MPSoC – System Architecture
ARM Multiprocessing

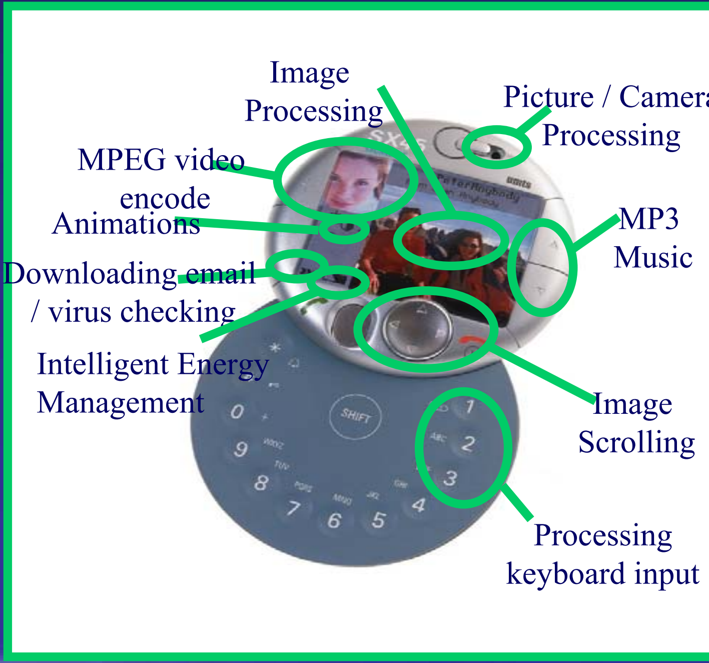
John Goodacre

Program Manager
Multiprocessing

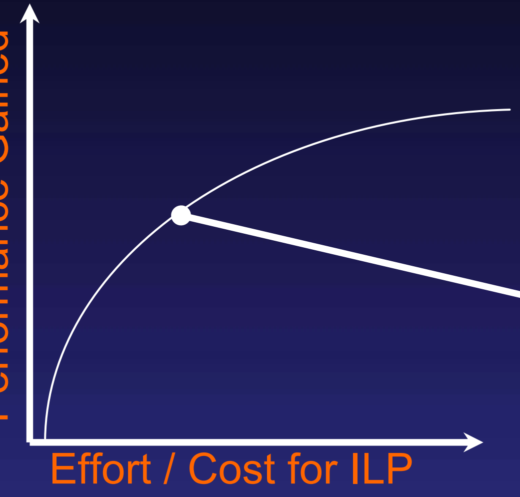
ARM Ltd. U.K.

Balancing the need for Parallelism

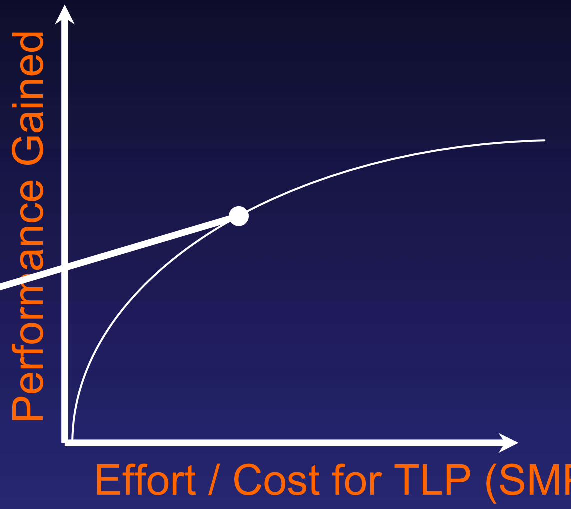
- More bits to more places
  - Signal processing instructions
  - SIMD, MOVE™ media instructions
  - Multi-layer bus architectures
- More instructions per cycle
  - Deeper pipelines
  - Superscalar, VLIW
- More algorithms at once
  - Coprocessors, accelerators
- More applications
  - Multiple processors



Where to best address parallelism



‘Optimal’ support for both ILP and TLP brings the most performance for the least cost / effort

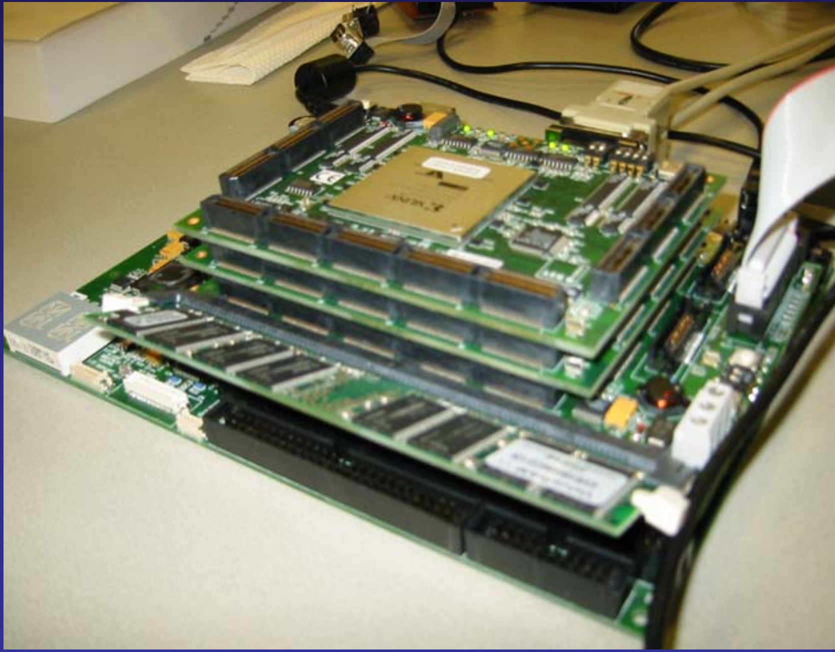


Expensive on hardware to continue to attempt to extract instruction level parallelism (ILP)

