Formal Analysis of the ACE Specification for Cache Coherent Systems-On-Chip

Abderahman KRIOUILE
PhD student, STMicroelectronics – Inria Rhône-Alpes – LIG

Wendelin SERWE
Research scientist, Inria Rhône-Alpes – LIG

Acknowledgements. Radu MATEESCU (Inria) & Massimo ZENDRI (STMicroelectronics) for their contribution and collaboration.

FMICS'2013
Madrid, Spain, 23-24/09/2013

life.augmented

L I G

informatics mathematics

Agenda

- **Introduction: STMicroelectronics**
- **Motivation**
- **ACE specification**
- **Formally modeling an ACE-compliant SoC with CADP**
- **Validation work**
- **Conclusion & On-going work**

Who we are

- A global semiconductor leader
- The largest European semiconductor company
- 2012 revenues of **$8.49B**(1)
- Approx. **48,000** employees worldwide(1)
- Approx. **11,500**(1) people working in R&D
- **12** manufacturing sites
- Listed on New York Stock Exchange, Euronext Paris and Borsa Italiana, Milano

(1) Including ST-Ericsson, a 50:50 joint venture with Ericsson

Flexible and Independent Manufacturing

A stronger, more focused product portfolio

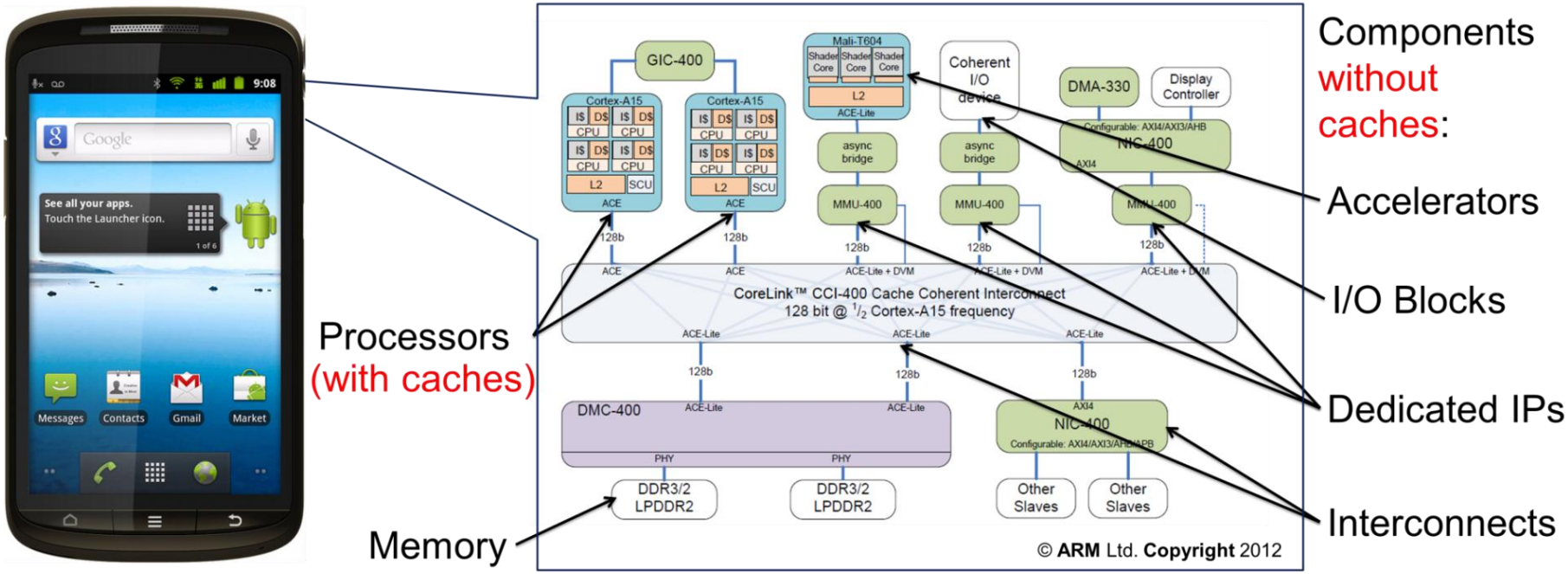
Sense & Power and Automotive Products

- MEMS and Sensors
- Power Discrete and Modules
- Advanced Analog, Power Management and Standard ICs
- Automotive products

- General Purpose MCUs and Secure MCUs
- Application Processors and Digital Consumer products
- Imaging ICs and Modules
- Digital ASICs

Embedded Processing Solutions

Heterogeneous System-on-Chip



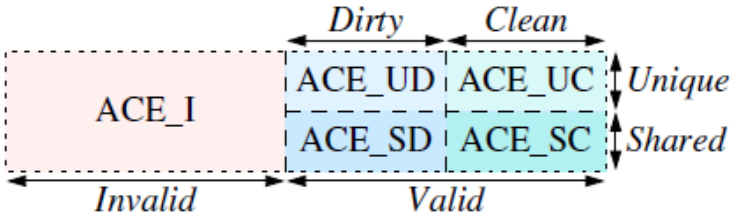
- Need for **System-Level Cache Coherency**
- **ARM** proposed ACE specification: standard for system level cache coherency

