16. SoC and Microarchitecture Verification

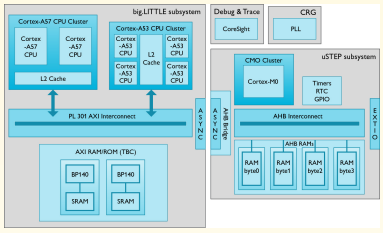
Jacob Abraham

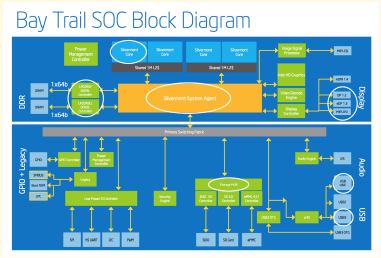
Department of Electrical and Computer Engineering
The University of Texas at Austin

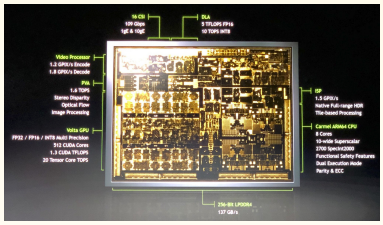
Verification of Digital Systems
Spring 2020

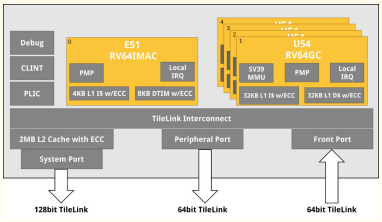
March 12, 2020

Systems on Chip

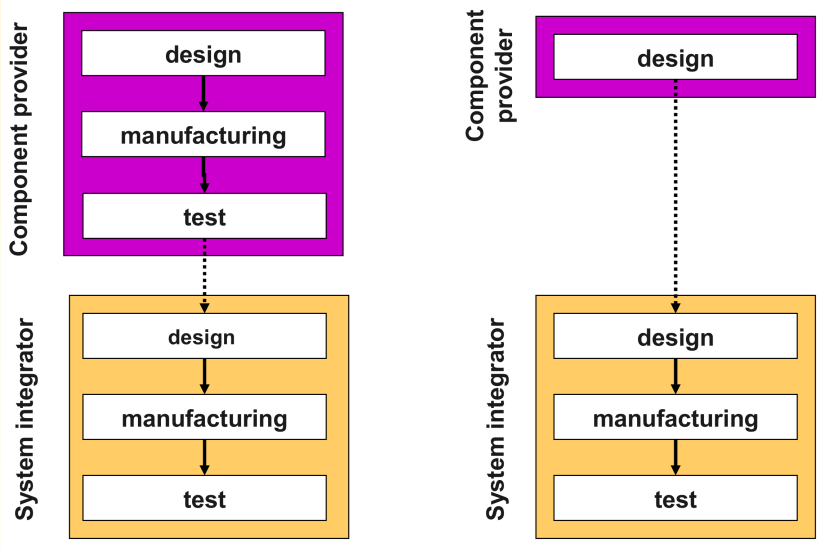








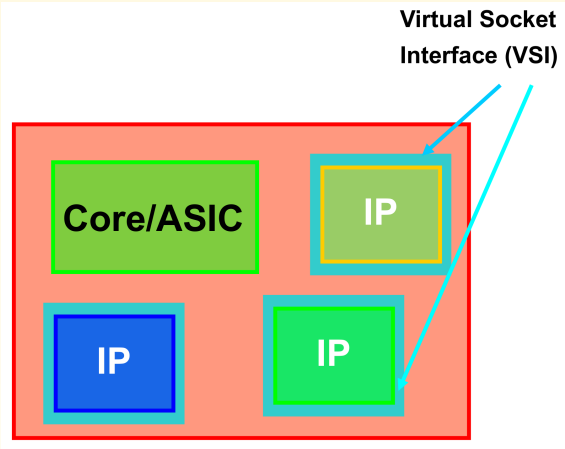
System-in-Package (SiP) vs. System-on-Chip (SoC)



Intellectual Property (IP) Reuse

- Different types of IP cores
- Hard IP – full layout
  - Area, performance, power known
  - No changes or optimization possible
- Soft IP – synthesizable HDL
  - Can be tailored for specific application
  - Harder to characterize
  - May not be as fast or small as hard IP
- Firm IP – netlist in target technology
  - Some optimizations possible

IP Reuse



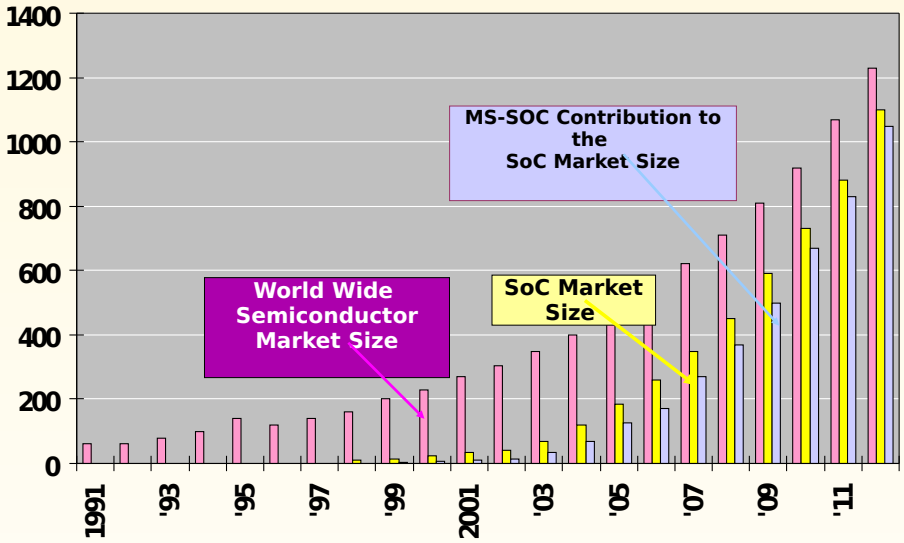
Virtual Socket Interface Alliance (VSIA): association of companies

