Menhir: Generic High-Speed FPGA Model-Checker

Emilien FOURNIER
Emilien.fournier@ensta-bretagne.org

Ciprian TEODOROV
Ciprian.teodorov@ensta-bretagne.fr

Loic LAGADEC
Loic.lagadec@ensta-bretagne.fr
• Model-checking
  • Generic, intuitive automated formal method..
  • .. but limited by the state-space explosion problem

• New trend: Decomposition of the verification task
  ➢ Trades-off memory resources for computation time

• Large research effort to accelerate model-checking
  • Shared Memory Systems, Distributed Systems, SIMD

Significant speedups, but no game-changing breakthrough
• Hardware acceleration
  • PHAST : safety model-checker, 200x speedup
  • FPGASwarm : Swarm safety model-checker, 1000x speedup

• Problems
  • Different hand-coded HDL models
  • No support for existing modeling languages
  • Implicit interleaving between the semantics and the algorithm

Difficult to use in practice and the results are hard to compare
• Isolate the model semantics from the verification engine
  • Support for off-the-shelf modeling languages
  • Easy to change the verification algorithm

• Parametric verification engine
  ➢ Multiple algorithms: continuum from exhaustive to partial verification
  ➢ Allow precise performance characterization

• Menhir prototype [today]: 3 languages & 6 verification algorithms
Algorithm

• Definitions
  • F: Frontier states to be processed
  • K: Known states
  • N: Frontier’s states Neighborhood

• Initialisation:
  • Initial states added to the Neighborhood

• Fixed-point iterations unrolling the state-space

```python
def safety_checker (m : M) : bool :=
  \mathcal{K} ← \emptyset
  \mathcal{F} ← \emptyset
  \mathcal{N} ← m.\text{initial}
  \text{do}
    \text{if } \exists \ n ∈ \mathcal{N}, \ ¬ m.\text{is_safe}(n) \text{ then}
      \text{return false}
    \text{\mathcal{K}, \mathcal{F} ← \mathcal{K} \cup \mathcal{N}, \mathcal{N} \setminus \mathcal{K}}
    \mathcal{N} ← \{ \ n \ | \ \forall \ x ∈ \mathcal{F}, \ n ∈ m.\text{next}(x) \ \}\n  \text{while } \mathcal{F} \neq \emptyset
  \text{return true}
```
• Pipeline structure

• Model Frontend
  • Isolated through the **Generic Language Interface** (GLI)
  • Encapsulates the model and the property
  • The only variable part between verification tasks

• Verification engine
  • Flexibility through generic interfaces
  • IKnown : Hardware set representation
  • IFrontier : Hardware priority queue representation
• 6 safety algorithms implemented
  • Exhaustive safety
  • Partial safety (bitstate hashing)
  • Bounded model-checking
Experimental setup

• Zynq 7020
  • Dual-core arm A9 CPU, 666MHz
  • 7-series FPGA, 100MHz
    • 85K Logic Cells
    • 4.9Mb BRAM

• Parametric model

\[
def\ nbits\ (w\in\mathbb{N}^+) : \quad \mathcal{M}_H\ C\ := \\
\langle\text{initial} \leftarrow \{ (n,T) | \forall i\in[0,w), \ n_i=0 \} , \\
\text{next} \leftarrow \lambda s, \{ (n,T) | \exists i\in[0,w), \ n_i=-s_i \} \rangle
\]
Experimental setup

• Baseline :
  • Divine3 model-checker running on one arm-A9 core
  • OBP2 running on the arm

• Scenarios :
  • EMI-UML
    • C-compiled baremetal interpreter for UML
  • DVE
    • C-compiled model from DVE
    • Used by Divine3
  • GLI native
    • VHDL handwriten model
    • 1 Clk « response time »
Results:
Exhaustive verification
Results:
Exhaustive verification

Divine / DVE-SoC
Divine / GLI-H

20/10/2020
FOURNIER Emilien  Emilien.fournier@ensta-bretagne.org
Results: Algorithms comparison
Conclusion

• Highly modular hardware model-checker

• 50x speedup vs high-performance software model-checker (Divine 3)

• Future work:
  • Performance optimisation
  • LTL model-checking support