MEMOCODE 2009

Verification of an Industrial SystemC/TLM Model Using LOTOS and CADP

Hubert Garavel, Claude Helmstetter, Olivier Ponsini,

and Wendelin Serwe

INRIA / VASY

http://www.inrialpes.fr/vasy

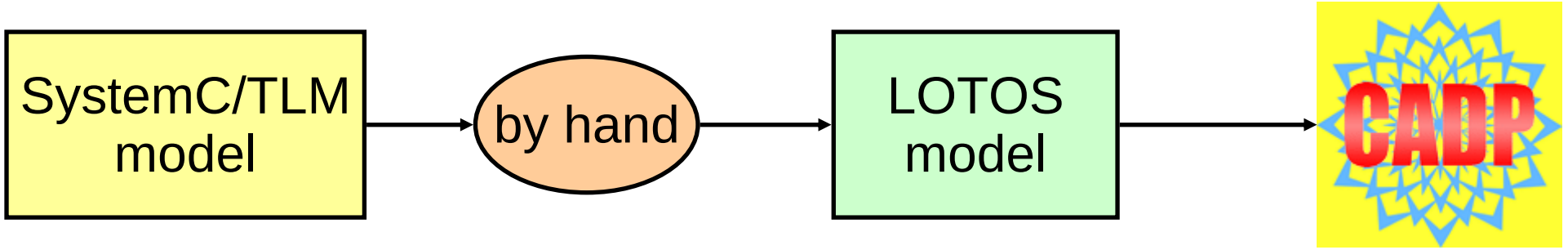
Transaction Level Modeling (TLM)

- Abstraction level for hardware modeling aiming at
  - Early availability
  - Fast simulation
- 2 main sub-levels:
  - **Loosely Timed** (TLM-LT) and **Approximately Timed** (TLM-AT)
- Applications:
    - Functional verification
    - Software development
    - Performance analysis (using TLM-AT)
    - Golden model for hardware verification
- No replacement for low level descriptions (e.g., RTL)
  - No automatic synthesis

Validation at the transaction level

- OSCI's SystemC simulator allows fast simulations
  - But some features are lacking
    - Cannot simulate all possible behaviors (no interactive scheduler)
    - No coverage guarantee
    - No backtracking
    - Cannot check complex properties
- CADP: Construction and Analysis of Distributed Processes
  - Provides: model checking, interactive simulation, …
  - Based on explicit state manipulation
  - Entry points: LOTOS (process algebra) or LTS (explicit graph)

Previous work(1/2)



- Translation rules (+benchmarks) described in:
  - FM'08, O.Ponsini & W.Serwe
  - MEMOCODE'08, C.Helmstetter & O.Ponsini
- Many others similar translations
  - To synchronous automata (LusSy v1: M.Moy, F.Maraninchi, ...)
    - Automatic translation
    - Connected to SMV, NBAC, SCADE prover
  - To Promela/SPIN (C.Traulsen, J.Cornet, ...)
  - To Petri nets, to Finite State Machines, to Kripke structures, ...

Previous work: experiments (2/2)

- Academic benchmarks
  - At most a few hundreds lines of code
  - Using LusSy and SMV (2005): up to 13 processes
  - Using Promela/SPIN (2007): up to 17 processes
  - Using Lotos/CADP (2008): up to 21 processes
- Few realistic case studies
  - "EASY platform" (7 modules, 8 processes, 3500 lines of code)
    - LusSy succeeded to translate this TL model to the SMV input language
    - But SMV failed to prove any property, and to find any bug
- Aim of this paper: a real case study

The BDisp

- Hardware component designed by STMicroelectronics
  - Computes video streams
  - 6 instructions queues with configurable priorities
    - 2 composition queues : real-time jobs (the result is immediately displayed)
    - 4 application queues : non real-time jobs (the result is stored)
  - Connected to
    - CPU + embedded software
    - VTG (video test generator): sends an interruption on every new screen line
- The BDisp SystemC/TLM model
  - one SystemC thread
    + one thread in the CPU, and one thread in the VTG
  - Contains fixed durations
  - About 26,000 lines of code