Develop open standards ("virtual sockets") to allow mix-and-match of IP cores

Developing a System-on-Chip

- Application requirements
- Software, hardware (function) partitions
- Select processor cores
  - ARM, MIPS, Tensilica, DSP
- Memory and interfaces
  - DRAM, SRAM, Flash, Rambus, etc.
- System interfaces
  - USB, PCI, PCMCIA, Ethernet, 802.11, Firewire, Bluetooth, etc.
- Glue ASICs

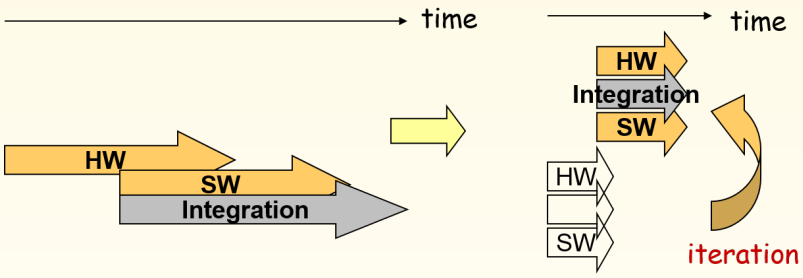
The World is not all Digital: System-on-Chip Market Size ($ Billions)



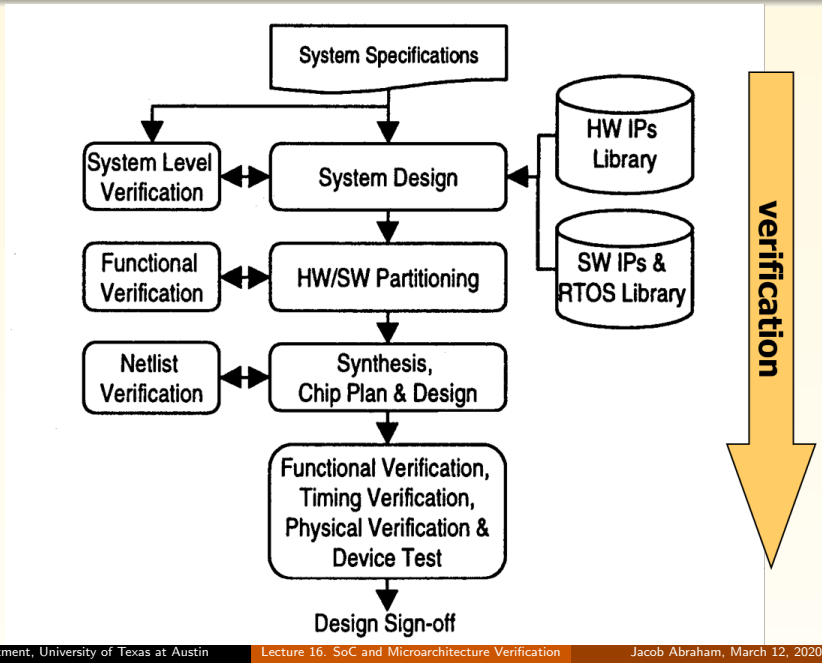
SONY Corp & Market Estimates

Hardware/Softwere Co-Design

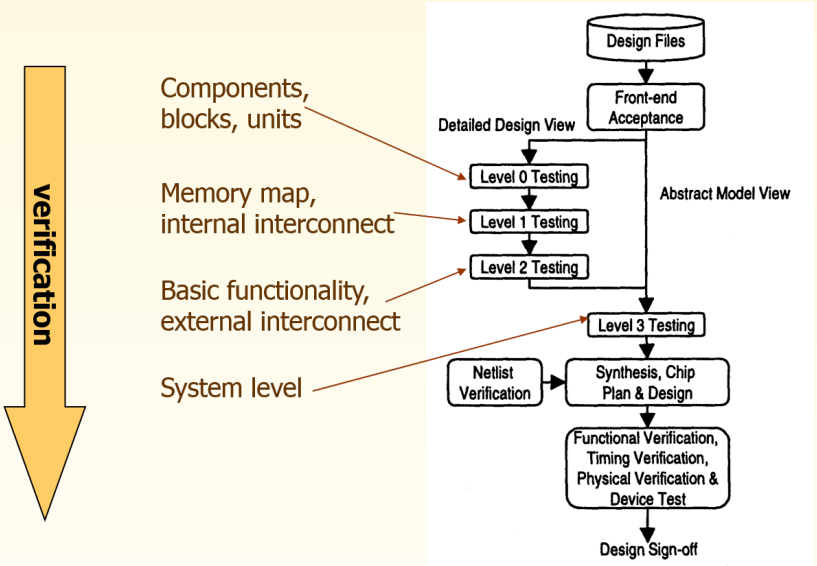
- Concurrent design of Hardware/Software components
- Evaluate the effect of a design decision at an early stage by "virtual prototyping" **Co-Verification**