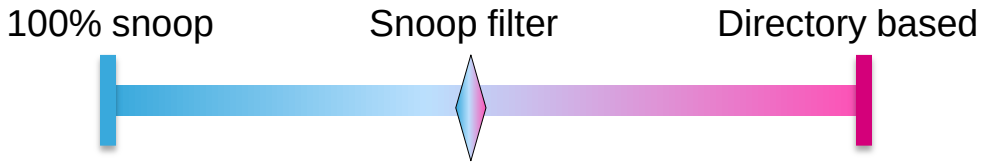
ACE Specification

- **ACE** (**A**XI **C**oherency **E**xtension): more than 300 pages specification
  - http://infocenter.arm.com/help/topic/com.arm.doc.ihi0022e
- ACE: a hardware support for System-Level Cache Coherency
- ACE specification
  - Interface communication protocol
  - Interconnect responsibilities

ACE Specification

- ACE states of a cache line:
- ACE channels
  - Read Channels (AR, R)
  - Write Channels (AW, W, B)
  - Snoop Channels (AC, CR, CD)
- ACE supported policies
  - 100% snoop
  - Directory based
  - Anything between (snoop filter)



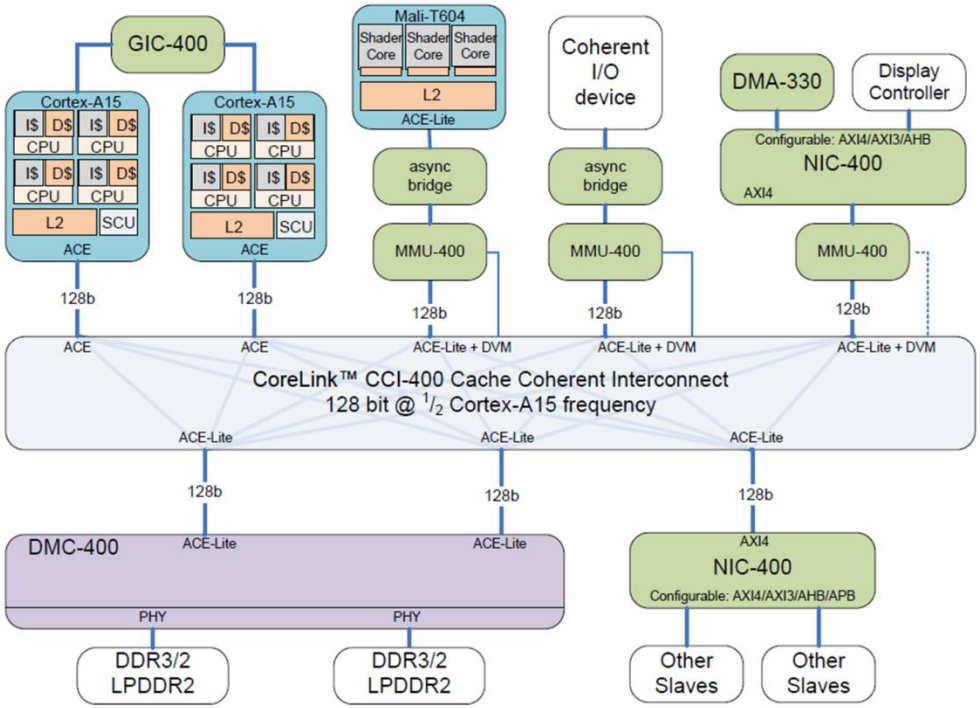


Different meanings of “protocol”

- Cache coherent protocols
  - System communication policies
- ACE protocol
  - Interface communication protocol
  - Interconnect responsibilities
- ACE protocol does not guarantee coherency
  => ACE is a support for coherency