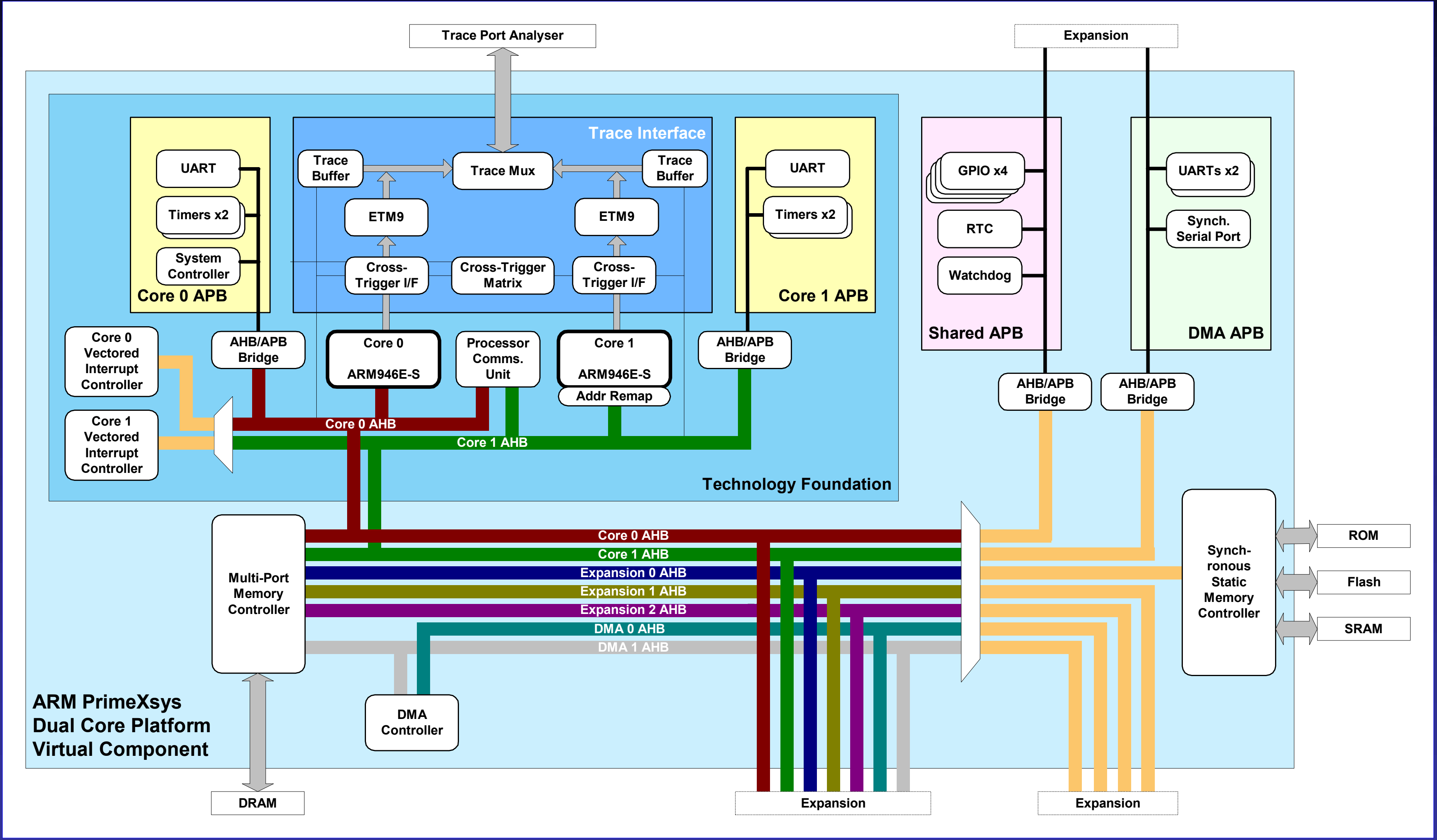
Expensive on hardware and software to continue to attempt to share work between more instances of thread level parallelism (TLP)

Dual Core PrimeXsys™ Platform (DCP)

- Targeted at ‘partitioned system’ designs
  - Channel processor + applications processor
  - Inter-processor communication mailbox
  - Debug cross trigger and trace multiplexing
- Licensable product
  - Includes model using ARM Integrator™ CP
  - Cores running VxWorks
  - Integrated ARM RealView® debug
- Market Targets
  - SOHO switches/routers, Wireless AP, auto-powertrain



DCP Virtual Component



Creating an MP Ecosystem

- ARM RealView Tools
  - Multicore support; ARM plus DSP; ARM plus ARM
  - Debug and Trace, Multi-ICE®
- AMBA™
  - Multi-layer AHB for multi-master system design
  - AMBA design kit including various SoC blocks
  - PrimeCell® Peripheral library
  - AXI (new bus protocol) for high speed
- Software
  - Architecture optimized middleware and libraries
- Platform based design
  - PrimeXsys

Assessing MP (Software Viewpoint)

- Two main models in considering an MP design
  - Asymmetric operation (AMP)
    - 'Static' task allocation
    - Distributed or common view of memory
      - Synchronization and communication via explicit message passing mechanism
    - Either homogeneous or heterogeneous CPUs
      - Manual allocation of work-items within definition of a SoC design
  - Symmetric operation (SMP)
    - 'Dynamic' task allocation
    - Shared view of memory
      - Synchronization and communication via shared state in memory
    - Normally homogeneous CPU arrangement
      - Automatic allocation of work-items within an abstract definition of the SoC design
- Both models are well understood and supported

ARM Multiprocessing

- Hybrid Symmetry Model
  - Support for both SMP and AMP software models
    - Bringing application portability and flexibility
  - Consideration for homogeneous and heterogeneous designs
    - Sponsoring a program with DSP vendors to ensure suitability of formalized standards
    - Architectural Partner reviews for homogeneous MP
  - An ecosystem problem
    - Debug / trace / tools / etc
- A program of working with the ARM Partnership
  - Silicon manufactures and 3rd party IP providers
  - Operating system vendors
  - Device and software manufactures

Outline of what makes an MP Core

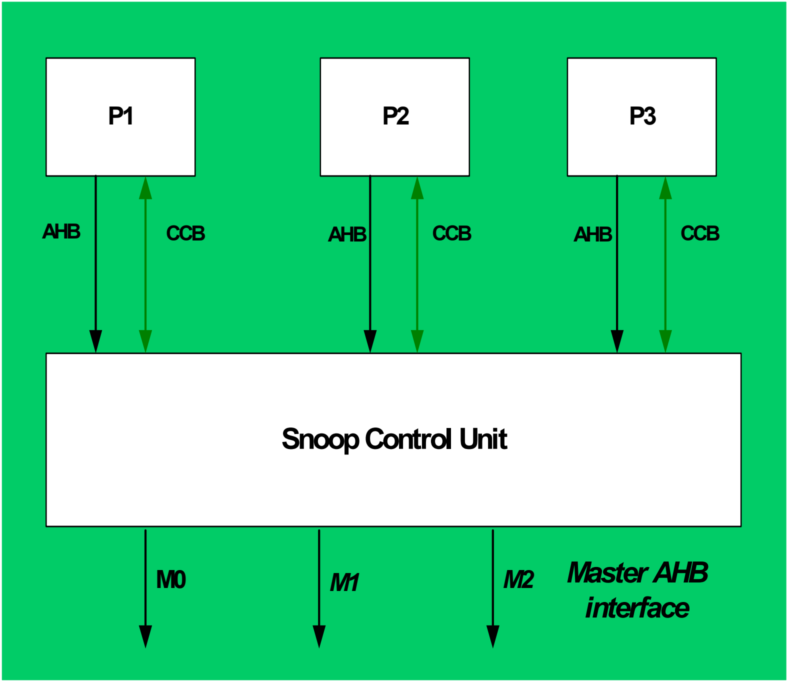
- Can identify itself as MP capable
  - Has a CPUID to uniquely identify CPU to software
  - Ability to indicate need to make memory coherent
- Can maintain memory coherency
  - Caches can participate in a MESI protocol
  - Physical tagged cache (VA to PA translation)
- Provides a consistent view of memory
  - With a defined memory ordering
  - Atomic and synchronization primitives (SWP etc)
- Communication with peer processors
  - Inter-Processor Interrupts (IPI)
  - Message Passing

