SoC Architectures for Hardware Designers

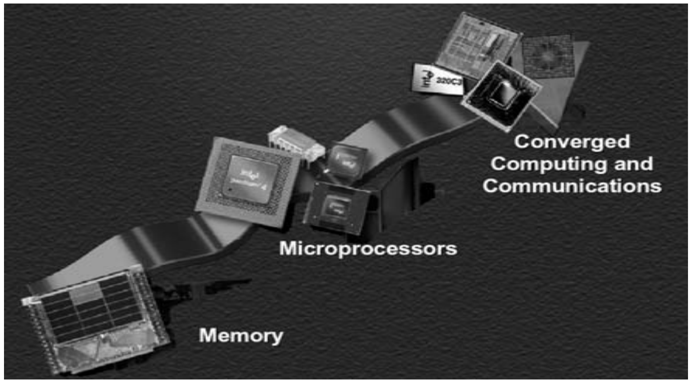
3rd International Seminar on
Application-Specific Multi-Processor SoC
7 - 11 July 2003, Hotel Alpina, Chamonix, France

Trevor Mudge
Bredt Professor of Engineering
The University of Michigan, Ann Arbor
http://www.eecs.umich.edu/~tnm

Outline of Tutorial

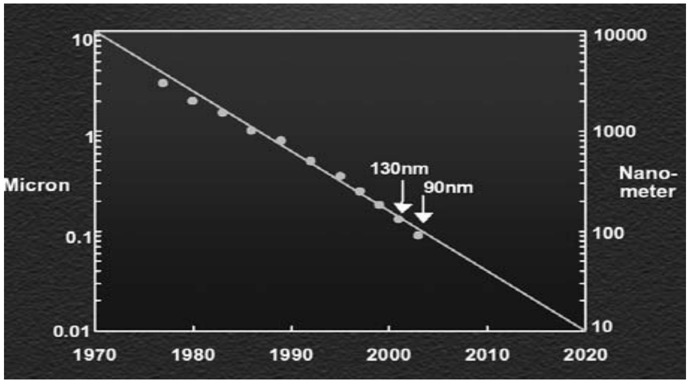
- Technology opportunities and limits
- What is a System-on-a-Chip – SoC

Silicon is the Engine: Andy Grove's Address at Dec. 2002 IEEE International Electron Devices Meeting



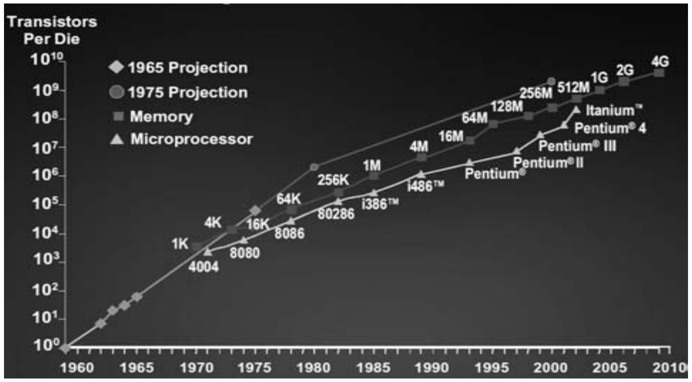
source: Intel

Where is technology heading?



source: Intel

Where is technology heading?



source: Intel

Flavors of Integrated Circuits

- Digital – signals are quantized to 2 levels
  - permits “infinite” precision
  - microprocessors etc.
- DRAM – dynamic random access memory
  - variant of above specialized for high density
- Analog – value of voltage models quantity exactly
  - low precision
  - only use when digital is not feasible
  - radio receivers and transmitters
- Difficult to mix any two in one die

