

# SYSTEMS ON CHIP (SOC) FOR EMBEDDED APPLICATIONS

---

Victor P. Nelson

“Leap Day”, 2012

This is not a “defense”  
So – please enjoy these photos of food.



# Outline

- What is an embedded SoC?
- SoC Intellectual Property (IP)
  - ARM processors
  - ARM support modules
- SoC Design Flow
  - Modeling and simulation
  - Physical design

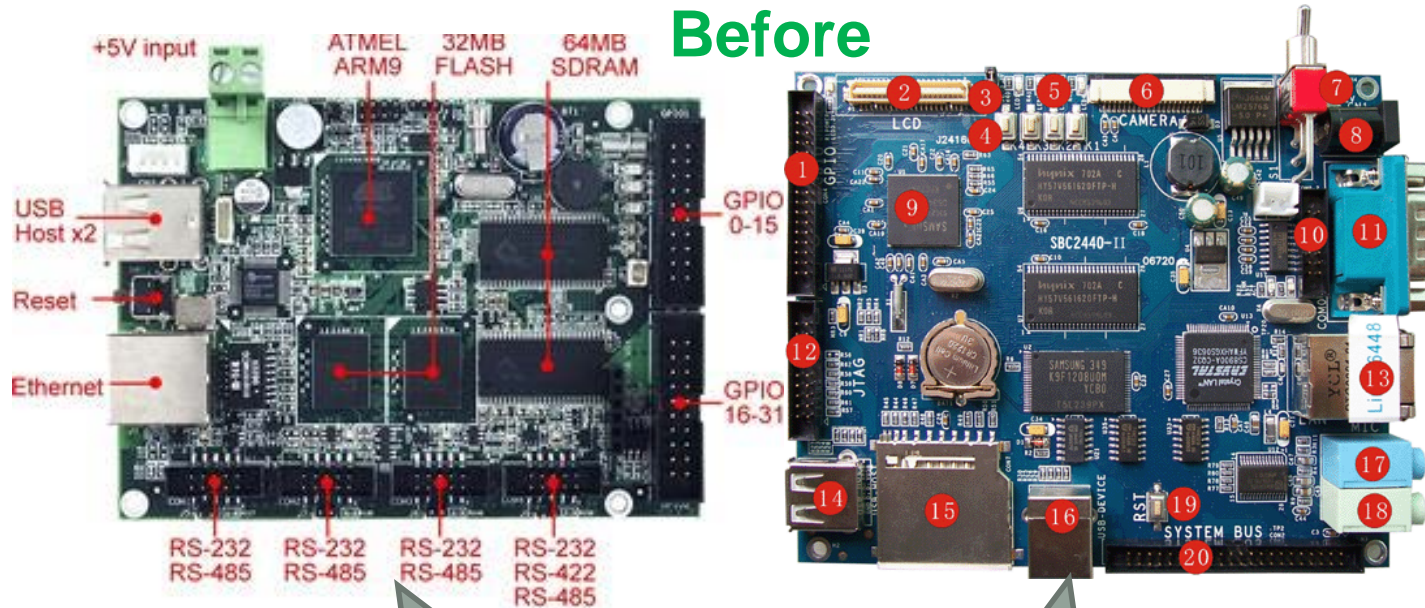
# Embedded System

- “System”
  - Set of components needed to perform a function
  - Hardware + software + ....
- “Embedded”
  - Main function not computing
  - Usually not autonomous
  - Usually a computer inside a system
  - Application specific
  - Subject to constraints

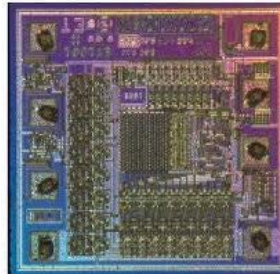
# System on chip

- Definition
  - (nearly) complete embedded system on a single chip
- Usually includes
  - Programmable processor(s)
  - Memory
  - Accelerating function units
  - Input/output interfaces
  - Software
  - Re-usable intellectual property blocks (HW + SW)