Objectives

- Develop a LOTOS model of the BDisp
- Check whether CADP can
  - Prove correctness properties
  - Help to find errors
- We address the control part of the BDisp
  - Mainly: arbitration of the instruction queues
  - Complete removal of the graphical operations
- Abstraction of the timing annotations
  - Does the correctness of the BDisp SystemC/TLM model depend on the fixed durations

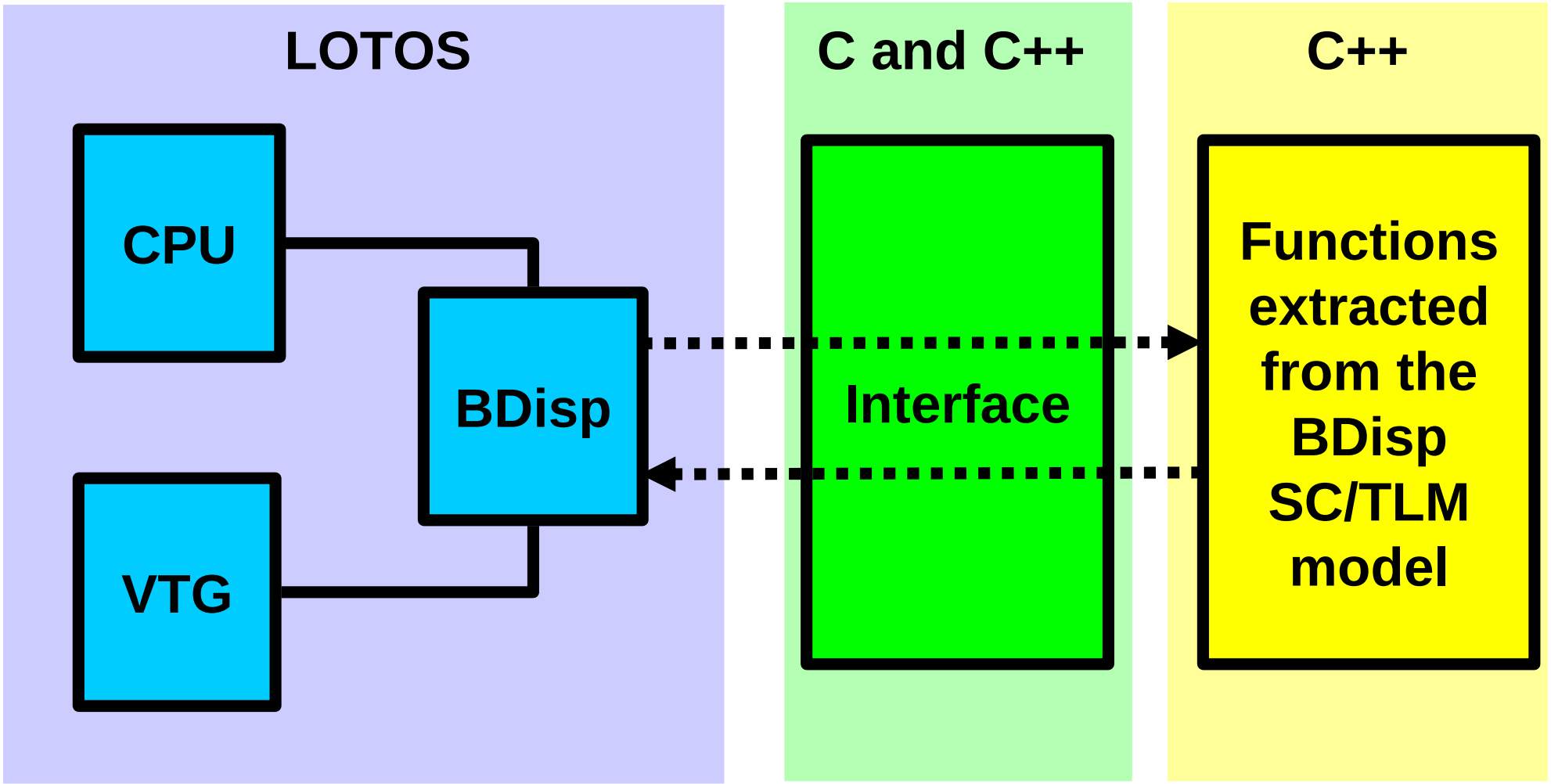
Outline

- Introduction
- Overview of the BDisp LOTOS model
- Abstractions and optimizations
- Experimental results
- Conclusion and future work