Limits

- Moore’s Law
- Power
- Mask Cost
- Complexity
- Return on Investment

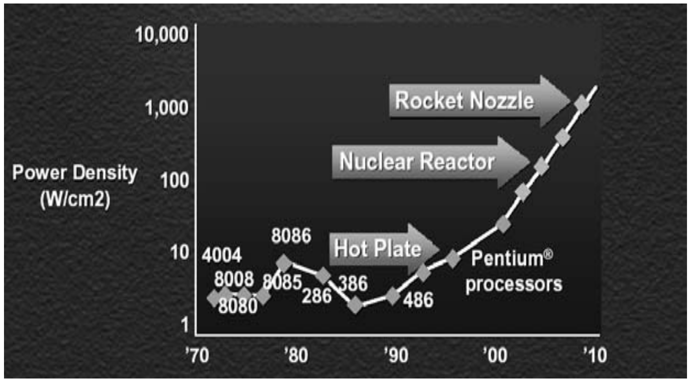
Limits: Moore’s Law

- Moore’s Law
  - the number of transistors on a given chip can be doubled every two years
  - principle of progress in electronics and computing since Moore first formulated the famous dictum in 1965
  - for the same amount of time, people have predicted it would hit a wall.
- Future Generations of Si Technology
  - double density = reduce line width by 0.7x
  - 130nm ➔ 90nm ➔ 60nm ➔ 45nm ➔ 30nm
  - 2 or 3 years between generations
  - ~10 ± 2 Years
  - after 2015 – paradigm shift to a non-Si technology
  - be careful about betting on that
- Moore’s law no limits for next 10 years

Limits: Power

- It’s not just transistor density that has grown exponentially ….

Power: The Current Battleground



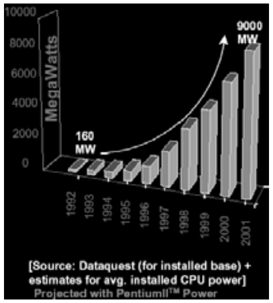
source: Intel

Total Power of CPUs in PCs

I992 – 90M CPUs @ 1.8W = 180MW

today – 500M CPUs @ 18W = 10,000MW

Four Hoover dams



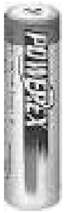
[Source: Dataquest (for installed base) + estimates for avg. installed CPU power]
Projected with PentiumII™ Power

Low power has other implications …

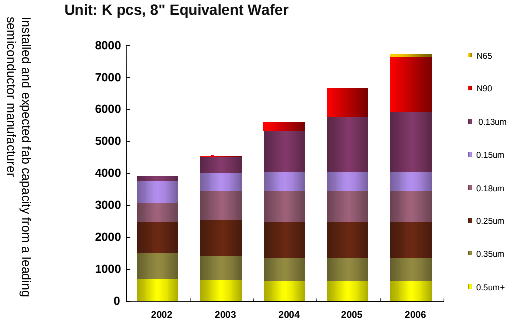
- Low power has been the technology that defines mainstream computing technology
  - Vacuum tubes → silicon
  - TTL → CMOS
  - microprocessors
- 1950’s “supercomputers” created the technology
- 1980’s supercomputer are the beneficiaries of microprocessor technology

What hasn’t followed Moore’s Law

- Batteries have only improved their power capacity by about 5% every two years



Limits: Mask Cost



- Today greatest volume in 0.25, followed by 0.18 and 0.15
- Next year perhaps 0.13 processes
- Older processes do not just disappear…

Limits: Mask Cost

- Closer to leading edge ➔ higher cost masks
- Volume is necessary
  - often means more programmable to achieve volume
- If application specific ness limits volume ➔ older process

Limits: Complexity

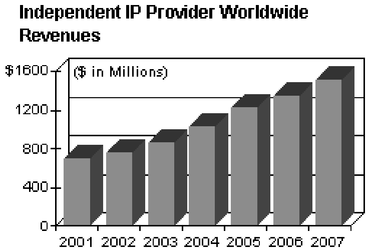
- Problems include
  - design time and effort
  - validation and test
- Hardware
  - SoC of previously defined parts
- Software
  - bigger challenge
  - 10x hardware costs
  - why run-time reconfigurable hardware may not be a good idea

Limits: Return on Investment

- Return on investment of fabs
  - Mid 60’s < $1M
  - Mid 70’s $3M
  - Early 90’s $1B
  - ’02 $3B
  - 2010 $??B
- Different business models
  - separate design and fab

Fabless IP Providers

- Business model is based upon the development and sale and/or licensing of pre-defined, fully-characterized, semiconductor functional cores
- In 2002, increased by 8.4% from 2001's $698.4 million
- Forecast to reach $1,503.3 million by end 2007



Source: In-Stat/MDR 4/03

What *is* An SOC?

- Its that part of a platform that can be cost-effectively integrated onto one chip
- Why not the whole thing?
- Because: Analog and DRAM

What *is* A Platform?

