Achieve Complete SoC Memory Map Verification through Efficient Combination of Formal and Simulation Techniques

Clemens Roettgermann,
Peter Limmer, Michael Rohleder

accellera SYSTEMS INITIATIVE

freescale™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Outline

- **Introduction**
  - Motivation
  - Verification challenges on SoC level
- State of the Art & Limitations
- Verification approach
  - Equivalence Classes
  - Recursive hierarchical break-down
- Results & Perspectives

Introduction

- SoCs with increasing **complexity**
  - Number of integrated cores, IP …
  - Communication complexity -> **Multi-master systems**

- **Safety** and **security requirements**
  - Adds to complexity of feature space
  - Underlines requests for **complete verification**

- SoC example:
  - **Lock-step architecture** with embedded flash
  - **Multi-core CPUs** on each lock-step area plus additional cores in peripherals
  - Automotive typical set of **communication controllers** (Ethernet, CAN, FlexRay, LIN)
  - Targeted for **ASIL-D** application
  - SHE compliant **encryption** support (e.g. SECURE_BOOT)
  - **Flash protection** mechanisms
  - **LifeCycle** concept - SoC state to control access to security information
    (e.g. PROD -> CUST_DEL -> OEM_DEL -> IN_FIELD -> FAILURE_ANALYSIS)

Motivation

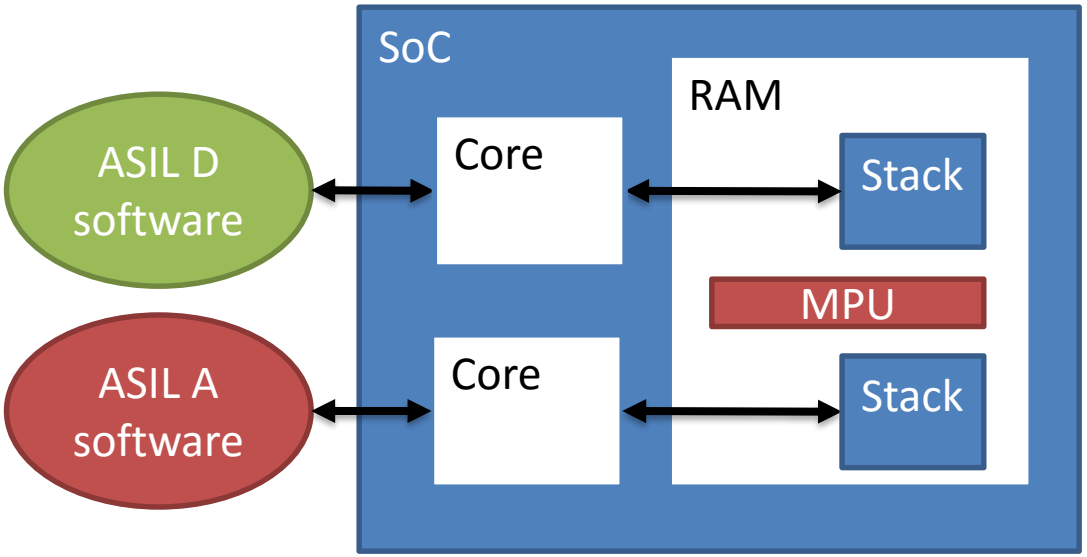
Multi-master:

- Partially different address map for individual masters

  e.g. Processor cores, DMA, Communication Controllers, Encryption Engines

Functional Safety aspects:

- "Freedom from Interferences"
  - ISO26262
  - SW task arguments rely on an address map that is **completely documented and verified**