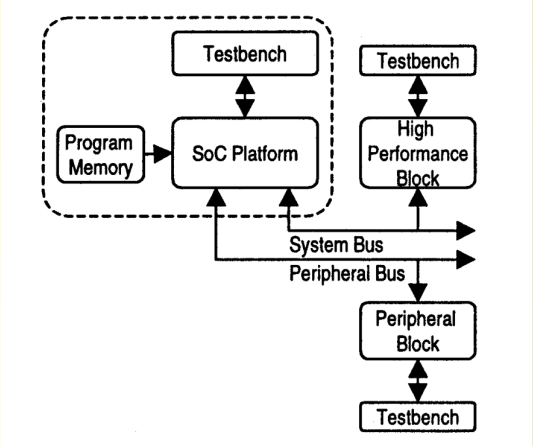
Top-Down SoC Verification



Bottom-Up SoC Verification



Platform-Based SoC Verification

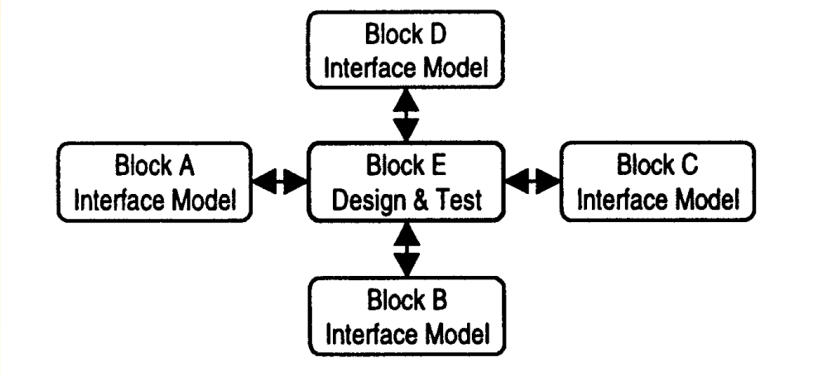


Derivative Design

Interconnect Verification between:

- **SoC Platform**
- **Newly added IPs**

System Interface-Driven SoC Verification

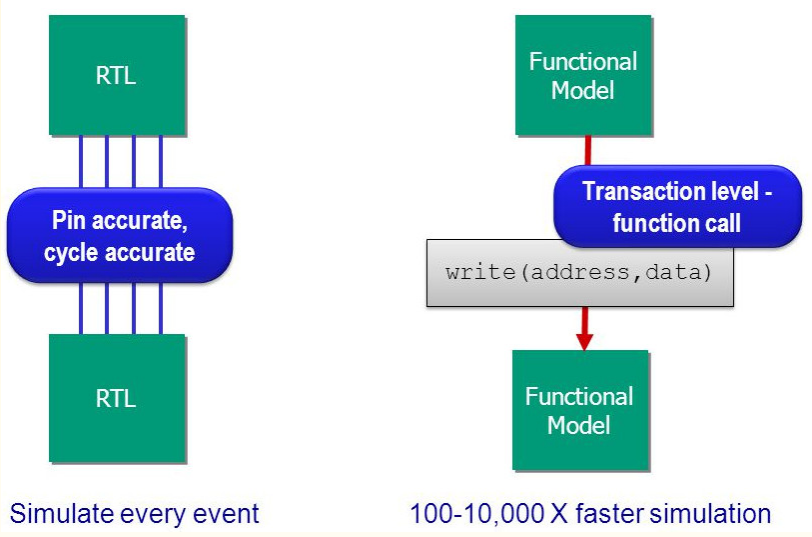


Besides Design-Under-Test, all others are interface models

Issues with Testbenches for SoC Verification

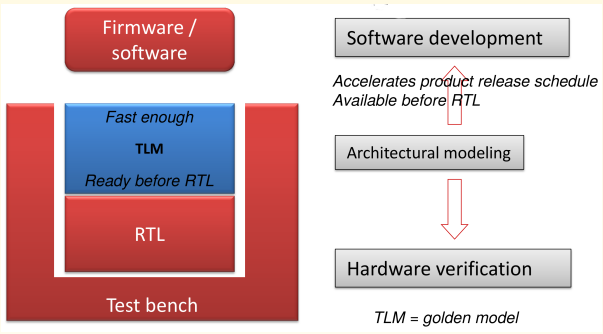
- Need for system-level stimuli
  - Generally comes from the application
- Need system-level models
  - Models for high-level algorithms
- Automating tests
  - Require constraints on tests – legal instructions, inputs to algorithms
- Dealing with IP blocks
  - "Hard" IP (ARM cores, etc.)
  - Analog/mixed-signal blocks, etc.
- Coverage metrics

Transaction Level Modeling (TLM)



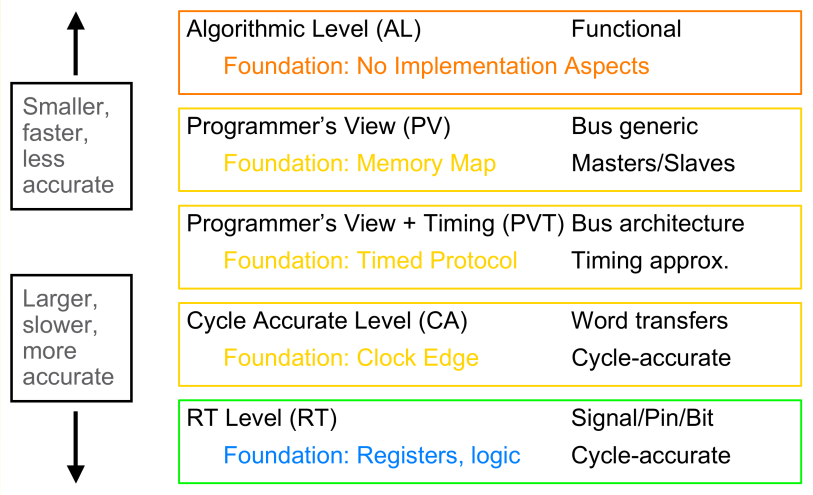
Source: Open SystemC Initiative

Benefits of TLM



Source: A. Khan and A. Sharma, DVCON India, 2015

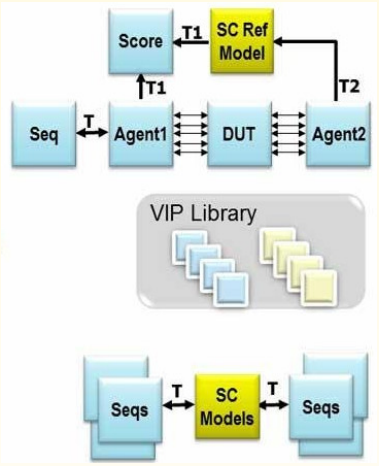
TLM Abstraction Levels



Source: S. Swan, Cadence, Inc.