- A programmable collection of digital components targeted to a class of applications
- Platforms are usually complete enough to load and boot an OS

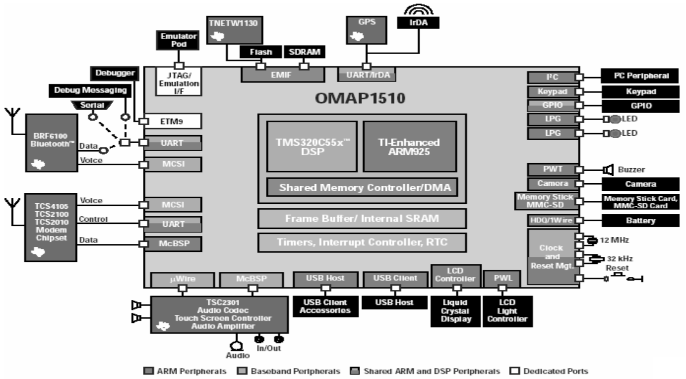
How Does a Platform Get Defined?

- Someone has an idea, sells it to a large tier-one OEM
- If the OEM thinks it's a good idea they ask their platform providers (i.e., ST and TI) to include that functionality in their platforms
- That someone with an idea of course could be: ARM with Jazelle, Nokia (i.e. an OEM), or ST with a coprocessor idea
- Typically the tier-one OEM limits ST or TI from selling the platform to anyone else in the same form
- The resulting ASSP (application specific standard part) that gets defined is slightly modified
- Another view:
  - tier-one OEMs get all the bits they really want in a platform
  - tier-two OEMs are usually satisfied with something that almost does that job and is cheap

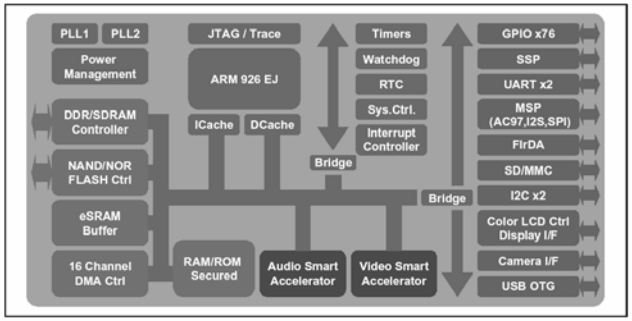
Four Examples

- Texas Instruments OMAP 1510
- STM Nomadik
- Intel PXA800F
- PDA/Communicator – University of Michigan
- Common features

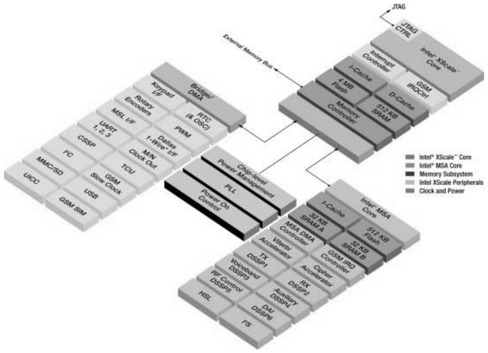
TI: OMAP



TI: Nomadik



Intel: PXA800F



PXA800F

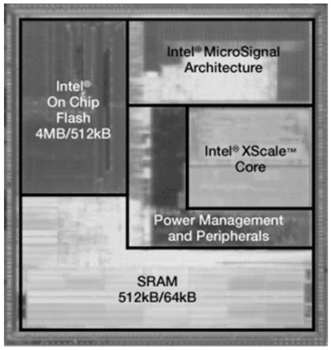
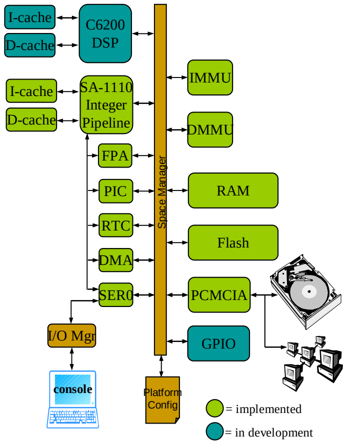

Intel® MicroSignal Architecture
Intel® On Chip Flash 4MB/512kB
Intel® XScale™ Core
Power Management and Peripherals
SRAM 512kB/64kB

Figure 1. Intel® PXA800F Processor

PDA/Communicator

iPAQ - like


I-cache
D-cache
C6200 DSP
I-cache
D-cache
SA-1110 Integer Pipeline
FPA
PIC
RTC
DMA
SER0
I/O Mgr
console
Space Manager
Platform Config
IMMU
DMMU
RAM
Flash
PCMCIA
GPIO
= implemented
= in development

