System-on-Chip Platform Security

Architecture, Design, Deployment, Debug

Sandip Ray and Swarup Bhunia

Department of Electrical and Computer Engineering

University of Florida

{sandip, swarup}@ece.ufl.edu

http://{sandip, swarup}.ece.ufl.edu

Acknowledgements

- Sean Baartmans
- Debra Bernstein
- Jay Bhadra
- Jeremy Casas
- Wen Chen
- Kai Cong
- Victor DelaGarza
- Monica Farkash
- Jason Fung
- Iwan Grau
- Raghudeep Kannavara
- Sava Krstic
- Li Lei
- Rebekah Leslie-Hurd
- Dhinesh Manoharan
- David Ott
- Jim Schwartz
- Mark Tuttle
- Jin Yang
- Abhishek Basak
- Srivalli Boddupalli
- Bo Chen
- Atul Prasad Deb Nath
- Domenic Forte
- Christopher Havlicek
- Sharad Malik
- Prabhat Mishra
- Mark Tehranipoor
- Fei Xie
- Hao Zheng

Platform Security Assurance

Review the overall platform
for attack opportunities
which undermine security objectives
in ways not covered by traditional approaches

Tutorial Goals

- Explain the scope of SoC platform security
- De-mystify and comprehend many of the security activities performed in practice today
- Discuss limitations in current state of practice
- Present some emergent solutions

A significant emphasis is on current industrial practice (and its limitations)

Traditional Computer Security
vs.
Modern SoC Security

How can we minimize vulnerability of our computing systems to malicious attacks?

Changing Computing Landscape…

"An embedded system is an electronic system that uses computer hardware and software, and perhaps additional electrical or mechanical parts, ~~intended to provide a dedicated function~~"

— PC Magazine, 2012

— Michael Barr, *Programming Embedded Systems*, 1999

General-purpose systems

- Complex, optimized architecture

- Customized design
- Unique use-case constraints
- Tight HW/SW integration
- Complex, optimized architecture
- Versatility and programmability
- Diverse use-case scenarios

How can we minimize vulnerability of our **highly complex** modern computing systems to malicious attacks?

….In a Highly Connected World….

"City of Hamburg and CISCO launch plan for smart city of the future"
BBC World News, May 1 2014

Things

Network Services

How can we minimize vulnerability to malicious attacks of diverse, highly complex computing systems in hands of possibly naïve users, **operating in an environment of billions of complex, error-prone, possibly malicious communicating devices**?

…Created By Complex Supply Chain…

How can we develop trustworthy, dependable, usable computing infrastructures with billions of smart, highly complex, connected computing devices, operating in a potentially malicious environment, and **developed by a complex supply-chain of untrustworthy players**?

…Built Under Aggressive Schedule…

How can we develop trustworthy, dependable, usable computing infrastructures with billions of smart, highly complex, connected computing devices, operating in a potentially malicious environment, and developed by a complex supply-chain of untrustworthy players within **a strict time-to-market requirement**?

Different Faces of Security

Device Security

- NFC and RFID
- Hardware primitives such as PUFs

Platform Security

- HW/FW/SW Interactions
- System-level Policies
- Design and development of security architectures

Cloud Security

- Mobile cloud
- Cloud clients and infrastructures

Scope of Platform Security

(10 mins)

Platform Security Assurance

Review the overall platform for attack opportunities which undermine security objectives in ways not covered by traditional approaches

- All reasonable attacker starting points
  - Software, physical access
- All reasonable attacker motivations
  - Data theft, jailbreaking, DRM bypass, remote access, DoS, ….

Platform Security Assurance

Review the overall platform
for attack opportunities
which undermine security objectives
in ways not covered by traditional approaches

- Minimize duplication of efforts
- Find gaps in current approaches
  - Third party interactions
  - ….

CanSecWest 2015

Attack on Intel BIOS for X86-Based Computers

- E
- 3

CanSecWest 2015

Attack on Intel BIOS for X86-Based Computers

- Exploits X86 architectural vulnerability
- 3161 Shipped BIOS images, 3 vulnerabilities, only 12 misses!

But BIOS can be patched, so I needn't worry, right!

- In reality, most people do not patch BIOS
- Since BIOS code is reused, infection is reusable and automatable

This all carries over to our smartphones and hand-held devices, while introducing more vulnerabilities

Mobile Devices: Attack Surface

Attacks on Privacy via malicious apps and in-app Ad libraries

Location

Pictures

SMS

Calender

Contacts

E-Mails

Music

Baseband and 3G

Mobile Network Attacks

Sensor Malware

Exploit Mobile Cloud Apps

Malware (e.g., Games)

Browser Attacks

Premium-Rate Services

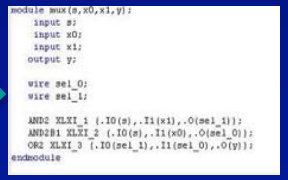
Hardware Attacks

SIM and SD Card, NFC

SoC Design Flow

SW/FW

HW



1st Silicon

Tape-out

PRQ

Exploration | Planning | Development | Production

Pre-silicon | Post silicon | Survivability

Security Assessment → Security Architecture → Security Validation → Security Validation → Security Validation

Risk Review
Architecture Definition

Threat modeling
Architecture review
Design review
Security Test plans
Architectural FV

Code review
RTL Testing
RTL FV

Penetration Testing

Assets, Policies, and Enforcement

The "CIA" Pillars

Confidentiality

- Read access control of sensitive data
- Authorization and authentication are essential

Integrity

- Write access control of sensitive data
- Authorization and authentication are essential

Availability

- Fault tolerance, Robustness, Handle bad inputs, DoS
- Fail safe, handle exceptions safely and securely

What's in Security Policy?

Assets

- Keys (Developer/OEM)
- Premium content
- End user information
- (Firmware) Execution flows
- Debug modes
- ….
- ….

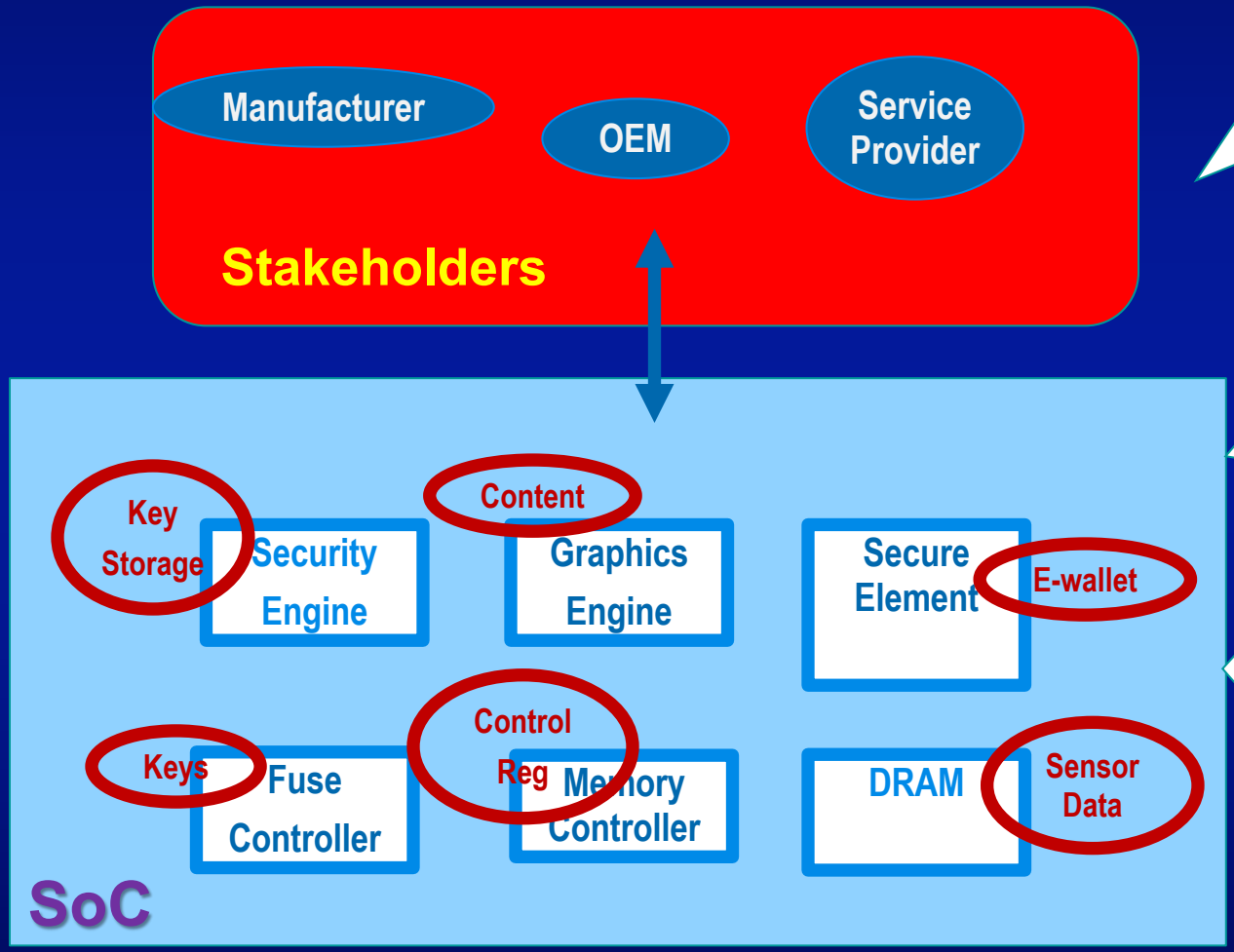
Policy

Protection Requirement

- Confidentiality
- Integrity
- Anti-replay
- Privacy Aspects
- ….
- ….

- **Multiple owners over lifetime of the parts**
- **Multiple IPs that do not trust each other**

Access to Assets



Stakeholders provision Assets such as Keys, E-wallet which reside in SoC building blocks such as Fuses, Secure Element etc.

As Assets flow through the various SoC building blocks during boot, additional assets are created at runtime

For every access to assets and building blocks holding and managing them, determine:

Who is accessing the asset?

What actions are requested?

When access is requested?

Example: Master Key

Master key is the root of all key blobs used by the cryptographic engines in the chip. Other keys are derived from it.

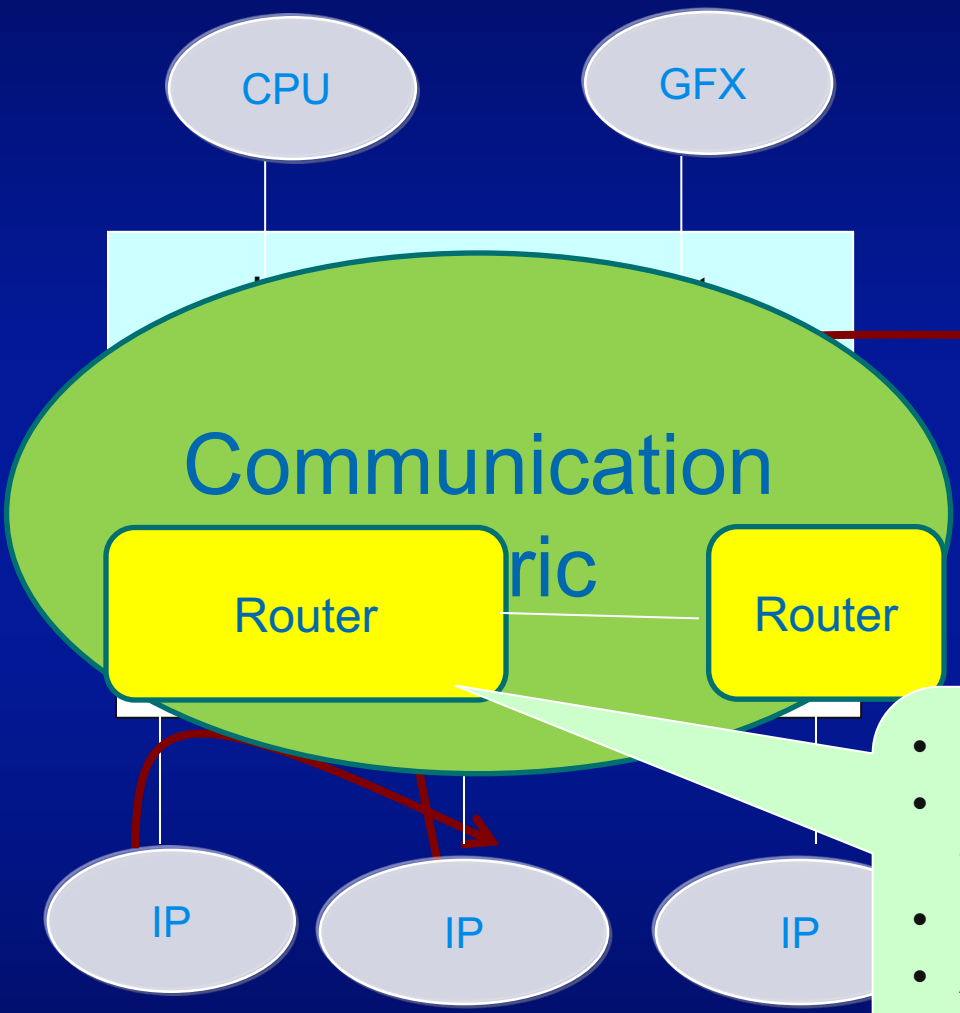
Who	When	How	Condition
Firmware	Boot	Read	Internal boot

Who	When	How	Condition
Any	After Boot	No read	Any

Who	When	How	Condition
OEM	Prod	Write	Authen-ticated

Who	When	How	Condition
Customer	Deploy	No write	Any

More Complexity: Fabrics



- Message immutability
- Redirection prevention
- Masquerade prevention
- Non-observability

- BARs used to get to the destination
- Transactions over some fabric subject to redirection by host OS
- Cannot trust OS controlled BARs
- Alternate routing mechanism necessary for secure communication

Industrial Primitives

- **ARM Trustzone**
  - Partition HW and SW resources into secure and insecure worlds
  - HW supports access control, permissions, and communications
  - SW supports secure system-call / interrupts for run-time execution
- **Intel SGX**
  - TEE to protect applications against malicious OS
  - Applications can create secure enclaves as “islands of trust”
  - Implemented as a collection of new CPU instructions
- **Samsung KNOX**
  - Partition between business and personal content
  - Hot swap between the two worlds

Enforcement Requirements

- Standardized language for security policies
- Parameterized, instantiable policy architecture
- Tools to synthesize policy implementations
- Effective validation strategies

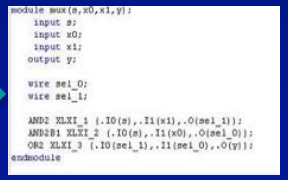
- **Some progress made on each vector by academic/industrial research**
- **But a large gap remains between the state of the art and what we need to be effective**

SoC Security Validation

SoC Design Flow

SW/FW

HW



1st Silicon

Tape-out

PRQ

Exploration | Planning | Development | Production

Pre-silicon

Post silicon

Survivability

Security Assessment → Security Architecture → Security Validation → Security Validation → Security Validation

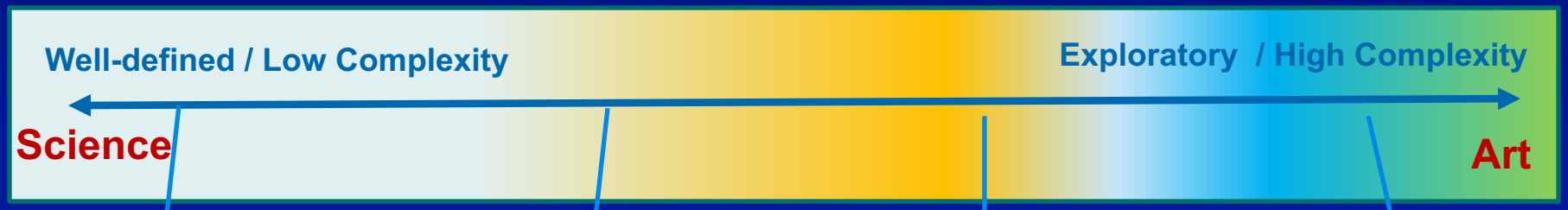
Risk Review
Architecture Definition

Threat modeling
Architecture review
Design review
Security Test plans
Architectural FV

Code review
RTL Testing
RTL FV

Penetration Testing

Science and Art of Security Validation



Functional validation of features providing security assurance

Crypto, secure boot, patching, etc.

