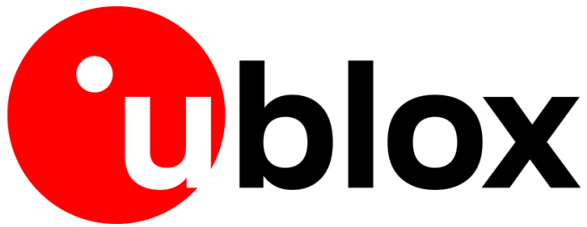
Covering the Last Mile in SoC-Level Deadlock Verification

Jef Verdonck, *Konstantinos Liatakis*, Khaled Nsaibia – u-blox AG

Dhruv Gupta, Sagar Dewangan, Tarun Upadhyay,

HarGovind Singh, Roger Sabbagh – Oski Technology



accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction to u-blox mobile SoCs
- Level-5 Formal Methodology
- Mobile SoC Case Study
- Results
- Conclusions

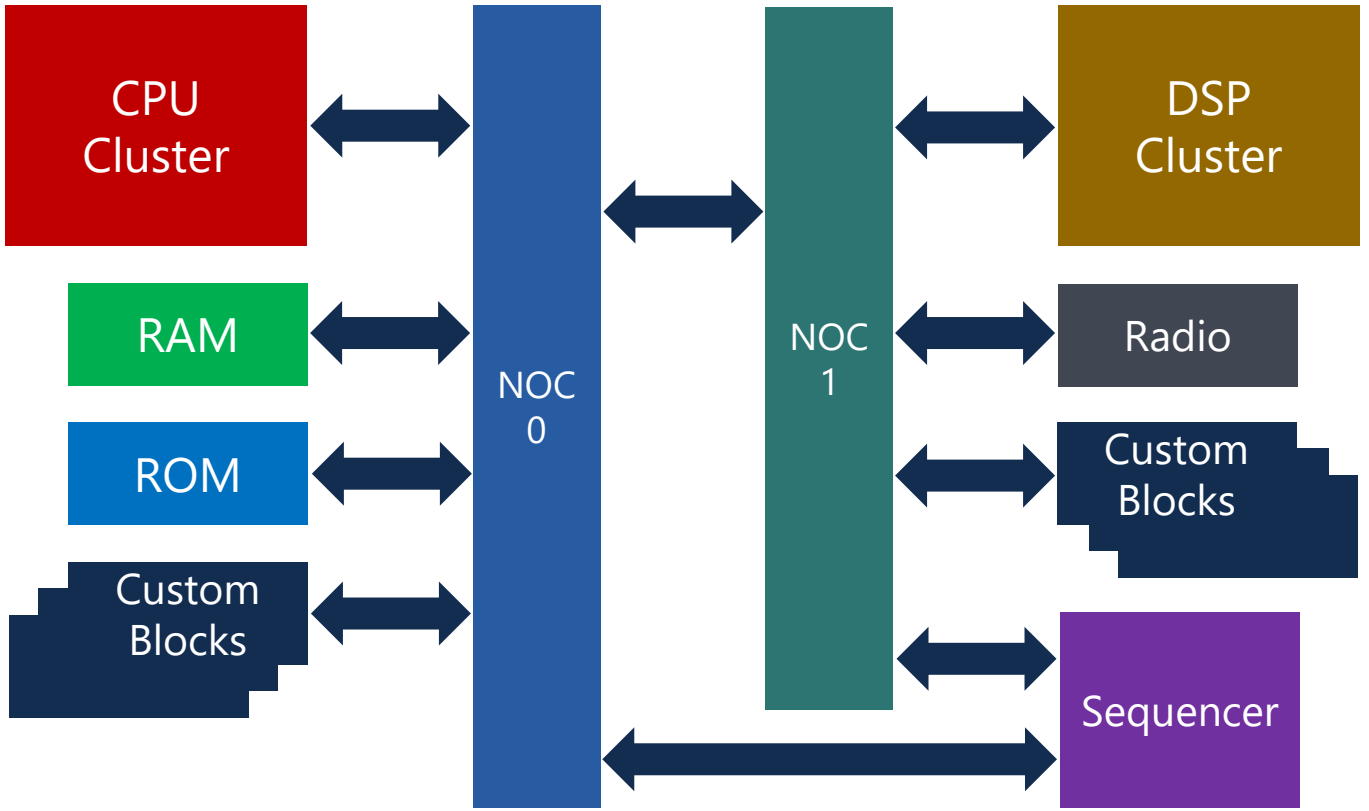
About u-blox

- IoT positioning and wireless communication solutions
- Technology and product line-up:

	P Positioning	C Cellular Communication	S Short Range Communication
Integrated Circuits			
Modules			
Services and Solutions	CellLocate® (modem based positioning) AssistNow™ (world wide GNSS assistance service) GNSS Correction Data (for high precision)* FOTA (Firmware over the air) Lifetime Security *through Sapcorda, a Joint Venture with industry partners		

u-blox IoT SoC Designs

- Multiple compute clusters
  - CPU – ARM®
  - DSP
- Multiple AXI® NoCs
- Wireless communication
  - LTE, Bluetooth, WiFi
- RAM/ROM
- 3rd party IP blocks
- Custom blocks



u-blox SoC Verification Methodology

- Simulation
  - UVM
  - S/W test cases
  - Assertions
  - Coverage (code + functional)
- Static and Formal
  - Linting, CDC
  - Formal apps
- Emulation
- FPGA prototype

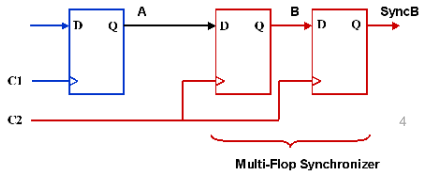
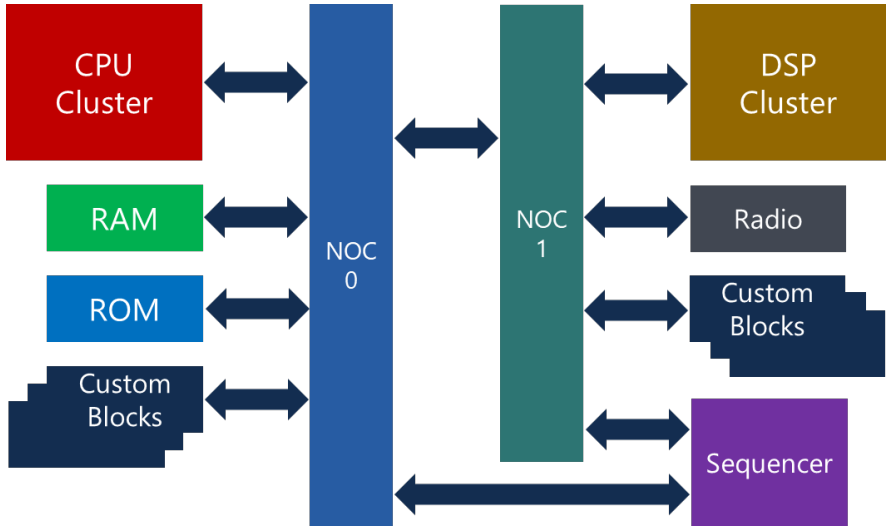


Image sources:
1. https://upload.wikimedia.org/wikipedia/en/thumb/e/ef/SystemVerilog_logo.png/240px-SystemVerilog_logo.png
2. https://media.digikey.com/Photos/Xilinx%20Photos/MFG_CK-V7-VC7215-G-J.jpg
3. https://www.accellera.org/images/community/uvm/uvm_logo.png
4. https://m.eet.com/media/1071283/Apafig3.gif
5. https://www.cadence.com/content/dam/cadence-www/global/en_US/diagrams/tools/system-design-verification/palladium-z1.jpg