Commonality: Heterogeneous Multiprocessors

- Control processor
- “Data plane” processor
- Analogous to the control and data of a program – not a pure separation either
- Data plane ➔ digital signal processor
- Other components are usually small but essential ingredients if OS is to be booted or to interface to the external world

Major Components

- Interconnect
  - current architectural paradigm uses buses
  - AMBA
- Control processors
  - standard general purpose processors
  - 1-2 generations behind state-of-the-art architecture
- Data plane processors
  - standard DSPs

Why Standard Part Processors

- Software (10x hardware)
- Tool chain – more software

Interconnect: Buses

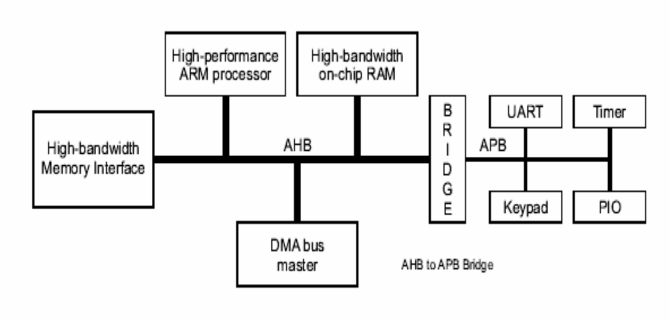
- What is a bus?
- A definition of a set of signals for broadcasting signals
- Strengths
  - inexpensive support for many-to-many connections provided they don’t overlap in time
  - multidrop
- Weakness
  - bandwidth limitation
  - high drive needs
- Future alternatives
  - point-to-point communication
    - essential for streaming data
- Network on a chip
  - leverage existing communications technology
  - need to simplify

Open Standard Bus: AMBA

- Advanced Microprocessor Bus Architecture
- On-chip bus proposed by ARM
- Very simple protocol
- High bandwidth bus
  - AHB – Advanced High-performance Bus
  - AXI protocol
- Low bandwidth bus
  - APB – Advanced Peripheral Bus
- Next generation high performance bus

On-Chip Bus (OCB)

- Interconnect components inside a single chip


High-performance ARM processor
High-bandwidth on-chip RAM
B R I D G E
UART
Timer
High-bandwidth Memory Interface
AHB
APB
Keypad
PIO
DMA bus master
AHB to APB Bridge

AMBA AHB Features

- Burst transfers
- Split Transactions
- Single cycle bus master handover
- Single clock edge operation
- Non-tristate implementation
- Wide data bus configurations supported
  - 64/128 bits

AMBA APB Features

- Low power
- Latched address and control
- Simple interface
- Suitable for many peripherals
- No wait state allowed
- No burst transfers
- No arbitration (bridge the only master)
- No pipelined transfer
- No response signal

AMBA AXI Features

- Separate Address / Control and data phases
- Supports Unaligned data transfers
- Burst-based Transactions
- Separate read / write channels for DMA
- Ability to issue Multiple outstanding Addresses
- Out-of-order Transaction Completion
- Easy Addition of Register Stages

Processors

- Control-type
  - – parallelism
  - – ARM processors
  - – Initially thought of as a low power solution
- Data plane
  - – Texas Instruments TMS32C6200
  - – Early DSP vendor – libraries & solutions

Architectural Approaches to Parallelism

- Process level parallelism
  - – Homogeneous
    - Tessellations of processors
    - MMP
    - SMPs
  - – Heterogeneous
    - SOC
    - Control processor and application specific processors

Architectural Approaches to Parallelism

- Instruction level parallelism
  - – Pipelining and multiple instruction issue
- Superscalar processors
  - – Hardware detects dependencies
  - – Responsible for scheduling instructions
- VLIW processors
  - – No hardware overhead
  - – Parallelism detected in software

Pros and Cons

- Superscalar
  - – Pros: run-time parallelism detected
  - – Cons: complex and consumes area and energy
- VLIW
  - – Pros: simple hardware
  - – Cons: software is much more complex

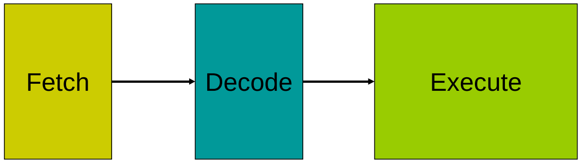
Where do they fit in an SOC

- Control Plane
  - – Superscalar – just
  - – Dominated by run-time conditional branches
- VLIW
  - – Digital signal processing
  - – Data parallel applications