# SoC Design Goal

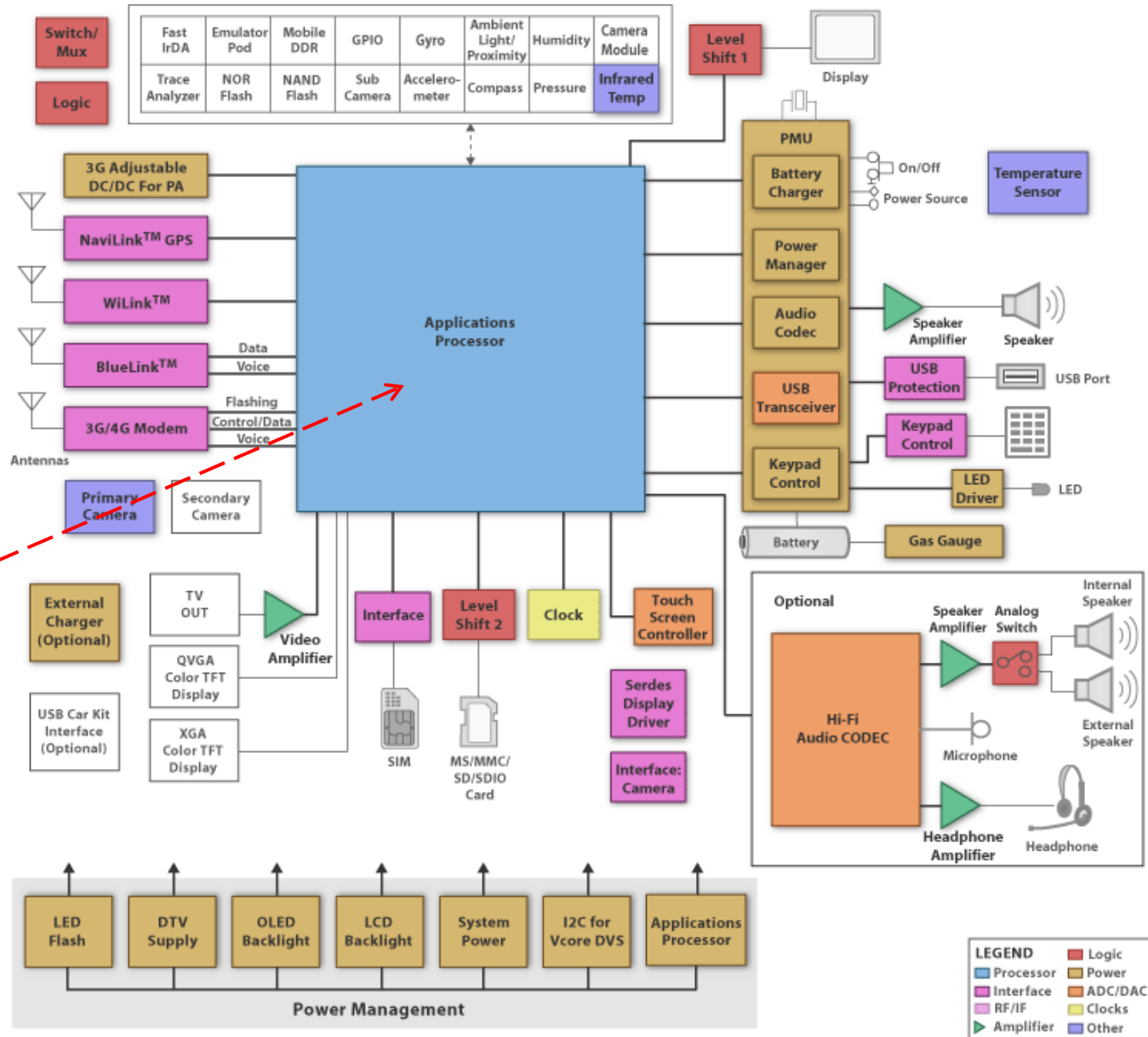


**After**

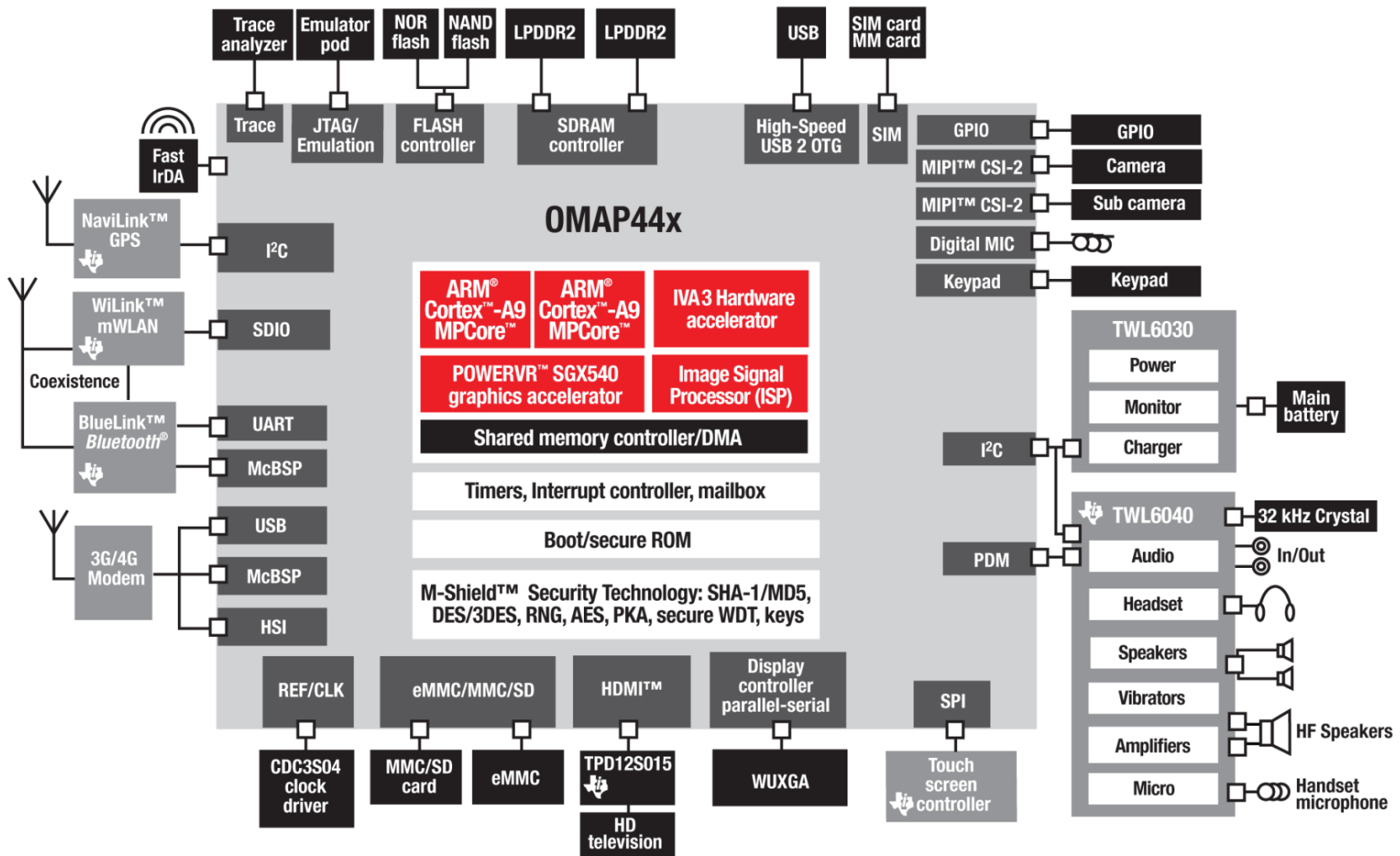


# T.I. smartphone reference design

Main SoC



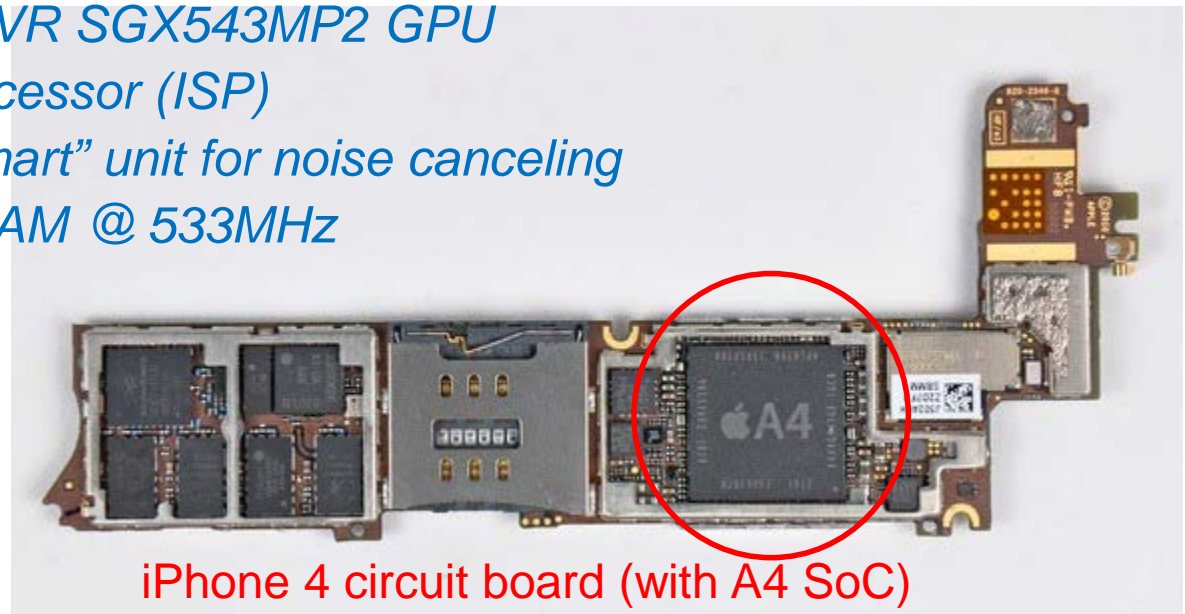
# Texas Instruments OMAP44x





# Apple "A5" SoC

- Used in *iPad 2* and *iPhone 4S*
- Manufactured by Samsung
  - 45nm, 12.1 x 10.1 mm
- *Elements (unofficial):*
  - ARM Corex-A9 MPCore CPU - 1GHz
    - NEON SIMD accelerator
  - Dual core PowerVR SGX543MP2 GPU
  - Image signal processor (ISP)
  - Audience "EarSmart" unit for noise canceling
  - 512 MB DDR2 RAM @ 533MHz

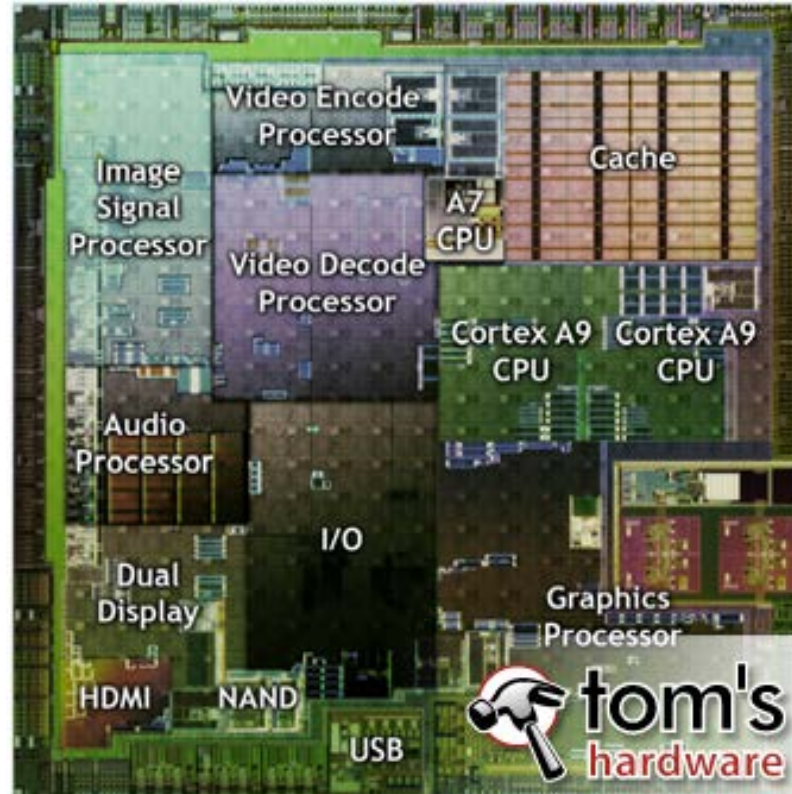


iPhone 4 circuit board (with A4 SoC)

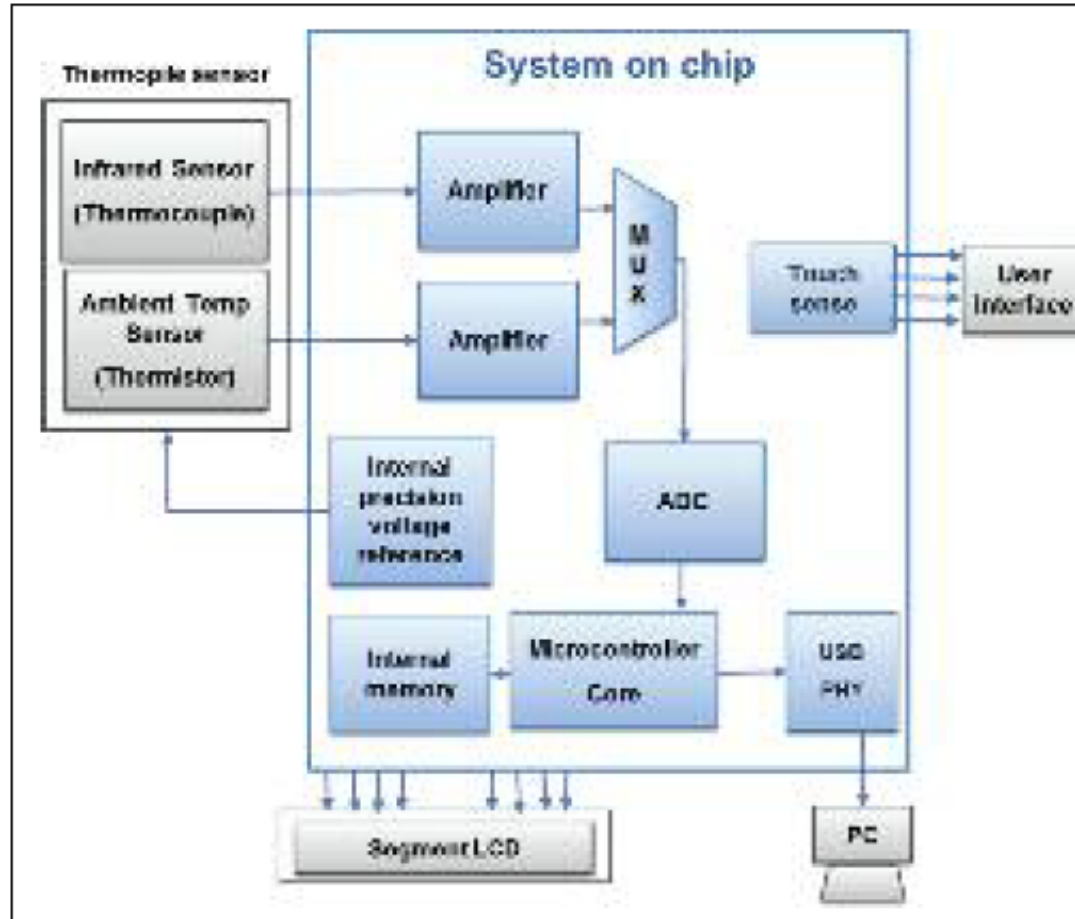
# Nvidia Tegra 2 SoC

## Tablet Applications:

- Asus Eee Pad
- Motorola Xoom
- Samsung Galaxy
- Acer Iconia Tab



# Example: blood pressure monitor SoC



# SoC Challenges

- SoC Designs
  - More complex, more functions, higher gate counts
  - Faster, cheaper, smaller
  - More reliable
- How to handle complexity?
  - System design at multiple abstraction levels
  - Integration of heterogeneous technologies & tools
  - Signal integrity & timing
  - Power management
  - SoC test methodology