Deadlock Verification Challenge

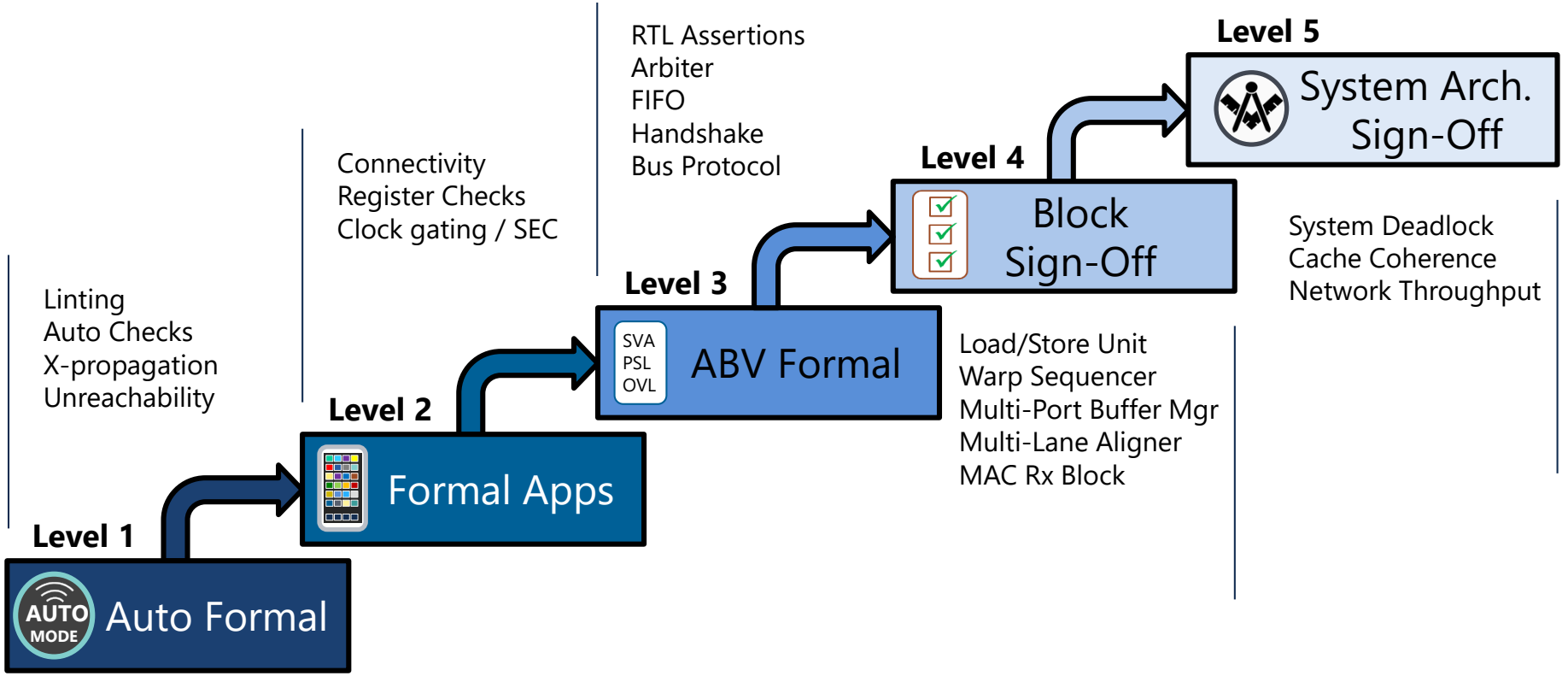
- SoC-level deadlock not well tested with traditional methods
  - Deadlocks occur in rare cases
  - Corner-case coverage is limited
- Simulation
  - High code coverage is ineffective
  - High functional coverage is insufficient
- Emulation / FPGA prototypes
  - Non-exhaustive
  - Used late in the design cycle
  - **Requires SW representative of the real target application**
- Slow turn-around-time for verification of design changes



Agenda

- Introduction to u-blox mobile SoCs
- Level-5 Formal Methodology
- Mobile SoC Case Study
- Results
- Conclusions

Formal Capability Maturity Model



Level-5 Formal Methodology

STEP 1

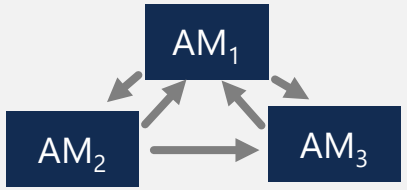
Create Architectural Models for the Blocks

- Model the block level "contract" that contributes to the system level requirement
- Abstract internal details not relevant to system-level requirement verification
- Example: For deadlock, model only the passing of control between blocks

STEP 2

Formal Verification of System of Block Models

- Proves block-level contracts imply system-level proofs
- Example: Absence of deadlock



STEP 3

Validate Block Models Against RTL

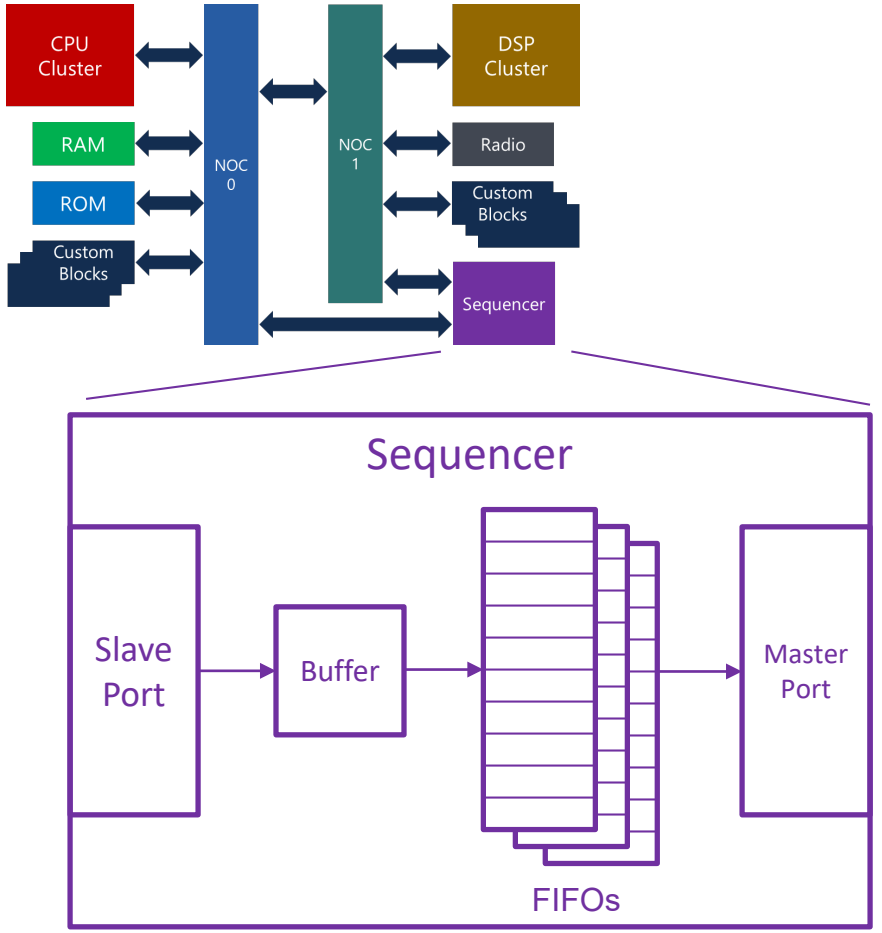
- Run formal verification to prove RTL implementation guarantees required contract
- Check that assumptions made about block AMs are true of the RTL