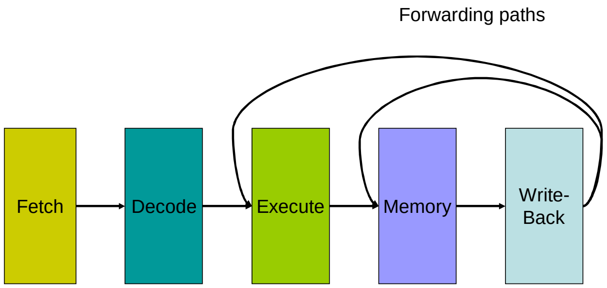
ARM Architecture Comparison

Feature	ARM7	ARM9E	ARM11
Architecture	ARMv4	ARMv5TE(J)	ARMv6
Pipeline Length	3	5	8
Java Decode	None	(ARM926EJ)	Yes
V6 SIMD Instructions	No	No	Yes
MIA Instructions	No	No	Yes
Branch Prediction	No	No	Dynamic
Independent Load-Store Unit	No	No	Yes
Instruction Issue	Scalar, in-order	Scalar, in-order	Scalar, in-order
Concurrency	None	None	ALU/MAC, LSU
Out of Order completion	No	No	Yes
Target Implementation	Synthesizable	Synthesizable	Synthesizable and Hard Macro
Performance Range	Up to 150Mhz	Up to 250Mhz	350Mhz - >1GHz

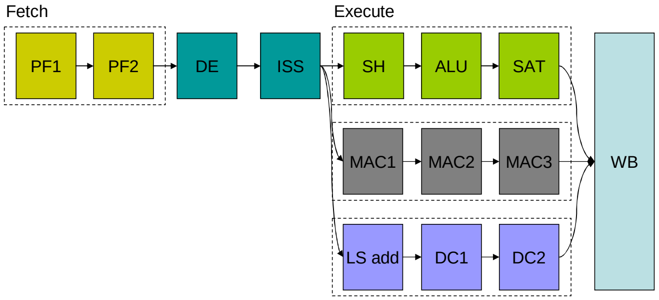
ARM Version 4



ARM Version 5



ARM Version 6



Data Hazards

add r1 ,r2,r3 **RAW**
sub r4, r1 ,r3

or r8, r1 ,r9 **WAR**
and r1, r6 ,r7

load r1, [r10] **WAW**
xor r1, r10 ,r11

Data Hazards (cont.)

ANDS R0,R2,R3 **RAW**
MOVCC R0,R4

ADD R2,R1,R4,LSL #8 **WAR**
STR R1,[R9],#4

LDR R7,[R9],#-4 **WAW**
SMULL R7,R9,R4,R4

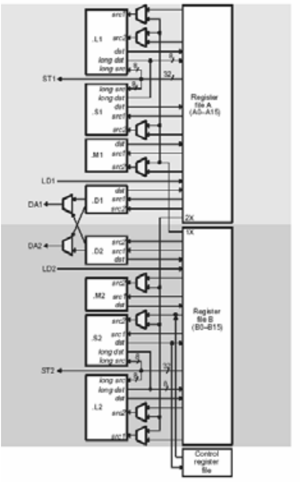
Data Plane Processors

- History
- Register file feeding multiply accumulate unit(s) – MACs
- MAC is the “basic” unit of an inner product
- inner (dot) product = $\sum$ a[i] × b[i]
- sum = sum + a[i] × b[i]
- move to VLIW from less high level language friendly architectures

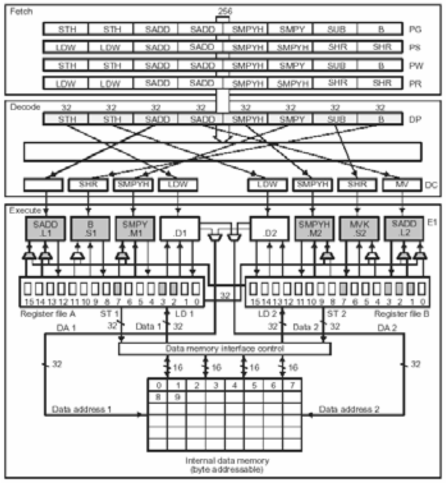
Texas Instruments TMS320C6200 Main Architectural Features