# SoC design with re-usable IP modules

- IP = intellectual property
  - HW or SW block
  - Designed for reuse
  - Need for standards (VSIA)
- Platform-based SoC design
  - Organized method
  - Reduce cost and risk
  - Heavy re-use of HW and SW IP
- Steps in re-use
  - Block -> IP -> integration architecture

# ARM IP (ARM makes no hardware)

- Processors
  - Cortex A, Cortex R, Cortex M, ARM11, ARM9, ARM7, SecurCore
- Multimedia IP - graphics, video, audio
  - Mali-T604 GPU graphics processor
  - Mali-VE6 video engine
- System IP
  - CoreLink – interconnect & memory controllers
    - Supports Cortex and Mali processors
    - AMBA – Advanced Memory Bus Architecture
  - CoreSight – debug and trace IP (build into SoC)
- ARM “Artisan” Physical IP
  - Logic IP, Standard Cells, Memory Compilers, Interface IP
  - *Technology-specific*

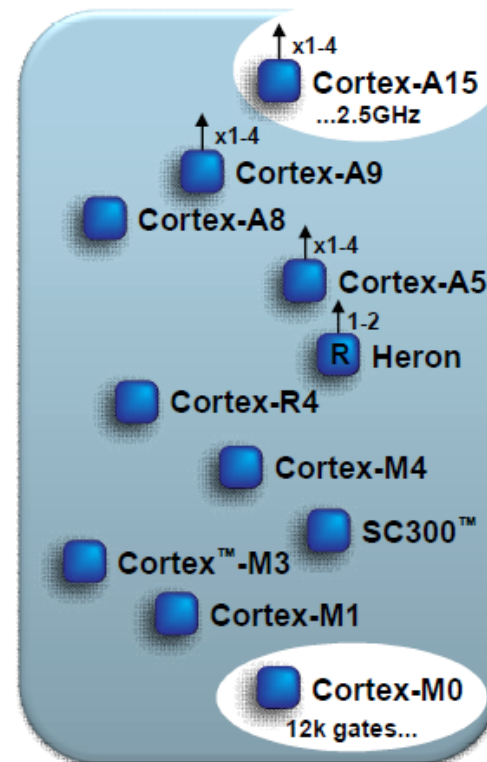
# ARM SoC-based products





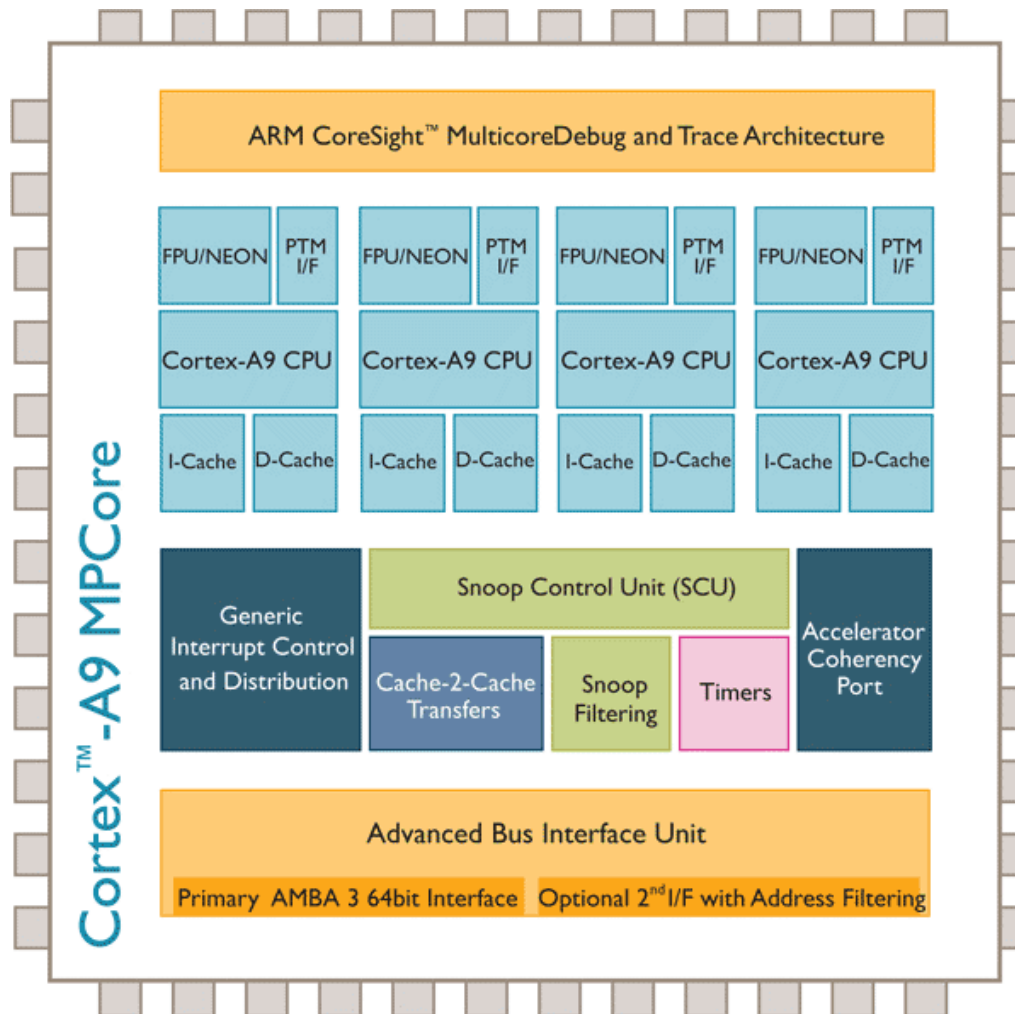
# ARM Cortex Processors

- ARM Cortex-**A** family (v7-A):
  - Applications processors for full OS and 3<sup>rd</sup> party applications
- ARM Cortex-**R** family (v7-R):
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-**M** family (v7-M):
  - Microcontroller-oriented processors for MCU and SoC applications