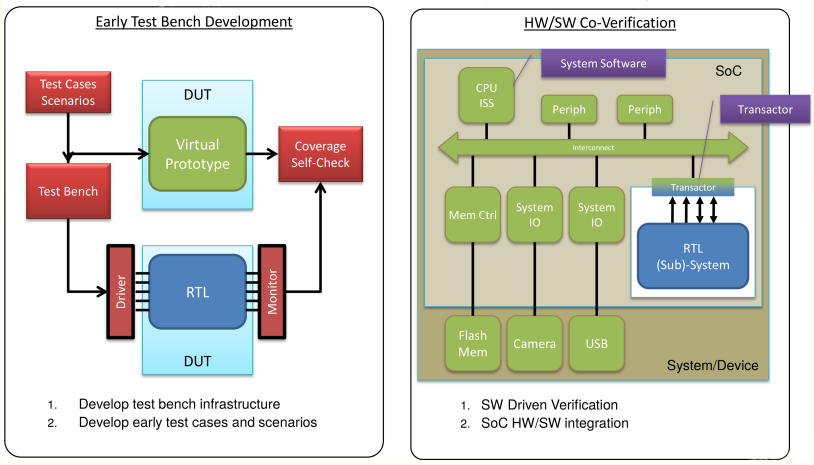
TLM in a UVM Framework

- **UVM-Connect** library provides TLM connectivity and object passing between SystemC and SystemVerilog UVM modules and components
- Allows abstraction refinement, reusability
- Can drive SystemC models with random stimulus from SystemVerilog



Source: Verification Academy, Mentor Graphics

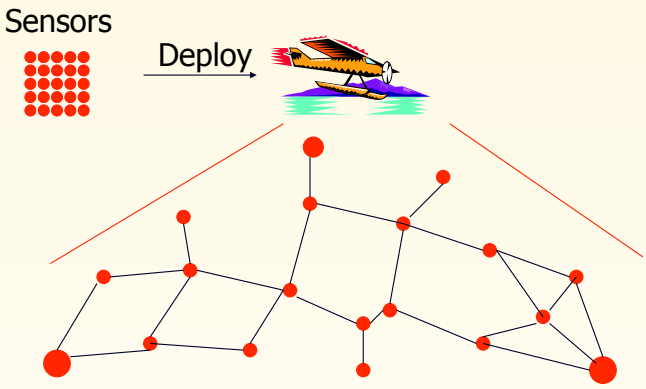
Leveraging TLM for Hardware-Software Integration



Source: A. Khan and A. Sharma, DVCON India, 2015

Networked Devices Require Their Own Identity

- Node-to-node identification in sensor networks



W. Du, "Securing Wireless Sensor Networks," Syracuse University

Device Fingerprints: A Review

MOSFET drain current difference
[Loftstrom, *ISSCC* 2000]

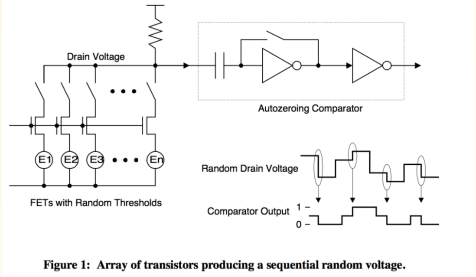
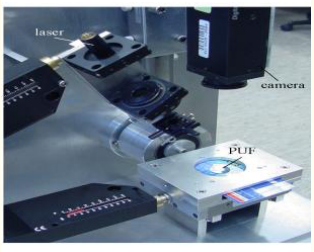


Figure 1: Array of transistors producing a sequential random voltage.

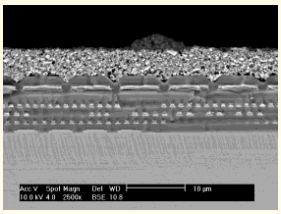
Optical PUF
[Pappu, *Science* 2002]



Device Fingerprints: A Review (Cont'd)

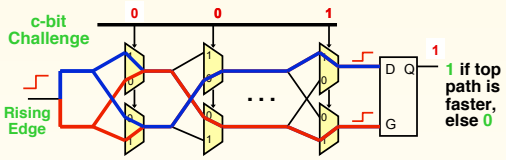
Coating PUF

[Tuyls, *CHES* 2006]



Arbiter PUF

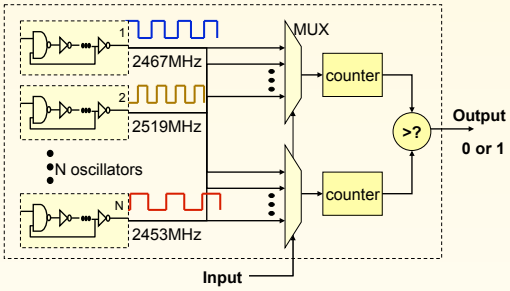
[Gassend, *CCS* 2002]



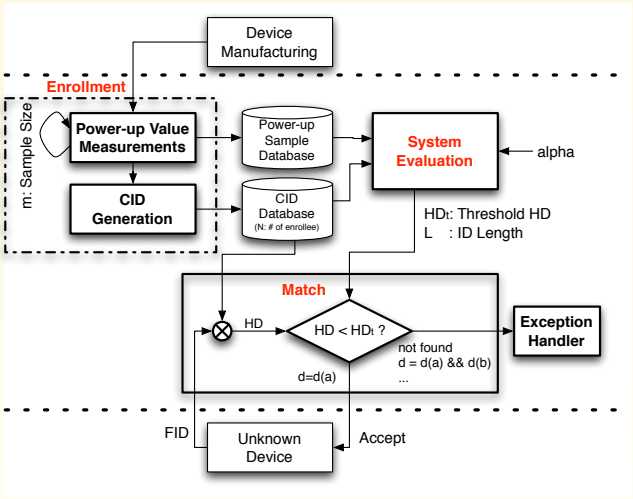
Device Fingerprints: A Review (Cont'd)

Ring Oscillator PUF

[Suh, *DAC* 2007]



SRAM-based Identification System



Constraints on Embedded Systems

Myriad of Intelligent systems

- Cost, power consumption constraints
- In critical applications, **Safety and Resiliency** are keys

Example: self-driving cars

- 100 Million lines of code for software, sensing and actuation
- 64 TOPS for cognition and control functions

Fundamental Requirements for Resilience

- **Redundancy**
- **Diversity**

"The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods"

Dionysius Lardner, "Babbage's calculating engine," *Edinburgh Review*, vol. 59, no. 120, pp. 263–327, 1834.