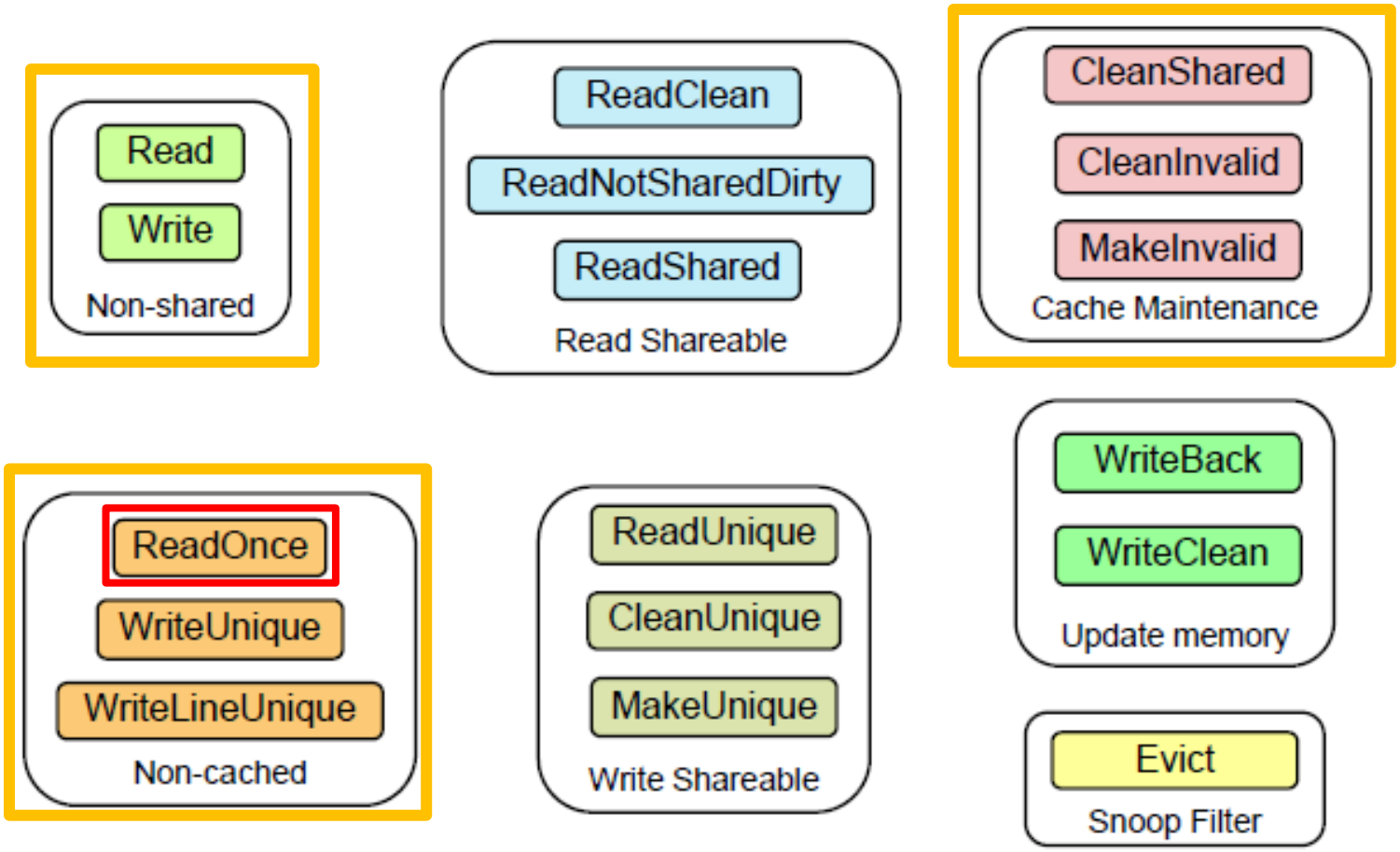
Different Kinds of Components

- **Interconnect**: called CCI (Cache Coherent Interconnect)
- **ACE Masters**: masters with caches
- **ACE-Lite Masters**: components without caches snooping other caches
- **ACE-Lite/AXI Slaves:** components not initiating snoop transactions