ARM MP Core Identification

- The ARM architecture defines registers within its coprocessor interface (CP15)
  - For discovering the type of processor and capability
  - For identification of a specific processor in a chip-multiprocessor (CMP)
  - For controlling of aspects of MP behaviour
- The MP system definition also addresses
  - How heterogeneous cores can identify themselves within a ARM core-based design
  - How a platform can understand the interaction of a MP processing core
  - How multiple cores can share SoC services
    - Such as JTAG for debug and trace

Need for Coherency

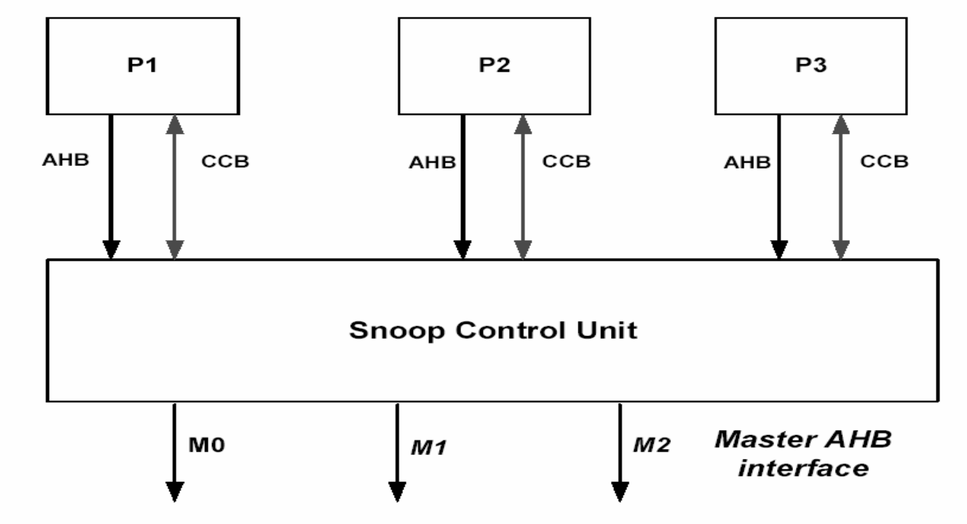
- MPSoC enables SMP to consider designs with a centralize coherency unit
  - Arbitrating access to common memory
  - Supports features such as Direct Data Intervention (DDI)
    - Unifies L1 between CPU
    - Common L2 system
- Maintains a coherent and consistent view of memory



P1/2/3	MP capable cores
M0/1/2	Multi-layer AMBA bus to system memory
AHB	AMBA buses from CPU
CCB	Coherency Control Bus

Snoop Control Unit

- Multiple homogeneous processing cores (Pn)
  - Consider embedded SMP need for maximum of 8
- Uses a MESI coherency protocol in a weak ordered consistency model with Berkeley extensions
  - Can resolve coherency events locally
- Supports applications with hybrid based symmetry
- Support for DDI
  - Direct Data Intervention
- Standard AMBA bus connections



Coherency Control Bus (CCB)

- AMBA sideband signals on CPU data and control interface requests
  - Implements the extended MESI protocol
- Notification of coherent effecting operations from CPU to SCU
  - Changes in cache line state (from running software)
- Assertion of coherent commands from SCU via private channel to CPU
  - Communication of changes that effects all cores
- SCU requesting data from CPU
  - Support for DDI

Memory Consistency

- Important both at the CPU and the system level
- Today's ARM1026EJ-S™ core is (mostly) strongly ordered
  - Between PSO and TSO
- ARM-MP (system) defines:
  - Weak Ordering Model
  - Need for software synchronisation and memory barriers

With respect to program order relaxations, we distinguish models based on whether they relax the memory order:

- **From a Write to a following Read (W=>R)**
- **Between two Writes (W=>W)**
- **From a Read to a Following Read or Write (R=>RW**

In all cases, the relaxation only applies to operation pairs with different addresses.

ARM-MP Provides a 'relaxed' program order for memory consistency

Relaxation	W=>R order	W=>W order	R=>RW order	Read Own Write early
Cacheable Access	Yes (WT/WB)	Yes (WB mode only)	No	Yes
Non-Cacheable Access	No*	No*	No	No*

*Assuming ARM1026EJ-S core; maybe possible with future cores

Memory Barriers

- Hardware Snoop Control Unit (SCU) maintains consistency among private L1 cache
  - Includes consistency over Eviction Write Buffers
- Software through system co-processor interface (CP15) can:
  - Explicitly drain write buffers
  - Manipulate cache state
- Implicit barriers under following instruction
  - SWP (as used in synchronisation)
- Read barriers
  - Not required within relaxed program order

ARMv5 Architecture - Spinlocks

- Atomically serializing SWP instruction

Type of region	Updated behaviour for MP
NCNB	Locked Non cacheable LDR, Locked Non bufferable STR
NCB	Locked Non cacheable LDR, Locked Non bufferable STR
WT Hit	Locked Non cacheable LDR in parallel with the read from the cache, read value taken from the cache, update the cache, locked Non-buffered STR back to main memory.
WT Miss	Locked read, do not update the cache, locked Non-buffered write, do not update the cache
WB Hit	Locked Non cacheable LDR in parallel with the read from the cache, read value taken from the cache, update the cache, locked Non-buffered STR back to main memory.
WB Miss	Locked read, do not update the cache, locked Non-buffered write, do not update the cache

```
typedef struct spinlock
{
    volatile unsigned int lock;
} ____cacheline_aligned spinlock_t;

#define __setlock(x)                    \
          ({                            \
              unsigned int __ret = 1;   \
              __asm__ __volatile__(     \
                    "swp %0, %0, [%1]"  \
                    : "+r" (__ret)      \
                    : "r" (x));         \
                    __ret;              \
          })
#endif

#define spin_lock_init(x) \
    do { *(x) = SPIN_LOCK_UNLOCKED; } while (0)

#define spin_unlock_wait(x) \
    do { barrier(); }
    while(((volatile spinlock_t *)(x))->lock != 0

#define spin_lock(x) \
    do { } while (__setlock(x) == 1)

#define spin_unlock(x) \
    do { ((spinlock_t *)(x))->lock = 0; } while (
```

Cache Management

- Software required to maintain TLB coherence
  - Part of task migration
- Page colouring scheme required to enforce correspondence between virtual and physical index
  - Dependent on cache size and associatively
  - Solved in software by imposing high-bit correspondence between VA and PA
- Additional coprocessor instruction to map VA-to-PA
  - Optimizes cache manipulation (line clean)
- Hardware manages the coherence protocol
  - Additional MESI states and manipulation

