- Independent of type and number of cores, type of inter-connects, and operating systems
- Forms layer on top of which other abstractions or applications may be built

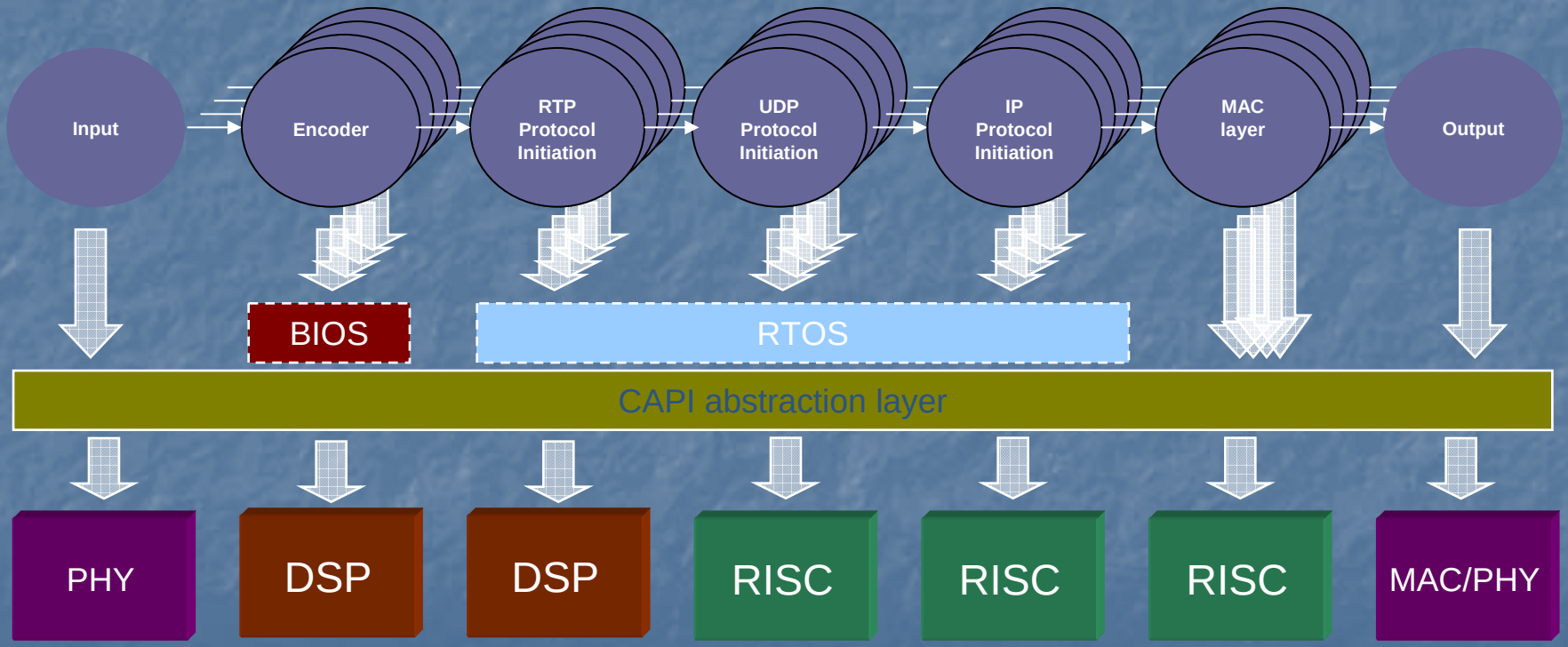


Courtesy of PolyCore Software

Potential Metrics - CAPI

- Throughput
  - Actual vs. hardware
- Latency
  - Processing
  - Delivery
- Overhead
  - Processing
  - Envelope
- Footprint
  - Code
  - Data

Platform/Application Scalability and Global Power Management



What CAPI Does for You?

- An OS vendor
  - A uniform foundation for present/future products
  - Abstracts the platform capabilities
  - Larger addressable market
- A processor and/or chip company
  - More application performance achievable
  - Better determinism compared to MPI
  - Reduced support burden through standardization
- A software developer
  - Better ease of use
  - More powerful development methodologies

Multicore System Developer is Biggest Beneficiary

False Claims on Multicore Debug Support

- Vendors have tools that intrinsically support multicore systems using a direct mapping approach per processor
  - Typically require additional plugins per processor type
  - User interface limited to a few cores (limited by screen size and human comprehension)
- Need for higher level of abstraction with a common debug semantics
  - No one would want 16 different debugger instances running to debug a 16 core system!
- Multicore Association plans to develop a standard debug protocol/API so vendors could support this type of development.