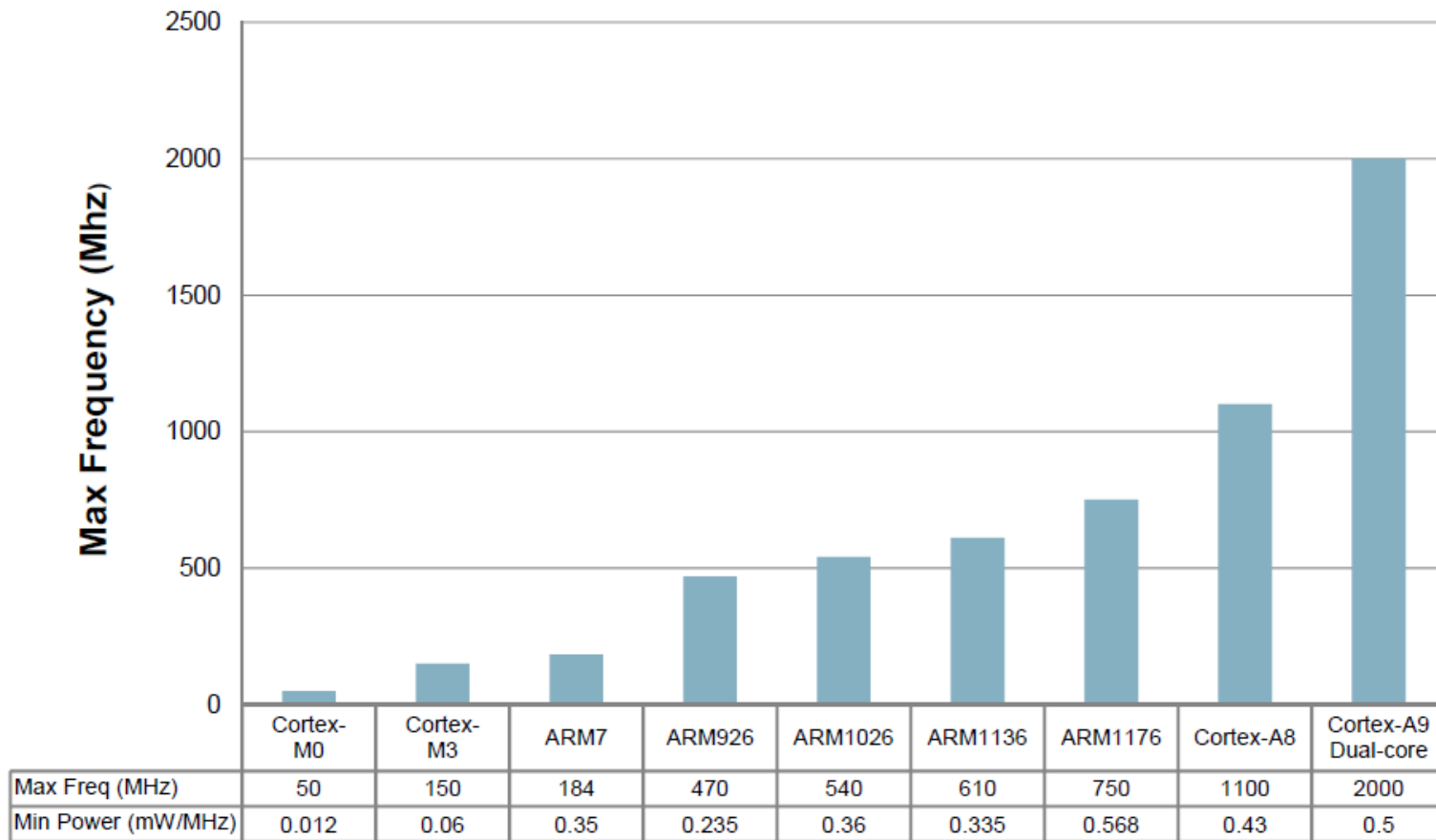




# ARM Cortex-A9 MPCore

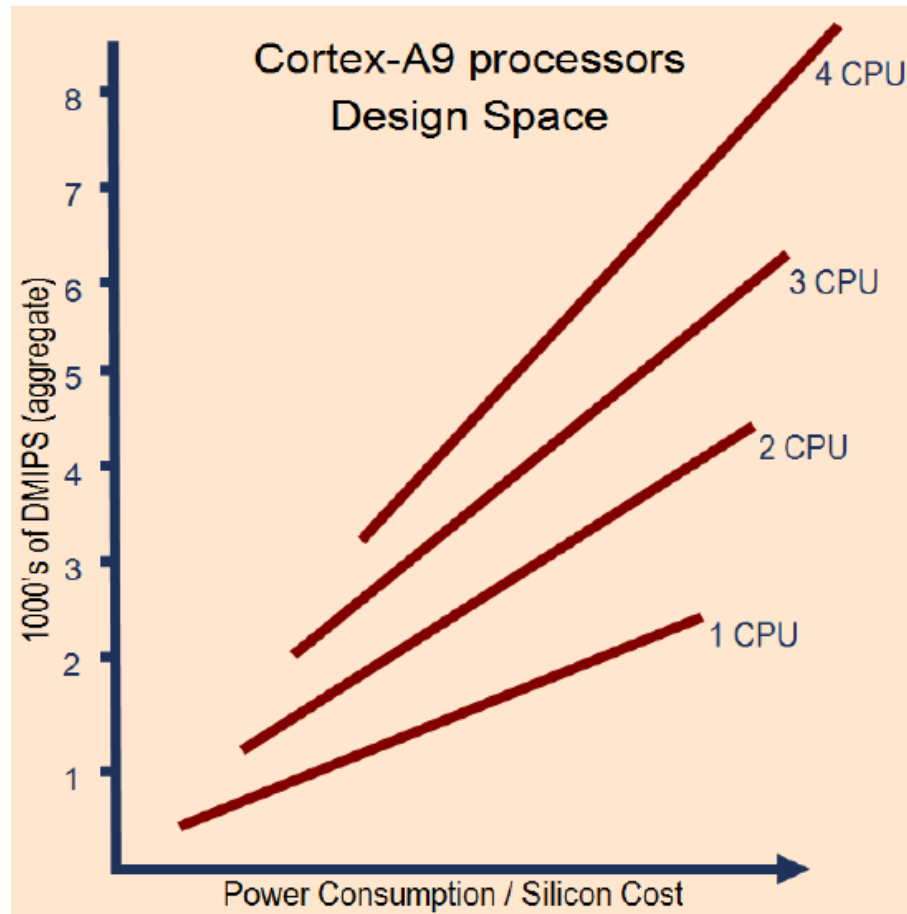


# Relative Performance



\*Represents attainable speeds in 130, 90, 65, or 45nm processes

# ARM Cortex-A9 MPCore performance



# Licensing ARM IP

- **Perpetual** (implementation) license
  - ARM partner may perpetually design and manufacture ARM-based products
- **Term** license
  - Design a limited number of ARM-based products within a specified time period (usually 3 years)
  - Perpetual manufacturing rights
- **Per use** license
  - Selected ARM IP, right to design a single ARM-technology product within a specified time frame (3 years)
  - Manufacturing rights perpetual
- **University “DesignStart” Program**
  - Some physical and selected processor IP downloadable for academic study

# ARM Processor Foundry Program

- Fabless semiconductor companies design ARM-based SoCs with approved ARM semiconductor foundry (90nm to 180nm)

- **Design Kit**

- Design Flow Guide
- ARM processor deliverables
- Design Sign-off (Simulation) Models (DSMs) and Test Vectors
- AMBA Design Kit
- RealView Development Suite
- Powerful JTAG-based run control device for non-intrusive debug of embedded processors
- RealView Integrator<sup>®</sup> Development Platform Training, support and maintenance.



- **Available processors:**

- ARM922, ARM946, ARM926EJ
- Provided as hard macrocells

# Cortex A9 System IP

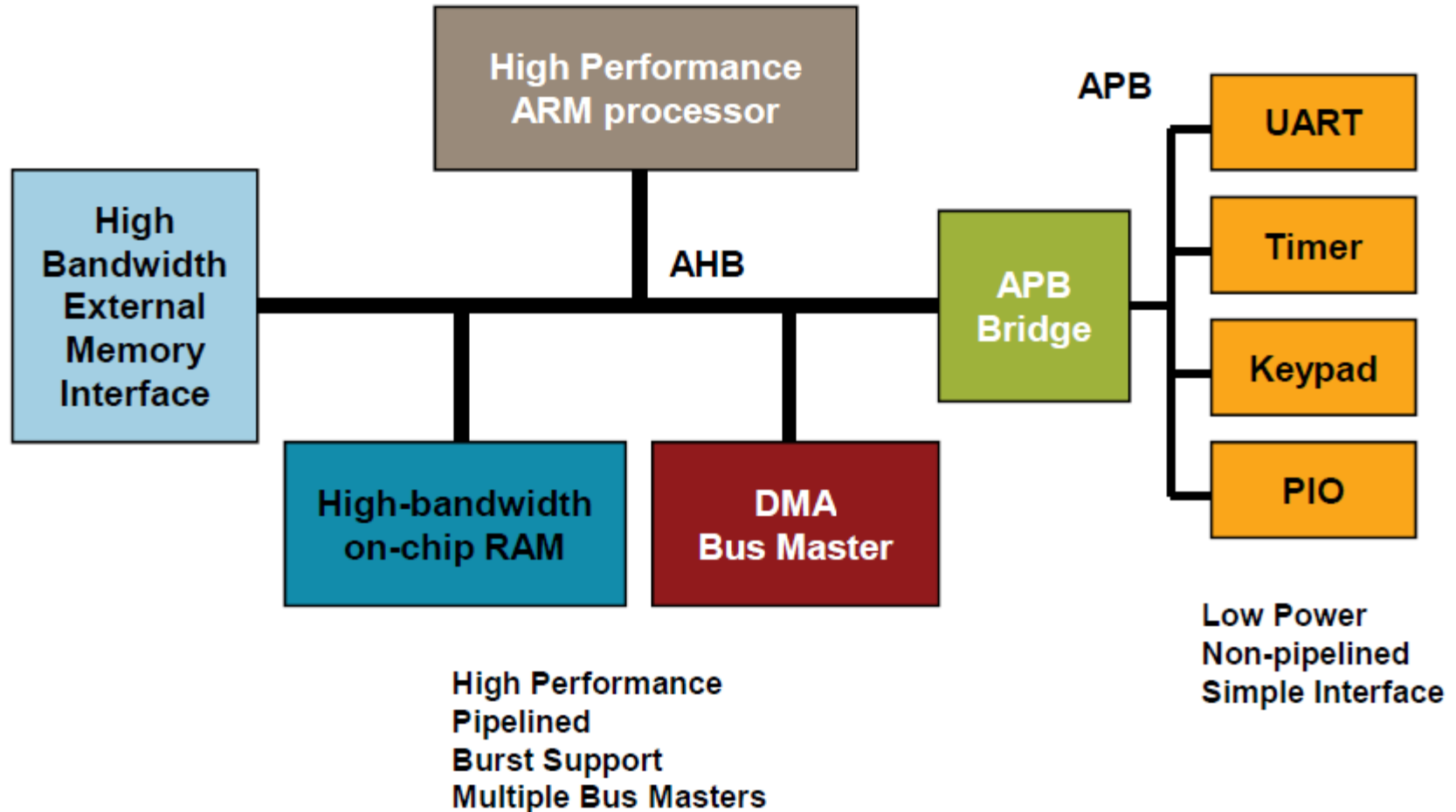
## Interconnect SoC components

Description	AMBA Bus	System IP Components
Advanced AMBA 3 Interconnect IP	AXI	<a href="#"><u>NIC-301, PL301</u></a>
DMA Controller	AXI	<a href="#"><u>DMA-330, PL330</u></a>
Level 2 Cache Controller	AXI	<a href="#"><u>L2C-310, PL310</u></a>
Dynamic Memory Controller	AXI	<a href="#"><u>DMC-340, PL340</u></a>
DDR2 Dynamic Memory Controller	AXI	<a href="#"><u>DMC-342</u></a>
Static Memory Controller	AXI	<a href="#"><u>SMC-35x, PL35x</u></a>
<a href="#"><u>TrustZone</u></a> Address Space Controller	AXI	PL380
CoreSight™ Design Kit	ATB	<a href="#"><u>CDK-11</u></a>