Interrupt Load Balancing

- SMP defines dynamic task assignment
  - Ability for OS to balance work-items between cores
    - MP core to simultaneously resolve interrupt requests
    - Maintain prioritization semantics for software
  - A need to understand current system load and anticipated future work load
    - ‘Warmness’ of caches
    - Cost of task migration
    - Task deadlines and schedules
- To work within the definition of a hybrid-symmetric, adaptively configured, MP ‘cluster’
  - No static definition of number of available CPU

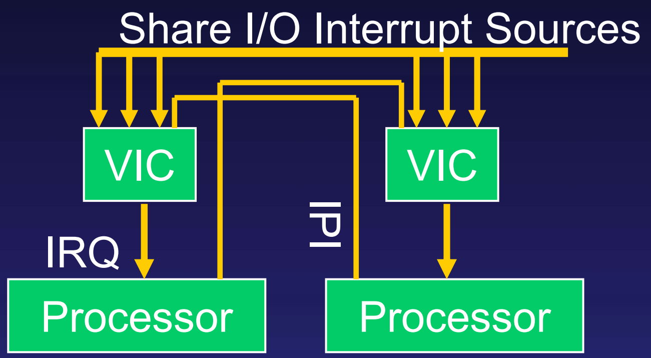
Interrupt Distribution Techniques

- Requirements from various aspects of MP
  - From both SMP and AMP software models
  - Need for software or hardware based load balancing
  - Priority, vectoring, scalability, latency demands
  - Trading existing SoC design (pin based VIC) against newer (emitter/collector) techniques
  - How to integrate IPI and message passing
- Considered high level designs
  - Place existing VIC functionality against each core
  - Insert interrupt routing between SoC VIC system and each core
  - Define a command (as opposed to pin) based emitter/collector routing scheme

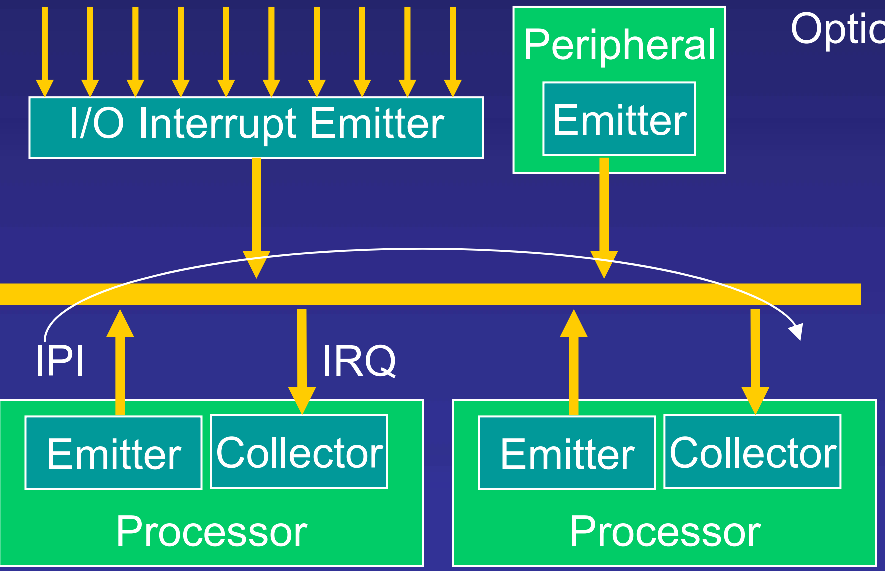
Options for Interrupt Distribution

- Final design for MP still open
  - Option 1: Available today
  - Option 2: Limited system advantage
  - Option 3: Architectural implications