- VLIW
  - Up to 8, 32-bit instructions per cycle
  - RISC-like ISA
- 2 - Cluster Architecture
- Per cluster:
  - 16 General Purpose Registers
  - 4 Fully-Pipelined Functional Units
  - One crosspath to other cluster
- Predicated execution
- Multi-cycle latency instructions

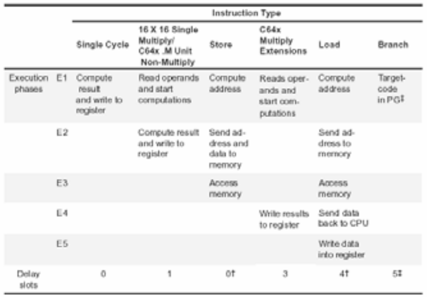
Execution Core



The Pipeline



Pipeline Execution of Instruction Types



		Instruction Type					
		Single Cycle	16 X 16 Single Multiply/ C64x .M Unit Non-Multiply	Store	C64x Multiply Extensions	Load	Branch
Execution phases	E1	Compute result and write to register	Read operands and start computations	Compute address	Reads operands and start computations	Compute address	Target-code in PG‡
	E2		Compute result and write to register	Send address and data to memory		Send address to memory	
	E3			Access memory		Access memory	
	E4				Write results to register	Send data back to CPU	
	E5					Write data into register	
Delay slots		0	1	0†	3	4†	5‡

Functional Units

- L-Unit
  - 32/40-bit Arithmetic
  - 32-bit Logical Operations
  - 32/40-bit Compare Operations
  - Leftmost 1 or 0 counting for 32-bit
  - Normalization count for 32 and 40-bit
- D-Unit
  - 32-bit Add and Subtract (linear and circular addressing)
  - Loads and Stores with 5-bit constant offset
  - Loads and Stores with 15-bit constant offset (.D2 unit only)

Functional Units

- S-Unit
  - 32-bit Arithmetic
  - 32-bit Logical Operations
  - 32/40-bit Shifts
  - 32-bit Bit-field Operations
  - Branches
  - Constant Generation
  - Control Register Access (.S2 unit only)
- M-Unit
  - 16x16 Multiply

Non Software Pipelined Loop

c code:

```
for (i = 0; i < L_WINDOW; i++) {
    y[i] = mult_r (x[i], wind[i]);
    move16 ();
}
```

- Cannot software pipeline loop
- Very little parallelism in assembly
- Does make use of auto increment load instructions
- MVK instructions setup return, no branch and link, plenty of delay slots to do this manually
- Notice the NOP 4 at the end of the loop, common for non software pipelined
- No overlap of caller and callee functions

```
assembly:
---------
L1:
           LDH     .D2T2   *B10++,B4
||         LDH     .D1T1   *A13++,A0

           MVKL    .S2     RL0,B3
           MVKH    .S2     RL0,B3
           NOP             1
           B       .S1     _move16
           SMPY    .M1X    B4,A0,A0
           NOP             1
           SADD    .L1     A0,A15,A0
           SHRU    .S1     A0,16,A0
           STH     .D1T1   A0,*A14++
RL0:       ; CALL OCCURS
           SUB     .D1     A11,1,A1
   [ A1]   B       .S1     L1
           SUB     .D1     A11,1,A11
           NOP             4
           ; BRANCH OCCURS
```

Function Unit Usage (non software pipelined loop)

D1	L1	M1	S1	D2	L2	M2	S2
■				■			
							■
							■
			■				
		■					
	■						
			■				
■							
■							
			■				
■							

Software Pipelined Loop

c code:

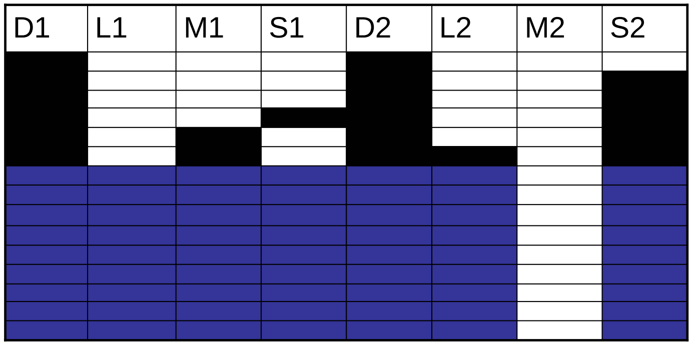
```
for (j = 0; j < L_WINDOW - i; j++)
{
   // L_mac is an intrinsic for the saturated multiply and accumulate
   sum = L_mac (sum, y[j], y[j + i]);
}
```