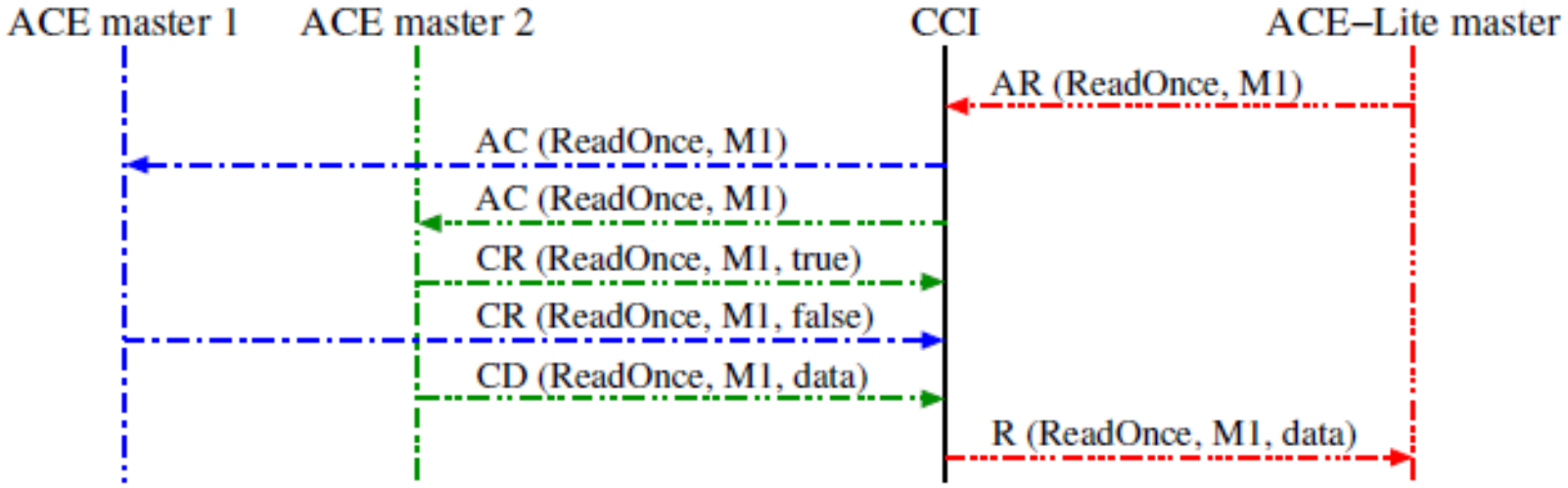
ACE transactions

ACE-Lite transaction subset



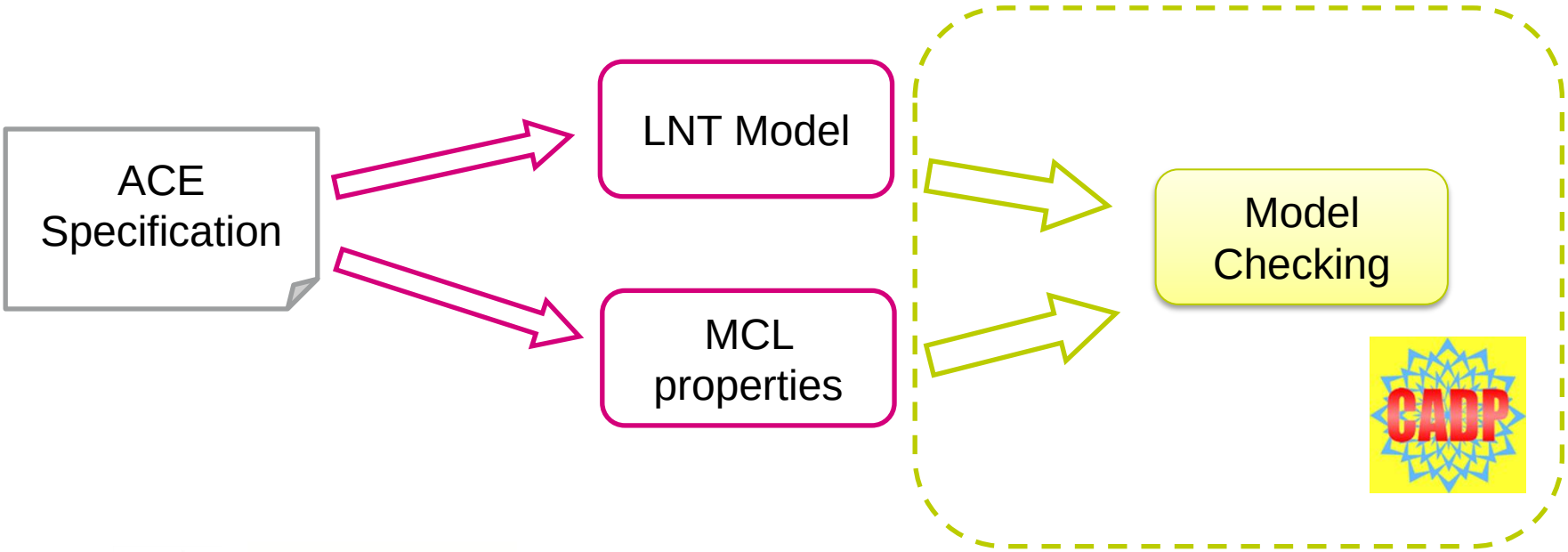
Example: transaction execution scenario

- Execution scenario of a ReadOnce transaction



Case study flow

- LNT specification language: modeling
- MCL temporal logic language: writing properties
- EVALUATOR4.0 model checker (CADP toolbox)



CADP verification toolbox

- Modular toolbox for formal modeling and enumerative verification of asynchronous systems (50 tools)
- Based on concurrency theory (process calculi)
- User-friendly input language (LNT) integrating features of
  - Imperative programming constructs: loops, variables, …
  - Concurrency theory: parallelism, formal semantics (LTS: Labeled Transition System)
- Several verification paradigms and techniques: model checking, compositional, on-the-fly, …
- Test & Code generation for rapid prototyping
- More information: http://cadp.inria.fr

Modeling Choices

- Focus on interactions between components (LTS => Black box view)
- Generic model: non-deterministic behavior
- Fully connected snoop topology
- Parametric model: different number of masters and slaves
- Transaction on a channel ⇔ LNT rendezvous on a gate (same name)

From ACE specification to LNT model

- Non deterministic choices
- If… then…else…

```
select
   if (  IsShared==0 ) then
      R (ReadOnce, …);
      CacheLine.state:= ACE_UC
   else
      R (ReadOnce, …);
      CacheLine.state:= ACE_SC
   end if
[ ]
   R (ReadOnce, …);
   CacheLine.state:= ACE_SC
[ ]
   …
end select
```

C4.5.2 ReadOnce

ReadOnce is a read transaction that is used in a region of memory that is shareable with other masters. This transaction is used when a snapshot of the data is required. The location is not cached locally for future use.