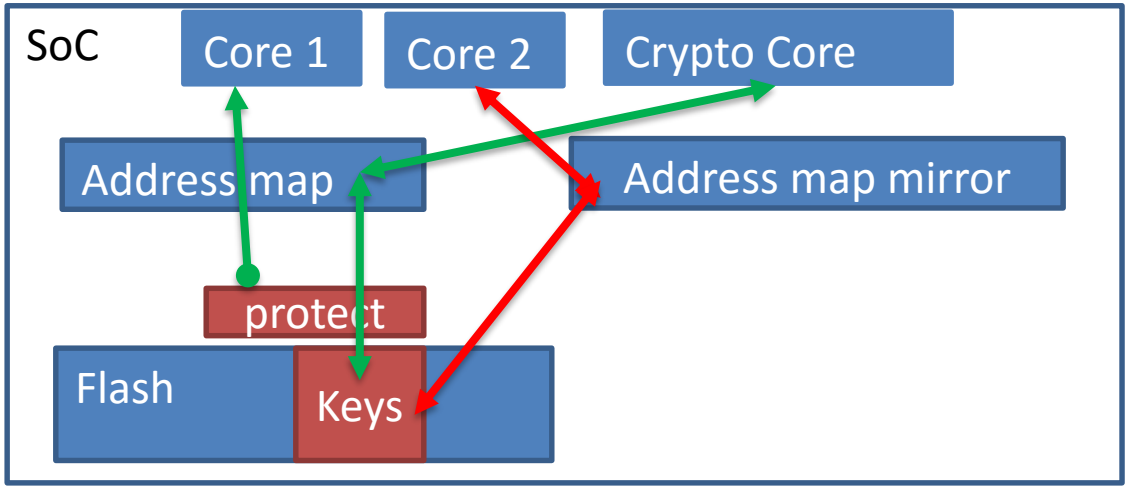
Motivation

Security aspects:

- Address map of security relevant areas dependent on
  - debugger connection, test mode
  - password challenges
  - Security life-cycle state

- Dangers from incomplete address decoding and undocumented registers
  - provide backdoors that can be exploit by attackers
  - Attack method: “Read the Manual”

Example:



Verification Challenges

- Completeness of address map verification is vital for fulfilling safety and security requirements

- Accessibility of a register or memory space
  - depends on multiple different aspects

$$\boldsymbol{accessibility} = F(\ \boldsymbol{address}, \boldsymbol{comMaster}, \boldsymbol{safetyMode}, \boldsymbol{securityMode}\ )$$

  - Verification effort (both, development and execution) multiplies

- Feasibility only by reduction of verification complexity

Outline

- Introduction
  - Motivation
  - Verification challenges on SoC level
- **State of the Art & Limitations**
- Verification approach
  - Equivalence Classes
  - Recursive hierarchical break-down
- Results & Perspectives

State of the Art & Limitations

Simulation-based verification

- Adopted for IP level and SoC level verification
- Benefits:
  - "real life" register access scenario - including effects from different clock domain, execution from core, MPUs, MMUs, lock-step logic
  - Debugging is straightforward
  - SW driven tests can be executed as well on real silicon -> reuse
- Problems:
  - Verification of complete address map not feasible on SoC level -> exploding simulation times
    - e.g. 5-10h/16K address window multiplies with $\#masters * \#safetyModes * \#securityModes$
  - Completeness cannot be achieved
    - side effects from accesses on one registers to other memory areas uncheckable

State of the Art & Limitations

Formal Verification

- Applicability
  - Well adopted for IP level register verification
  - SoC level feasibility suffers from rapid State-space explosion -> requires abstractions and constraining to reduce state space
- Benefits
  - Security and safety modes can be taken into account smoothly by using them as enabling/disabling conditions
  - Applicable for wide memory ranges. CPU effort scales with complexity of logic and not with width of memory range
- Problems:
  - Complete access path from master down to IP register not feasible to verify. The relevant logic and sequential aspects are hard to handle by formal engines
  - The required abstractions and constraints might hide problems

Outline

- Introduction
  - Motivation
  - Verification challenges on SoC level
- State of the Art & Limitations
- **Verification approach**
  - Equivalence Classes
  - Recursive hierarchical break-down
- Results & Perspectives

Systematic Approach

- Situation:
  - Project schedule constrains
  - Requirement to verify the complete address map under different safety and security modes
  - Strengths of simulation and formal methods are complementary

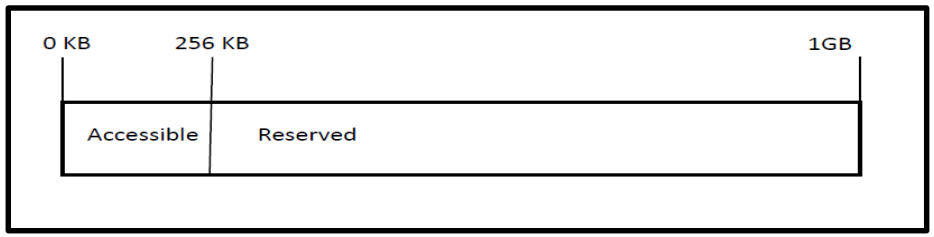
- Approach:

  - Systematic hierarchical partitioning of address map
    - ➔ Method of Equivalence Classes

  - Utilize different verification methods for different verification aspects
    - ➔ Formal Checks vs. Simulation