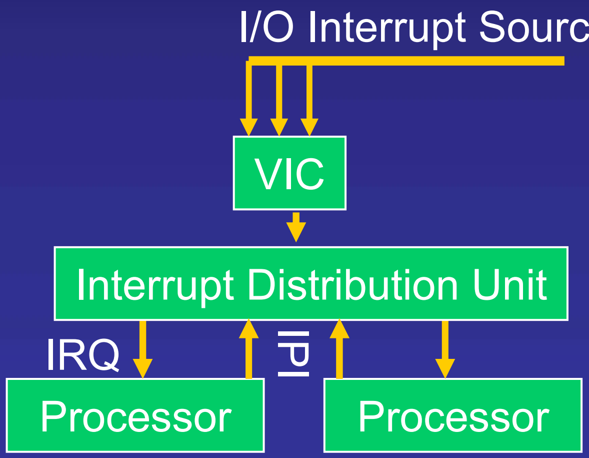
Option 1



Option 3



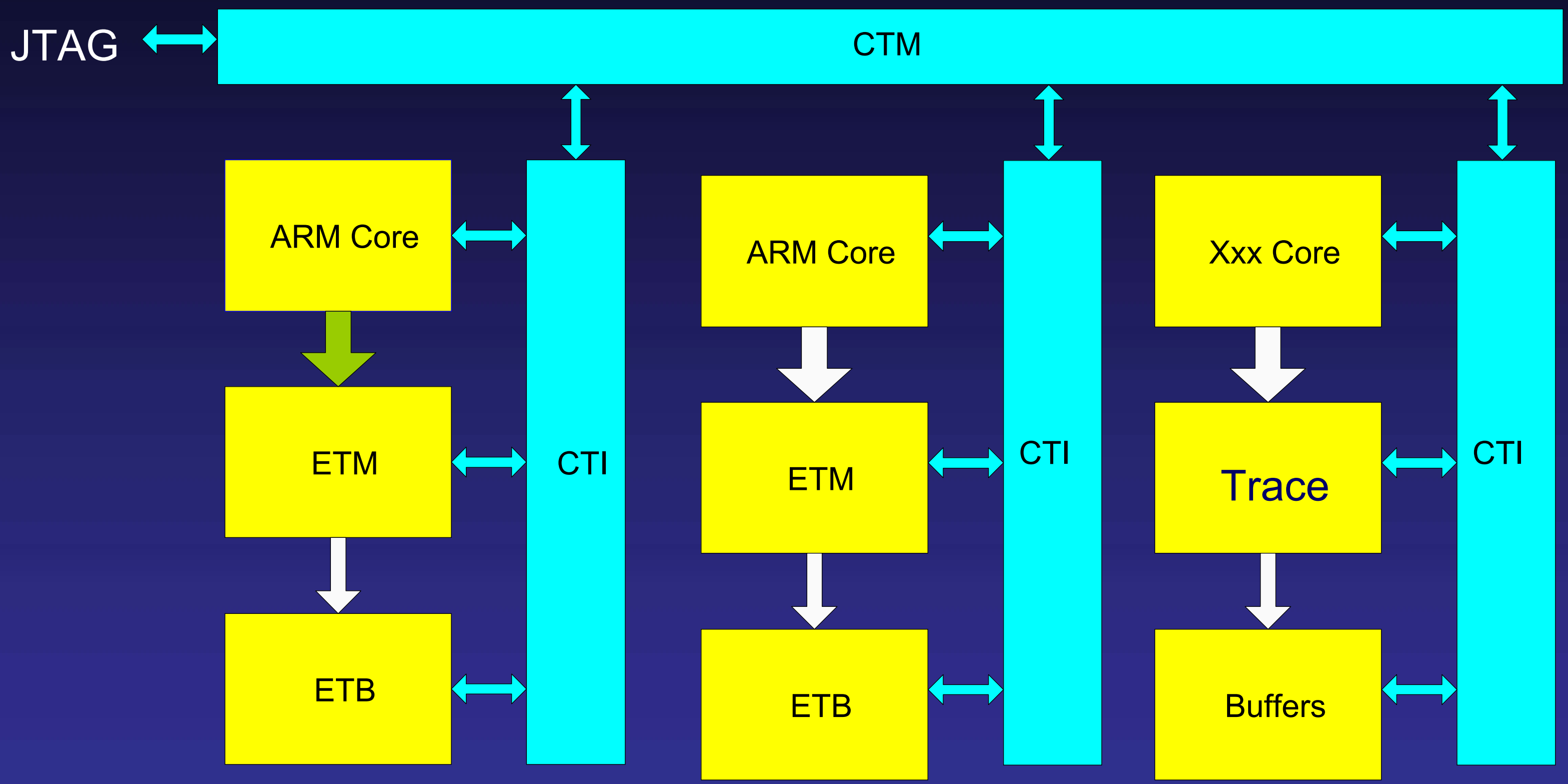
Option 2



CMP Debugging

- MP brings a new set of problems for SoC debug
  - Need to set breakpoints between CPUs
  - Need to synchronize (and compress) trace from each CPU
  - OS awareness for MP within tools
  - Core-based design, need to integrate debug features of hardened cores
- Effects
  - SoC design and IP blocks functionality
  - Tools need to ‘understand’ SoC design
  - Need for a system solution

Debug: Cross Trigger Matrix (CTM)



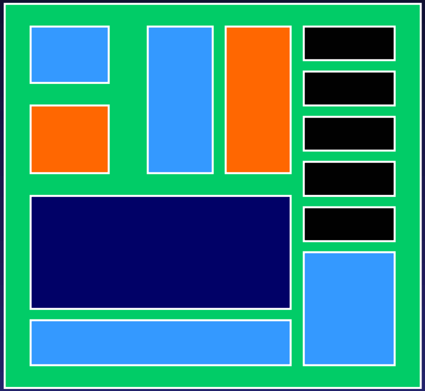
CTM	Cross Trigger Matrix
ETM	Embedded Trace Macrocell™
ETB	Embedded Trace Buffers™
CTI	Cross Trigger Interface

- Multi-core H/W Debug
  - Heterogeneous
  - Homogeneous

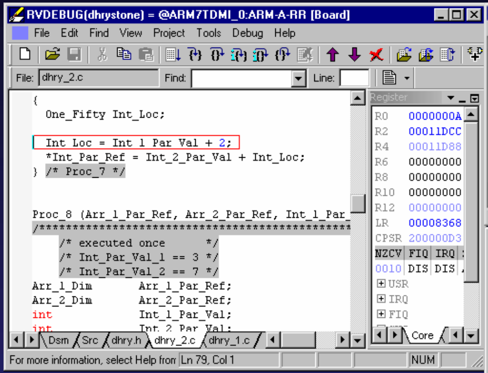
Trace Funnel

- Funnels multiple trace streams from any source to a single trace port or Embedded Trace Buffer
  - Asynchronous clock domains supported
  - All traces are time/cycle correlated
  - Prioritised stalling mechanism for when trace port is overloaded
  - Filters based on trace source ID
  - All ARM core families supported
    - ETM v3.x protocol
  - Other trace protocols supported
    - e.g. DSP or Nexus
- Very simple trace port data format
  - Compatible with any capture device
  - Flexible width and frequency

A System Solution



On-chip features



RVDEBUG(dhrystone) = @ARM7TDMI_0:ARM-A-RR [Board]
File Edit Find View Project Tools Debug Help
File: dhry_2.c
Find:
Line:
Register
Dsm
Src
dhry.h
dhry_2.c
dhry_1.c
Core

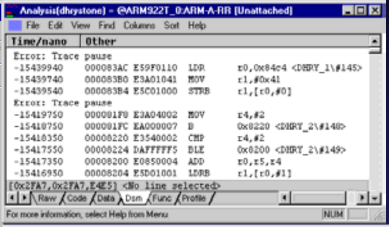
```
{
  One_Fifty Int_Loc;

  Int Loc = Int 1 Par Val + 2;
  *Int_Par_Ref = Int_2_Par_Val + Int_Loc;
} /* Proc_7 */


Proc_8 (Arr_1_Par_Ref, Arr_2_Par_Ref, Int_1_Par_
/**************************************************
    /* executed once      */
    /* Int_Par_Val_1 == 3 */
    /* Int_Par_Val_2 == 7 */
Arr_1_Dim       Arr_1_Par_Ref;
Arr_2_Dim       Arr_2_Par_Ref;
int             Int_1_Par_Val;
int             Int_2_Par_Val;
```

```
R0   0000000A
R2   00011DCC
R4   00011D88
R6   00000000
R8   00000000
R10  00000000
R12  00000000
LR   00008368
CPSR 200000D3
NZCV FIQ IRQ
0010 DIS DIS
⊞ USR
⊞ IRQ
⊞ FIQ
```

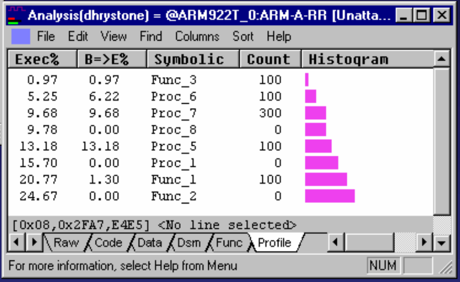
For more information, select Help fron Ln 79, Col 1 NUM



Analysis(dhrystone) = @ARM922T_0:ARM-A-RR [Unattached]
File Edit View Find Columns Sort Help
Raw
Code
Data
Dsm
Func
Profile

```
Time/nano    Other
Error: Trace pause
-15439940    000083AC E59F0110  LDR    r0,0x84c4 <DHRY_1\#145>
-15439740    000083B0 E3A01041  MOV    r1,#0x41
-15439540    000083B4 E5C01000  STRB   r1,[r0,#0]
Error: Trace pause
-15419750    000081F0 E3A04002  MOV    r4,#2
-15418750    000081FC EA000007  B      0x0220 <DHRY_2\#140>
-15418350    00008220 E3540002  CMP    r4,#2
-15417550    00008224 DAFFFFF5  BLE    0x0200 <DHRY_2\#149>
-15417350    00008200 E0850004  ADD    r0,r5,r4
-15416950    00008204 E5D01001  LDRB   r1,[r0,#1]
[0x2FA7,0x2FA7,E4E5] <No line selected>
```