Targets for the Debug Working Group

- Embedded Multicore Systems
- Many on-chip CPUs, perhaps many chips on a board/system
- Homogeneous or heterogeneous
- Inter-operation of multiple vendors & tool chains

Multicore Debug Challenges

- Parallel programming in general: multiple simultaneous threads of execution, such as synchronization bugs
- Differing user experiences for heterogeneous elements; tool-chain and vendor compatibility issues
- Mapping from lower-level implementation constructs (e.g. memory, registers) to higher-level programming model constructs (e.g. messages, synchronization; see CAPI)
- Scale: what do we do beyond 2, 4, 8 cores? Need aggregate control and state inspection

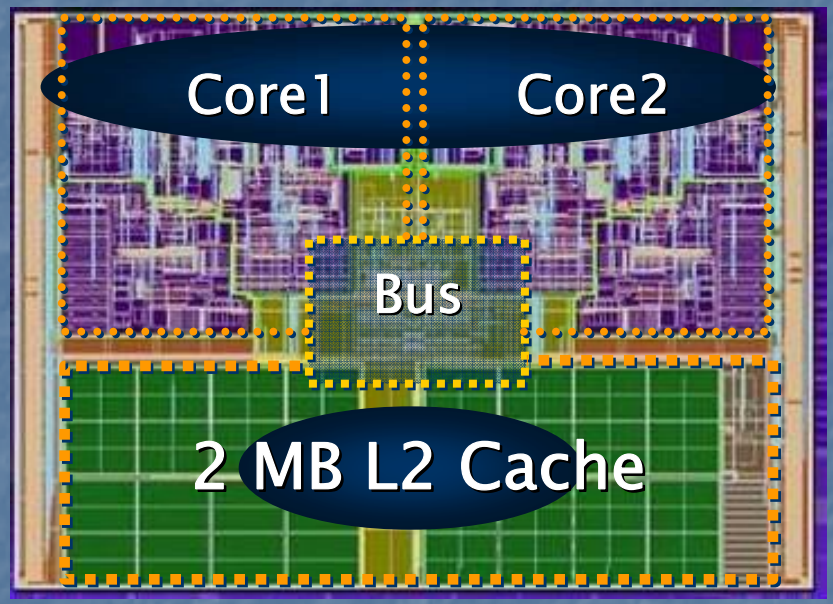
Current Debug Initiatives

- Identify/tie high-level requirements for multicore debugging to specific requirements on underlying infrastructures (i.e. OS or runtime layers)
- Extend debug interfaces in a standardized way to meet the needs of multicore debugging.
- Standardize connection between JTAG interfaces and debuggers
  - Enable 3rd-party debuggers to control systems with multiple cores with different JTAG interfaces from multiple vendors
- Identify appropriate interfaces and drive standards to enable richer and more uniform debugging experience on multicore systems

Extending EEMBC's Scope

- Increase benchmark complexity
- Includes the OS as part of the benchmark
- Moves outside the L1 into memory system to provide an even better view of expected performance
- Supports the industry in the evaluation and future development of MP architectures, including multiprocessor vs. uniprocessor
- MP benchmarks will need to run on top of an OS/RTOS
  - Assume a threading model is available and,
  - Assumes the benchmark can be threaded
- Accelerator based asymmetric heterogeneous MP solutions today lack common programming paradigm
  - Potential support from the Multicore Association