# ARM Advanced Microcontroller Bus Architecture (AMBA)

- On-chip interconnect specification for SoC
- Promotes re-use by defining a common backbone for SoC modules using standard bus architectures
  - AHB – Advanced High-performance Bus (system backbone)
    - High-performance, high clock freq. modules
    - Processors to on-chip memory, off-chip memory interfaces
  - APB – Advanced Peripheral Bus
    - Low-power peripherals
    - Reduced interface complexity
  - ASB – Advanced System Bus
    - High performance alternate to AHB
  - AXI – Advanced eXtensible Interface
  - ACE – AXI Coherency Extension
  - ATB – Advanced Trace Bus

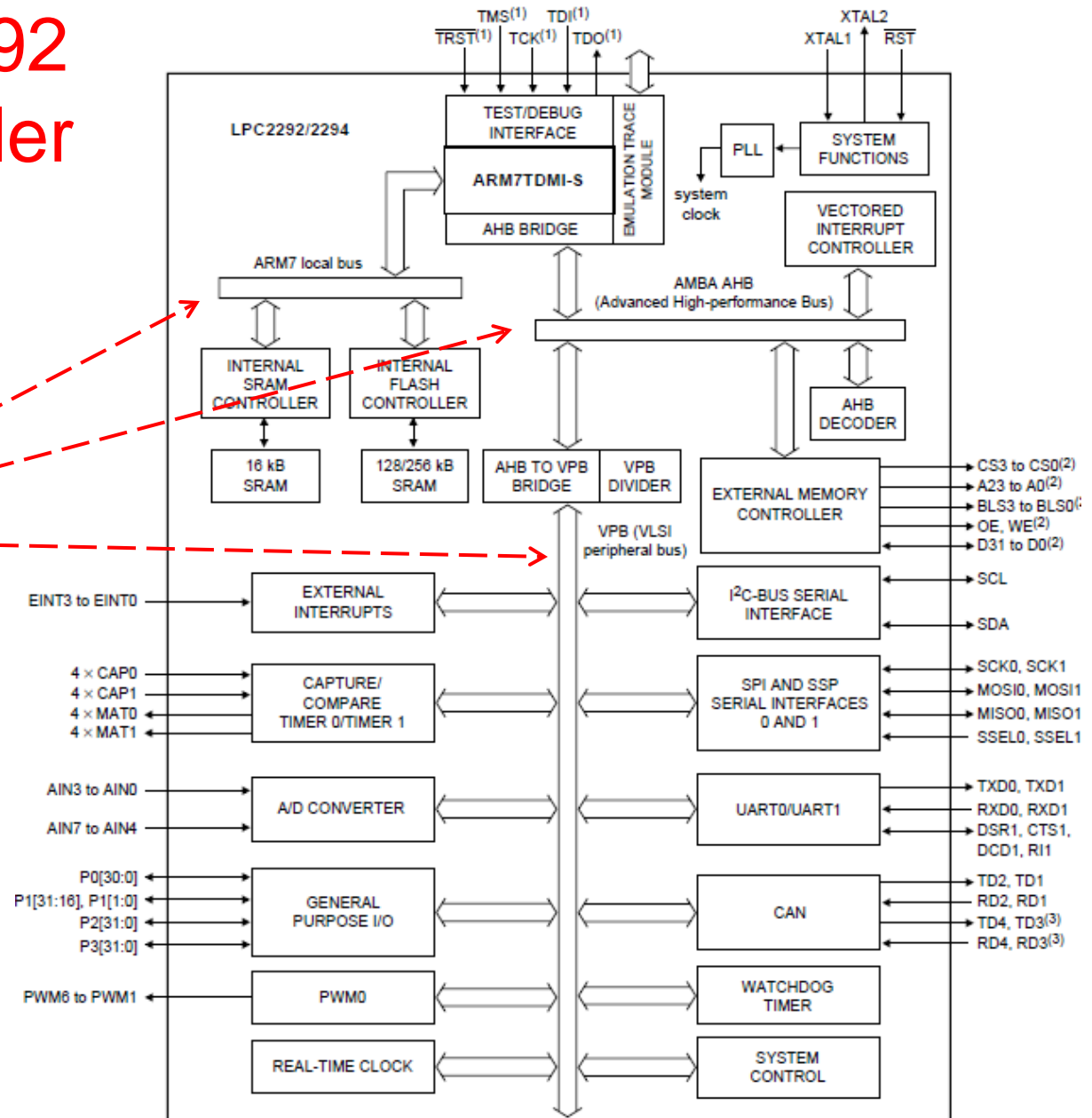
# Example AMBA System





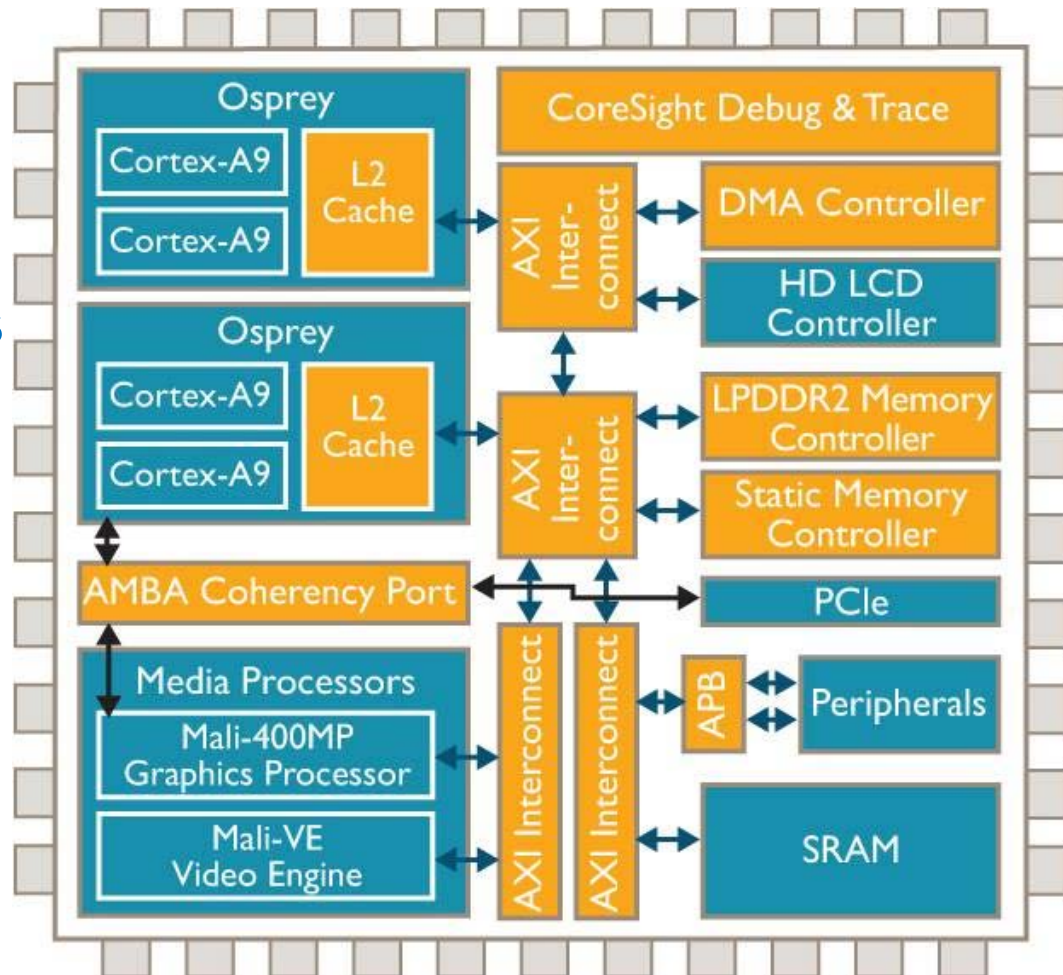
# NXP LPC2292 Microcontroller

ARM buses



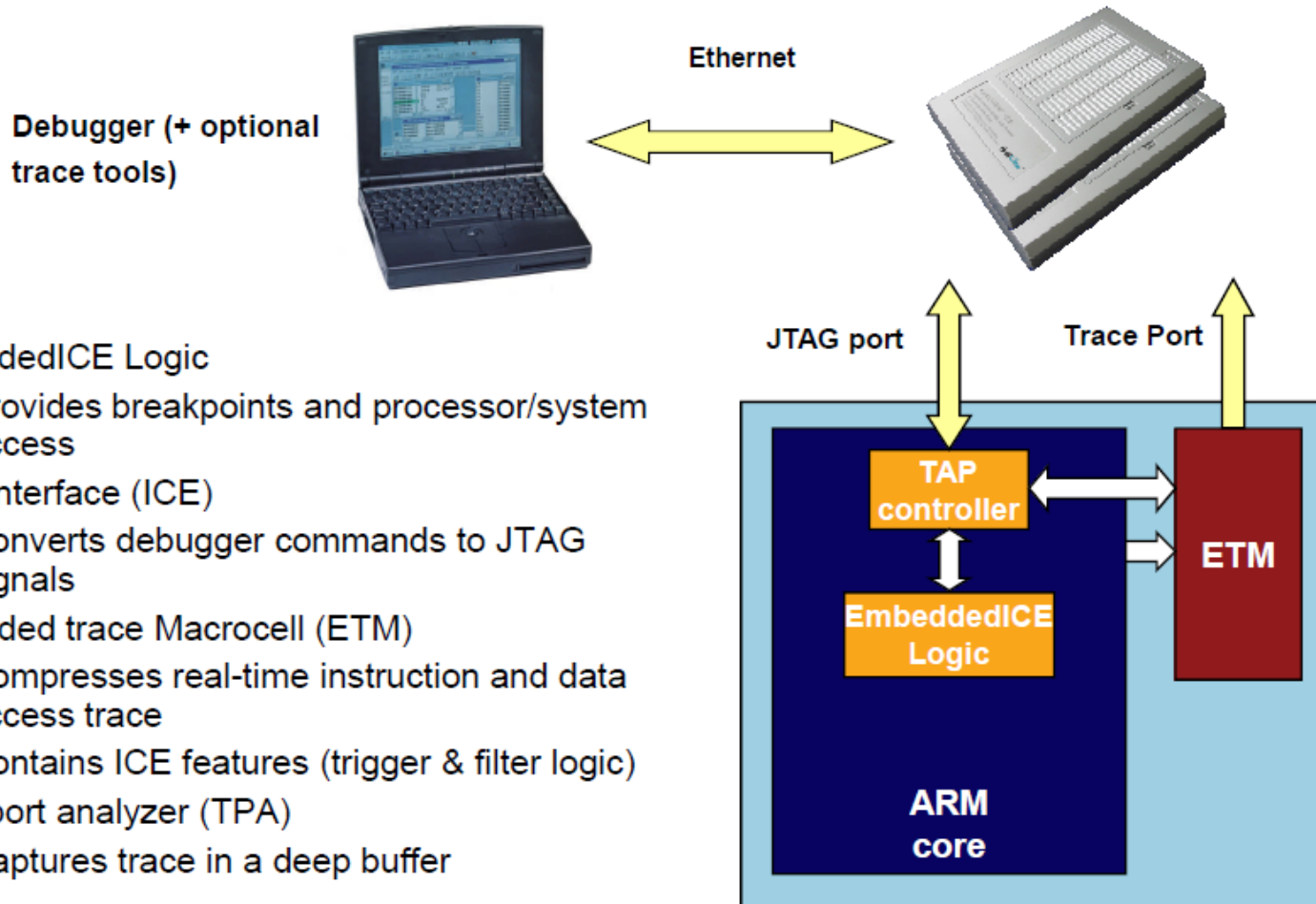