Agenda

- Introduction to u-blox mobile SoCs
- Level-5 Formal Methodology
- Mobile SoC Case Study
- Results
- Conclusions

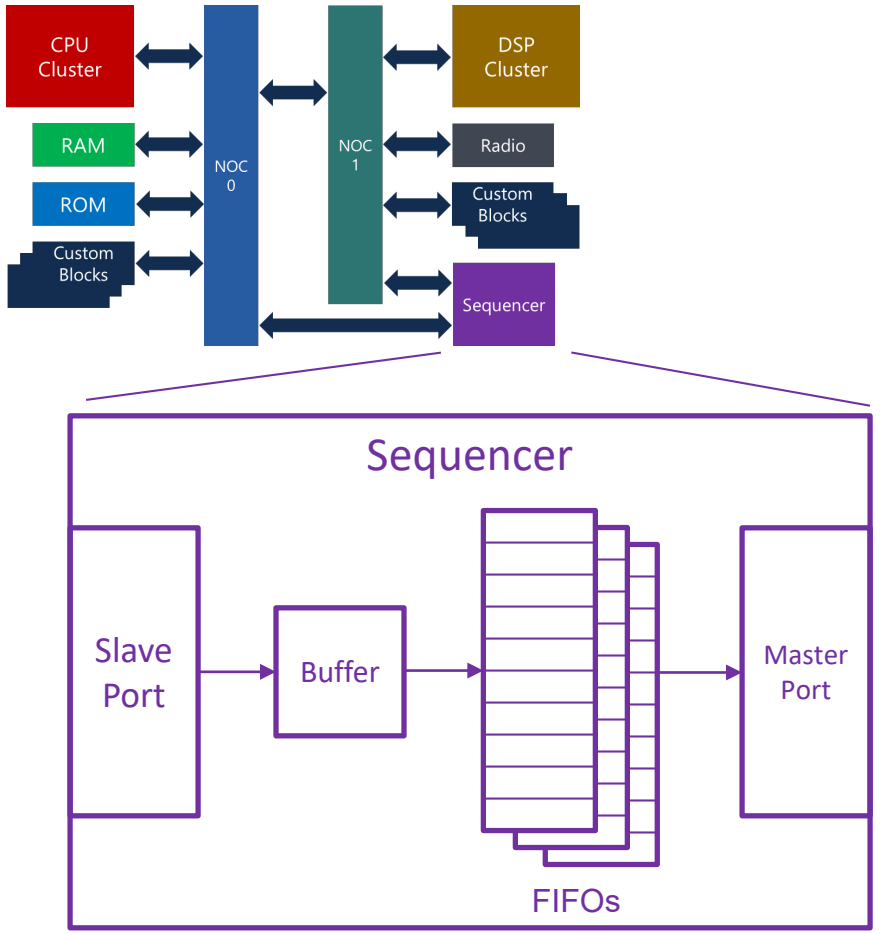
Step 1: Architectural Models

- Example: Sequencer (SQ)
  - Slave port receives transactions from NoC1
  - Master port initiates transactions to NoC0
- Slave port flow control property
  - If write transaction is not back-pressured by target FIFO, then
    - Accept the transaction and write the response to the FIFO
  - Else,
    - Do not accept the transaction until back-pressure is resolved in the relevant FIFO



SQ Architectural Model: Master Port

- Master port flow control property
  - **If** there are entries at the top of the FIFOs to be sent on the master port, **and**
  - There is no outstanding transaction in flight on the master port, **then**
    - The entry is popped from the relevant FIFO
    - A transaction is initiated on the master port



SQ Data Path Abstraction

- Abstracted data path
  - FIFOs track only the number of entries and meta data required for flow control