Example: Crypto engine encrypts and decrypts data correctly on all modes

Validation of Deterministic Security Requirements

Access control restrictions, address translation, etc. Bugs are likely overlooked during normal usage

Example: Calls to DMA access to DMA-protected memory range through DMA address translation regs. must be aborted

Negative Testing

Look beyond what is specified, see if security objectives can be subverted or are underspecified

Example: Are there other paths to protected DMA memory in addition to DMA address translation regs? How to activate them?

White Box Expert Hacking

Goal-oriented attempts by expert hackers at breaking security objectives (often at HW/FW/SW divide or chip boundary)

Three Easy Steps to Validation

- Identify the security objectives
- Model the security threats
- Validate objectives against threats

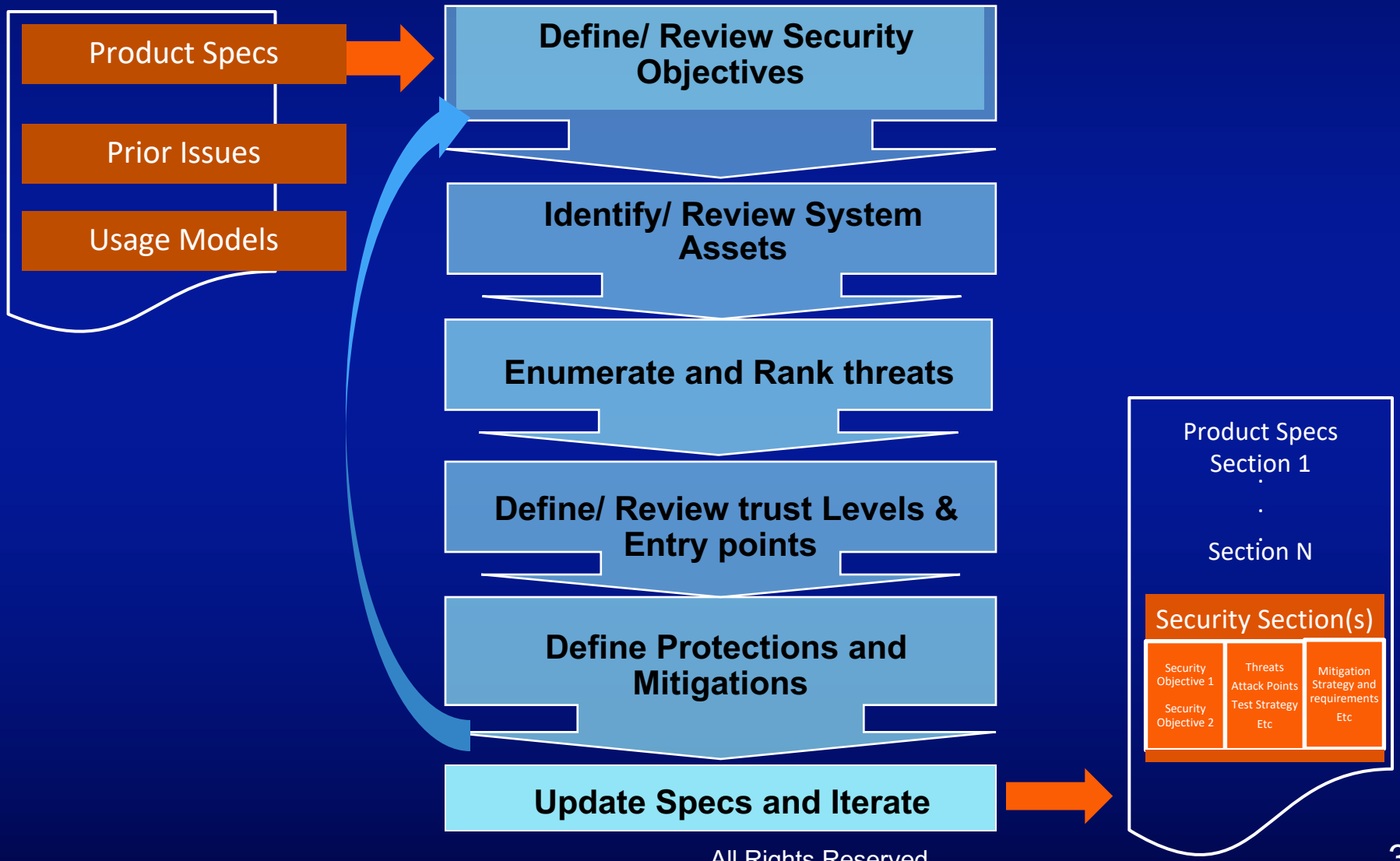
Adversaries in SoC Security

Asset owners
- Manufacturer, OEM, End user

Adversaries
- Manufacturer, OEM, End user

- Adversary Capabilities
  - Unprivileged software adversary
  - System software adversary
  - Software side-channel/covert-channel adversary
  - Simple hardware adversary
  - Skilled hardware adversary
  - Hardware reverse-engineer adversary

Modeling Threats

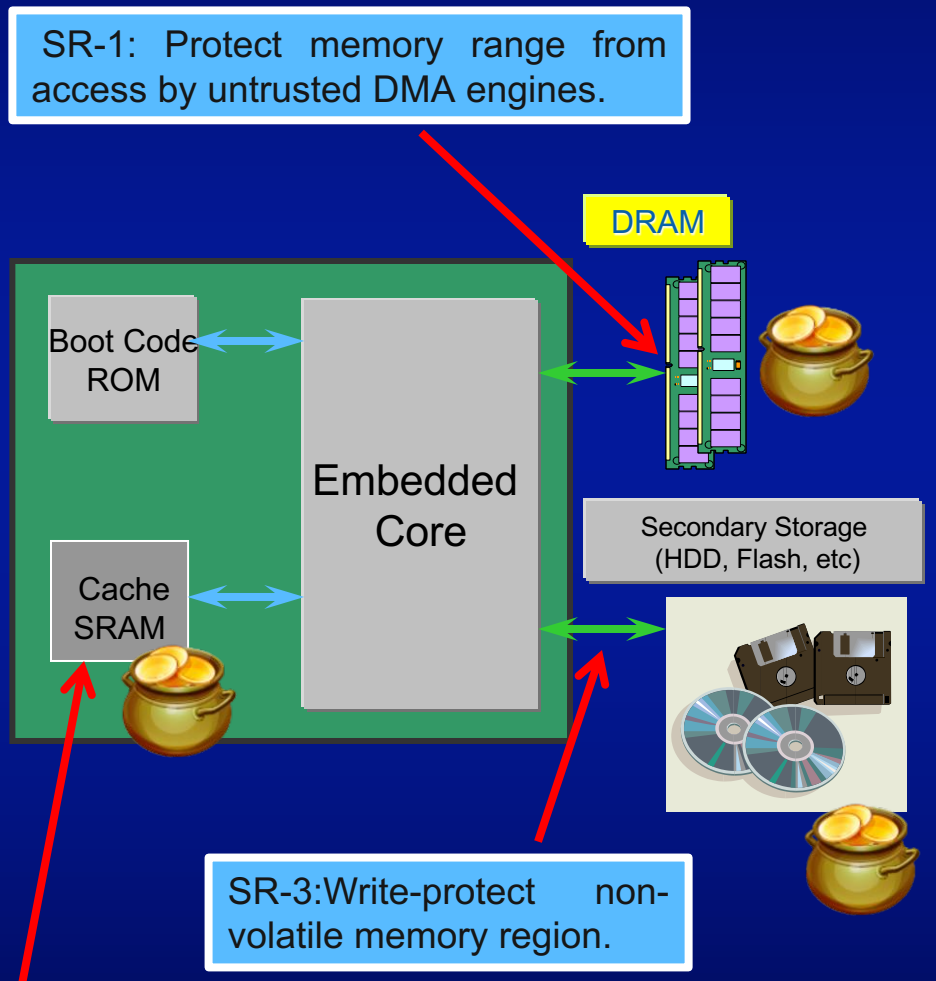


Example: Code Injection Threats

Security Objective:
Prevent Code Injection in storage and runtime memory

Assets:
Cache SRAM, DRAM, Secondary storage

Threats:
- Untrusted device access to DRAM through DMA
- Write to internal cache SRAM
- Corrupt secondary storage



Threat Analysis: Complexity

- Model perspective
  - Attacker-centric, starts with attacker motivation and ability
  - Asset-centric, starts with assets
  - System-centric, starts with system execution flow
- Entry point identification

How many entry points are there:
  - Debug feature? Crypto keys? Display ports?

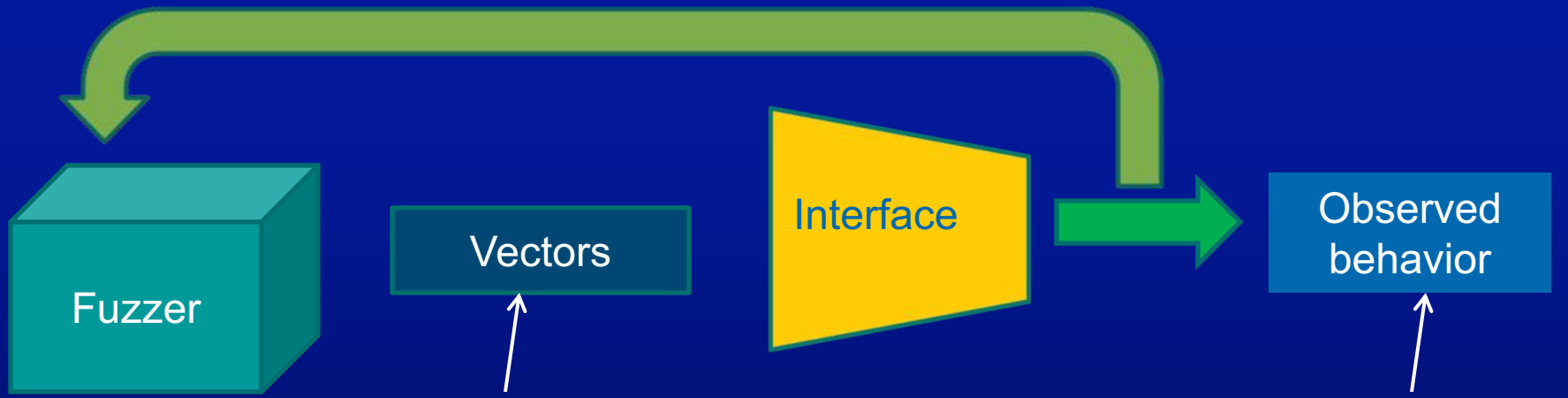
- Risk Assessment
  - **D**amage Potential
  - **R**eproducibility
  - **E**xploitabiliy
  - **A**ffected Systems
  - **D**iscoverability

Validation Activities Summary

- Fuzzing
  - Determine behavior on unexpected inputs
- Penetration Testing
  - Focused tests to exploit vulnerabilities
- Formal Verification
  - Using static/formal techniques
- Hack-a-thon
  - Concerted hacking by a group of experts

Fuzzing

- Buffer overflows
- Integer overflows
- Access violations
- Unhandled exceptions
- DoS
- Race/Timing errors
- Memory leaks
- …



- Long strings
- Symbols
- Bit flipping
- Large integers

- System hang
- Crash
- …

Fuzzing Approaches

Dumb/Mutated

- Random inputs or randomly mutated valid input, throw any and all at it
- Easy and fast to apply

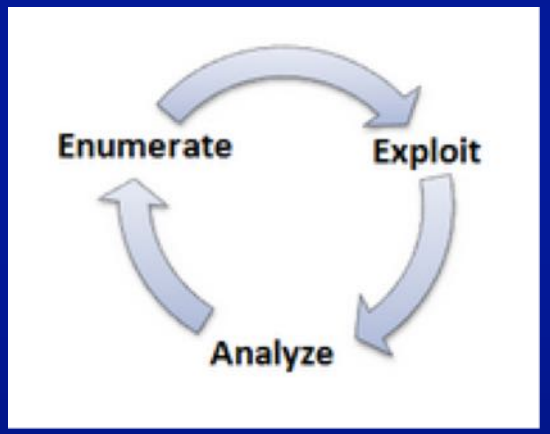
Smart/Generated

- Generated anomalies based on comprehensive knowledge of target
- Significant up front investment
- Greater coverage

Penetration Testing

A penetration test is an attack on a computer system with the intention to find security weakness, potentially gaining access to it, its functionality, and data

- Enumerate the attack surface
- Exploit/perform the attack
- Analyze results against objectives



Penetration Testing: Vulnerability Detection

"Easy":

- Documentation Review
- Known vulnerability
- Missing patches
- Out-of-date software
- Known Misconfigurations

Moderate

- Related Misconfigurations
- Related Vulnerabilities
- Tool Smorgasbord

Hard

- Component Analysis
- Vulnerability Classes
- Platform Horizontal/Integration Testing
- Vulnerability Research

Formal Verification

Use of mathematical and symbolic methods to formally prove a desired property of the system

Heavy-weight

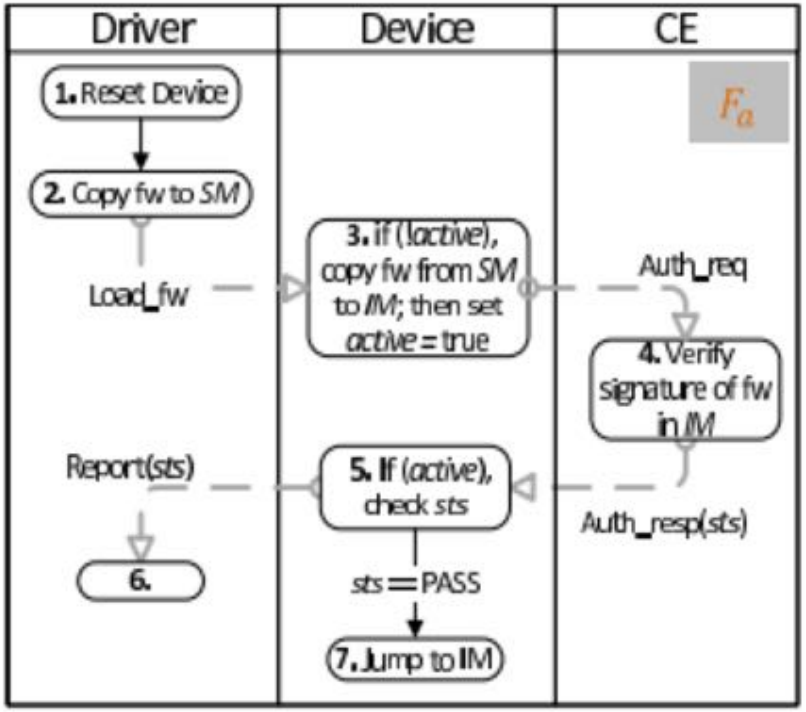
Full verification of critical modules, e.g., cryptographic core

Light-weight

Using static analysis methods to exercise different system paths and expose vulnerabilities

Formal Verification: Case Study

Krstic *et al., 2015*



Is the firmware that is finally loaded always the authenticated firmware?

No

- A counterexample requires 3 concurrently running flows with the right interleaving
- Difficult to achieve through penetration testing or fuzzing

Security and Debug Challenges

Issue at Hand

- Security is critical for modern computing systems
- **But also critical is our ability to validate and debug these systems**

Security and validation/debug requirements are, for the most part, not in conflict

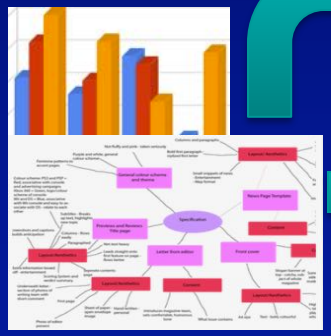
Big Exception: Post-silicon Validation

Basis of the Conflict

- Security requires protection of assets and secrets from indiscriminate/unauthorized access
- Post-silicon validation and debug require observability and controllability of internal signals of the design

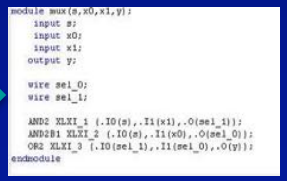
- **Can we satisfy both these requirements?**
- **If there is a trade-off, how can we resolve it?**
- **How serious is the problem?**
- **What do we do today? Why is it inadequate?**
- **What factors should a solution consider?**

What's in Validation?