- Iteration interval is 1
- 8 iterations in ||
- Needs a large prologue because iteration interval is less than the number of branch delay slots (notice there are 5 branches before the kernel to setup one branch resolving each cycle)
- Able to use A4 and B5 for each iteration because of load delay slots
- Out of order processor achieves pipelining by renaming and branch prediction
- Able to get lots of ||
- Uses predicates to stop loop and squash epilogue

```
Assembly :
----------

L11:    ; PIPED LOOP PROLOG

           LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5

           LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5
||         B       .S2     L12

           LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5
||         B       .S2     L12

           SUB     .S1X    B0,7,A1
||         LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5
||         B       .S2     L12

           B       .S2     L12
||         LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5
||         SMPY    .M1X    B5,A4,A5

           SUB     .L2     B0,6,B0
||         SMPY    .M1X    B5,A4,A5
||         LDH     .D1T1   *A0++,A4
||         LDH     .D2T2   *B4++,B5
||         B       .S2     L12
```

```
L12:    ; PIPED LOOP KERNEL

   [ A1]   SUB     .S1     A1,1,A1
||         SADD    .L1     A3,A5,A3
||         SMPY    .M1X    B5,A4,A5
|| [ B0]   B       .S2     L12
|| [ B0]   SUB     .L2     B0,1,B0
|| [ A1]   LDH     .D1T1   *A0++,A4
|| [ A1]   LDH     .D2T2   *B4++,B5

;** --------------------------------*
L13:    ; PIPED LOOP EPILOG
;** --------------------------------*
           NOP             3
```

Function Unit Usage (software pipelined loop)



Compiler Issues 1

- Compiler doesn't generate || code for function epilogue
- Doesn't overlap code completely with branch delay slots

```
           LDW     .D2T2   *+SP(508),B3
           LDW     .D2T1   *+SP(528),A15
           LDW     .D2T2   *+SP(524),B13
           LDW     .D2T2   *+SP(520),B12
           LDW     .D2T2   *+SP(516),B11
           LDW     .D2T2   *+SP(512),B10
           LDW     .D2T1   *+SP(496),A12
           LDW     .D2T1   *+SP(492),A11
           LDW     .D2T1   *+SP(488),A10
           B       .S2     B3
||         LDW     .D2T1   *+SP(500),A13
           LDW     .D2T1   *+SP(504),A14
           ADDK    .S2     528,SP
           NOP             3
           ; BRANCH OCCURS
.endfunc
```

Compiler Issues 2

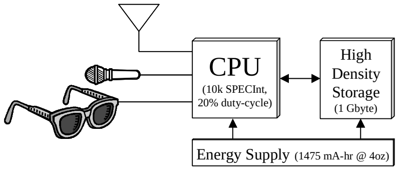
- Compiler doesn't overlap the load delay slots and the branch delay slots
- VLIW much more difficult for a compiler
- Compilers are already very complex and hard to create/debug entities
- Very difficult to fill 5 branch delay slots unless software pipelining a loop

```
  LDW     .D2T1   *+SP(180),A0
  NOP             4
  STW     .D2T1   A0,*+SP(220)
  B       .S1     _move16
  MVKL    .S2     RL312,B3
  MVKH    .S2     RL312,B3
  NOP             3

------------------------
; Change it to this

   LDW     .D2T1   *+SP(180),A0
   B       .S1     _move16
   MVKL    .S2     RL312,B3
   MVKH    .S2     RL312,B3
   NOP             1
   STW     .D2T1   A0,*+SP(220)
   NOP             1
```

Control vs. Data Plane

- Merge?
  - lower cost systems?
  - lower power systems?
- Complicates real-time deadlines
- Add a MAC unit to a general purpose processor – ARM's Piccolo
- Low end solution

A Challenge for the Near Future: Wireless Supercomputing



CPU
(10k SPECInt, 20% duty-cycle)
High Density Storage
(1 Gbyte)
Energy Supply (1475 mA-hr @ 4oz)

Workload	Performance Req'ed (relative to fastest current design)
Soft-radio	4x
Crypto-processing	4x
Augmented reality	4x
Speech recognition	2x
Mobile Applications	2x