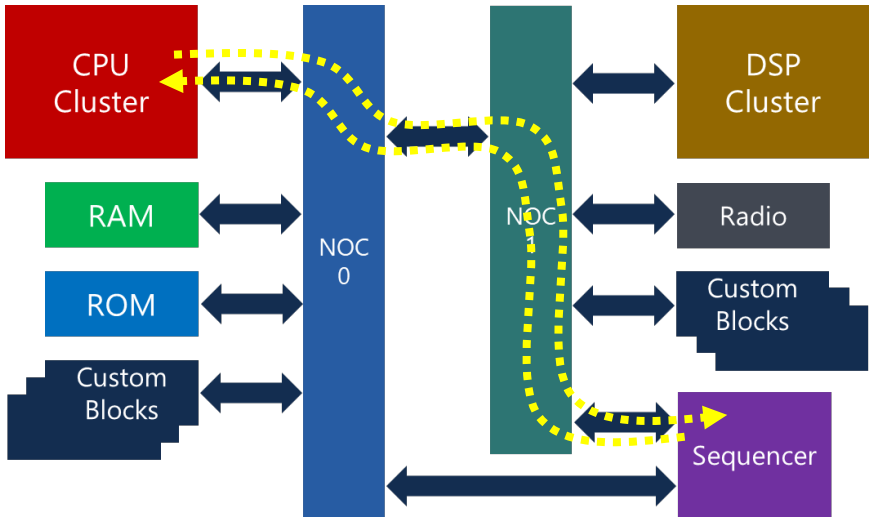
```
// SQ Abstraction Model code example
// Instance for one of the SQ target FIFO
// NOTE: No data is stored for this FIFO.
// AM tracks only number of entries
FV_fifo #(
    .DEPTH(SQ_FIFO_DEPTH)
) sq_fifo (
    .clk(clk)
   ,.resetn(resetn)
   ,.in_valid(sq_fifo_in_valid)
   ,.out_valid(sq_fifo_out_valid)
   ,.fifo_full(sq_fifo_full)
   ,.fifo_empty(sq_fifo_empty)
);
```

```
// No write to SQ FIFO if full
asrt_int_push_0_if_full: assert property(
   @(posedge clk) disable iff(!resetn)
   sq_fifo_full |-> !sq_fifo_in_valid
);

// No pop from SQ FIFO if empty
asrt_int_pop_0_if_empty: assert property(
   @(posedge clk) disable iff(!resetn)
   sq_fifo_empty |-> !sq_fifo_out_valid
);
```

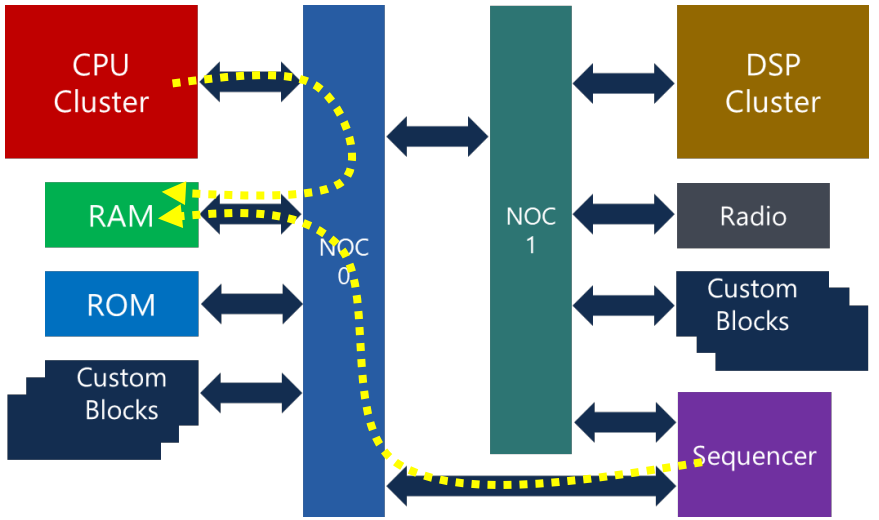
Step 2: System-Level Verification

- System-level DUT
  - Assemble DUT of Architectural Models
- Forward progress checkers
  - **If** a master initiates a transaction to a slave, **then**
    - The slave should receive the request within a finite time window, **and**
    - The master should receive the response within a finite time window
  - Example
    - CPU write requests on AW channel to SQ should appear at SQ slave port within **n** cycles
    - SQ slave port write responses on B channel should appear at CPU master port within **m+n** cycles



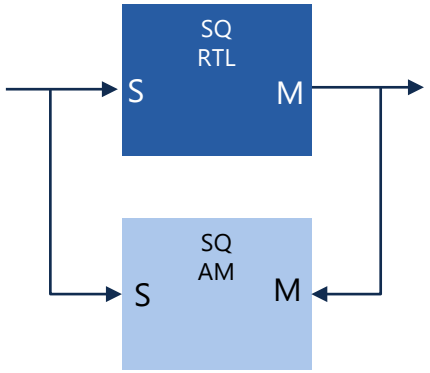
System-Level Constraints

- System-level constraints
  - Model real-world S/W use cases
- Examples
  - Masters can only initiate write transactions to a sub-set of slave ports
    - SQ can only initiate writes to the RAM
  - Transactions can only be of a specific type and sequence
    - CPU can never initiate single beat (AWLEN == 0) write transactions to the RAM
- Constraints validation
  - Assumptions used as assertions in later stages of verification (simulation, emulation)