Translating the BDisp into LOTOS

- A complete translation would require too much work
- Different handling of different parts of the code
- Communication code
  - Concerns: transactions, SystemC events, shared variables
  - **Translated to LOTOS according to systematic rules** [FM'08]
- Local computations
  - Concerns: sequential control, data manipulations, …
  - **The LOTOS model imports C++ code of the original model**
    - **BDisp access: execution of the corresponding C++ code**
  - Write functions to store and compare the state of the C++ types

Architecture of the LOTOS model



Compact representation in C of the BDisp state

- `BDisp`: C++ class describing the BDisp (provided by STM)
  - This class is memory consuming: ~40 kilobytes
  - Cannot be modified without modifying all the C++ code
  - No copy, no hash, and no comparison functions
- `C_State`: C type to store efficiently a BDisp state
  - Store only relevant data

    (e.g., parameters for graphical operations are not stored)
  - 2 conversion functions
    - LOTOS (C) to SystemC (C++)
    - SystemC (C++) to LOTOS (C)
  - Copy and comparison using `memcpy()` and `memcmp()`

Interface between TLM and LOTOS

- LOTOS code

```
let new_state:Lotos_state = compute_X(old_state)
```

- Interface code

```
C_state compute_X(C_state state) {
```

1. Expand the C state to a C++ state (~original class)
2. Call the corresponding C++ method
3. Convert the C++ state to the C compact representation

```
}
```

- C++ code extracted from the SystemC/TLM model
  - Contains a method `void BDisp::compute_X() {…}`
  - When a communication is encountered (e.g., transactions), we split the method in smaller methods without SystemC/TLM code

Outline

- Introduction
- Overview of the BDisp LOTOS model
- Abstractions and optimizations
- Experimental results
- Conclusion and future work

Abstractions reducing the graph size

- Goal: validate BDisp synchronization issues
- Focus on the control part
  - Instruction node: we keep only informations related to arbitration
- Instruction queues
  - Instruction nodes are generated when read by the BDisp

    (instead of generated when written by the CPU)
  - So at most one instruction node per queue is stored at a time

Reduction of the state size without loss of information

- BDisp state size reduced to 52 bytes
  - Some ‘int’ variables replaced by ‘bool’ (`sc_signal<int>` simulates faster than `sc_signal<bool>` due to template specialization)
  - Use of bit fields
  - Removal of padding bytes
- The whole system state was still larger than 1000 bytes
  - Reduced to 104 bytes after changing how the BDisp is accessed

Accessing the BDisp state: 1st version

- Usual solution for shared variables: sending the state

```
process User[READ,WRITE,...

...
READ ?a:T;
let b:T = modify(a) in
let c:Bool = test(b) in
WRITE !b;
...
```