Dealing with Security – Very Different From Dealing with Physical Faults or Errors

Attacks are Intentional

- Faults and Errors related to design or physical causes are **systematic** or **random**
- Attacks are **deliberate**
  - Initiated by a clever adversary

Security Attacks

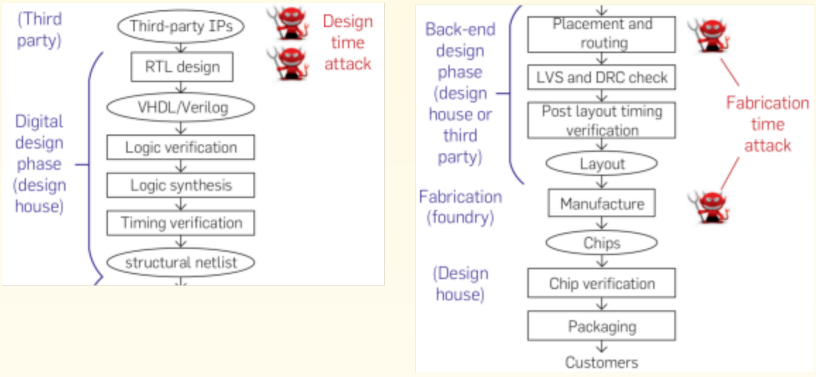
Hardware Trojans

- Malicious modification of designs
- Example of analog circuitry modifying a digital chip – extremely difficult to identify
- Design diversity may be a solution

External attacks

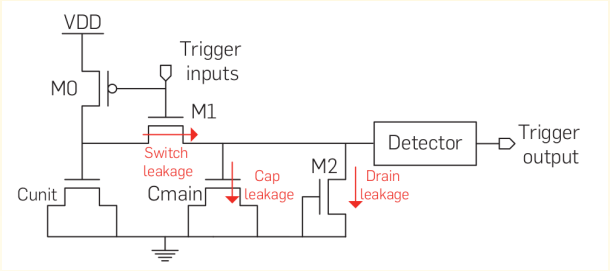
- Classic work (Abadi) suggested control flow checking to detect execution of undesired code
- Effects of attacks could include modification of data, execution sequences, denial of service, etc.
  - Require data checks in addition to control-flow checks
  - Need to detect DoS attacks during operation – example, shutting down GPS system (or spoofing GPS position)

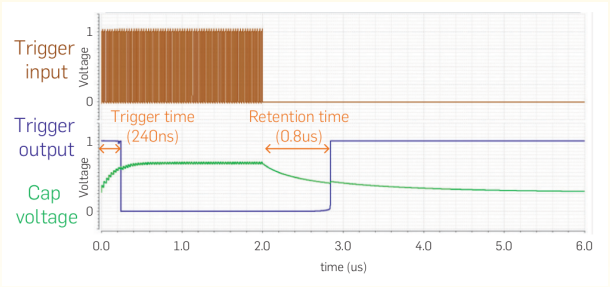
IC Design Process with Possible Attacks



Source: Yang et. al, Communications of the ACM, September 2017.

Schematics and Simulation of Analog Trigger Circuit

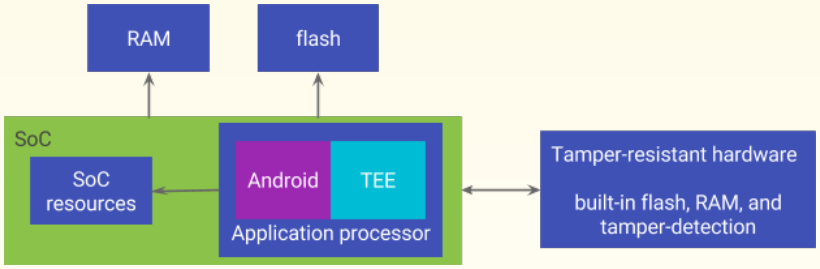




Android Pixel 2 Security Module

Tamper Resistant Hardware

- Discrete chip separate from the SoC, with its own flash, RAM
- Can control its own execution, and is robust against side channel information leakage attacks
- Loads its OS and software directly from internal ROM and flash, and controls all updates
- Resilient against fault injection and side channel attacks



Source: X. Xin, Android Developers Blog, 13 November 2017

Control Flow Deviation Detection for Application Level Security

Attacks subvert the control flow of the software

- Insert control-flow checks in the code (particularly useful for embedded software)
- Run-time signatures and checks can be inserted automatically during compile time

Implementation

- Signature update instructions inserted at the beginning and end of each function, as well as before and after the call instructions
- Illegal branches will result in signature mismatches

Proposed in 2005 (Abadi)

Example – Detection of Illegal Jump

Pre-computed signatures

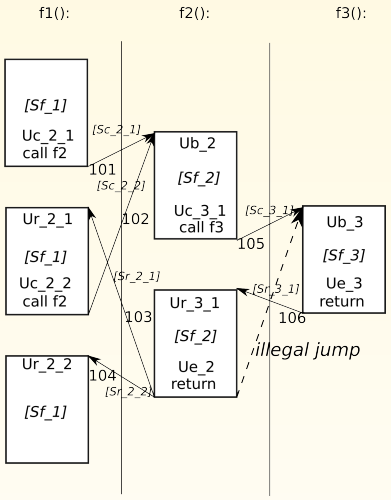
```
Sf_1   = 000
Sf_2   = 001
Sf_3   = 010
Sc_2_1 = 011
Sc_2_2 = 011
Sr_2_1 = 100
Sr_2_2 = 100
Sc_3_1 = 101
Sr_3_1 = 110
```