Step 3: Validate Block AMs vs. RTL

- Close the loop
  - Ensure that assumptions made in the AM are valid
- Containment check
  - RTL behavior must fall within the range of assumptions made for the AM
- Block-level formal sign-off
  - Use Level-4 methodology to prove block-level RTL properties
- Results
  - All the properties were proven (unbounded)

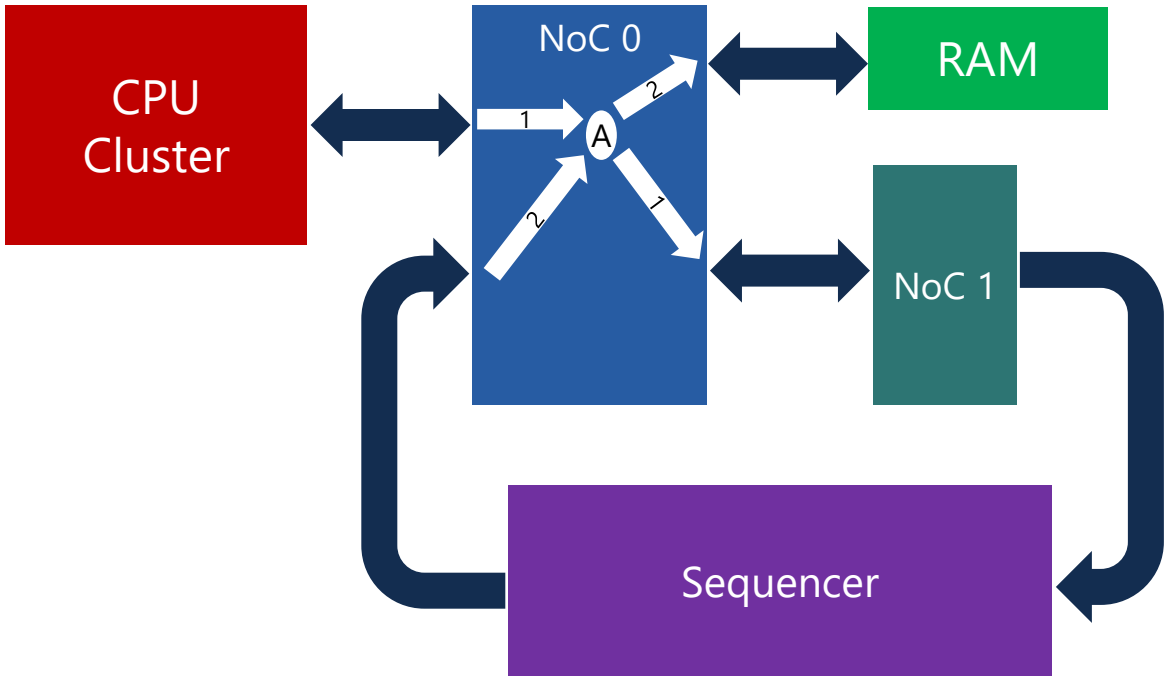


Agenda

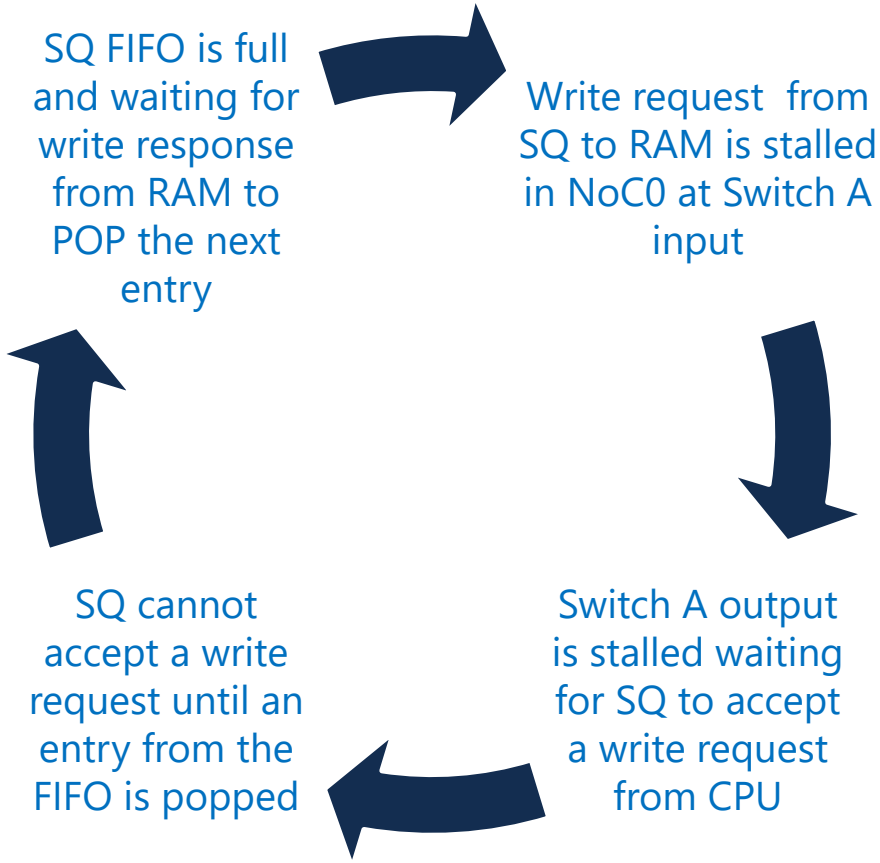
- Introduction to u-blox mobile SoCs
- Level-5 Formal Methodology
- Mobile SoC Case Study
- Results
- Conclusions