SW/FW

HW



Tape-out

1st Silicon

PRQ

Exploration | Planning | Development | Production

Pre-silicon

Post silicon

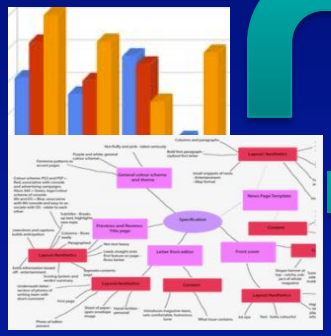
Survivability

Code Reviews
Simulation and Testing
Hardware Acceleration / FPGA
Formal Analysis

\+ High observability
\+ High control

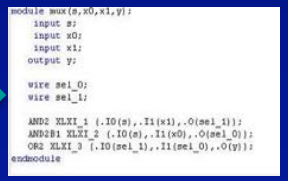
\- Inaccurate physical models
\- Poor tool scalability

What's in Validation?



SW/FW

HW



Tape-out

1st Silicon

PRQ

Exploration | Planning | Development | Production

Pre-silicon

Post silicon

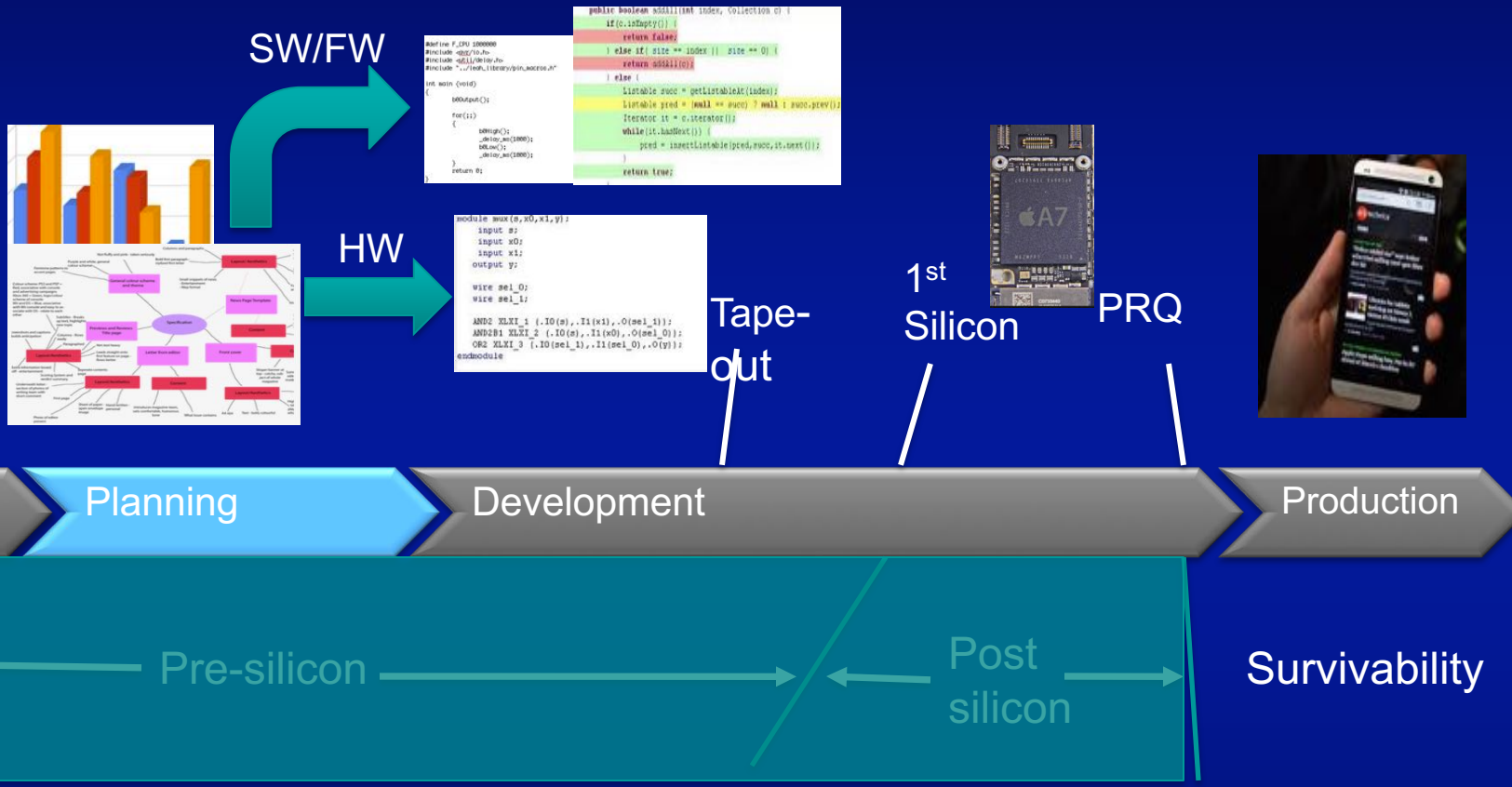
Survivability

Focused Logic Tests
Large Software Applications
Electrical Parameters
Physical Stress Conditions

+ High execution speed
+ Exploration of deep states
+ Accurate physical behavior

– Limited observability and control
– Error sequentiality
– Noise

What's in Validation?



Firmware/software patches
Reduce features

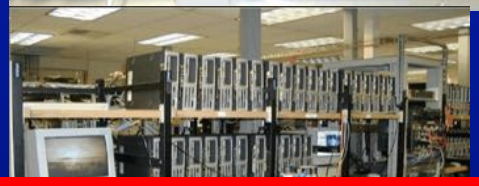
- Very limited observability/control
- Limited defeature bits
- Time

What is Post-silicon Validation?

Ensure that (pre-production) chip behaves as expected

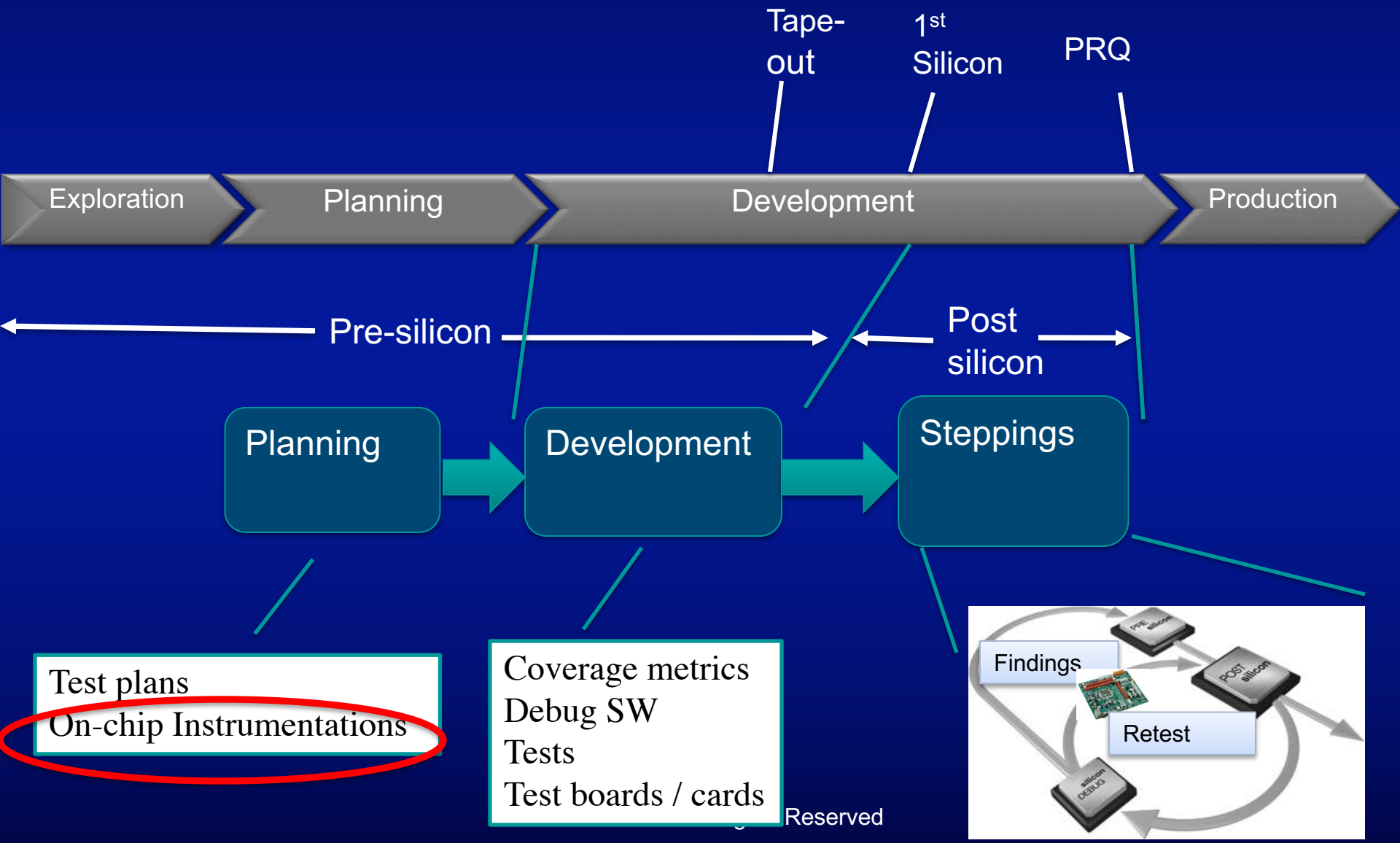
- With real silicon
- Typically running actual software and application

- **System Validation (SV)**
  Logic correctness (using specialized synthetic tests)
- **Compatibility Validation (CV)**
  Compatibility with apps, OS, add-ons, etc.
- **Electrical Validation (EV)**



- **Performed under aggressive schedule**
- **Requires expensive, elaborate setup**
- **Requires significant up-front planning**

Post-silicon Validation Flow



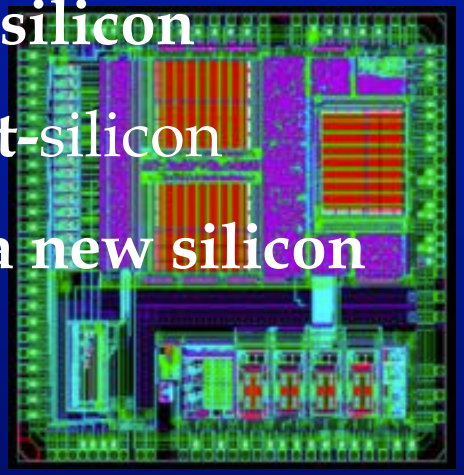
On-Chip Instrumentation

- To validate/debug a design we need to observe updates to different signal values during execution
- To observe a signal [illegible] design to funnel its value to an obser[illegible]

Millions of multi gigahertz internal wires 120K block-to-block wires

- We can only select a very small number of signals
- Observable signals must be **determined pre-silicon**
- Observability problems are **experienced post-silicon**
- **Fixing inadequate observability requires a new silicon stepping (and is often impossible)**

Approx. 100 external wires



On-Chip Instrumentation Practice

Custom Instrumentation

- Used for critical properties
- Currently purely manual, based on designer intuition

Generic Instrumentation Architectures

- Too many to enumerate or even comprehend fully
  - **State observability**, e.g., scan, memory dump
  - **Trace observability,** e.g., for events, messages
  - **Access infrastructure,** e.g., JTAG, microcode patch
  - **Trigger,** e.g., hardware assertions
  - **Programmable microcontrollers**
  - **Misc.,** e.g., defeatures, coverage monitors, counters

Some Consequences

XBOX 360 JTAG Hack

- **Use JTAG port to write to firmware memory**
- **Update firmware**
  - New FW permits unauthorized code execution
- **A common approach to attack SoC**
  - Similar approaches to jailbreak smartphones

Solution is NOT to disable update via JTAG

Why is it not just a security Policy?

Firmware upgrade policy: Restrict type of firmware upgrade permissible through JTAG

Not much different from any other security policy!!

- **Ambiguity**
  - Observability requirements are rarely clear-cut
- **Feed-through**
  - Routing signals through high-security modules
- **Lack of centralization**

State of the Practice

Primarily based on human creativity, although there is some method to the madness

- Start with high instrumentation and little security during design and early validation phase

- **Highly complex design requirement**
- **Does not account for**
  - 3rd party IPs with assets protected from validators
  - Rogue IPs
  - …

Considerations for Solution

- **HVM Challenges**
  - Reuse same test patterns, provide simple access, high coverage
- **Validation Challenges**
  - Support efficient functional debug while protecting secrets
- **Reusability Considerations**
  - Across different types of assets and usage models
- **Control logic Issues**
  - Centralized, easy to follow, validate, check for leaks, etc.
- **Observability Considerations**
  - Self-securing, reduce/eliminate temporal attacks
- **Variability Challenges**
  - Robust against late changes

Security in IoT

The Internet of Things Regime

The IoT Regime is the **point of time** when IoT involves a **connected network of physical objects** or "things" embedded with electronics, software, and sensors, to enable it to **behave smartly** and achieve greater value.

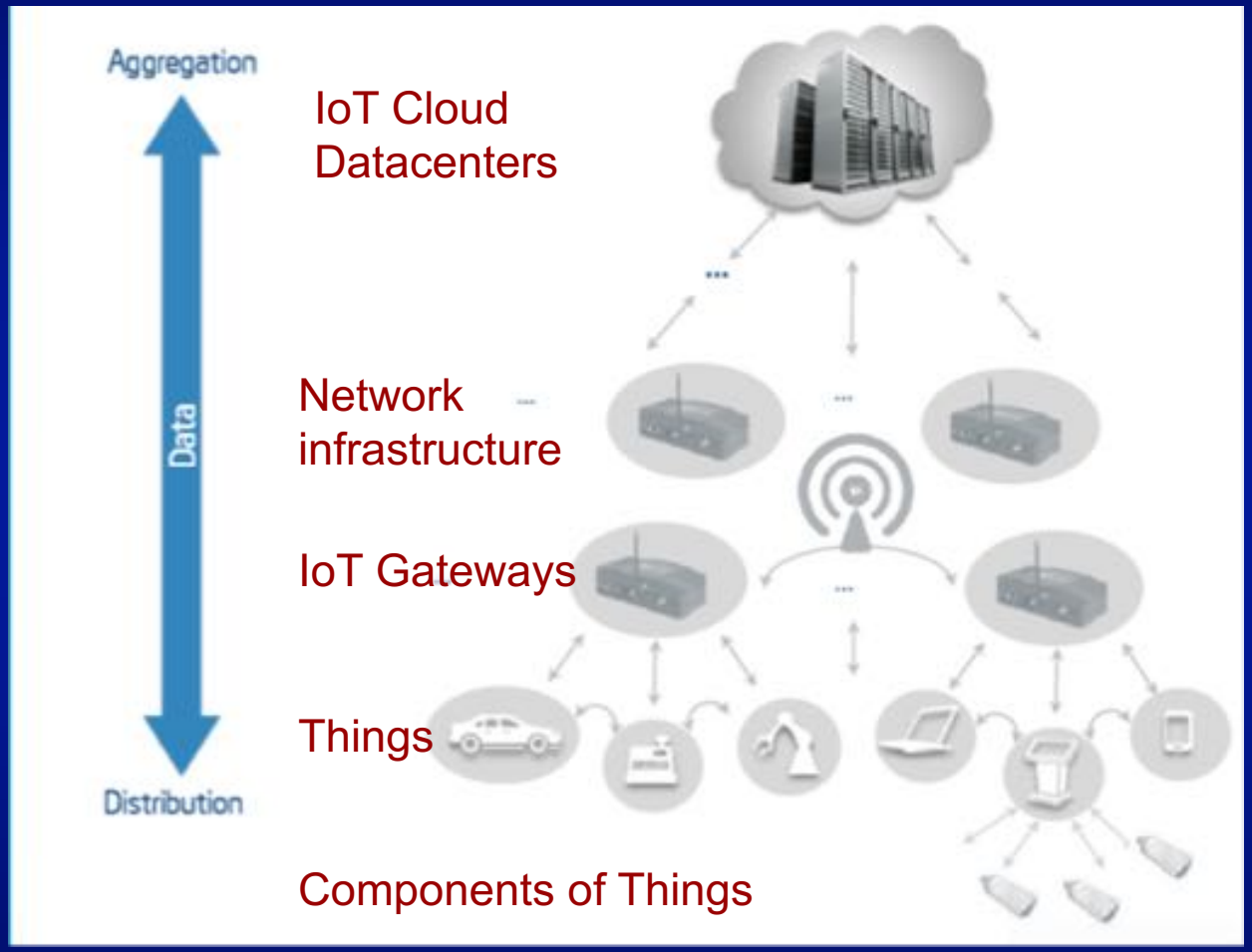
--- Wikipedia

The number of connected computing devices exceeds the human population.