For more information, select Help from Menu NUM



Analysis(dhrystone) = @ARM922T_0:ARM-A-RR [Unatta...
File Edit View Find Columns Sort Help
Raw
Code
Data
Dsm
Func
Profile

Exec%	B=>E%	Symbolic	Count	Histogram
0.97	0.97	Func_3	100	
5.25	6.22	Proc_6	100	
9.68	9.68	Proc_7	300	
9.78	0.00	Proc_8	0	
13.18	13.18	Proc_5	100	
15.70	0.00	Proc_1	0	
20.77	1.30	Func_1	100	
24.67	0.00	Func_2	0	

[0x08,0x2FA7,E4E5] <No line selected>

For more information, select Help from Menu NUM

Debugger

JTAG Emulator
& Trace capture

Power, Performance, Area (PPA)

- Low power is as, if not more, important than performance for mobile embedded devices
  - CMP designs look to offer a single design that
    - Can be run with very low power consumption, or
    - Can be run with high performance levels
- Each aspect has been researched
  - **The Case for a Single-Chip Multiprocessor**
    - Olukotun, Hayfeh, Hammond, Wilson and Chang Computer Systems Laboratory, Stanford University
  - **An Adaptive Chip-Multiprocessor Architecture for Future Mobile Terminals**
    - Mladen Nikitovic and Mats Brorsson Department of Microelectronics and Information Technology, Royal Institute of Technology

CMP – Premise for Performance

- It takes less hardware for a programmer to have expressed thread / application level parallelism than for hardware to extract instruction level parallelism
  - Uses less silicon for higher performance
- It is easier to implement multiple simple designs at higher frequencies than one complex design
  - More efficient use of advanced silicon processes

Information referenced from research paper:

The Case for a Single-Chip Multiprocessor

Olukotun, Hayfeh, Hammond, Wilson and Chang
Computer Systems Laboratory, Stanford University

Stanford, CA. October 1996

CMP - Performance Findings

- Up to double the performance from the same die area from a CMP design running applications with large grained thread-level parallelism
- CMP designs are simpler to implement and can run at a higher frequency than the similarly sized uniprocessor

UP vs CMP extrapolated actual processor

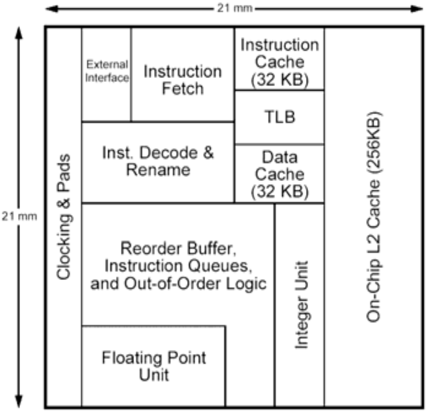


Figure 2. Floorplan for the six-issue dynamic superscalar microprocessor.

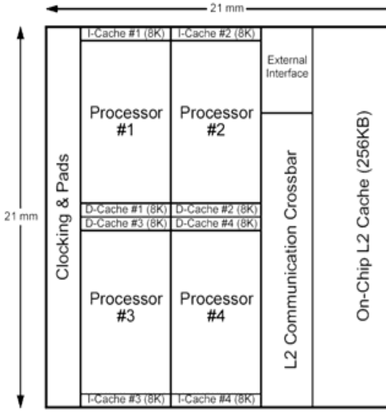
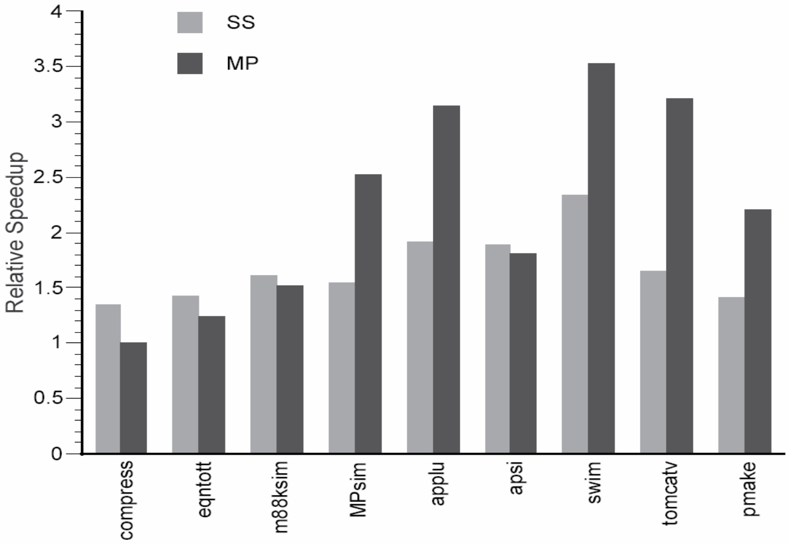


Figure 3. Floorplan for the four-way single-chip multiprocessor.



CMP – Premise for Low Power

- Higher core frequencies drive for higher voltages leading to cubic increase in power requirements
  - Intelligent Energy Management (IEM) provides cubic savings in power consumption on CMP's already lower core frequency and voltages.
- Devices use energy by both consuming power statically (leaking) and dynamically (switching)
  - Adaptive Power Management shuts off entire CPUs within a CMP when not required for performance

Information referenced from research paper:

An Adaptive Chip-Multiprocessor Architecture for Future Mobile Terminals

Mladen Nikitovic and Mats Brorsson
Department of Microelectronics and Information Technology, Royal Institute of Technology
Kista Sweden. October 2002

CMP - Power Consumption Findings

- Adaptive CMP is up to twice as power efficient as a *perfectly* Intelligently Energy Managed Uniprocessor
- IEM plus Adaptive CMP could save more power

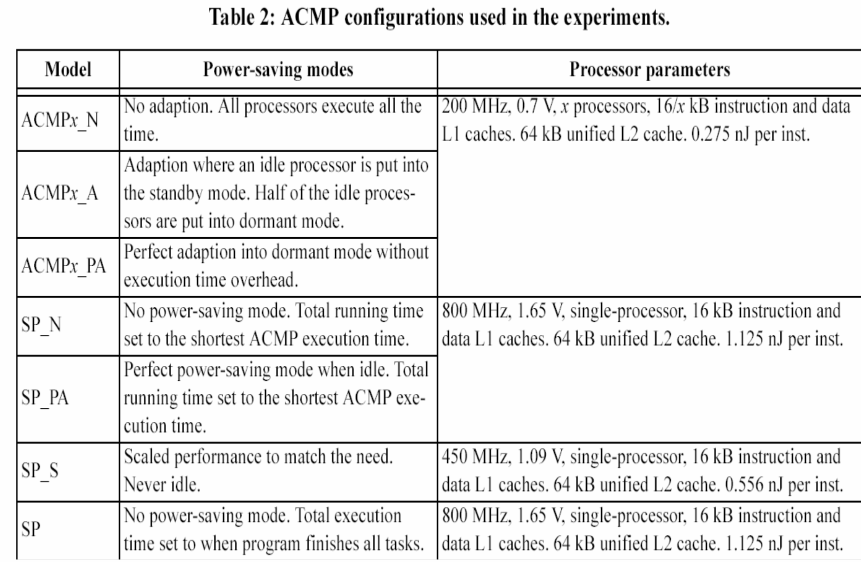


Table 2: ACMP configurations used in the experiments.