The transaction response requirements are:

- the IsShared response indicates if the cache line is shared or unique
- the PassDirty response must be deasserted.

Table C4-3 shows the expected cache line state changes for the ReadOnce transaction:

Table C4-3 Expected ReadOnce cache line state changes

Transaction	Start state	RRESP[3:2] IsShared/PassDirty	Expected end state	Legal end state With Snoop Filter	Legal end state No Snoop Filter
ReadOnce	I	00	I	I	I
		10	I	I	I

Table C4-4 shows the other permitted cache line state changes for the ReadOnce transaction:

Table C4-4 Other permitted ReadOnce cache line state changes

Transaction	Start state	RRESP[3:2] IsShared/PassDirty	Expected end state	Legal end state With Snoop Filter	Legal end state No Snoop Filter
ReadOnce	UC	00	UC	UC, SC	I, UC, SC
	UD	00	UD	UD, SD	UD, SD
	SC	00	UC	UC, SC	I, UC, SC
		10	SC	SC	I, SC
	SD	00	UD	UD, SD	UD, SD
		10	SD	SD	SD

From ACE specification to LNT model

- But how to model this requirement ?

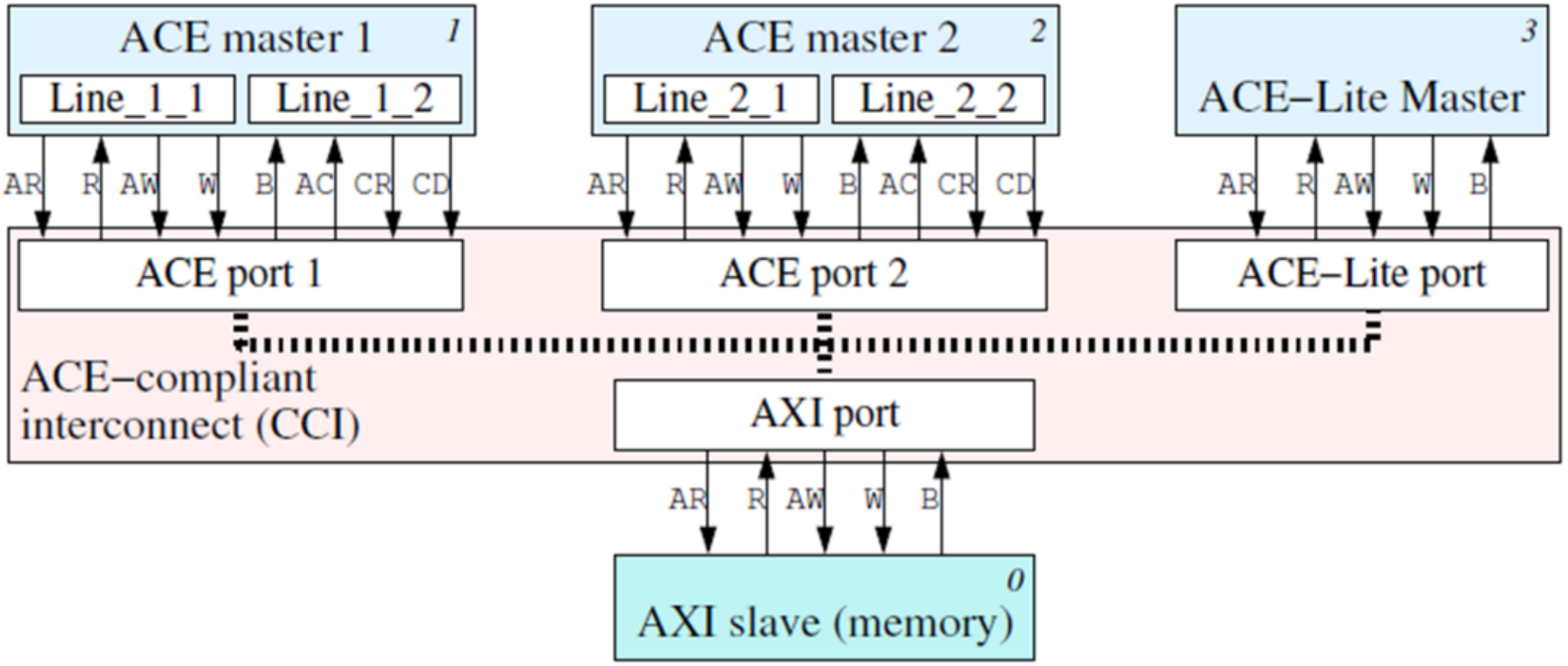
C6.5.3 Permission to update main memory

The interconnect must ensure that all updates to main memory, both from cached masters and the interconnect itself, are performed in the correct order. The interconnect must only give a cached master permission to update main memory when it is guaranteed that any earlier updates to main memory are ordered.