--- Cisco White Paper 2011

Source: Intel

E2E View of IoT



Enterprise Cloud, DC

ISP Routers, Firewalls

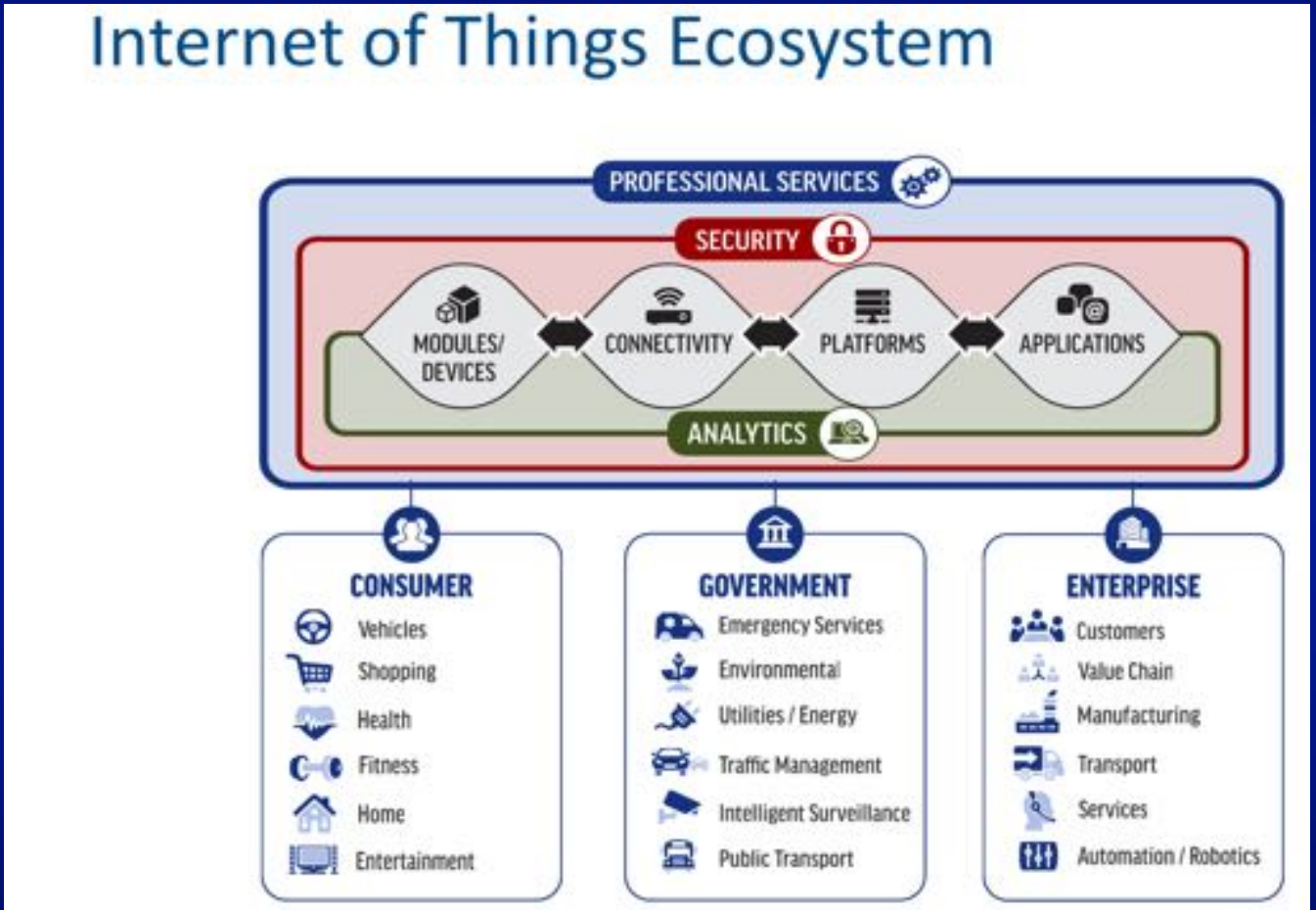
Factory, Building, Car

HVAC, Fitbit

Sensors
Active/Passive Tags

Too many configurations of sensors, devices, and gateways connected and delivering data

An All-Encompassing Ecosystem



Nothing in computing (or anywhere else) left behind

IDC Analyze the Future

A Disparate Stakeholder Pool

Type of Developer	Considerations
Hobbyists; Students; Makers	• Low Cost solutions; open development environment • Dev Support
Entrepreneurs; Start-ups	• Accelerated TTM • Build Cost • Support/ Maintainability • Productization support
Traditional OEMs	• Interoperability/Connectivity with existing solutions.
Existing Producers	• Interoperability/Connectivity with existing solutions • Ability to connect with devices outside closed ecosystem

The Security Story

Cloud increases data exposure to security threats

Internet of Things magnifies the amount of data and diversity of data collection sites

Walden C. Rhines, Mentor Graphics, 2016

Some Unique Features of IoT

- Long, complex life cycle
- Mass produced in same configuration
- Equipment never intended to be connected
- Machine-to-machine
- Normal C-I-A often reversed
- Requires holistic view of device to gateway to cloud and the communications between them
- Many traditional protection mechanisms not applicable due to form factor, deployment, power constraints

Challenge from Long Life: Post-Quantum Crypto

Problem:

- Quantum computers may come into existence between 2030 and 2050
- They will
  - break RSA and ECC
  - find n-bit AES keys and pre-image of n-bit hash in time $2^{n/2}$
- Designs that depend on crypto are at risk

IoT systems that are expected to survive till 2030 and beyond should account for this risk

Why is this hard?

There are of course known ways to address the crypto issue

Algorithm	Attack	Ops on Classical Computer	Ops on Quantum Computer
AES – 256 bit key	Cryptanalysis	2^{256}	2^{128}
SHA2 or SHA3 – 384 bit hash	Find Pre-image	2^{384}	2^{192}
	Find Collision	2^{192}	2^{128} with 2^{128} RAM

- Replace all use of symmetric crypto with AES-256
- Replace all use of cryptographic hash with SHA2 or SHA

For all uses of crypto that cannot be modified on field, are these assumptions sufficient?

How do we validate such a requirement?

Configurability Needs

- **Long device life**
  - Policy and implementation upgrades, on-field adaptation
- **Large OEM ecosystem**
  - Flexibility for implementing (and upgrading) policies
- **Product/platform diversity**
  - Quick policy configurability for different platforms and form factors and derivatives
- **Dynamically changing user security needs**

- How do we capture such requirements?
- How do we add resilience implementing such requirements?
- How can we validate them, with aggressive TTM schedule?

Infrastructure IP for Security: A Scalable Solution for Secure SoC

Background: Infrastructure IP

- System-on-chip (SoC) design using reusable IP blocks is a prevalent practice!
- Most IPs are functional
  - CPU, memory, DSP, crypto, comm., analog, ……

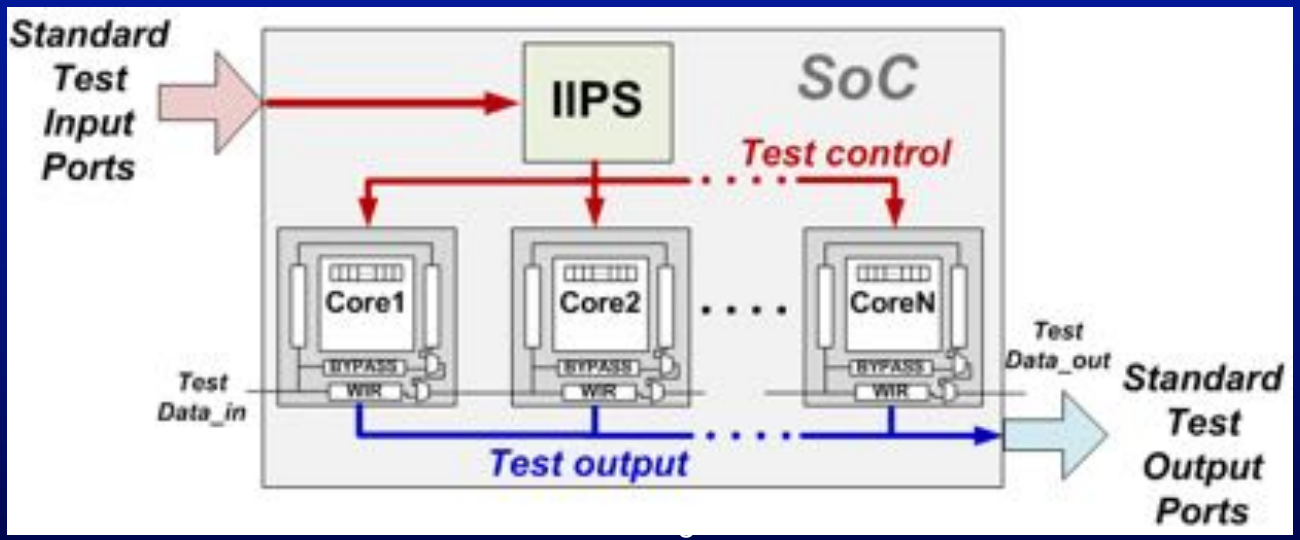
Infrastructure IP – *A special class of non-functional IPs incorporated during SoC integration to facilitate test/debug/verification!*

Infrastructure IP for Security (IIPS)* – A *non-functional IP that interfaces with existing IPs in an SoC to implement hardware security features!*

* *Wang, Basak, & Bhunia, IEEE Tcomp, 2015*

Infrastructure IP for SoC Security (IIPS)

- IIPS contains multiple security primitives to provide various security protections for an SoC
- Features of IIPS
  1. *Ease of integration; plug-n-play using IEEE 1500 Standard*
  2. *Centralized*
  3. *Minimal performance/power/area overhead*
  4. *Functionally scalable and flexible*
  5. *Configurable*
  6. *Does not affect IP level design & IP integration in SoC*
  7. *Can be merged with other (e.g. test) infrastructure IPs*



IIPS: Summary

Flexibility:

- Flexible in interfacing with enhanced configurations of IEEE 1500 arch.
- Effectiveness can be improved with advanced features of IEEE 1500
  - ScanPUF & Trojan detection can be adapted to use the parallel interface
  - Broadside capture can be applied for detecting Trojans in system buses

Functional Scalability:

- Can provide protection against other attacks / other protection schemes
- Can be configured at both design & run time
- Can be integrated with test support logic to reduce overhead & effort

Configurability / Sunthesizability:

1. Configurable IIPS IP block
2. Amenable for automatic synthesis of IIPS in a SoC design

Using IIPS for Security Policy Enforcement

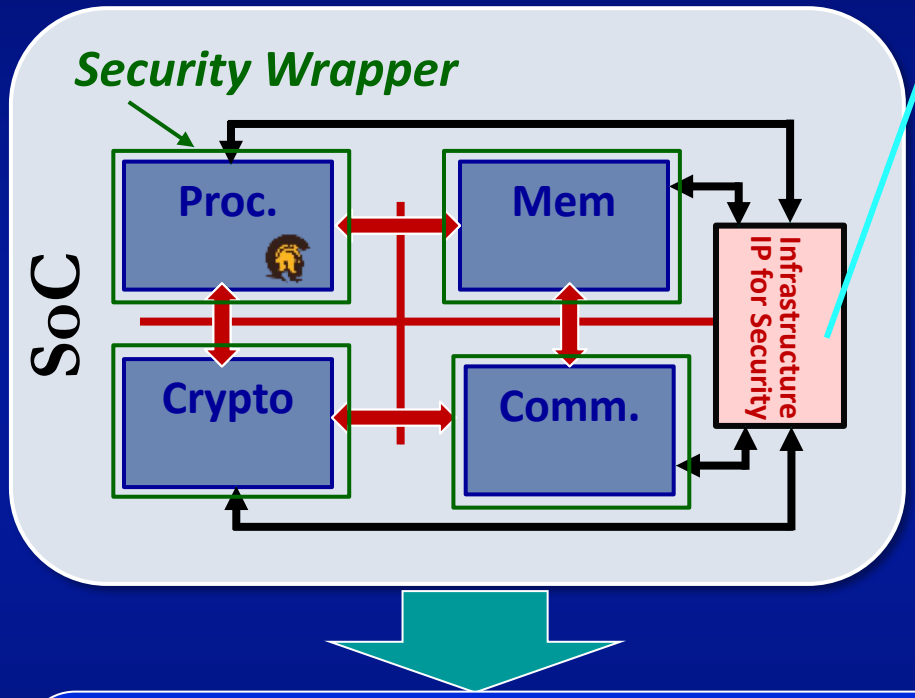
Basak et al., DAC 2015
Basak et al. ICCAD 2015
Basak et al., IEEE TIFS 2017
Ray et al., IEEE Spectrum 2018
Ray at al., PIEEE, 2018

E-IIPS: Extended IIPS

- *Security assets in SoC spread across different IP blocks*
- **Assets**: Crypto cores, programmable fuses, DRM keys, firmware, user data etc.
- Access restrictions governed by SoC security policies
- Policies often involve subtle interactions between IPs
- Natural language representation in architecture documents
  - Often ambiguous & complex
  - Often continuously refined during SoC integration
  - *No systematic method*

E-IIPS enables systematic & flexible implementation of diverse SoC security policies!