Update instructions

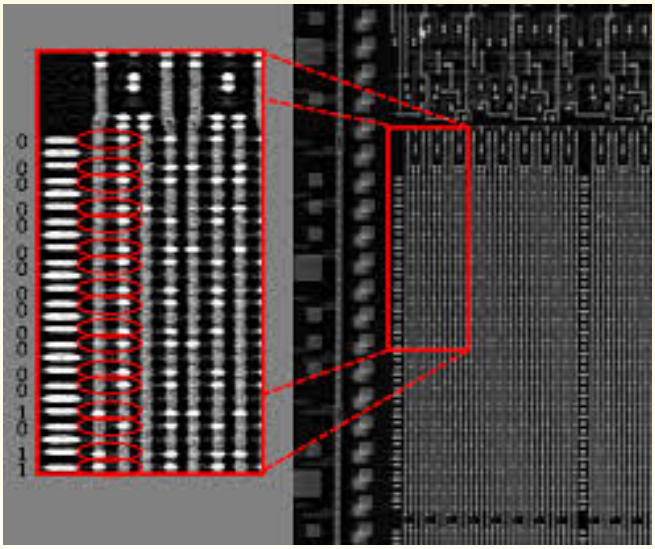
```
Ub_2:   S = S XOR 010
Ue_2:   S = S XOR 101
Ub_3:   S = S XOR 111
Ue_3:   S = S XOR 100
Uc_2_1: S = S XOR 011
Uc_2_2: S = S XOR 011
Ur_2_1: S = S XOR 100
Ur_2_2: S = S XOR 100
Uc_3_1: S = S XOR 100
Ur_3_1: S = S XOR 111
```

Detection of illegal jump

In case of illegal jump,
S inside f3 = Ub_3(Ue_2(Sf_2))
= 001 XOR 101 XOR 111
= 011
S inside F3 != Sf_3

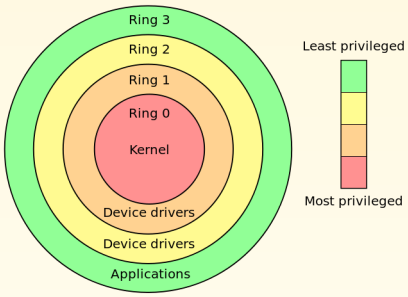


Partial View of Intel Microcode



Source: Koppe et. al, 26th USENIX Security Symposium, August 2017

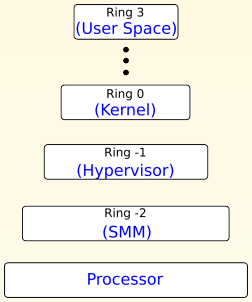
Security Vulnerabilities in Hardware and Microcode



x86 Protection Rings (Wikipedia)

- **System Management Mode (SMM)** introduced in 1990
- CPU executes code from separate area of memory (SMRAM)
- SMRAM is only accessible by the processor, (not even OS)
- SMM handles system-wide functions like power management, hardware control, proprietary OEM code
- Intended for use only by system firmware

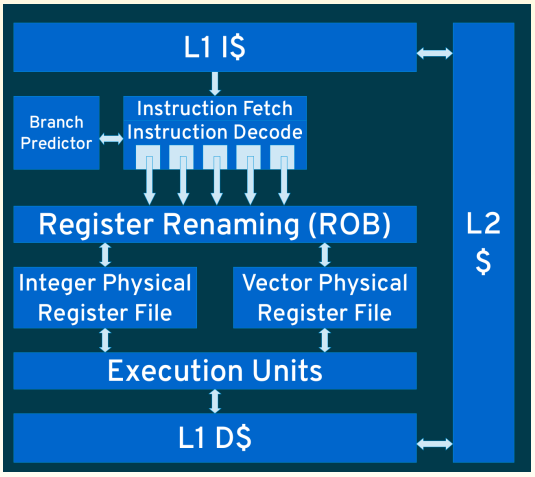
Sinkhole Security Vulnerability



- Remap Local Advanced Programmable Interrupt Controller (APIC) over SMRAM range during critical execution states
- Cause memory accesses that should be sent to the MCH to be prematurely accepted by the APIC instead
- Permits malicious ring 0 code influence over the SMM

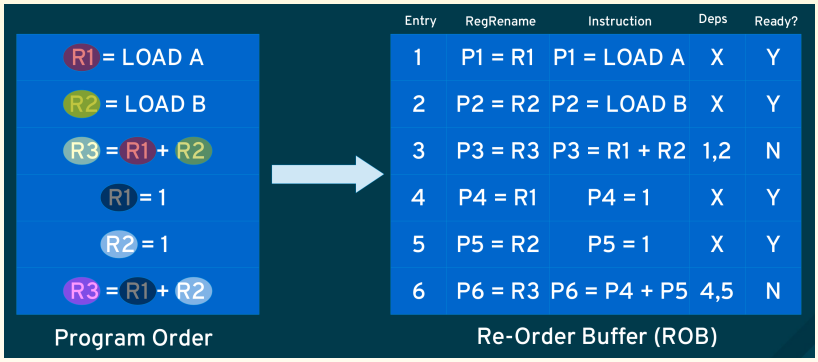
`https://www.blackhat.com/docs/us-15/materials/us-15-Domas-The-Memory-Sinkhole-Unleashing-An-x86-Design-Flaw-Allowing.pdf`

Exploits on Modern Microarchitectures: Meltdown, Spectre, etc.