# CoreLink peripherals for AMBA

“CoreLink” =  
interconnect +  
memory controllers  
for Cortex/Mali



# ARM Debug Architecture

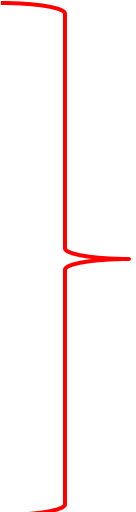
(For Cortex: “CoreSight” debug & trace IP)



- EmbeddedICE Logic
  - Provides breakpoints and processor/system access
- JTAG interface (ICE)
  - Converts debugger commands to JTAG signals
- Embedded trace Macrocell (ETM)
  - Compresses real-time instruction and data access trace
  - Contains ICE features (trigger & filter logic)
- Trace port analyzer (TPA)
  - Captures trace in a deep buffer

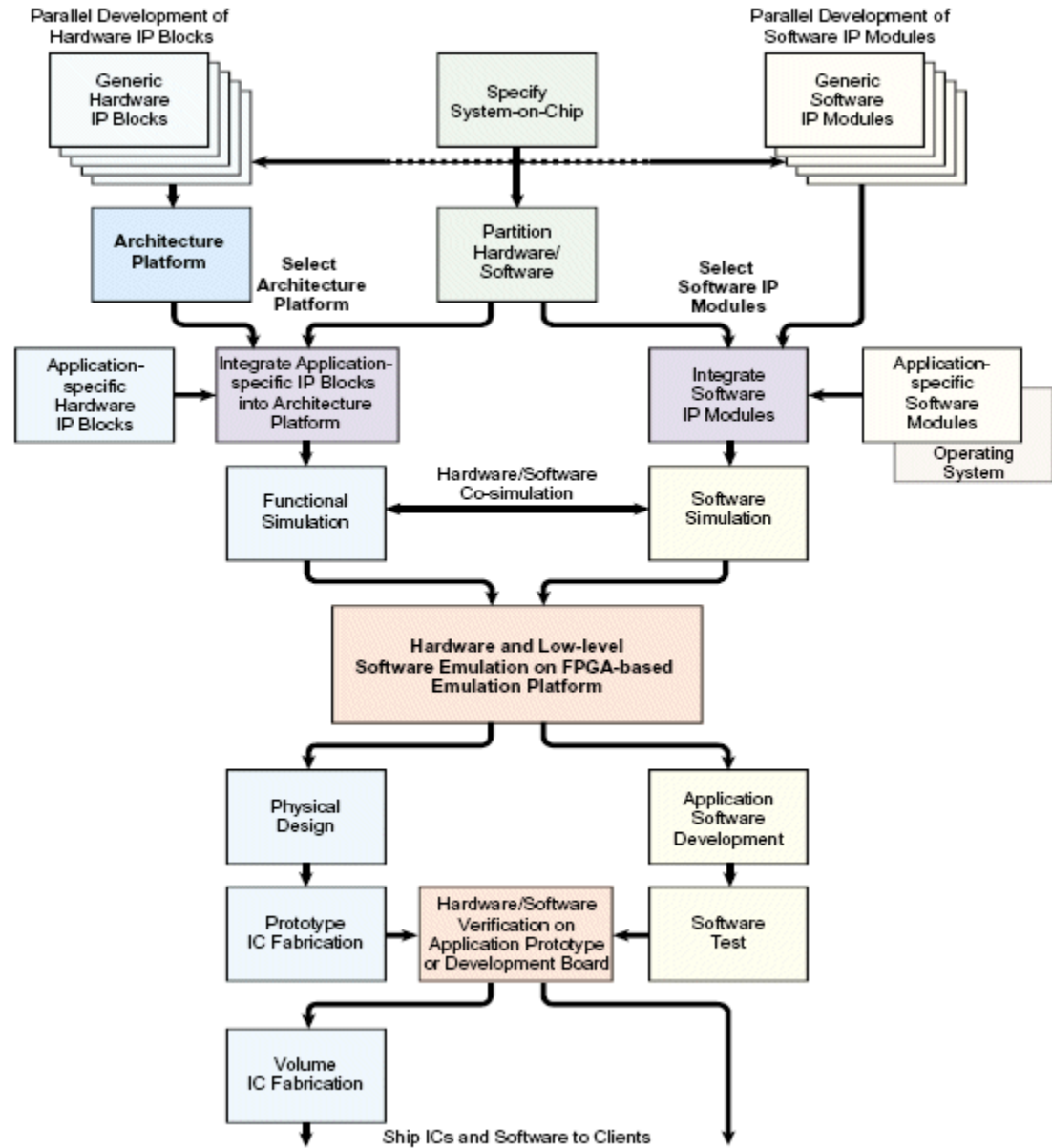
# SoC Design Process

- Customer requirements
- System specification
- Architecture design
  - Hardware vs. software
- Component design
- Integration
- Verification
- Manufacture
- Test