A systematic way to address diverse SoC security Issues



Centralized, plug-n-play, configurable ***security-brain***

Trust/Security Issues:

- Authentication
- Hardware Trojan
- Scan based attacks
- Side-channel attacks
- Diverse ***security policies***

- Reduced design effort/cost!
- Improved security
- Improved debug
- Protection against ***unanticipated*** attacks (*a "hardware patch"*)

- *Wang et al., IEEE Tcomp, 2015*
- *Ray et. al., DAC 2015*
- *Basak et al., ICCAD 2015*
- *Basak et al., TIFS 2017*

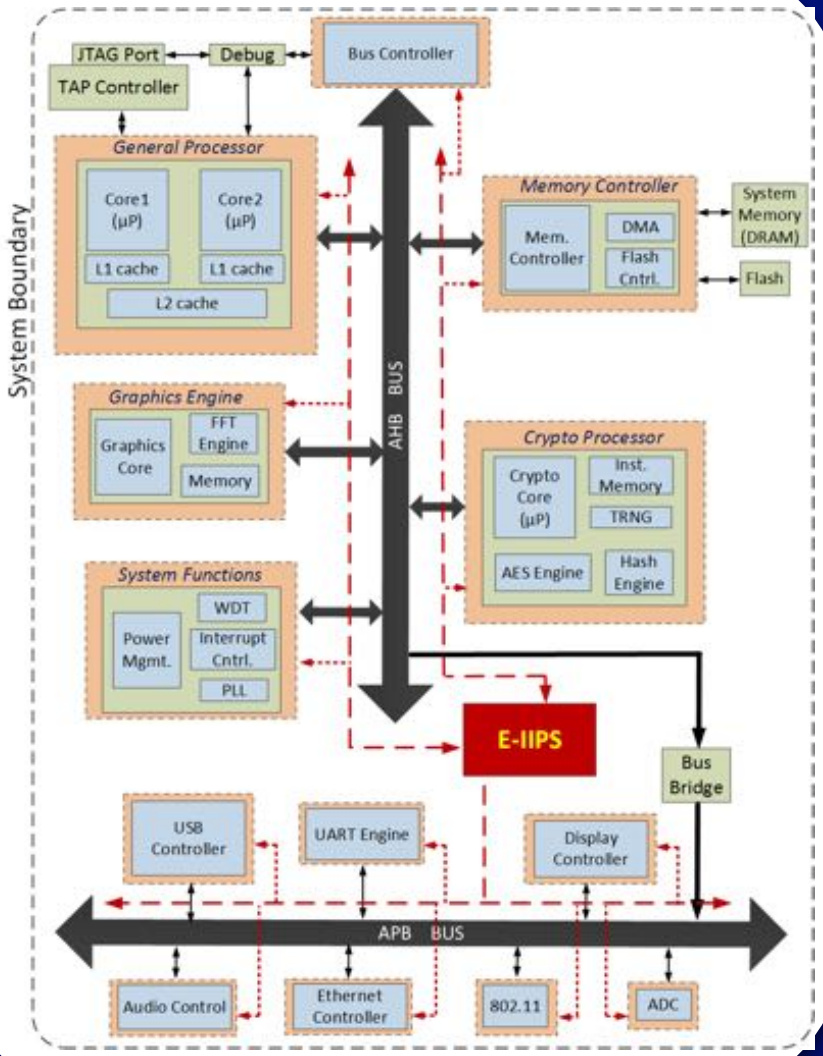
Security Policies

- *Security assets in SoC spread across different IP blocks*
- **Assets**: Crypto cores, programmable fuses, DRM keys, firmware, user data etc.
- Access restrictions governed by SoC security policies
- Policies governing *confidentiality*, *integrity* & *availability* of assets
- Policy Categories: *(1) Access Control; (2) Information Flow; (3) Liveness; and (4) Time-of-Check Time-of-Use (ToC-ToU)*

Ex. 1 – *During boot, data transmitted by crypto-engine cannot be observed by any IP in the SoC other than its intended target (Confidentiality)*

Ex. 2 – *A secure key container can be updated during silicon validation, but not after production (Integrity)*

Proposed Architecture



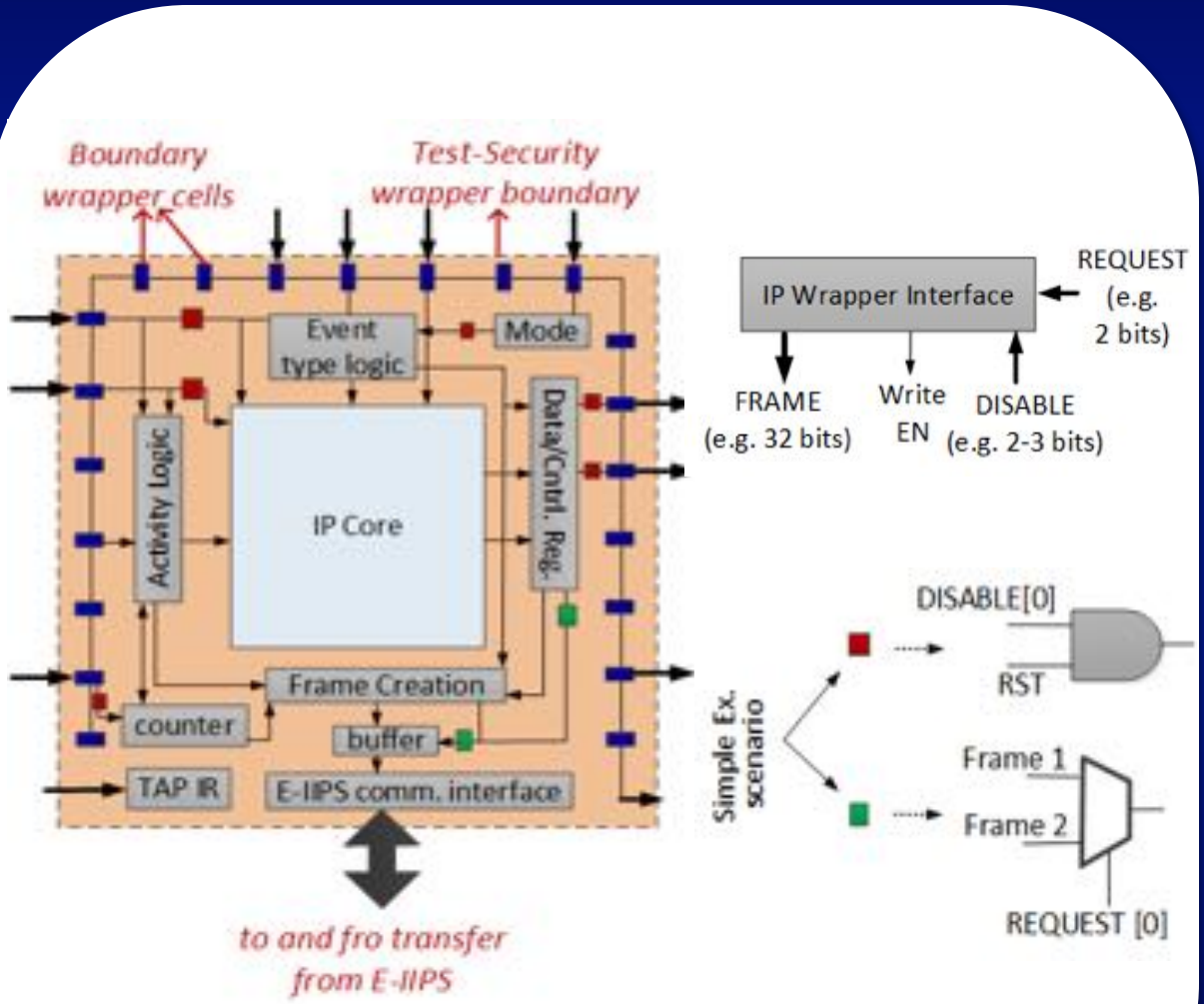
"E-IIPS" (Extended IIPS)

- Implements SoC security policies
- µC based design
- Policies programmed as F/W in secure NVM – easy to patch/ upgrade!
  - *– Authentication during upgrade*

"Security Wrapper" around IPs

- Extension of std. test/debug wrappers
- "*Smart*" – only security relevant events communicated to E-IIPS
  - *– Reduces comm. bottlenecks*

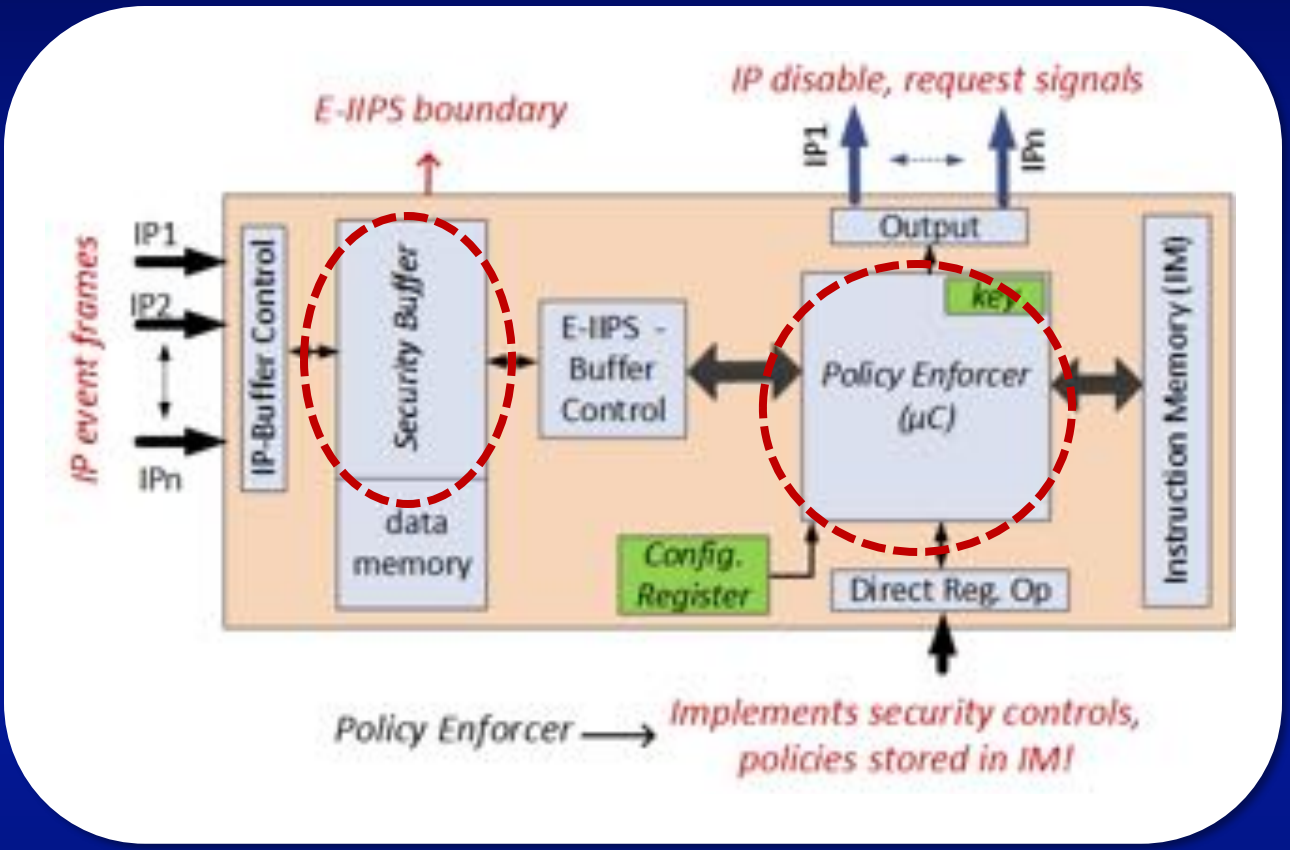
Security Wrappers



Active/ StandBy	Frame length	E_id	E_type	E_metadata

- Wrappers abstract out internal IP details!
- Std. variable length frame based comm. w/ E-IIPS
- *Request & Control* signals from E-IIPS
- E-IIPS configures wrappers at boot time
- Events standardize within IP types e.g. memory, processor, comm. core
- Memory IP: memory/cache controllers etc.
  - **Events:** IP read/write requests, power down
  - **Metadata:** address, DMA channel, burst size

E-IIPS – Security Policy Controller



E-IIPS Functions

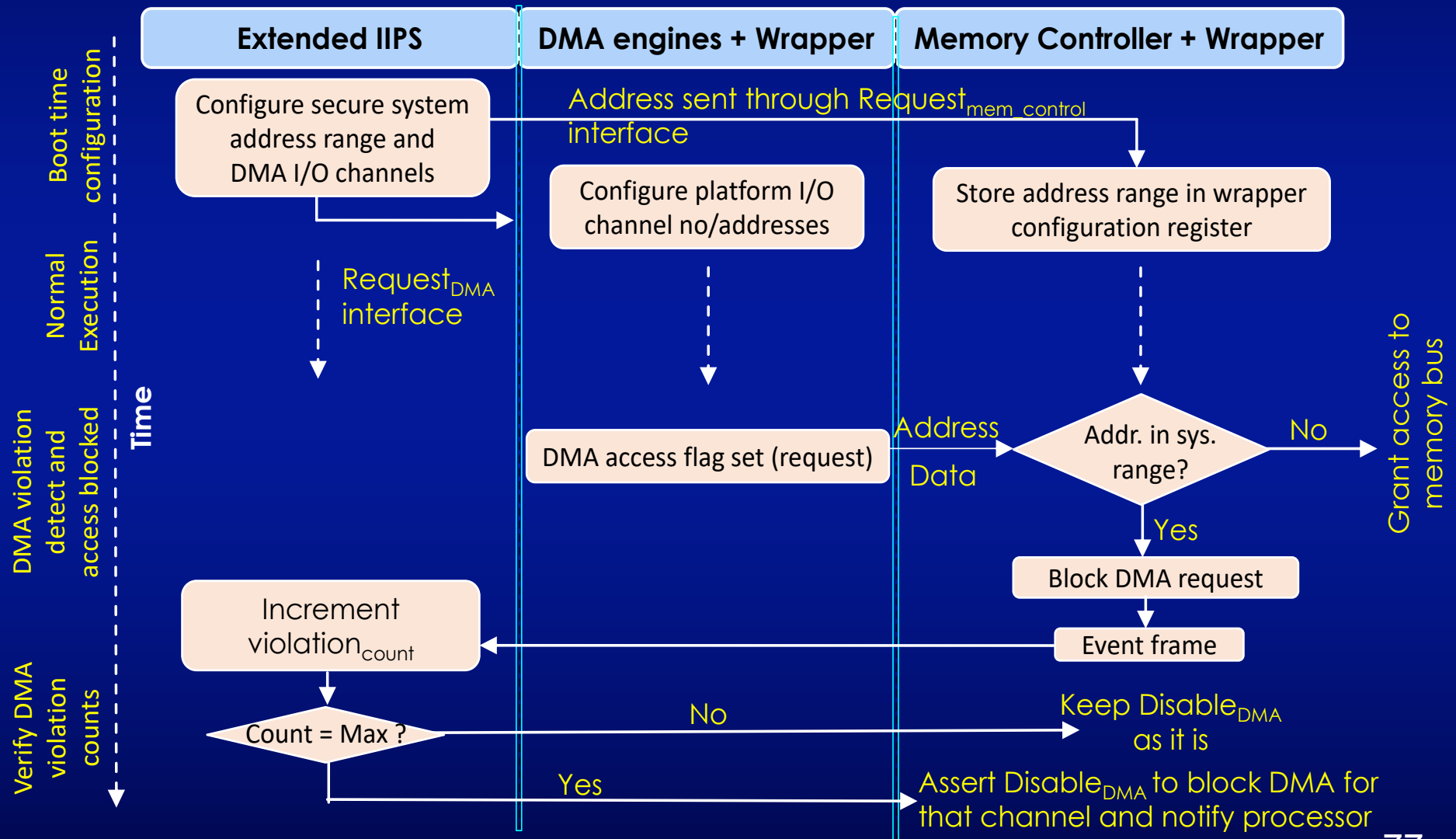
- Analyze events from IP wrappers
- Determine system security state
- Communicate IP specific request & disable