Source of slides on this topic, J. Masters, RedHat, Inc., presentation at FOSDEM 2018

Out-of-Order (OoO) Microarchitectures

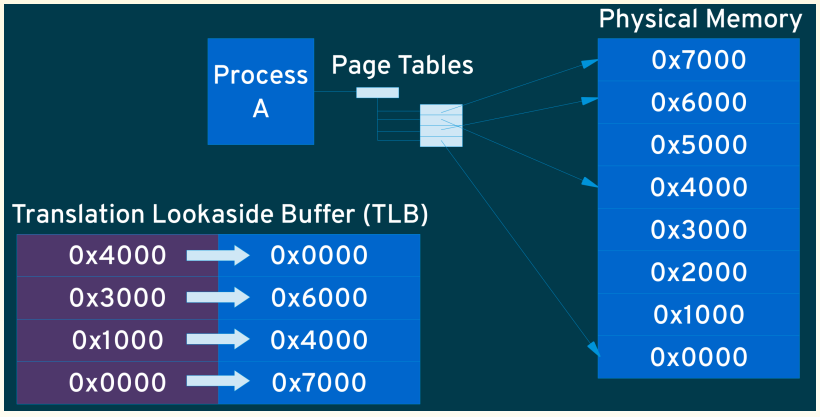


Invented by R. Tomasulo and used in the IBM System/360-91 FPU

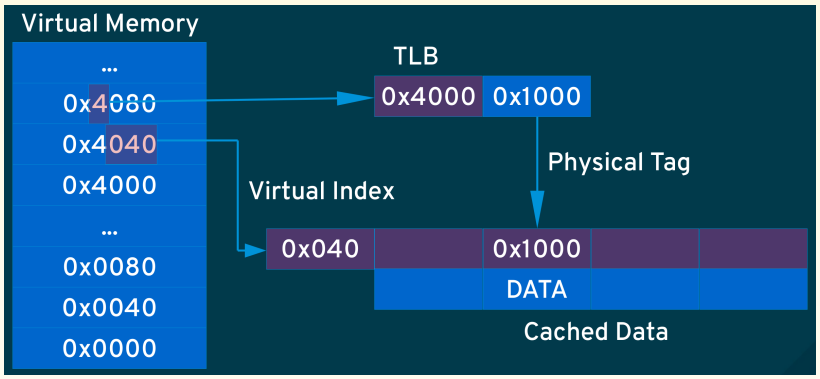
Instructions are decoded in-order and send to the OoO backend

Reorder Buffer defines an execution window

Virtual Memory



Caches



Side-Channel Attacks

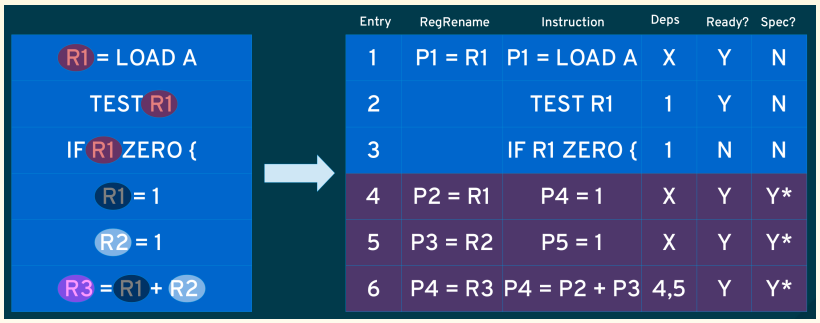
Attack based on information gained from the physical implementation, rather than weaknesses in the implemented algorithm

- Monitoring EM radiation ("Tempest" remote attack)
- Measuring power consumption (differential power analysis)
- Timing the length of operations to derive machine state
- etc.

Caches can be exploited as side channels

- Difference in access time for a given address can be measured by software
- Thus, possible to determine whether a specific address is in the cache

Branch Prediction and Speculation

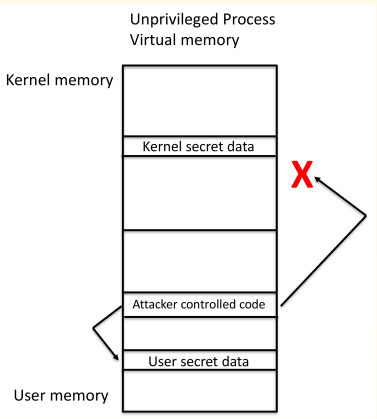


Entry	RegRename	Instruction	Deps	Ready?	Spec?
1	P1 = R1	P1 = LOAD A	X	Y	N
2		TEST R1	1	Y	N
3		IF R1 ZERO {	1	N	N
4	P2 = R1	P4 = 1	X	Y	Y*
5	P3 = R2	P5 = 1	X	Y	Y*
6	P4 = R3	P4 = P2 + P3	4,5	Y	Y*

The branch is speculatively executed before the condition is resolved

If the predicted branch was incorrect, speculated instruction can be discarded, and does not become architecturally visible

Example: Spectre Attack



- User process reads a byte of arbitrary kernel memory
- This will eventually cause an exception, but, due to out-of-order exection, will leak the data to a side channel before the exception handler is invoked

Detecting Hardware Security Vulnerabilities by Unique Program Execution Checking (UPEC)