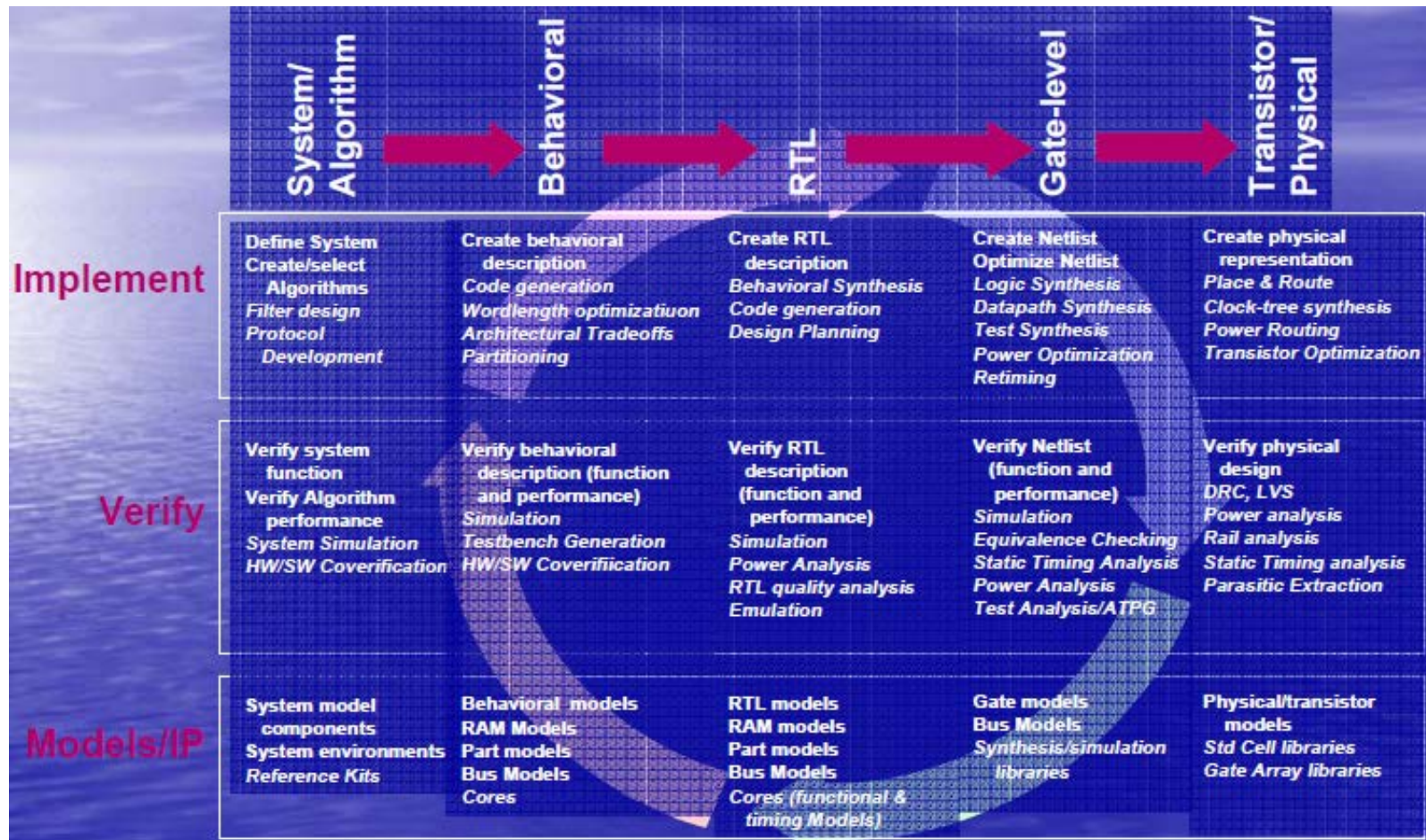
Model, simulate,  
and evaluate  
at each stage

# SoC Design Flow





# Typical design flow tasks



# High-Level Performance Modeling

- **Identify Workloads**

- Based on target market
- Standard benchmarks (spec, EEMBC)
- O/S based “real” benchmarks – browser, real apps

- **Performance Models**

- C-based, highly configurable
- Internally developed (no EDA vendor)
- Fast Instruction-Set Based Model
  - No timing information, but very fast
  - Used for statistics collection and coarse algorithm development (i.e. branch prediction scheme, load/store address patterns)
- Abstracted Pipeline
  - Reasonably accurate, longer development time
  - More specific to microarchitecture

# Higher levels of abstraction for SoC

- ESL (Electronic system level)
  - RTL (register transfer level) to TLM (transaction-level modeling)
  - VHDL to SystemC to UML
- HW/SW co-design
  - Simulation models/emulators of hardware to develop software while hardware is being developed
  - Need new tools
  - Consider the whole system
  - Large optimization potential
  - Combination of formal, semi-formal and non formal techniques



# Unit RTL

- Synthesizable HDL models
- Split work into units based on functionality
  - Verilog language of choice
  - Write low-level constructs only (assign, case)
  - Why?
    - Portability; we target multiple partners and have to target
    - ‘lowest-common denominator’ design tools
  - Know your RTL! Easier to count gates “on-the-fly”
- Orderly bring-up; integration as soon as possible

# Unit Validation vs. Top Level Validation

- Validation is done at both the unit- and top-level. Both are important and doing one isn't a substitute for doing the other!
- **Unit Level Pro'S**
  - Faster runtimes (more cycles -> more throughput)
  - Easier to generate "strange" corner case stimulus
- **Top Level Pro'S**
  - Interfaces between units are tested
  - Stimulus re-use; often top level tests are assembly and can be ported from other projects
  - This is what is being delivered!
  - Performance validation

# Validation

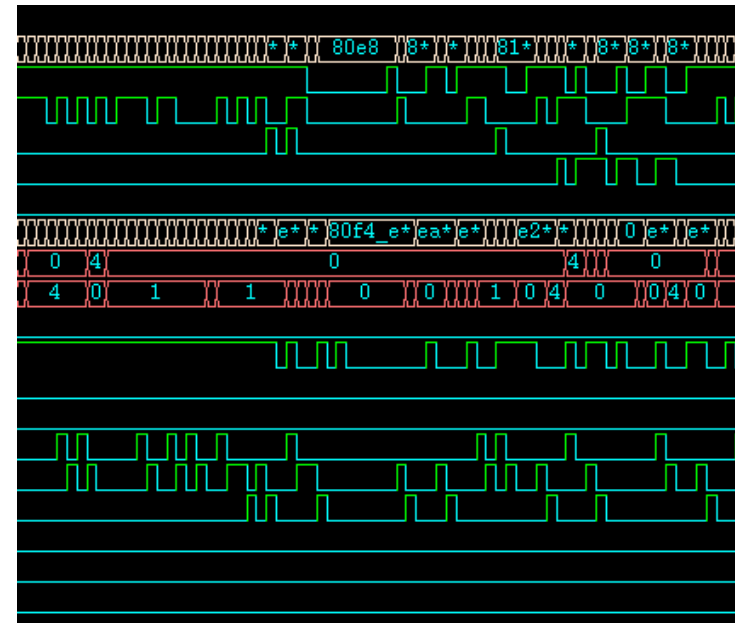
- Stimulus
  - random – RIS, traffic generators for system
  - directed – assembly language tests to check specific features
- Irritators
  - artificial constraints to stress certain features
  - i.e. 1 entry TLB to test table walk logic, forced stalls to test queues
- Assertions
  - constantly check that logic cannot do things specified as “illegal”
  - i.e. bus protocol checkers
- Formal
  - uses properties to check that system can't get into certain state via sequence (lots of constraints required)
  - lots of research ongoing about formal

# Validation - Coverage

- How do I know that I've tested everything?
- Coverage provides validation metrics
  - Line Coverage
    - hit all RTL lines (easy to run)
  - Condition Coverage
    - hit all RTL terms (easy to run)
  - Functional Coverage
    - Only way to determine sequences hit
    - Require coverage points to be written
    - For example, test pipeline flush under other stall conditions (Cache miss, TLB miss...)
    - RTL unit designer and validation person collaborate to determine "points" or "matrices"

# Validation – Simulation/Emulation

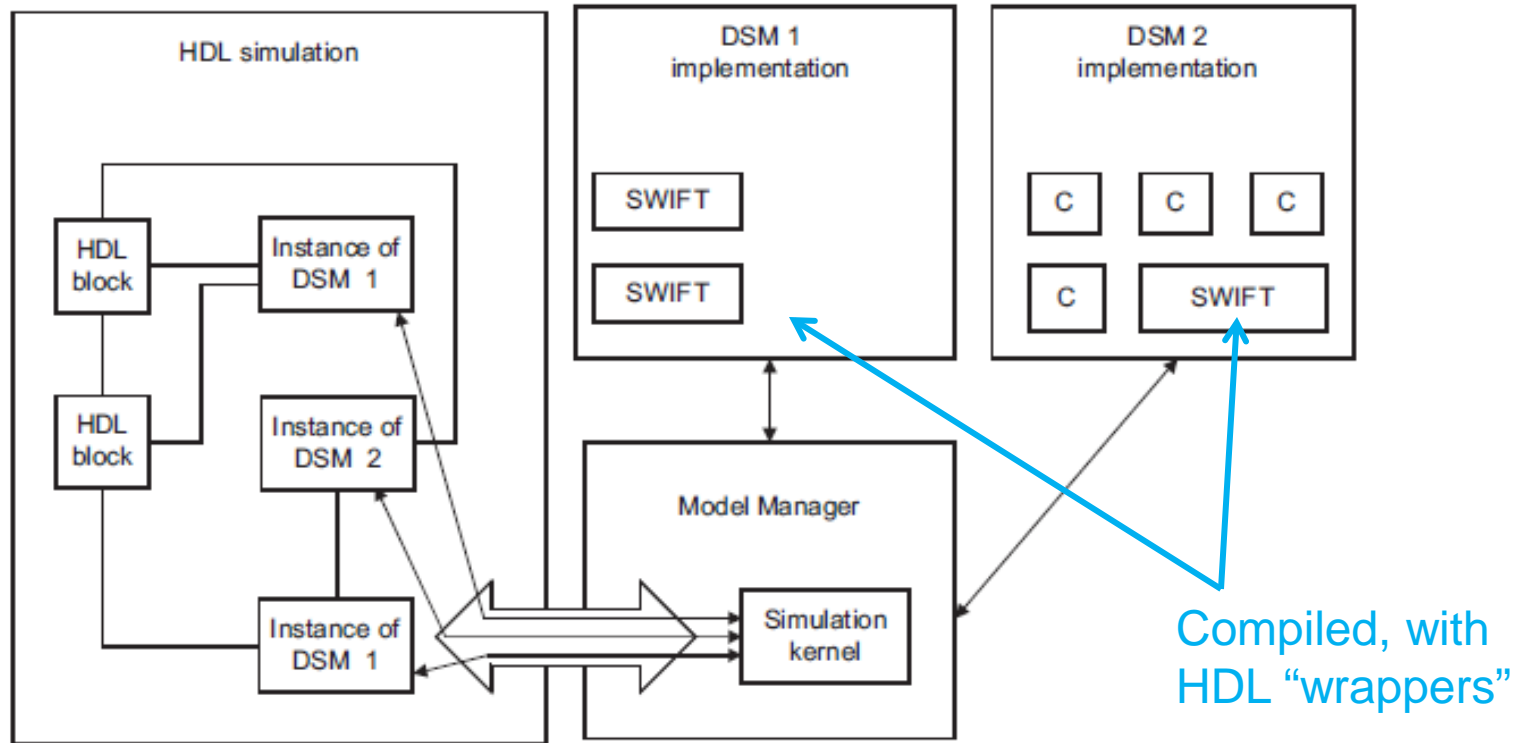
- Software simulation – compile RTL and testbench
  - testbench can be Verilog/VHDL or newer extensions (SystemVerilog)
  - slow – 100's of Hz, slower if simulating multi-core system
  - full visibility (dump waveforms) – but this is even slower
  - best for debugging/rapid turnaround
- Emulation
  - FPGA – moderate \$, large setup time, 1's of MHz, low to no visibility into failures
  - Quickturn/hardware emulator – big \$, large capacity, 1 MHz
    - Can do O/S boots and complex system validation
    - Visibility – can see all RTL signals
    - Can use virtual device drivers for peripherals (LCD)