Major Components

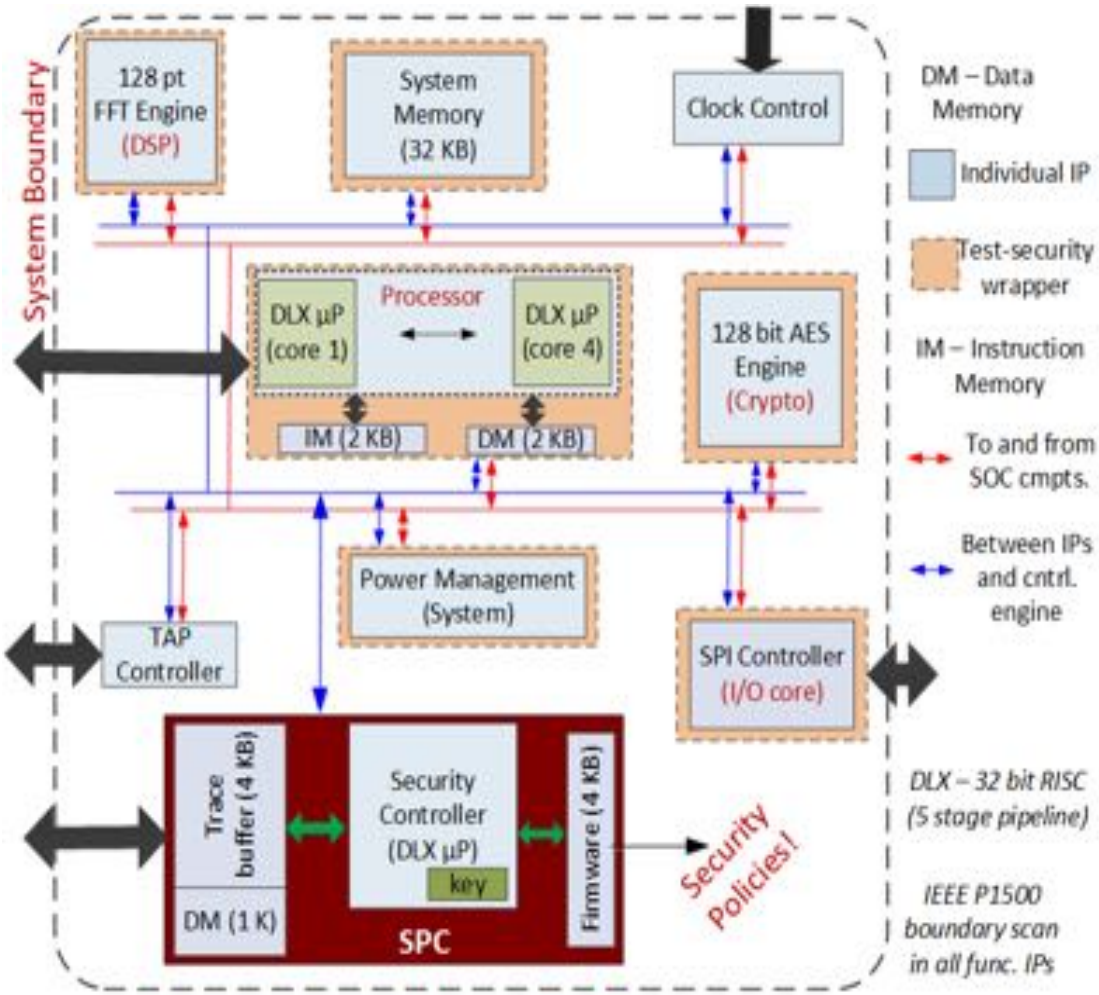
- *Security Buffer* – Storage for IP event logs
- *Policy Enforcer* – Execution engine (μC)
  - Configurability at design time

An Example Mapping

Policy: *DMA accesses prohibited for system specific addresses*



Overhead Analysis



Functional SoC Model

- IP cores (Verilog RTL) from open source
- DLX RISC µP (5 stage), 128 pt. FFT engine, 128-bit AES core, SPI controller core
- IEEE 1500 boundary scan incorporated
- IPs have addresses mapped to memory
- Point-to-point connections
- *Functionally validated with Modelsim*

Results

E-IIPS

- ***Policy Enforcer:*** DLX RISC µP core
- ***Security Buffer:*** 4 KB, 32b frames
- ***Instruction Memory:*** 4 KB
- ***Data Memory:*** 1 KB
- 2 bit req. & disable sig.
- Memory areas from CACTI SRAM models

Security Wrapper

- ***Example Events:*** Memory RD/WR (memory controller), transfer start/stop (SPI) etc.

Wrapper Area Overhead

IP	Org. Area (μm^2)	Wrapper Overhead (%)
AES engine	101620	2.1
SPI controller	3947	9.2
DLX RISC µP	290496	6.8
FFT engine	1810	10.1

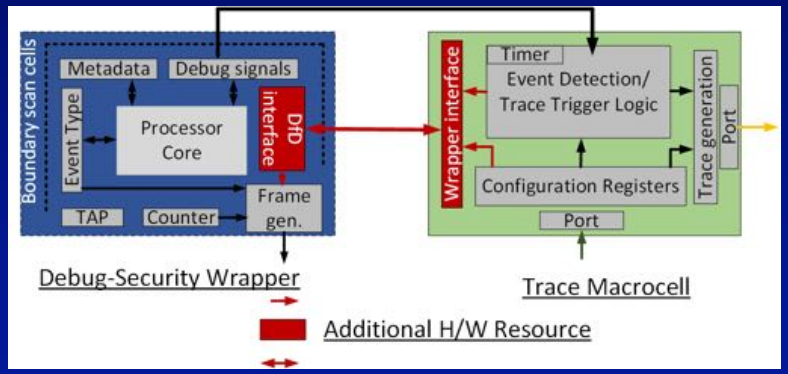
E-IIPS Overhead

SoC	Org. Area (μm^2)	E-IIPS Overhead (%)
Model	13.1×10^6	21.7
Apple A5 (APL2498)	69.6×10^6	4.06
Intel Atom Z2520	41×10^6	7.1

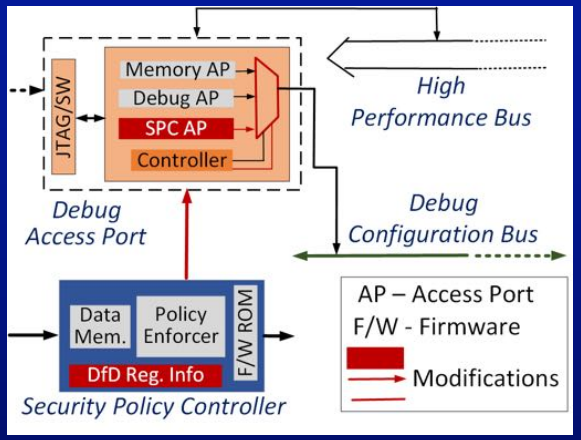
Can on-Chip Debug Architecture Help?

Basak et al., DAC 2016

Details on Security-Debug Integration

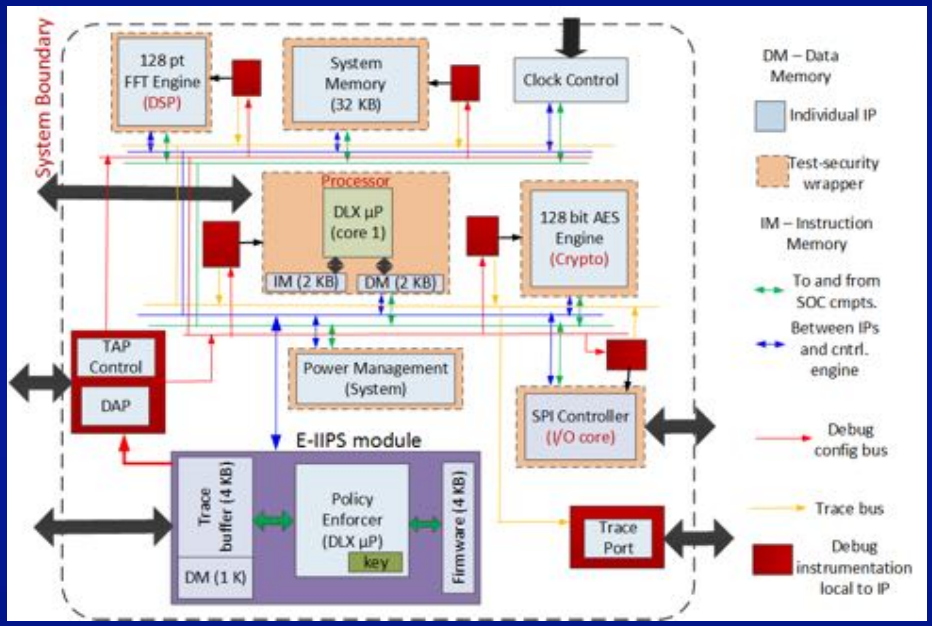


Minimal (typical) H/W overhead in security wrapper to local trace cell interface



Representative Central Policy Controller to DfD Interface

Experimental framework for H/W Overhead/Power Analysis



- Certain use case scenarios of security policy implementation using DfD instrumentation have been analyzed
- At boot, E-IIPS configures these DfD cells
- Debug functionality / usages not hampered

Debug Interface with Security Architecture

Methodology

- Required local DfD configuration register address /values extracted from debug program model
  - SoC designer stores them in E-IIPS
- At boot, E-IIPS configures these DfD cells
- During normal execution, the local DfD detects events of interest and sends them to E-IIPS

DfD Re-purposing Constraints

- Debug functionality / usages not hampered
- System energy / power profiles not significantly affected
- Small H/W overhead for DfD-security interfacing

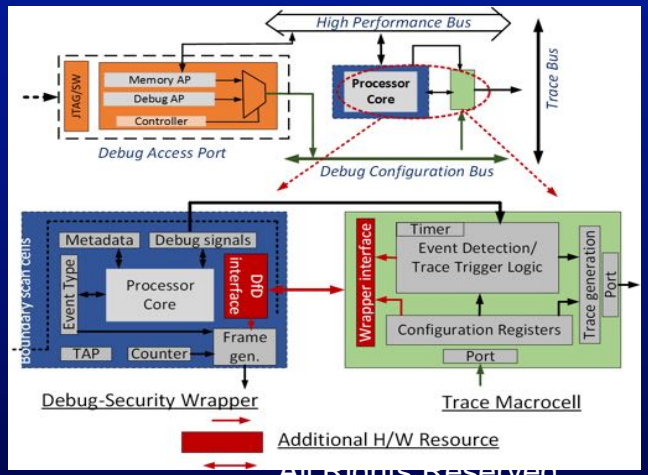
Wrapper-local DfD

- Separate triggered DfD port to wrapper (event transport)
- Configuration register based unique event identifier

SPC-DfD interface

- Addition in Debug Access Port (DAP)
- Configuration link from SPC to DAP

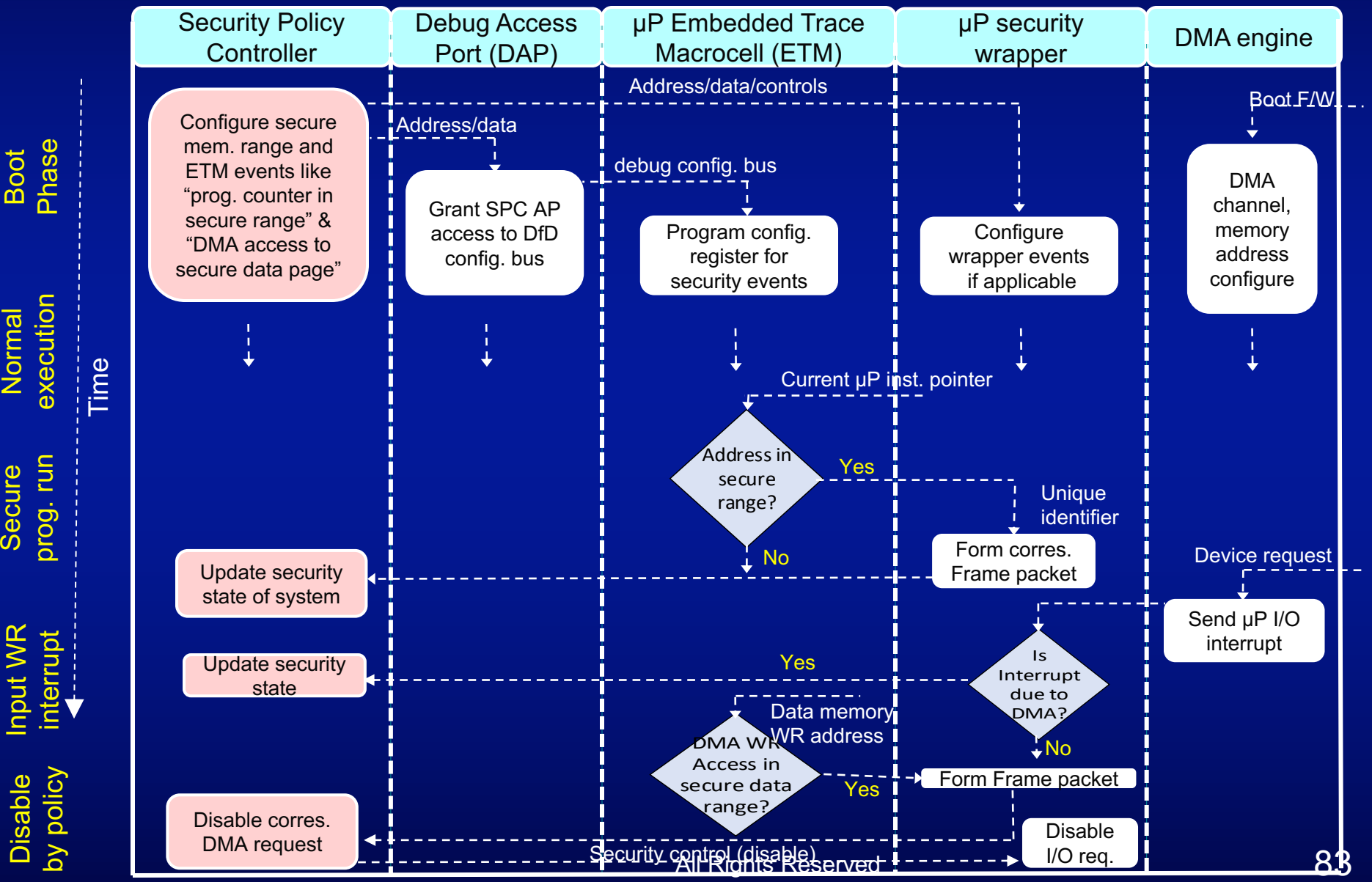
Wrapper to local DfD Comm. Interface



A. Basak et. al, DAC'16

An illustrative Use Case Scenario

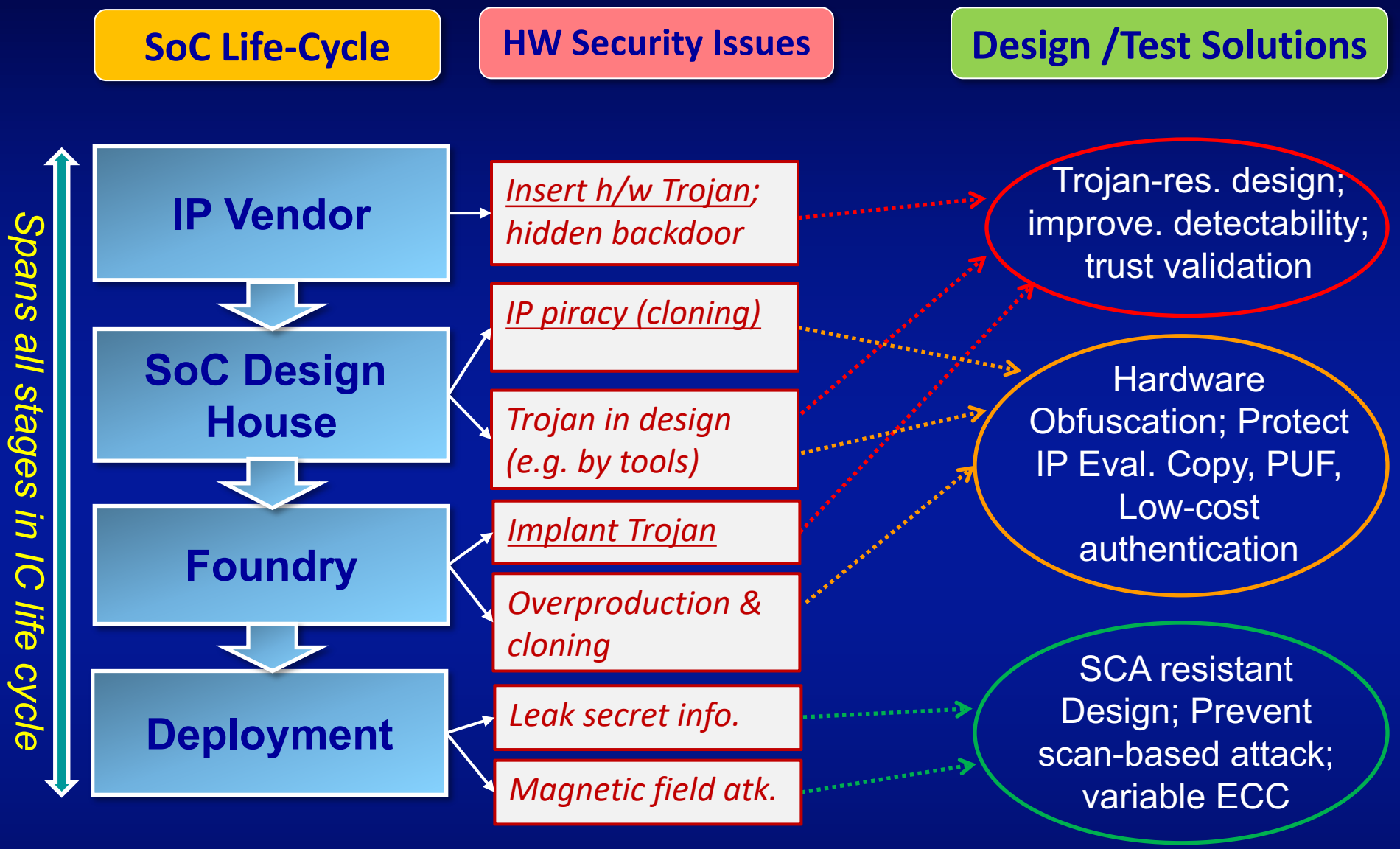
I/O Non-Interference Policy: When CPU is executing in high security mode, I/O devices on SoC platform cannot access protected data memory