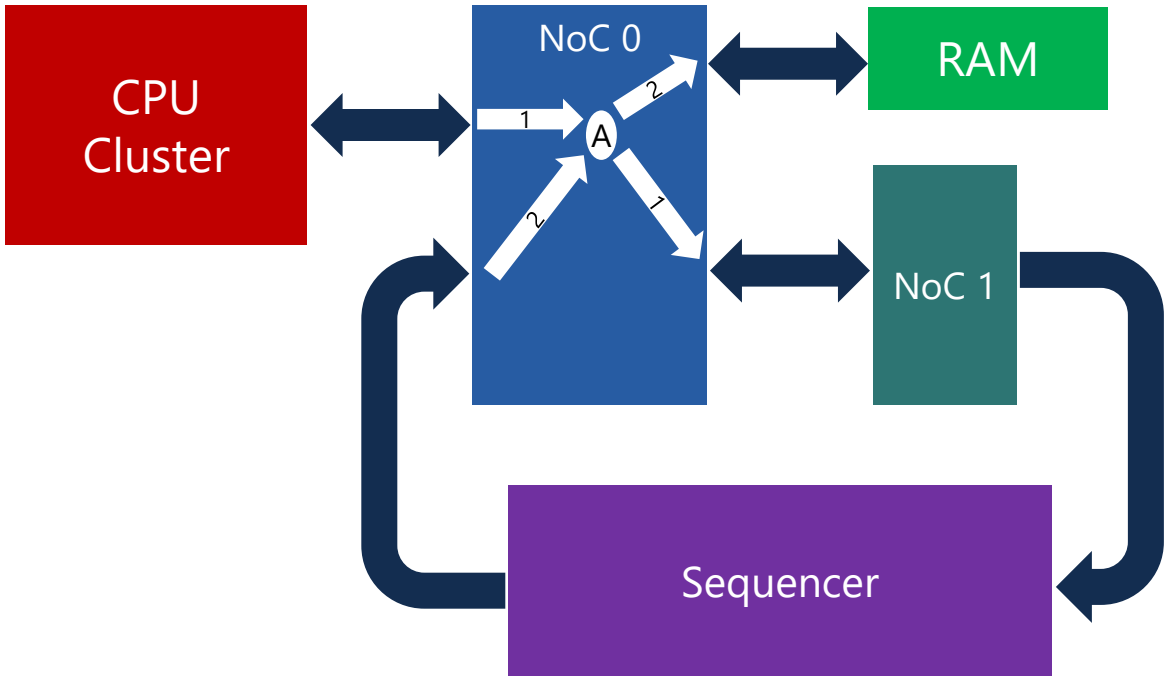
Found SoC Deadlock Bug

- Bug scenario
  - CPU sends multiple write requests to SQ
  - SQ FIFO becomes full, which back-pressures the NoCs
  - A CPU write request to SQ is stalled at Switch A
  - In parallel, SQ sends multiple write transactions to RAM
  - An SQ write request to RAM is stalled behind CPU transactions at Switch A