Model	Power-saving modes	Processor parameters
ACMP*x*_N	No adaption. All processors execute all the time.	200 MHz, 0.7 V, *x* processors, 16/*x* kB instruction and data L1 caches. 64 kB unified L2 cache. 0.275 nJ per inst.
ACMP*x*_A	Adaption where an idle processor is put into the standby mode. Half of the idle processors are put into dormant mode.	
ACMP*x*_PA	Perfect adaption into dormant mode without execution time overhead.	
SP_N	No power-saving mode. Total running time set to the shortest ACMP execution time.	800 MHz, 1.65 V, single-processor, 16 kB instruction and data L1 caches. 64 kB unified L2 cache. 1.125 nJ per inst.
SP_PA	Perfect power-saving mode when idle. Total running time set to the shortest ACMP execution time.	
SP_S	Scaled performance to match the need. Never idle.	450 MHz, 1.09 V, single-processor, 16 kB instruction and data L1 caches. 64 kB unified L2 cache. 0.556 nJ per inst.
SP	No power-saving mode. Total execution time set to when program finishes all tasks.	800 MHz, 1.65 V, single-processor, 16 kB instruction and data L1 caches. 64 kB unified L2 cache. 1.125 nJ per inst.

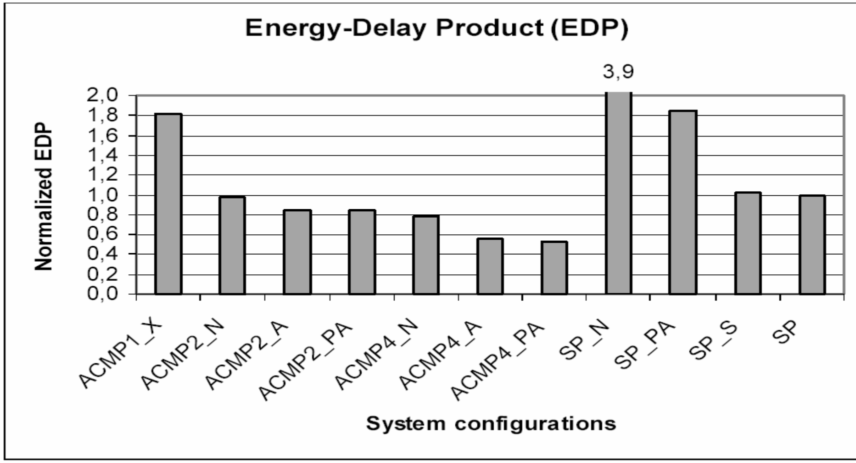


Figure 5. The Energy-Delay Product of the different models

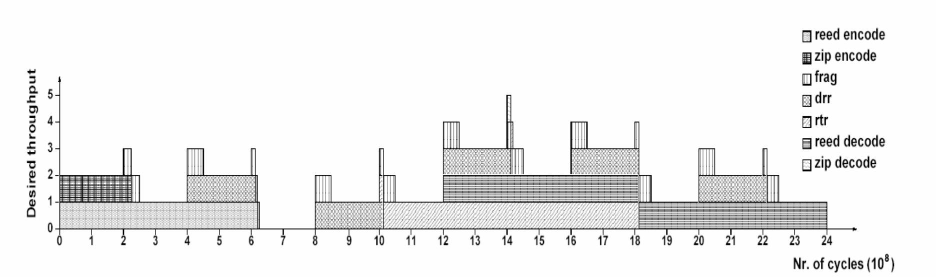
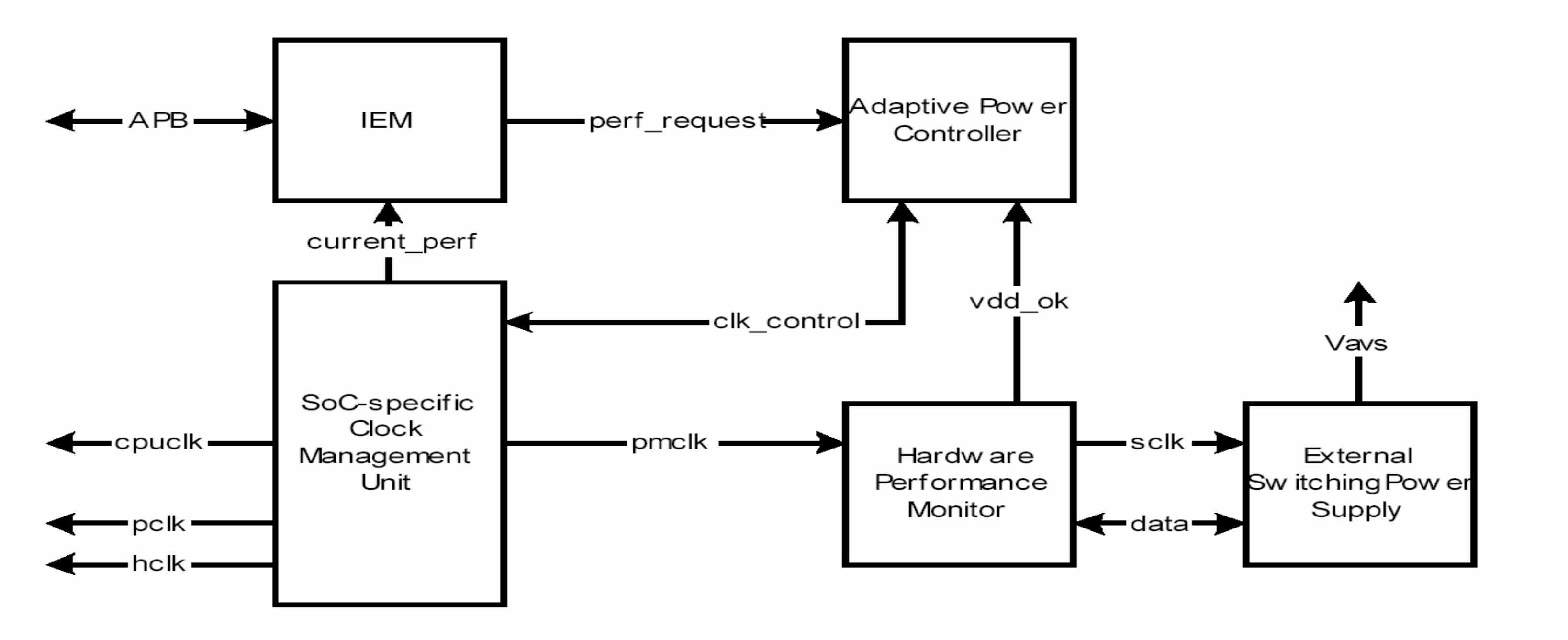


Figure 2. Workload scenario using the Commbench applications.