- Is this the definition of data integrity ?
- We use “Constraint-oriented specification style”
  - Global processes in parallel to the model to constrain the behavior
- We can generate the LTS with or without global constraints

Formally Modeling an ACE-compliant SoC

- Formal model: about 3200 lines of LNT code
- Masters: non-deterministic agents including all ACE-conform behavior



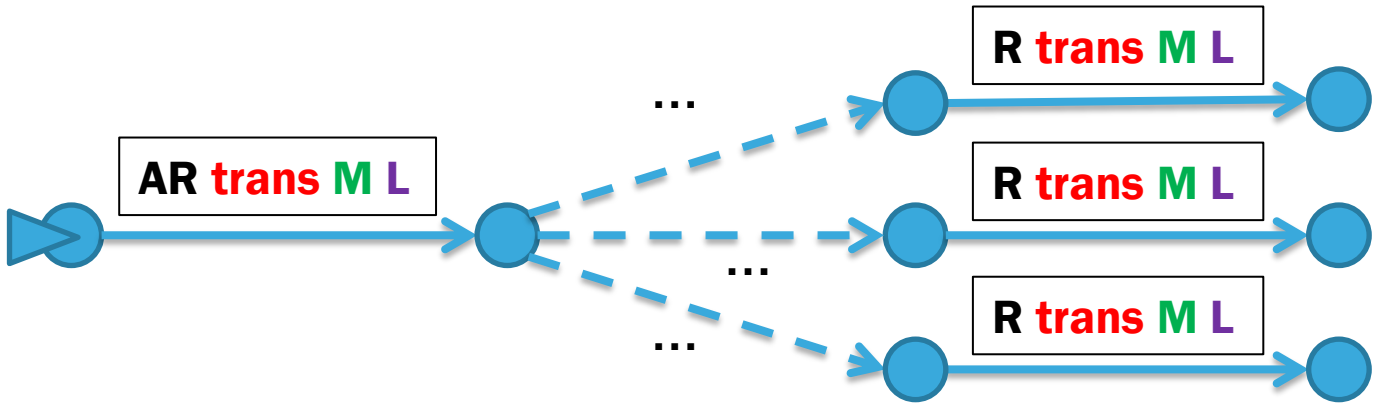
Verified Properties in MCL

- **Complete execution of transactions**

  every transaction inevitably finishes

Liveness formula

[true * . { AR ?trans:String ?M:Nat ?L:Nat ... }] inev ({ R !trans !M !L })



macro inev (Action) = mu X . (< true > true and [not Action] X) end_macro

From state based to action based properties

- **Cache coherency:** inter-caches coherency Safety property

  coherency of the ACE states of all caches

- If a cache line is on **ACE_UD** state (**M1**,**L**,**ACE_UD**) All caches of other masters (**M2** ≠ **M1**) which have the same memory line **L** have to be on an **ACE_I** state.

  => State based property

From state based to action based properties

- **Cache coherency:** inter-caches coherency Safety property

  coherency of the ACE states of all caches

- If an action **C M1 L ACE_UD** happen
  while there is no action **C M1 L s1 where s1≠ACE_UD**
  if an action **C M2 L s2 where M2≠M1 & s1≠ACE_I** happen
  then FALSE

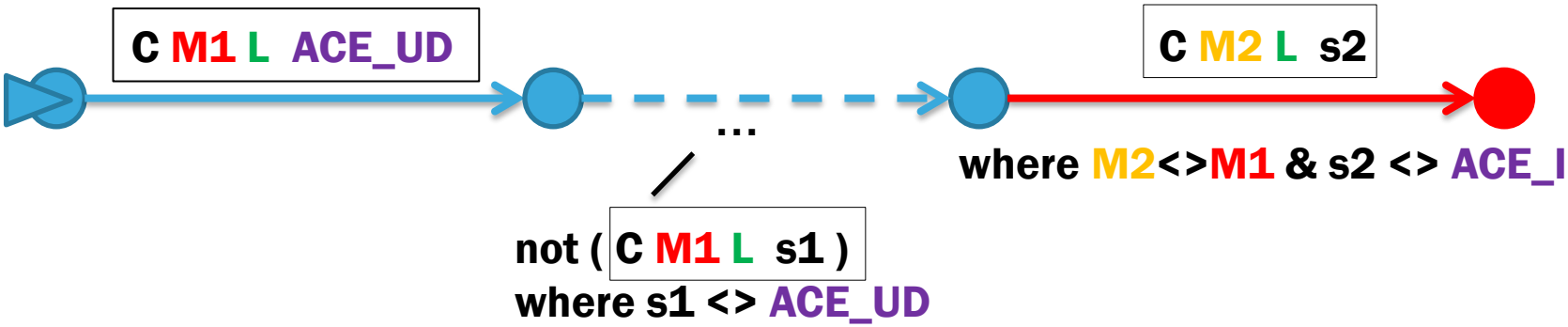
  => Action based property

Verified Properties in MCL

- **Cache coherency:** inter-caches coherency — Safety formula

  coherency of the ACE states of all caches

```
[ true * .
  { C ?M1:Nat ?L: Nat "ACE_UD" } .
  ( not ( { C !M1 !L ?s1:String where s1<>"ACE_UD"} ) ) * .
  { C ?M2:Nat ! L ?s2:String where (M2<>M1) and (s2<>"ACE_I") }
] false
```