  - Combine the complementing advantages

Equivalence Class Example



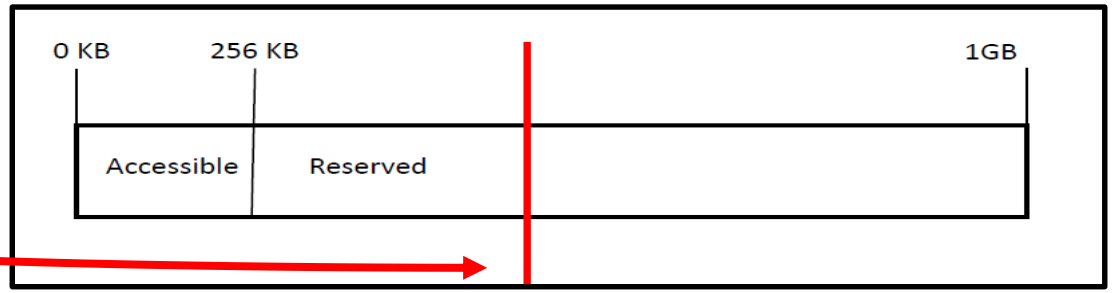
- Assume only two possible behaviors: **[accessible]** OR **[reserved]**
- Verification scope breaks down into just 2 classes - Within each of the classes is the behavior regarded as equivalent (with respect to that aspect)

- However, method not straightforward to apply
  - consider the following bug:

```
if (addr<0x40000):
          accessible
else if (addr==0x4cafe)
          I_AM_THE_BUG
else
          reserved
```

Equivalence Class Usage Approach

```
if (addr<0x40000):
          accessible
else if (addr==0x4cafe)
          I_AM_THE_BUG
else
          reserved
```