# ARM Design Simulation Models (DSM)

- HDL behavioral models of ARM cores
  - for functional and “in some cases, timing” simulation
  - derived from ARM core RTL code
  - full device functionality
  - register visibility
  - configure cache and memory sizes
  - compatible with VHDL/Verilog simulators (ex. ModelSim)
- Back-annotation capable, timing accurate
  - accept timing through SDF files
  - min/typ/max pin-to-pin delays
  - setup/hold/pulse checks
- **Also called “Design Sign-Off Models”**
  - generated from technology-specific netlist of a core

# ARM DSM simulation structure



# Implementation Style

- **Custom**
  - Squeeze last % of performance from design
  - Larger teams required
  - Best suited to datapath elements:
    - adders
    - memories
    - search [CAM] structures; reservation stations
- **Automated Flow**
  - Synthesis, place and route
  - Time to market
  - Tools have gotten pretty good
  - Dependent on good libraries (standard cell)
  - Initial RTL is portable between processes

These are not mutually exclusive; even teams that use “custom” design will use automated flow for non-critical or control blocks.



# Standard Cells

- Important to have rich set of standard cells for synthesis tool
  - drive strengths
  - complex gates (for area)
  - biased threshold (faster P, faster N)
- Static Power (leakage) minimization
  - This power is burned when doing NOTHING!
  - Multi-Vt
    - Incremental cost; 4-5 more masks out of 45 (implant)
    - higher Vt – slower, but less leakage
    - lower Vt – faster, but MUCH more leakage (use sparingly if at all)
  - Long channel
- Characterization
  - Synthesis – function, pins, timing, power
  - Place and route – pin location, blockages, power rails (abstracts)

# Implementation Step - Synthesis

- Maps RTL to gates (standard cells) – “netlist”
- Constraints – input/output arrival time, drive strength, load, physical floorplan
- Logic optimization – moves early arriving signals to front of stack, late to back of stack
- Selects arithmetic operators based on timing requirements (i.e. + types)
- Cost function – can end up local minima

**Late Arriving**

```
assign cancel_f1 = btb_predtkn_f3 |
    ubtb_predtkn_f2 | ic_miss_f4 | itlb_miss_f3;

assign fetch_f2_ns = fetch_f1 & ~ cancel_f1 &
    ~ if_flush;

always @(posedge ck_gclkcr)
    fetch_f2 <= #`DFE_DELAY fetch_f2_ns;
```

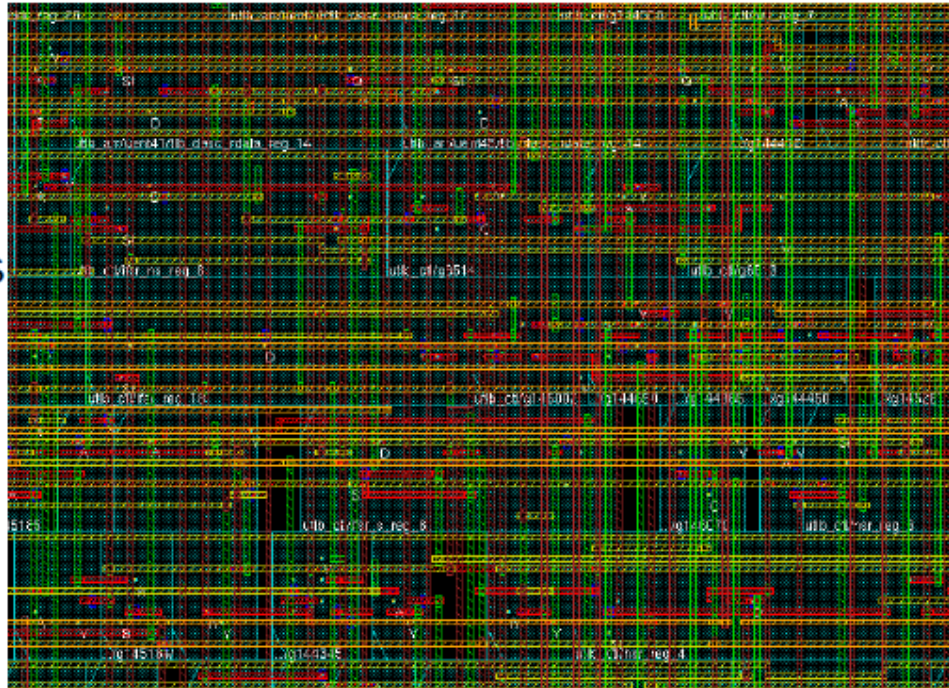
```
ANDN2_X4M g2014 (.A(if_flush),
    .BN(nI724), .Y(n2028));

NOR2_X1M g2023 (.A(n2028),
    .B(btb_predtkn_f3), .Y(fetch_f2_ns)

SDFFQ_X2M fetch_f2_reg
    (.D(fetch_f2_ns)...
```

# Implementation – Place and Route

- Placement has such massive effect on synthesis that tools combine the two – “DEF physical netlist”
- Routing – manage congestion; route critical signals first
- Post-route optimization – resize or add buffering for signals
- Clock tree
  - mesh, H-tree
  - clock tree synthesis
  - skew – useful sometimes
- Signal integrity
  - coupling
  - degradation of long runs



# Implementation Feedback

## ■ Timing report – does design meet timing?

Path 1: VIOLATED Setup Check with Pin utlb\_ctl/tlb\_stat\_match\_fl\_reg\_0/CK

Endpoint: utlb\_ctl/tlb\_stat\_match\_fl\_reg\_0/D (^) checked with leading edge of 'ck\_golkr'

Beginpoint: utlb\_ctl/dvm\_invtlb\_seq\_reg\_0/QN (v) triggered by leading edge of 'ck\_golkr'

```
Other End Arrival Time      0.400
- Setup                    -0.009
+ Phase Shift              1.000
+ CPPR Adjustment          0.000
- Uncertainty              0.100
= Required Time            1.291
- Arrival Time             1.169 **
= Slack Time               -0.065 **
```

Instance	Cell	Arc	Delay	Arrival	Slew	Load	Fanout
		ck_golkr ^		0.400	0.060		
utlb_ctl/dvm_invtlb_seq_reg_0	SDFFXRPQN_X2M_A12TS_C31	CK ^ -> QN v	0.074	0.474	0.027	0.004	1
utlb_ctl/plcwnsbuf_st6333	INV_X4B_A12TS_C31	A v -> Y ^	0.025	0.499	0.016	0.008	4
utlb_ctl/plcwnsbuf_st6330	INV_X2M_A12TS_C31	A ^ -> Y v	0.028	0.527	0.023	0.009	5
utlb_ctl/g6509	AND3_X6M_A12TS_C31	A v -> Y v	0.045	0.572	0.017	0.011	3
utlb_ctl/g120055	NOR2_X6A_A12TS_C31	A v -> Y ^	0.025	0.597	0.024	0.011	1
utlb_ctl/plctnsbuf_star7495	INV_X16M_A12TS_C31	A ^ -> Y v	0.022	0.619	0.016	0.036	5
utlb_ctl/tlb_stat_match_fl_reg_0	SDFPQH_X4M_A12TS_C31	D ^	0.000	1.169	0.018	0.002	

## ■ Power Report

# SoC integration

- Once core is built, integrated with other cores into chip
- Many millions of gates; can we abstract this out?
- System Design
  - SystemC model – transaction level, no timing
  - Can chain processor/peripheral models together to test OS
- Cycle-level system simulation
  - compiled model
  - no internal visibility
  - faster runtimes
  - smaller, simulator won't run out of memory!

# ARM Development Tools

- Software development
  - [ARM Development Studio 5 \(DS-5\)](#)
    - For ASICs and ASSPs
    - Compilers, debugger, system performance analyzer, real-time system simulator
  - [Keil Microcontroller Development Kit \(MDK\)](#)
    - For embedded microcontrollers
    - Cortex M, Cortex-R4, ARM7, ARM 9 devices
    - Compilers, debugger, simulators
- Models
  - ARM Fast Models – virtual platforms for software development before silicon

# Conclusions

- SoC design requires different design approach than traditional ASICs
  - More modeling & simulation at higher abstraction levels
- Heavy use of IP, re-usable modules, platform-based design
  - SoC design team must work with IP vendor and foundry
- Use platform design & standard interfaces between IP
- Hardware/software co-design
- Many design challenges