The Issue of Untrusted IPs

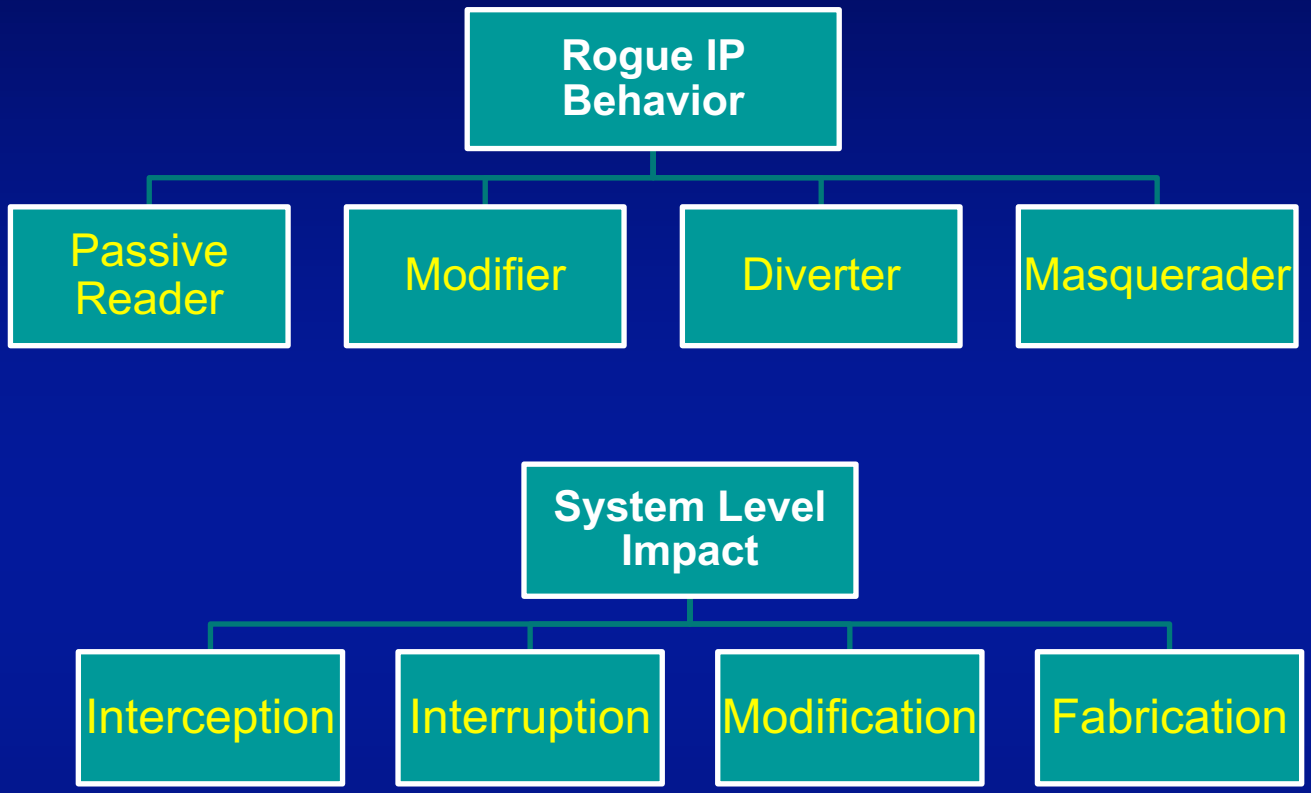
- Trustworthy Computing in SoC with Untrusted Components

Basak et al., IEEE TIFS 2017



DFT 2012, Tcomp 2012, CHES 2009, D&T, 2012, CHES 2011; ASP-DAC 2013, DAC 2013, VTS 2007, DAC 2013, ICCAD 2008, DAC 2014, DAC 2015, TCAD 2009, VTS 2014, VTS 2015, PIEEE 2014, TIFS 2017

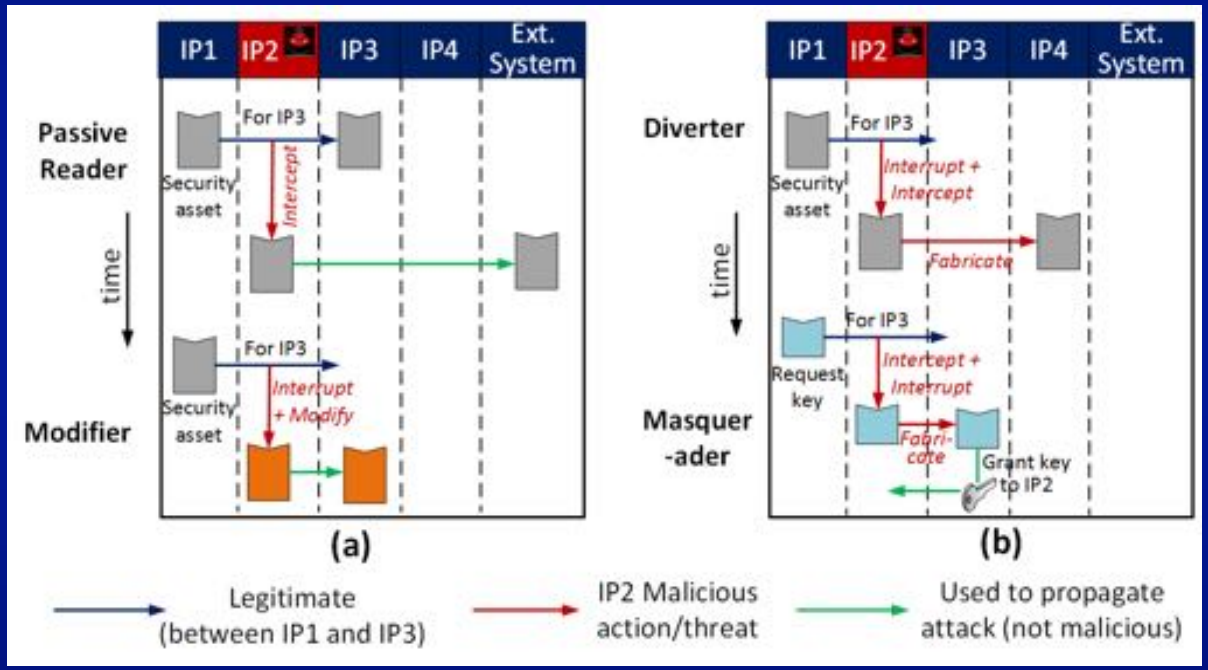
Trust Issues in SoC due to Untrusted IPs



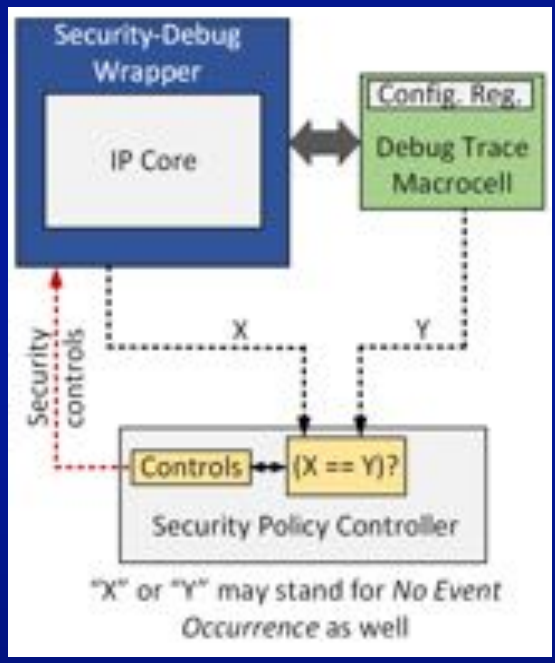
- System level effects of IP level Trojan in SoC – domain of interest!
- Often, visible effects of Trojan only at system level – info. leakage, data corruption or DoS of system
- Cannot be detected with standalone IP trust validation

SoC Security Architecture Resilient to Untrusted IP

- Third-party IPs can have various trust issues.
- How E-IIPS can ensure trust with untrusted IPs?



IP Trust Issues in a SoC

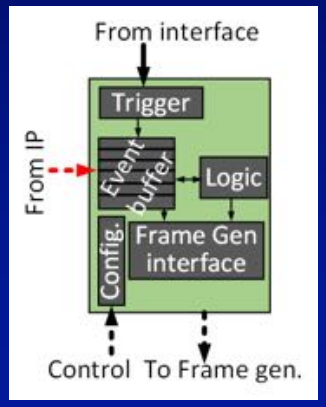


Verify Integrity of Wrapper & Fabric

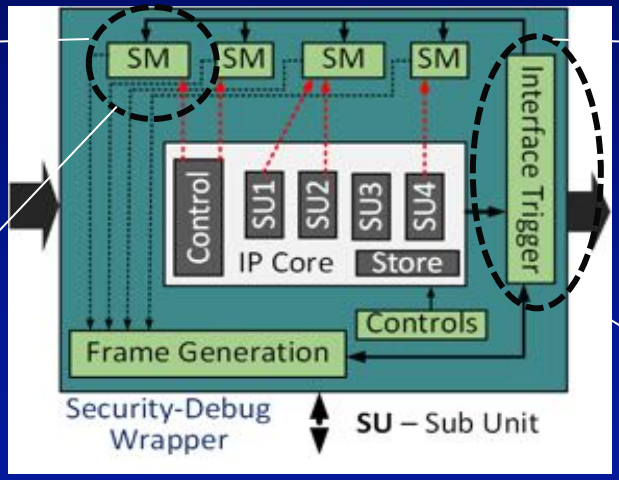
Key Insight: Develop fine-grained, IP-trust aware security policies

Basak et al., IEEE TIFS, 2017

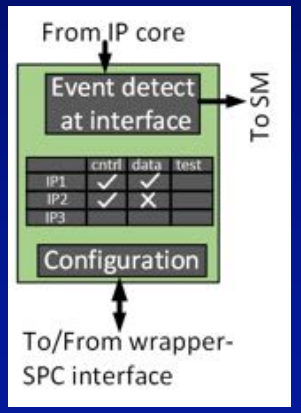
Resilience to Untrusted IP



Micro-architecture of Security Monitor



Enhanced IP-Trust aware security wrapper

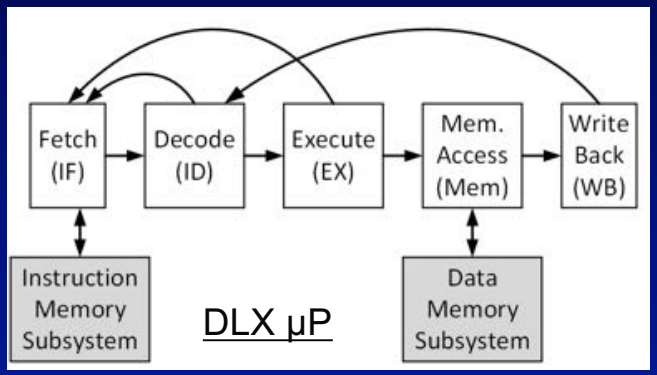


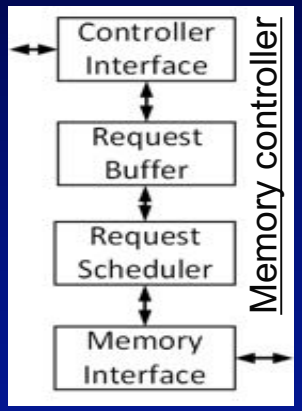
Micro-architecture of Interface Trigger

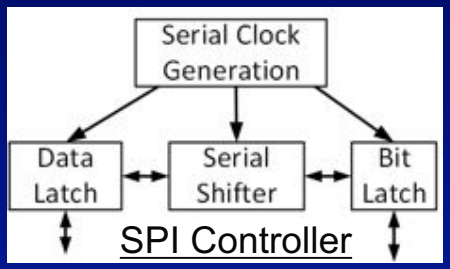
- Security Monitor: Inserted as part of security wrappers to monitor and store recent spatio-temporal events – i.e. the "MCE" (Microarchitecturaly Correlated Events)
  - Triggered to send MCEs to SPC for correlation analysis; Configured by SPC (boot)
  - Inserted by IP provider and/or SoC designer; Can be validated and emulated by local DfD
- Interface Trigger: Detects untrusted IP attempts to communicate with interacting IP/SoC components and triggers the monitors
- IP Trust aware Security Policies in SPC: Decides what and between which MCEs, the correlation checks should be performed

Overhead Analysis

Security Monitors inserted in 3P IPs to analyze H/W overheads with different scenarios of increasing Trojan Coverage







Original Area and Power for DLX µP (at 32 nm) with 1 KB instr, data memory – 352405 µm^2 ; 12.56 mW

Diff. Security Monitor Scenario	Die Area Ovrhead (%)	Power Over-head (%)
Case I (32 b o/p)	6.68	6.92
Case I (256b o/p)	7.17	7.32
Case II	10.44	10.82
Case III	11.68	11.62

Overhead of Monitors for increasing Trojan coverage in µP

Low Overhead; Minimal increase for higher Trojan coverage and output frame width

Original Area and Power for Memory controller and SPI controller IP – 629433 µm^2 , 13.81 mW ; 5456 µm^2 , 0.298 mW

Diff. Monitor Scenario	Area Ovrhead (%)	Power Overhead (%)
Case I	10.77	14.04
Case II	11.16	18.53

Diff. Monitor Scenario	Die Area Ovrhead (%)	Power Overhead (%)
Case I	29.88	19.12
Case II	101.08	66.78

Overhead of Monitors in memory controller and SPI controller

Could be high for small IP cores (like SPI)