T

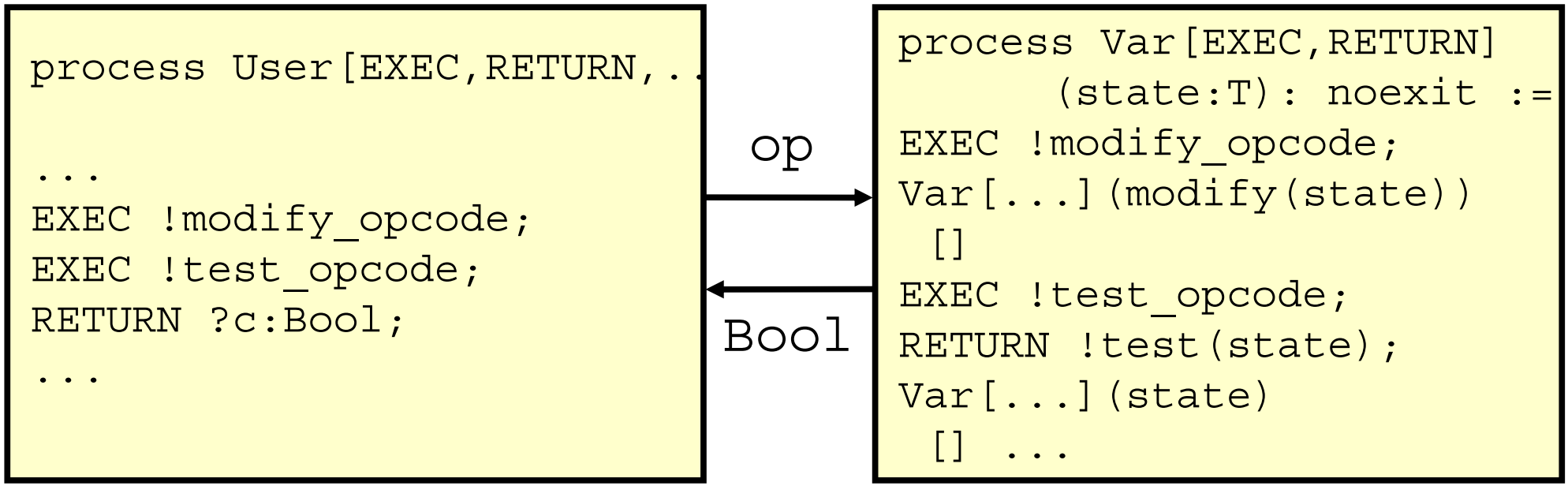
T

```
process Var[READ,WRITE]
        (state:T): noexit :=

WRITE ?new:T;Var[...](new)
  []
READ !state; Var[...](new)

endproc
```

- Problem:
  - Each process contains one local variable of type T
  - A state of the BDisp LOTOS model contained 17 copies of the BDisp state

Accessing the BDisp state: 2nd version

- Second solution: sending the operations



- Only one variable of type T for the whole system
- Problem: the number of transitions may increase
  - Solution: merge operations (*modify_and_test_opcode*)

Outline

- Introduction
- Overview of the BDisp LOTOS model
- Abstractions and optimizations
- Experimental results
- Conclusion and future work

Translation results

- We developed a LOTOS model of the BDisp
  - Less than 2 months of work
  - 1000 lines of LOTOS
  - 2500 lines of C/C++ written manually
  - 5500 lines of C++ code reused (among 26,000 lines)
    - Some changes to separate local computations and commnunication code
    - Minor change to make the BDisp C++ code compatible with 64 bit machines

First verification results

- Interactive simulation with backtracking
- Generation of the full labeled transition system (LTS)
  - Not possible using less than 16 GB of RAM
  - Up to 155,000,000 states and 371,000,000 transitions
- On-the-fly reduction did not help (reductor tool of CADP)
- Compositional verification cannot be applied
  - The SystemC/TLM description is too monolithic

Verification scenarios

- Verification scenarios and property checking
  - "verification scenario": restriction of some inputs to concentrate on something useful (e.g., trigger two queues, then stop one)
  - 10 verification scenarios and 5 correctness properties
  - Possible to generate and reduce the LTS of the scenarios
- Property checking returned one unexpected result:
  - one property was wrong on the untimed version,

    but correct one the original timed version
  - Can be replayed on the original SystemC/TLM model
    - Requires an interactive SystemC scheduler (SCRV)
    - Requires to remove some "`wait`(*duration*)" statements

Conclusion

- Possible to use CADP tools on the BDisp, which is an industrial case study
- Too large state space to be generated completely
- Successful verification of scenarios representing each a large set of behaviors
- Found a synchronization error in the untimed version of the BDisp model
- Started discussion with STMicroelectronics about a new case study (replacement of the BDisp in new SoCs)

Reducing the translation effort

- Still much manual work to connect SystemC/TLM with CADP
- What could be automated, using a C++ frontend:
  - The generation of the LOTOS code corresponding to the communications
    - Systematic rules have been described in previous papers
  - The compact C representation and the interface code
    - Reducing the state size may require static analysis or human help to bound integer variables
- However, it seems difficult to automate the abstractions