Intelligent Energy Management (IEM)


APB
IEM
perf_request
Adaptive Pow er Controller
current_perf
clk_control
vdd_ok
Vavs
SoC-specific Clock Management Unit
cpuclk
pclk
hclk
pmclk
Hardw are Performance Monitor
sclk
data
External Sw itching Pow er Supply

- ARM & National Semiconductor Collaboration
  - To bring licensable IP to market
  - Predictive closed loop / adaptive voltage scaling
  - Set optimal performance level to reach deadlines

However: Uniprocessor Designs

- Still very relevant in embedded devices
  - Many of today's OS and RTOS are not MP aware
  - Continuous need for more application performance
  - Huge libraries of non-TLP aware code
- Further resolves challenges for low power in a well understood system design
  - Software for adaptive closed loop voltage and frequency scaling
  - Optimized ILP extraction (useful in future for MP)
- Some performance hungry algorithms can't expose TLP
  - Most folks minds are single threaded !
  - The 'optimal' MP solution will balance ILP & TLP

Operating System support for MP

- Various levels of support for MP hardware
  - Full dynamic scheduling on SMP
  - Application visibility of additional cores
  - Additional cores ‘hidden’ behind library / application
- Various levels of maturity
  - Good support for course-grain TLP
    - POSIX thread etc
  - Limited exposure to CMP designs
    - Not optimized for ‘closeness’ of cores
  - Mostly research-level support for expressing fine-grained TLP
    - OpenMP or specific MP languages
- Emerging interest in abstracting TLP away from programmer (eg. Via speculative parallelism)

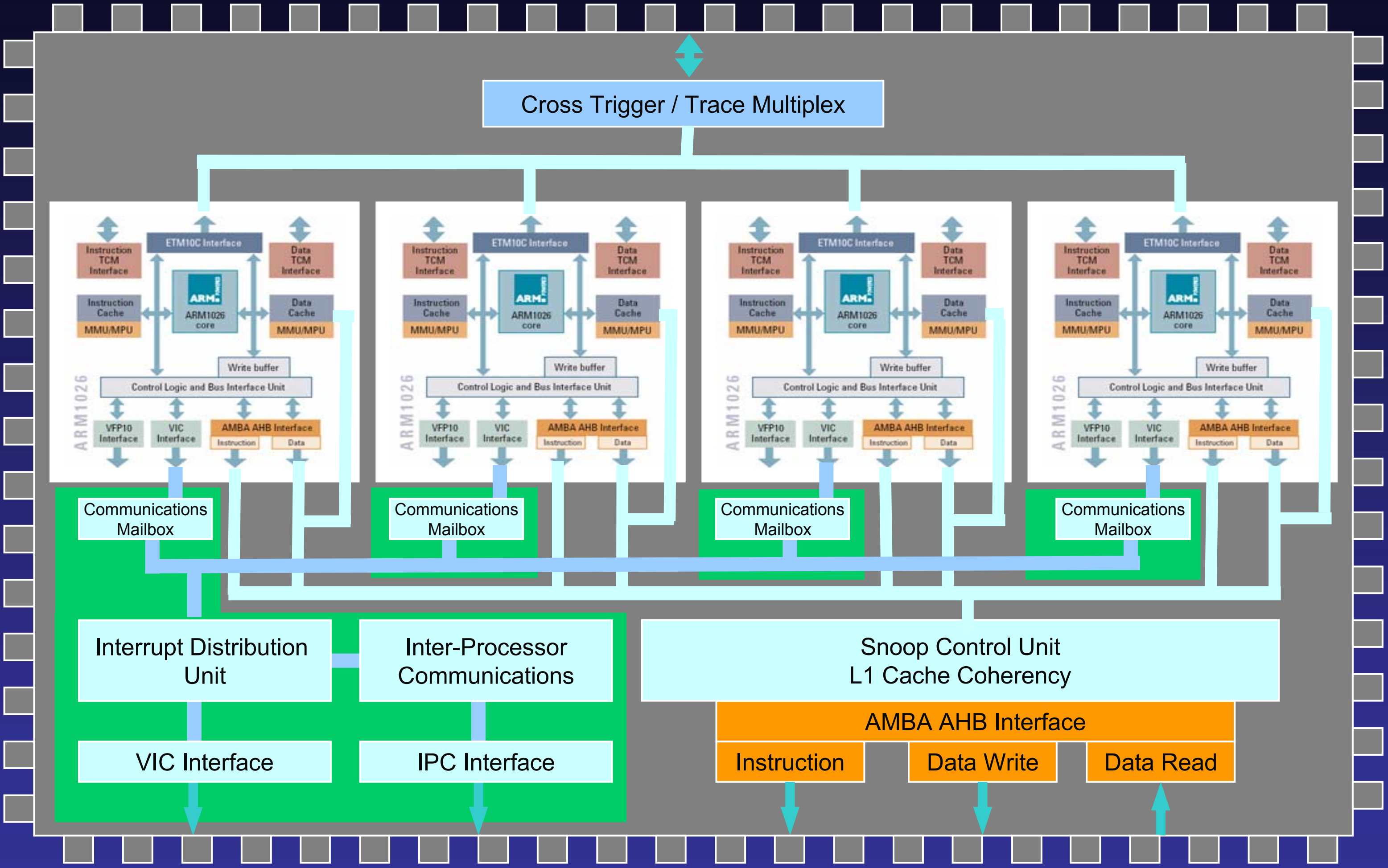
MP Middleware Considerations

- Message Passing Interface (MPI) on ARM-IPC mailbox for communication via common memory
  - Defines 4 x 32bit word direct core-core data path
  - Driver for ARM IPC interface (ADI veneer)
  - Collaborating with DSP vendors on heterogeneous interconnect standards
  - http://www-unix.mcs.anl.gov/mpi/
- OpenMP for expressing the finer-grained thread parallelism that can be supported on CMP designs
  - Mostly an issue for tool-chain vendors
  - http://www.compunity.org/
  - http://phase.etl.go.jp/Omni/home.html

Is Our Future Speculative?

- Can you automate (hardware or software) the extraction of TLP from linear applications?
  - Over general code
  - For 'media' processing
  - Within execution environments / virtual machines
- Does this effect a MP platform design ?
  - Need for hardware threads ?
  - A hardware scheduler ?
- ARM's current target is for declarative parallelism
  - Minimizing resources required to benefit from MP
  - Maximizing performance and energy conservation

Testing the Design



- ARM MP Trial Demonstrator Platform

Conclusions

- We’ve been building an MP ecosystem for a while
  - Releasing MP capable products
  - Partners are building lots of MP designs
- ARM believes (SMP) chip multiprocessing is very interesting for embedded devices both from the
  - Performance from a given area of silicon, and the
  - Energy required to get a specific performance
- So we’re now at the stage to:
  - Work with the partnership to formalize MP support
  - Introduce capabilities for SMP
  - Make available a trial implementation of a ARM MP platform design for partner evaluation / feedback