Circular Dependency

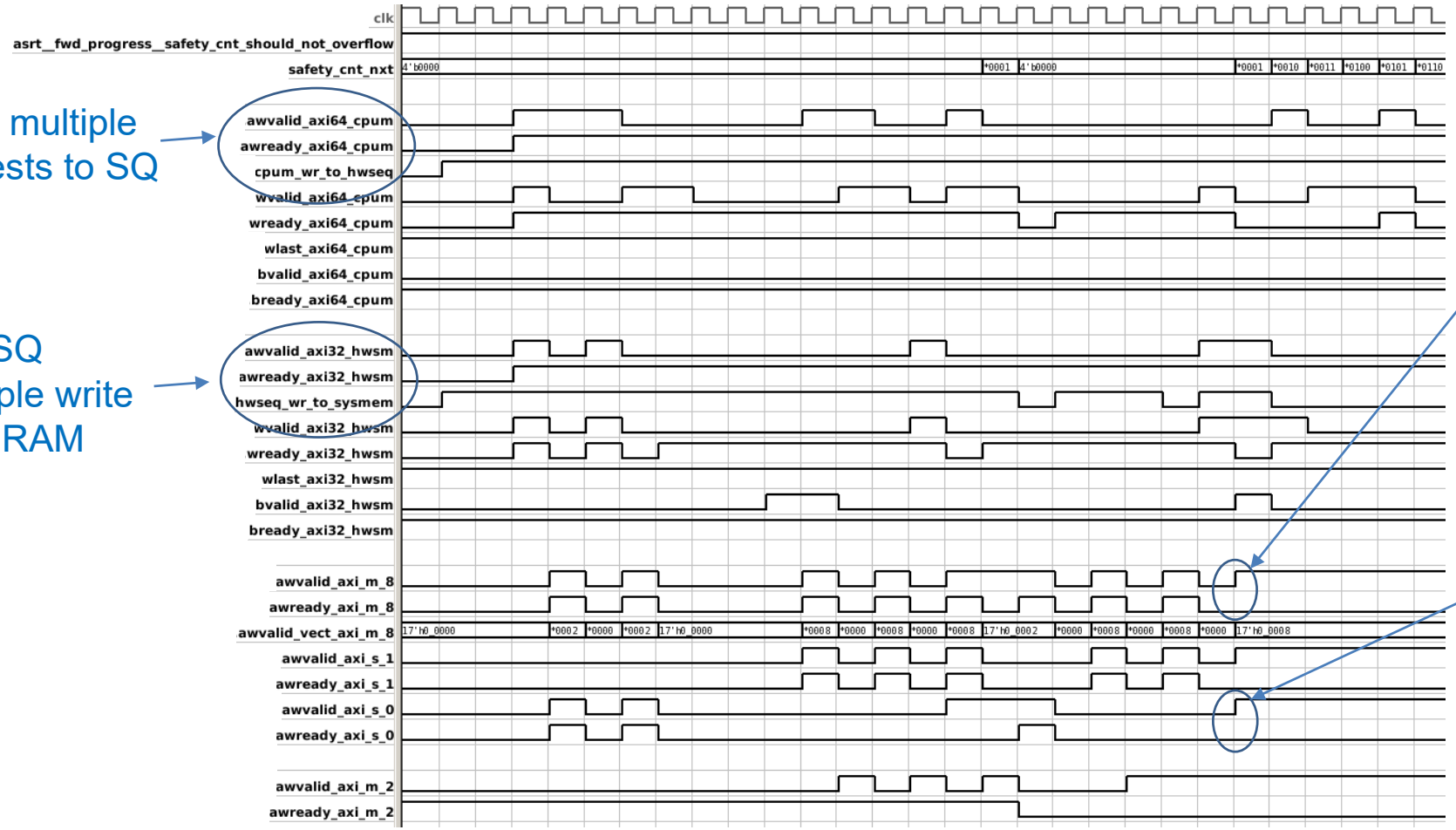




Counterexample Waveform

1. CPU sends multiple write requests to SQ

2. In parallel, SQ sends multiple write requests to RAM



3. CPU to SQ write is stalled at SwitchA output port

4. SQ to RAM write request is waiting behind CPU to SQ write request at Switch A

Agenda

- Introduction to u-blox mobile SoCs
- Level-5 Formal Methodology
- Mobile SoC Case Study
- Results
- Conclusions

Observations of Formal Approach

- Deadlock bug found after 2 weeks of formal effort
  - Undetected after many months of simulation
  - Undetected after 6 weeks of emulation
- AMs are much smaller than RTL
  - Example: #LOC of SQ AM is 10% of the #LOC of SQ RTL
- Formal tool runtime to find the bug was ~30 minutes
  - Architectural models enable effective formal analysis of system requirements
  - Not possible with the RTL models – runtime explosion
- Short turn-around-time to prove the bug fix
  - Achieved a high-level of confidence in the new design in 1 day
  - Would take weeks or months otherwise

Conclusions

- Level 5 formal sign-off
  - Effective solution for SoC-level deadlock verification
  - Addresses the risk of bugs that are not detected with traditional methods

- Successful use on mobile SoC
  - Found critical deadlock bugs
  - Fast verification of bug fixes
  - Provided confidence for tapeout

Current and Future Directions in Formal Verification

- Level 1 and Level 2 Formal
  - Auto-checks, Protocol Compliance checks, Connectivity checks, Register checks etc.
  - Part of u-blox maturity flow
  - Required for sign-off
- Level 3 Formal
  - Assertion based verification
  - Already employed for many blocks
  - Aim is to expand its usage in areas more suitable for formal – discover more bugs earlier
    - Handshakes
    - FSM logic and other control structures
    - FIFOs
- Level 5 Formal
  - Powerful to find the root cause of deadlocks and to prove bug fixes
  - Will be used in the future for relevant system level scenarios

Questions?