Example code for attack

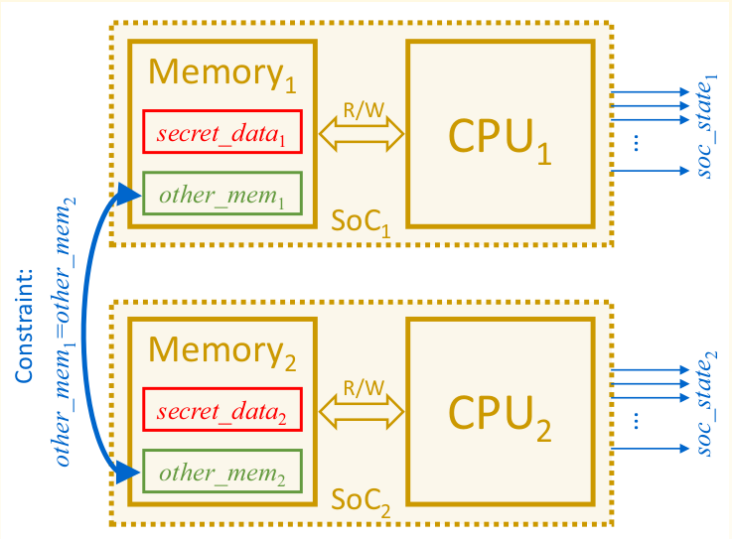
```
1: li x1, #protected_addr     // x1 ← #protected_addr
2: li x2, #accessible_addr    // x2 ← #accessible_addr
3: addi x2, x2, #test_value   // x2 ← x2 + #test_value
4: sw x3, 0(x2)               // mem[x2+0] ← x3
5: lw x4, 0(x1)               // x4 ← mem[x1+0]
6: lw x5, 0(x4)               // x5 ← mem[x4+0]
```

- Some secret data is stored in a protected location (address)
- Assume the cache holds a valid copy of the secret data
- The attacker program guesses the cache line with the secret data and sets the condition for a RAW hazard by writing to the line
- If the guess was correct, the hazard occurs, leading to a slightly longer execution time

Fadiheh et. al, DATE 2019, pp. 988–993

ECE Department, University of Texas at Austin | Lecture 16. SoC and Microarchitecture Verification | Jacob Abraham, March 12, 2020 42 / 50

Computational Model for UPEC



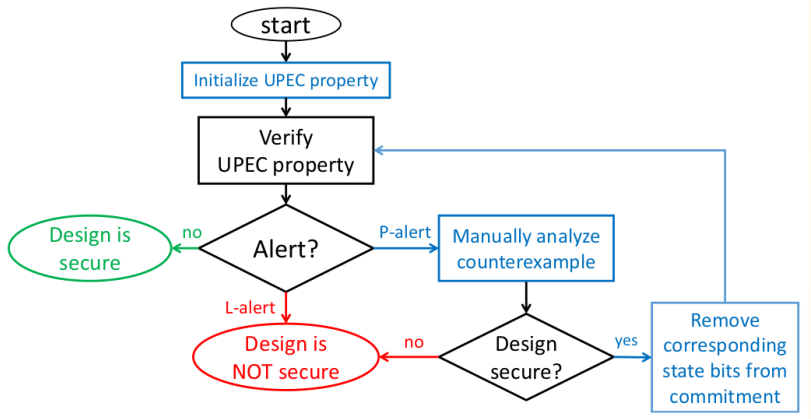
ECE Department, University of Texas at Austin | Lecture 16. SoC and Microarchitecture Verification | Jacob Abraham, March 12, 2020 43 / 50

Formal Property for UPEC

AG(secret_data_protected $\wedge$ micro_soc_state$_1$ = micro_soc_state$_2$
$\rightarrow$ AG soc_state$_1$ = soc_state$_2$

assume:
at t: *secret_data_protected*();
at t: *micro_soc_state*$_1$ = *micro_soc_state*$_2$;
at t: *no_ongoing_protected_access*();
during $t..t+k$: *cache_monitor_valid_IO*();
prove:
at $t+k$: *soc_state*$_1$ = *soc_state*$_2$;

UPEC Approach



Results of UPEC Experiments

	D cached	D not cached
d_{MEM}	5	34
Feasible k	9	34
# of P-alerts	20	0
# of RTL registers causing P-alerts	23	N/A
Proof runtime	3 hours	35 min
Proof memory consumption	4 GB	8 GB
Inductive proof runtime	5 min	N/A
Manual effort	10 person days	5 person hours

"Unfixable" boot ROM Security Flaw"

In the Special Purpose Engine Within Every Processor Chip

- The "Converged Security and Manageability Engine CSME"
- Has its own CPU, own RAM, own code in a boot ROM
- And, access to the rest of the machine
- Recent implementations based on 80486, running a free microkernel OS, *MINIX*

The CSME Performs Crucial Tasks

- Runs below the OS, hypervisor and firmware
- Controls power levels
- Starts the main processor chips
- Verifies and boots the motherboard firmware
- Provides cryptographic functions

Information in this and following slides from S. Nichols in *The Register*, March 5, 2020

The Exploit

When Powered Up, CSME ...

- Sets up memory protections on its owm built-in RAM
- Now, other hardware and software can't interfere with it
- The protections are disabled by default

The Problem

- There is a tiny timing gap between the system turning on, and the CSME executing the code in its boot ROM that installs the protections
  - The code is in the form of I/O memory management data structures (page tables)
- During the timing gap, other hardware (even on the motherboard) can initiate a DMA transfer into the CSME's private RAM, overwriting variables and pointers
- In this case, the CSME can be commandeered for malicious purposes, out of view of the software running above it

The Problem, Cont'd

Hacking a Processor

- The exploit can be attempted when the machine is switched on, or it wakes up from sleep (which resets the protections)
- Would need local access to a box to exploit this
- The boot ROM is read only, and cannot be patched
- The IOMMU's reset defaults cannot be changed without a respin

Report of vulnerability and fixes

- Reported to Intel by Positive Technologies (details being withheld till a white paper is ready)
- Intel developed a software patch that prevents the chipset's integrated hub from attacking the CSME
- Positive thinks there may be other ways in

Impacts

Multiple security features in CSME

- "Enhanced Protection ID" for anti-piracy DRM protections and IoT attestation
- Provides TMP functions which allow OS and applications to securely store and manage digital keys for file system encryption, etc.
  - At the heart of this cryptography is a "Chipset Key" that is encrypted by another hard-coded key

Managing the exploit

- To fully compromise the protection ID, hackers would need to extract the hardware key used to encrypt the chipset key, and this resides in the "Secure Key Storage"
- Problem: the key is not platform specific (used for an entire generation of chipsets)
- Probably only a matter of time, by exploiting the ROM vulnerability