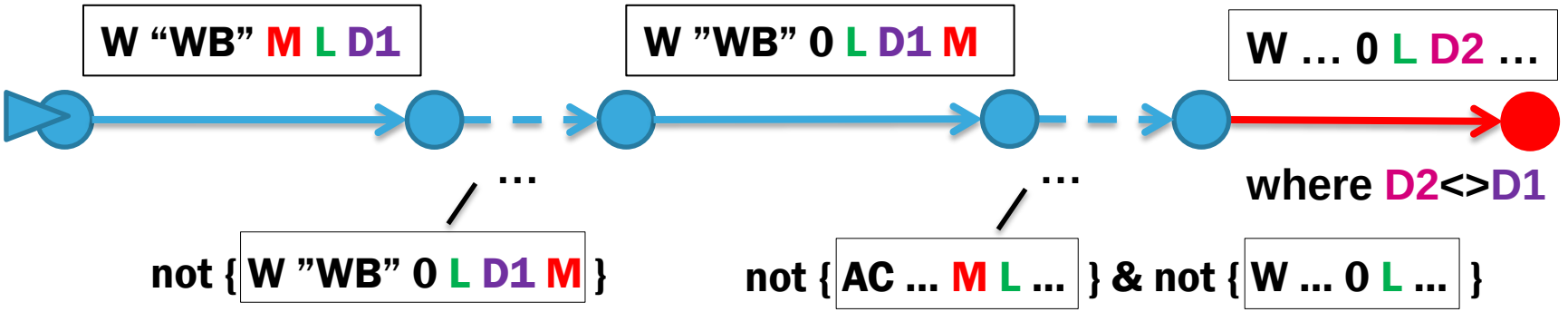
Verified Properties in MCL

- **Data integrity:** memory-caches coherency

  correct order of write operations to the shared memory

  Safety formula

```
[ true * .
  { W !"WRITEBACK" ?M:Nat ?L:Nat ?D1:Nat } .
  ( not { W !"WRITEBACK" !"0" !L !D1 !M } * .
  { W !"WRITEBACK" !"0" !L !D1 !M } .
  (
     ( not { AC ... !M !L ... } ) and
     ( not { W ... !"0" !L ... } )
  ) * .
  { W ... !"0" !L ?D2:Nat ... where D2<>D1 }
] false
```



State Space Generation & Verification

Experimental results: state space generation and verification

allowed transactions			global	LTS size		properties				
m1	m2	lite	constraints	states	transitions	φ_1	φ_2	φ_3	φ_4	φ_5
S_0	$\{\mathcal{A}\}$	S_0	yes	93,481,270	308,087,560	√	√	√	√	√
S_0	$\{\mathcal{A}\}$	S_0	no	105,376,971	351,344,207	√	√	√	√	×
S_0	$\emptyset$	S_0	yes	7,518,552	21,227,610	√	√	√	√	√
S_1	$\emptyset$	S_1	yes	3,685,311	10,649,422	√	√	√	√	√
S_1	$\emptyset$	S_1	no	3,127,707	9,121,134	√	√	×	×	×
S_2	S_2	$\emptyset$	yes	3,545,801	11,122,536	√	√	√	√	√
S_2	S_2	$\emptyset$	no	2,819,505	9,095,620	√	√	×	×	√
S_3	$\emptyset$	S'_3	yes	1,834,195	5,170,829	√	√	√	√	√
S_3	$\emptyset$	S'_3	no	1,437,412	4,547,398	√	√	√	√	×
S_4	S_4	$\emptyset$	yes	560,299	1,669,886	√	√	√	√	√
S_4	S_4	$\emptyset$	no	599,971	1,780,634	√	√	×	×	×
S_5	S_5	$\emptyset$	yes	40,983	63,922	√	√	√	√	√
S_5	S_5	$\emptyset$	no	55,439	98,688	√	√	√	√	√

In the table above, we use those sets of allowed transactions:
S_0 = set of all ACE (respectively ACE-Lite) transactions
$S_1 = \{MakeUnique, ReadOnce, ReadUnique, WriteBack\}$
$S_2 = \{MakeInvalid, MakeUnique, ReadShared, ReadUnique, WriteBack\}$
$S_3 = \{MakeUnique, WriteBack\}$, $S'_3 = \{ReadOnce\}$
$S_4 = \{CleanInvalid, CleanShared, ReadUnique, WriteBack\}$
$S_5 = \{MakeInvalid, MakeUnique, WriteBack\}$