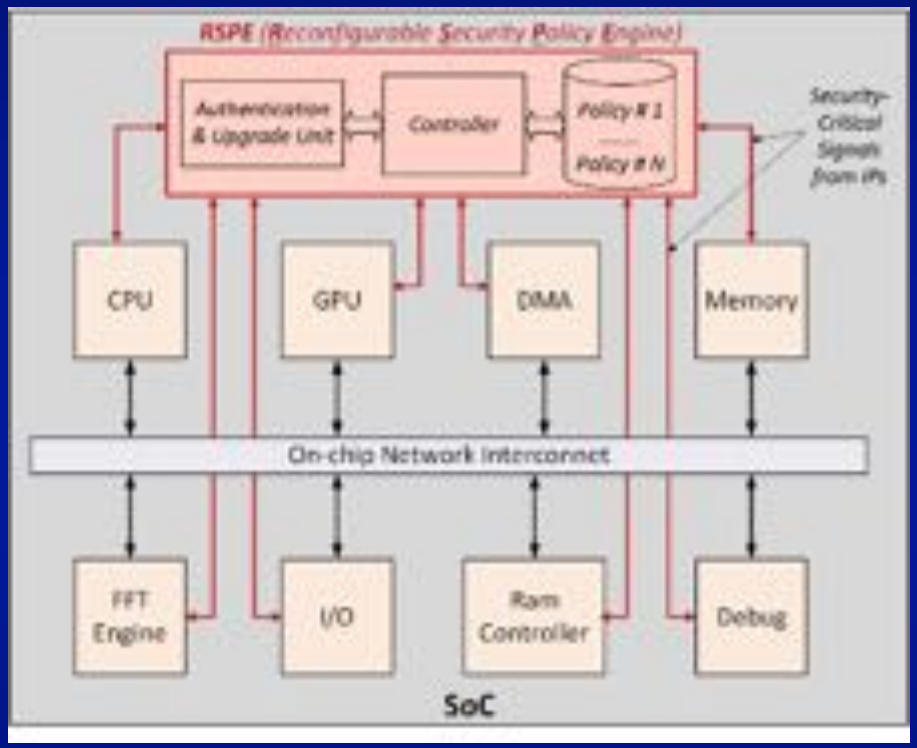
IP Core	OVH(%) in model	OVH(%) in Apple A5	OVH(%) in Intel Atom
Processor	0.31	0.059	0.1
Memory Controller	0.543	0.103	0.175
SPI Controller	0.043	0.008	0.014

Negligible w.r.t. full SoC die area

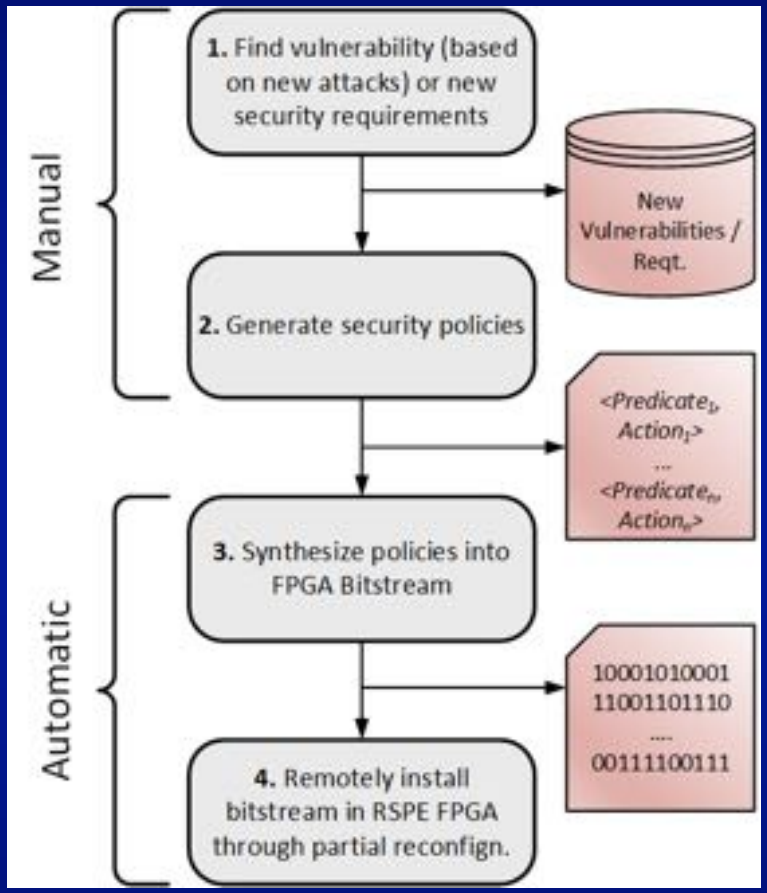
Patchability Analysis

Prasad et al., ASPDAC 2018

Hardware Patch



Overall architecture



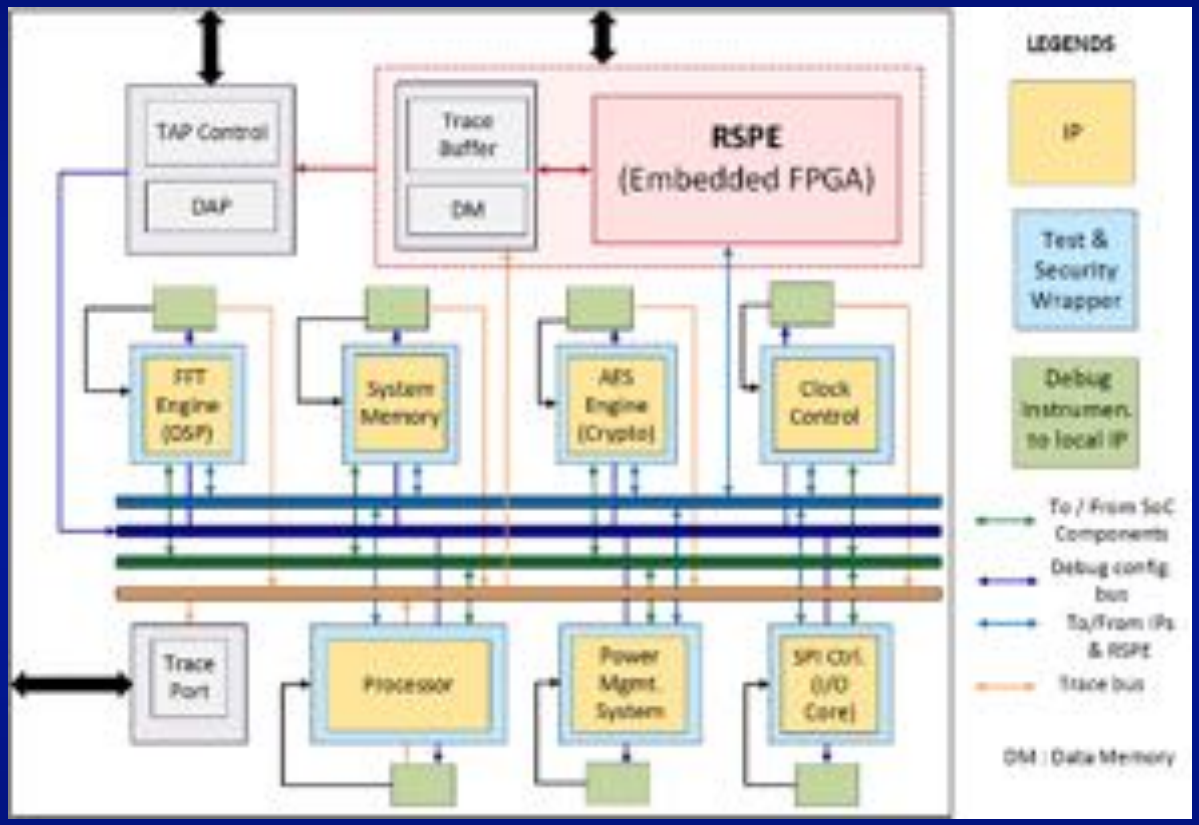
Software Flow

Debnath et al., ASPDAC 2018, Under Review

Hardware Patch

What enables patching?

- An upgradable security policy engine
- Access to (all) security-critical events
  - Interface w/ on-chip debug infrastructure
  - Interface with security wrapper
- Remote authentication & upgrade install hardware (inside RSPE)



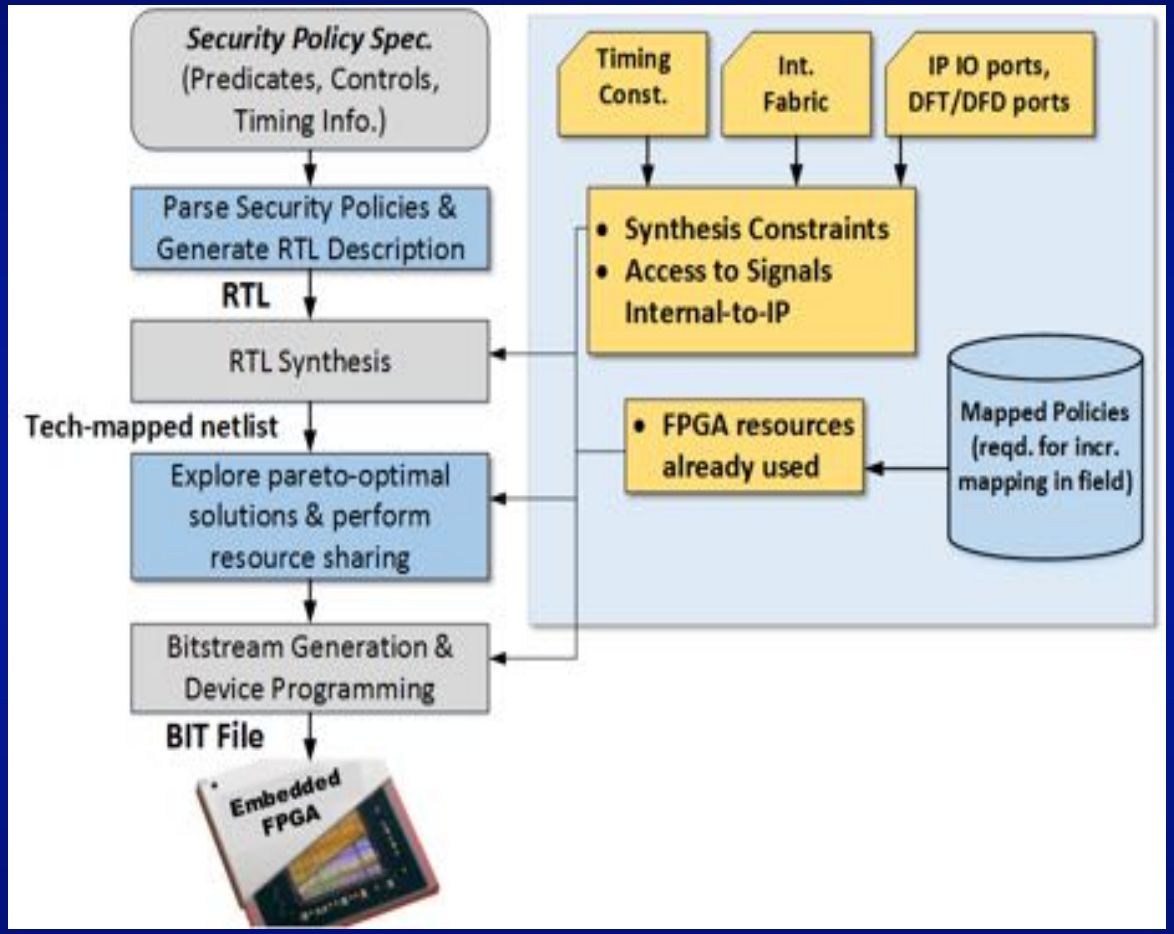
Debnath et al., ASPDAC 2018, to appear

CAD Framework

- Systematic approach to synthesize policies into FPGA based RSPE

Key Features:

- Amenable for automatic synthesis of arbitrary policies
- 3-tuple format: <timing, predicate, action>



Mapping Diverse Security Policies on Embedded FPGA-based RSPE

Representative SoC Security Policies

Policy #	Predicate Part	Action Part	IPs Involved
1	*User mode & (Mem RD/WR Req. by User — Mem RD/WR Req. by IP A — ...)*	*RD/WR Addr. within specified range*	DLX µP & any other IP with access to system memory
2	*Supervisor mode & (Mem RD Req. by User — Mem RD/WR Req. by IP A — ...)*	*RD Addr. within shared memory range & No WR*	DLX µP & any other IP with access to system memory
3	*Debug mode & (Trace cells busy — power mgmt. module busy)*	*No update in power control firmware & no changes in SPI controller Config. Reg*	Power mgmt. module & SPI controller
4	*!(Supervisor mode) & (Inst. Mem Update Req. through test access port or SPI controller)*	*No update of Inst. Mem. allowed*	DLX µP
5	*Active Crypto mode*	*No interrupt or Memory Access Req. from the DLX core or any IP is allowed*	Crypto module, processor and other IPs access to processor

Results Analysis

- **Number of Arbitrary Security Policies**
  - Observable signals : Predicate tuples
  - Controllable signals : Action tuples
  - DfD Integration demonstrates superior performance

Tuple Type	Test Wrappers (Number of policies)	Security Wrappers (Number of policies)	Design-for-debug Infrastructure (Number of policies)
2P, 1A	570	490046760	2987015850
4P, 1A	14535	7.91E+13	1.59E+15
8P, 1A	377910	1.75E+23	3.89E+25
8P, 2A	377910	4.42E+25	1.81E+28

Results Analysis

- **Energy and Latency:**
  - FPGA-based design vs MCU-based Design
  - FPGA-based design:
    - 5.02 times more energy efficient
    - 5.5 times faster

	Die Area (μm^2)	Clock Freq. (MHz)	Cycle Count (10 Policies)	Total Latency (µs)	Dynamic Power (mW)	Static Power (mW)	Total Energy (nJ)
DLX µP	0.724	203	210	1.04	14.27	63.48	80.86
FPGA	1.06	138	26	0.189	64.9	20.43	16.13
Ratio	0.68	1.47	8.07	5.49	0.22	3.11	5.02

Area, Performance, Power, and Energy Values for DLX uP Core and FPGA Based RSPE Module

Summary

- Developed and evaluated a novel infrastructure IP for security!
- Protects against both HW and SW Security Issues
- Developed a novel architecture for **efficient** implementation of SoC security policies
  - Flexible (& *upgradable* in field)
  - Enables systematic implementation
  - Lower overhead
  - Easy-to-debug
  - Minimal impact on the IP blocks (standardized *security wrapper*)

[Recent] **Developed formal verification flows for policies**

Conclusion

"It's good to do something that scares you"

--- Ellen DeGeneres

- Designing and validating security of complex modern embedded systems is a **critical problem**
- Addressing the problem requires strong collaboration among several areas in Computer Science and Engineering
- We have made some progress, but our research has only **scratched the surface** of this vast domain
- The future road in this area is **uncertain**, **exciting**, and **crucial** to our well-being

98

THANKS!

#PowerfulYetSecure

Questions ?

References:

1. A. Basak, S. Ray, and S. Bhunia, "A Flexible Architecture for Systematic Implementation of SoC Security Policies", Intl. Conf. on Computer-Aided Design (ICCAD), 2015
2. T. Fox-Brewster, "Voodoo Hackers: Stealing Secrets from Snowden's Favorite OS Is Easier than You Think, http://www.forbes.com/sites/thomasbrewster/2015/03/18/hacking- tails-with-rootkits
3. A. Basak, S. Bhunia, and S. Ray, "Exploiting Design-for-Debug for Flexible SoC Security Architecture", Design Automation Conference (DAC), 2016.
4. A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security Assurance for System-on-Chip Designs with Untrusted IPs", IEEE Transactions on Information Forensics & Security (TIFS), 2017.
5. A. Debnath et. al., "SoC Security Architecture for Hardware Patch", ASPDAC, 2018.
6. L. Greenemeier, "iPhone Hacks Annoy AT&T but Are Unlikely to Bruise Apple," *Scientific American*, 2007.
7. S. J. Greenwald, "Discussion Topic: What is the Old Security Paradigm," in *Workshop on New Security Paradigms*, 1998, pp. 107–118.
8. C. Kallenberg and X. Kovah, "How Many Million BIOSes Would You Like to Infect?" in *The 15th Annual CanSecWest Conference (CanSecWest 2015)*, 2015.
9. S. Krstic, J. Yang, D. W. Palmer, R. B. Osborne, and E. Talmor, "Security of SoC Firmware Load Protocol," in *IEEE HOST*, 2014.
10. S. Ray, J. Yang, A. Basak, and S. Bhunia, "Correctness and Security at Odds: Post-silicon Validation of Modern SoC Designs", Design Automation Conference (DAC), 2015.
11. S. Ray, T. Hoque, A. Basak, and S. Bhunia, "The power play: Security-energy trade-offs in the IoT regime", ICCD, 2016.
12. S. Ray, M. Tehranipoor, & S. Bhunia, "System-on-Chip Platform Security Assurance, Architecture and Validation", Proceedings of IEEE, 2018