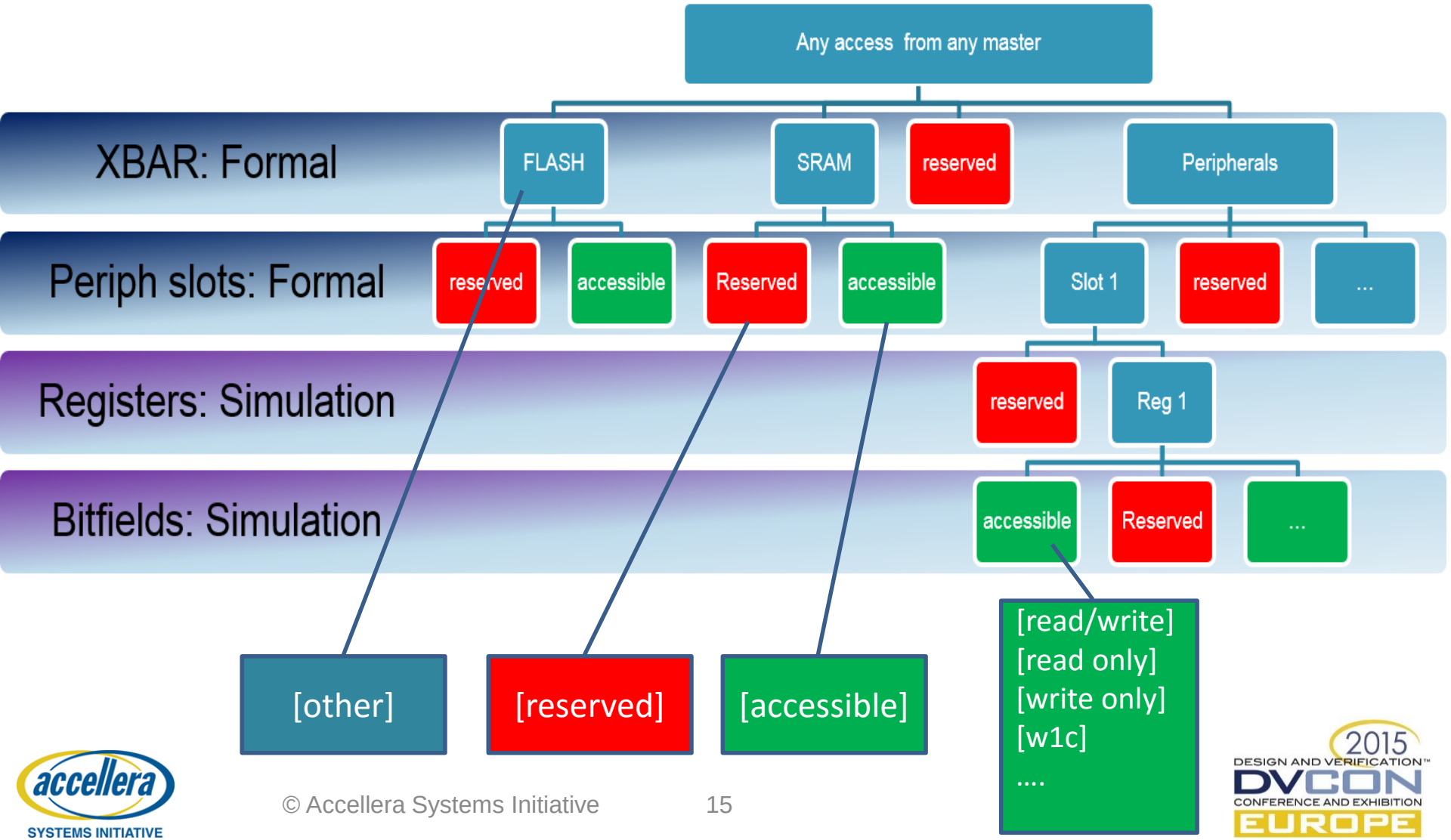


- To definitely find the bug (or verify its non-existence)
  - Simulation:
    - each address needs to be checked
    - back to full complexity **- not feasible**
  - Formal:
    - 2 properties to be defined
    - Formal Proof possible in minutes - **feasible**

Recursive hierarchical partitioning

- Approach to apply equivalence classes for complexity reduction:
  - ✓ **Define** equivalence classes based on Specification
  - ✓ **Confirm** the assumed equivalence classes using **formal checks**
  - ✓ Use the **confirmed** equivalence classes to **reduce** verification scope for simulation
- Recursively partition address map into 3 equivalence classes
  - **[accessible]** -> formal check
  - **[reserved]** -> formal check
  - **[other]** -> ranges not conclusive at one hierarchy level – further break-down necessary
- Recursively apply partitioning down to granularity of single IP slot
  - IP slot ranges are only a small subset (1/1000) of overall SoC address map
  - The reduced address ranges is feasible to explore in simulation

Recursive hierarchical partitioning



Outline

- Introduction
  - Motivation
  - Verification challenges on SoC level
- State of the Art & Limitations
- Verification approach
  - Equivalence Classes
  - Recursive hierarchical break-down
- **Results & Perspectives**

Results and Perspectives

- ✓ Completeness:
  - ✓ Each address of the SoC memory map is covered by approach
  - ✓ Safety and security requirements related to address map can be ensured
  - ✓ “Freedom from Interference” arguments as used by customers can be proven to hold true
- ✓ Efficiency:
  - ✓ Successfully combined the complementing strengths of established verification methods
  - ✓ Formal check of wide ranges and Simulation of “deep” register accesses through full access path
  - ✓ Verification runtimes and resource utilization (CPU&Memory) became feasible
    - Simulation: 8 testcases per peripheral slot, each running 10min-4h
    - Formal: 8CPU with 80G Mem per formal run, about 20 runs, each running 30min-8h

Results and Perspectives

- ✓ Project Effects:
  - ✓ Security vulnerabilities can be detected and fixed before RTL freeze
  - ✓ Validation, Field Application and finally our Customer identify less address map defects

- ➢ Perspectives
  - ➢ Apply standardized memory map specification formats
  - ➢ Improve automation of partitioning ➔ automate generation of assertions
  - ➢ Improve automation of testcase generation
  - ➢ Adapt to other bus protocols (e.g. AHB→AXI)
  - ➢ Adapt to other RAM/Flash interfaces, e.g. external

Thank you for your attention …

Questions?

accellera SYSTEMS INITIATIVE

freescale™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Backup: Benefit

Assume

- 4h CPU time per IP slot
- 60 IP slots

➔4h*60 = 2400h

Consider: IP slots are just 1/1000 of 4G address map

➔ Hypothetical CPU time for 4G: 2400h*1000 = 100days *1000 = 100000 days = 273 year (CPU time)

Backup: Invariant Conditions

- For some aspects of the hierarchical break down it is essential to verify the invariance of accesses while being processed in bridge modules (routing modules such as cross-bars). Aspects:
  - AHB access if routed from a master to a slave port shall not change it's attributes
  - AHB access shall only be routed to one slave port
  - Each AHB access appearing at slave port shall correspond to exactly one master AHB access
  - Routing of an AHB access to a specific slave port does not depend on master port number
  - Each AHB access transformed by an AHB_to_OtherProtocoal bridge shall following a specific set of transformation rules
- Proving such invariants can be used to further reduce the simulation space
- Due to importance of Invariant Conditions we hosted a dedicated master thesis just for this aspect