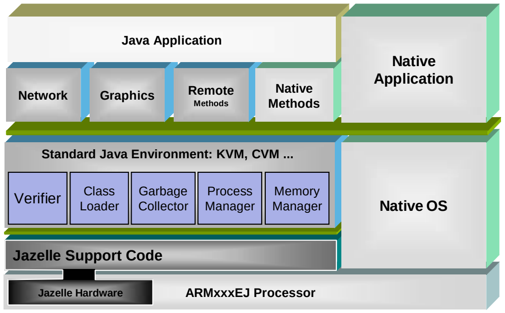
- All with v tiny batteries
- Ambient power

Advanced Topics

- JAVA accelerators
- Secure Cores

JAVA Accelerators

- Jazelle Hardware and Software



Java Application

Network

Graphics

Remote Methods

Native Methods

Native Application

Standard Java Environment: KVM, CVM ...

Verifier

Class Loader

Garbage Collector

Process Manager

Memory Manager

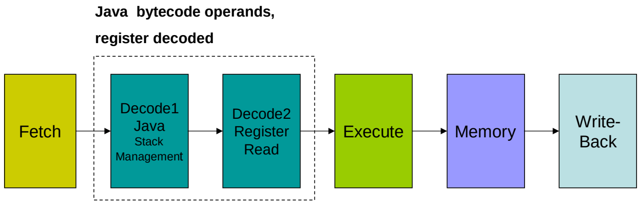
Native OS

Jazelle Support Code

Jazelle Hardware

ARMxxxEJ Processor

ARM v5 Jazelle Mode Pipeline



Java bytecode operands, register decoded

Fetch

Decode1 Java Stack Management

Decode2 Register Read

Execute

Memory

Write-Back

ARMv5 Jazelle

- Fetches Bytecode from I-Cache
  - D-Cache fetch for JVM execution
- Variable length Instruction Fetch
  - Length for each Bytecode variable
- Internal Translation Buffer store translated native code
  - Bytecodes tend to expand in translation
- Branch back to Normal Mode for VM execution

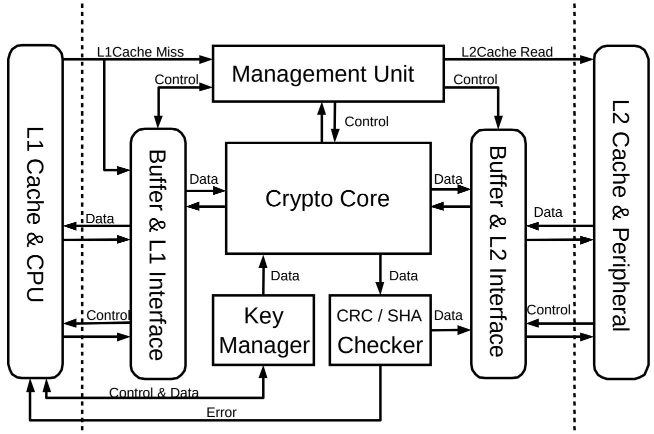
Jazelle Translation Example

dup	⇨	LDR t0, [SP,#4]
		STR [SP], t0
		SUB SP,SP,#4
iload_1	⇨	MOV t0,#1
		LDR t1, [LP,t0,LSL #2]
		STR [SP], t1
		SUB SP,SP,#4

Secure Cores

- Off-chip information un-trusted
  - OS, External I/O also un-trusted
  - On chip components only trusted
- Security must be application or thread based
  - Security should be managed per application
  - Inter-application communication should also be secure

System Architecture



Ideal Goal for Hardware Security

- Detection
  - Detect tampered applications
  - Applications found to be tampered not executed
  - SHA, CRC components for detection
- Prevention
  - Use proven Encryption / Decryption methods
  - AES, RSA
- Low overhead
  - Minimal Increase of Latency

Trade-Off for Security

- Detection
  - SHA Block expensive to implement
  - No Detection results in system crash
  - Detect partial parts of Application
- Prevention
  - RSA Block expensive to implement
  - Simple crypto cores unreliable
  - AES Reasonable
    - Reused for network transmissions
  - Partial encryption / decryption may also be deployed

Trade-Off for Security (cont.)

- Overhead
  - – Crypto Cores add large overhead
    - Ex) Typical AES units take 10 cycles to complete
  - – Prefetch / Speculation should be explored
  - – Private / Public Keys are added for speculation parameters

Other Issues

- Key management
  - – Key revocation
  - – Acquiring a Key, Currently assuming TCPA key obtaining method
- Memory Management Unit
  - – Sticky business
    - DLLs, malloc issues
    - Adding and deleting secure and unsecure pages