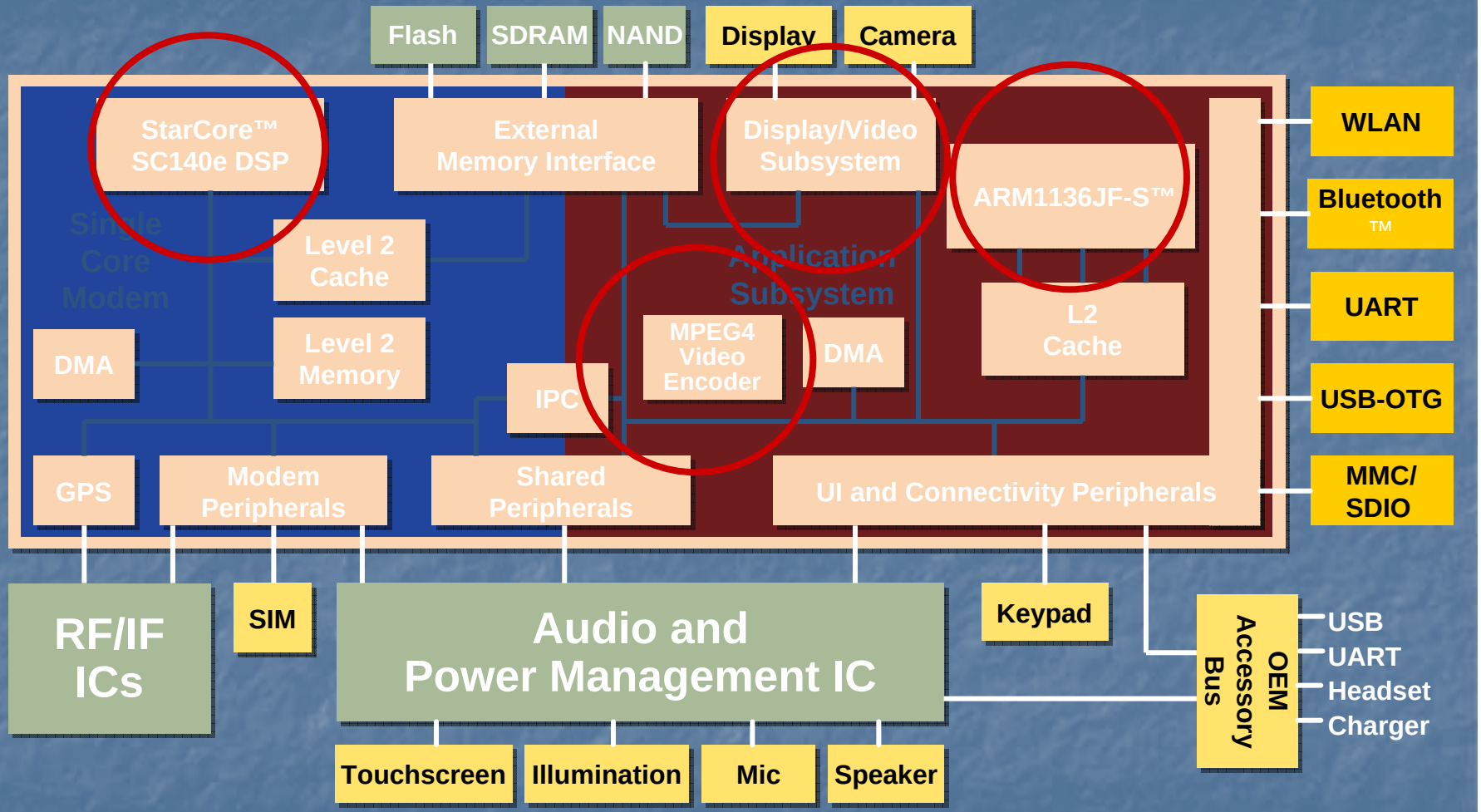
SMP Implementation



Theoretical Maximum Performance = x GHz

Practical Maximum Performance = ?

Highly Integrated AMP



Theoretical Maximum Performance = ?

Practical Maximum Performance = ?

Benchmarks Requirements

- To demonstrate the effectiveness of an application to exploit parallelism as per the forms of concurrency
  - Across multiple data streams
  - When breaking up a bigger problem
  - For running multiple concurrent algorithms
- Benchmarks must also
  - Use standard tool flows, ideal within current EEMBC test harness
  - Also able to run on a uniprocessor to allow MP/uP comparisons
  - Be abstracted to operate across multiple OS and/or hardware platforms

Possible Approaches

- Shared memory
  - Semantics of a thread-based API such as POSIX
- Message Based
  - Implement CAPI support in the EEMBC test harness
  - Define and/or develop test cases based on using CAPI
- EEMBC sees both approaches as valid to support embedded applications which are supporting coherency and distributed memory architectures.

Forms of Concurrency

- EEMBC concentrating on three main forms of concurrency decomposition
  - Concurrency due to the processing of multiple data streams
    - Uses common code over multiple threads
    - Shows how well a solution can scale over scalable data inputs
  - When a single algorithm is decomposed into multiple sub-tasks
    - Multiple threads work on a common data set
    - Shows how well a solution can support fine grain parallelism
  - When a solution does more than one thing at once
    - Concurrency over both the data and instructions
    - Shows the scalability of a solution for general purpose processing

Outline of Shared Memory Approach

Software can assume homogeneity across processing elements

- Where MP is being used for the scalability of general purpose processing, as apposed to accelerator offloading

- Threading is the standard technique to express concurrency
  - Where each thread has symmetric visibility of memory
- Where the test harness abstracts the hardware into basic software threading model

Outline of Message Based Approach

Software can assume heterogeneity across processing elements

- Where MP is being used to offload and accelerate units of work offloading
- Invokes controversy on granularity and units of decomposition (i.e. all specialist cores can be very diverse)

- Messaging is the standard technique to synchronize and communicate between tasks
  - Where each task has a distributed view of memory
- Where the test harness abstracts the hardware to offer a communications infrastructure (i.e. CAPI)

The Importance of Standard APIs

- Standard interfaces simplify development and management of multicore software and reduces time to market
- Standard interfaces broaden the availability of products
- Standard interfaces enable software re-use
  - From ESL modeling to real target hardware
  - From product to product
- Standard interfaces enable more vendor choice

How to Get Involved

- Join the Members of the Multicore Association
  - Participate in an existing working group
  - Start a new working group
  - www.multicore-association.org
- Join the EEMBC membership
  - Gain access to a benchmark standard
  - Participate in the development of next generation benchmarks
  - www.eembc.org