Complete execution

State Space Generation & Verification

Experimental results: state space generation and verification

allowed transactions			global	LTS size		properties				
m1	m2	lite	constraints	states	transitions	φ_1	φ_2	φ_3	φ_4	φ_5
S_0	$\{\mathcal{A}\}$	S_0	yes	93,481,270	308,087,560	√	√	√	√	√
S_0	$\{\mathcal{A}\}$	S_0	no	105,376,971	351,344,207	√	√	√	√	×
S_0	$\emptyset$	S_0	yes	7,518,552	21,227,610	√	√	√	√	√
S_1	$\emptyset$	S_1	yes	3,685,311	10,649,422	√	√	√	√	√
S_1	$\emptyset$	S_1	no	3,127,707	9,121,134	√	√	×	×	×
S_2	S_2	$\emptyset$	yes	3,545,801	11,122,536	√	√	√	√	√
S_2	S_2	$\emptyset$	no	2,819,505	9,095,620	√	√	×	×	√
S_3	$\emptyset$	S'_3	yes	1,834,195	5,170,829	√	√	√	√	√
S_3	$\emptyset$	S'_3	no	1,437,412	4,547,398	√	√	√	√	×
S_4	S_4	$\emptyset$	yes	560,299	1,669,886	√	√	√	√	√
S_4	S_4	$\emptyset$	no	599,971	1,780,634	√	√	×	×	×
S_5	S_5	$\emptyset$	yes	40,983	63,922	√	√	√	√	√
S_5	S_5	$\emptyset$	no	55,439	98,688	√	√	√	√	√

In the table above, we use those sets of allowed transactions:

S_0 = set of all ACE (respectively ACE-Lite) transactions

$S_1 = \{MakeUnique, ReadOnce, ReadUnique, WriteBack\}$

$S_2 = \{MakeInvalid, MakeUnique, ReadShared, ReadUnique, WriteBack\}$

$S_3 = \{MakeUnique, WriteBack\}$, $S'_3 = \{ReadOnce\}$

$S_4 = \{CleanInvalid, CleanShared, ReadUnique, WriteBack\}$

$S_5 = \{MakeInvalid, MakeUnique, WriteBack\}$

Cache coherency

State Space Generation & Verification

Experimental results: state space generation and verification

allowed transactions			global	LTS size		properties				
m1	m2	lite	constraints	states	transitions	φ_1	φ_2	φ_3	φ_4	φ_5
S_0	$\{\mathcal{A}\}$	S_0	yes	93,481,270	308,087,560	√	√	√	√	√
S_0	$\{\mathcal{A}\}$	S_0	no	105,376,971	351,344,207	√	√	√	√	×
S_0	$\emptyset$	S_0	yes	7,518,552	21,227,610	√	√	√	√	√
S_1	$\emptyset$	S_1	yes	3,685,311	10,649,422	√	√	√	√	√
S_1	$\emptyset$	S_1	no	3,127,707	9,121,134	√	√	×	×	×
S_2	S_2	$\emptyset$	yes	3,545,801	11,122,536	√	√	√	√	√
S_2	S_2	$\emptyset$	no	2,819,505	9,095,620	√	√	×	×	√
S_3	$\emptyset$	S'_3	yes	1,834,195	5,170,829	√	√	√	√	√
S_3	$\emptyset$	S'_3	no	1,437,412	4,547,398	√	√	√	√	×
S_4	S_4	$\emptyset$	yes	560,299	1,669,886	√	√	√	√	√
S_4	S_4	$\emptyset$	no	599,971	1,780,634	√	√	×	×	×
S_5	S_5	$\emptyset$	yes	40,983	63,922	√	√	√	√	√
S_5	S_5	$\emptyset$	no	55,439	98,688	√	√	√	√	√

In the table above, we use those sets of allowed transactions:

S_0 = set of all ACE (respectively ACE-Lite) transactions

$S_1 = \{MakeUnique, ReadOnce, ReadUnique, WriteBack\}$

$S_2 = \{MakeInvalid, MakeUnique, ReadShared, ReadUnique, WriteBack\}$

$S_3 = \{MakeUnique, WriteBack\}$, $S'_3 = \{ReadOnce\}$

$S_4 = \{CleanInvalid, CleanShared, ReadUnique, WriteBack\}$

$S_5 = \{MakeInvalid, MakeUnique, WriteBack\}$

Data integrity

Conclusion

- Formal model for ACE-compliant SoC produced
- CADP/LNT: analysis heterogeneous coherent SoCs
- Constraint-oriented specification style helpful to model general requirements
- Model used by STMicroelecronics to simulate the behavior in system-level
- Counterexamples: scenarios to be tested on industrial test bench

On-Going Work

- Impact of a coherent interconnect in a concrete SoC
  - Combine generic interconnect with model of a concrete SoC
- Model-based test and validation:
  - automatic test-scenario extraction
  - guided (co-)simulation

Formal Analysis of the ACE Specification for Cache Coherent Systems-On-Chip

To contact us: Abderahman.kriouile@st.com & Wendelin.serwe@inria.fr

For more information

- CADP toolbox

http://cadp.inria.fr

- ARM. AMBA AXI and ACE Protocol Specification

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0022e