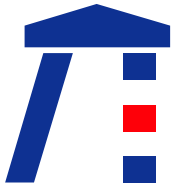
# A Computational Model for SAT-based Verification of Hardware-Dependent Low-Level Embedded System Software

Bernard Schmidt, Carlos Villarraga, Jörg Bormann,

Dominik Stoffel, Markus Wedler, Wolfgang Kunz



TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Yokohama, 1/25/2013

# Content

- Motivation
- Related Works
- Model Generation
  - Basis: Abstract HW/SW Model
  - Flow
- Advantages of Model
- Experiment
- Conclusion / Future Work

# Motivation

- Embedded System

  - Close interaction between HW and SW
  - Examples: drivers, communication structures

- Goal
  - Formal verification (FV) of combined HW/SW behavior

- Objective of this work
  - Computational model and algorithms for FV of *hardware-dependent*, low-level software

# Related Works

- Large research body on SW verification
- Our focus: New computational model for *hardware-dependent* SW


- [2] R. Jhala and R. Majumdar, “Software model checking,” *ACM Comput. Surv.*, vol. 41, pp. 21:1–21:54, October 2009.
- [3] T. Ball, A. Podelski, and S. K. Rajamani, “Boolean and Cartesian Abstraction for Model Checking C Programs,” in *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS 2001. London, UK: Springer-Verlag, 2001, pp. 268–283.
- [4] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar, “The software model checker Blast: Applications to Software Engineering,” *Int. J. Softw. Tools Tec*
- [5] P. Godefroid, “M *CM SIGPLAN-SIGACT Sympo-sium on Pri* 174–186.
- [6] K. Havelund and *tional Journal on Software Tools for Technolog*
- [7] G. J. Holzmann,
- [8] D. Babic and A. *h International Conference on Software Engineeri*
- [9] F.Ivancic, Z. Yar *oftware verification,” in Proc. In- ternational Symp*
- [10] E. Clarke, D. K *d Model Checking,” in Proceedings of the* *03, pp. 368–371.*
- [11] D. W. Currie, A *he 37th Annual Design Automation Conference*, ser. DAC ’00. New York, NY, USA: ACM, 2000, pp. 130–135.
- [12] B. Schlich, “Model checking of software for microcontrollers,” *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 4, pp. 36:1–36:27, Apr. 2010.
- [13] C. S. Păsăreanu and W. Visser, “A survey of new trends in symbolic execution for software testing and analysis,” *Int. J. Softw. Tools Technol. Transf.*, vol. 11, no. 4, pp. 339–353, Oct. 2009.
- [14] T. Arons, E. Elster, S. Ozer, J. Shalev, and E. Singerman, “Efficient symbolic simulation of low level software,” in *Design, Automation and Test in Europe, 2008. DATE ’08*, march 2008, pp. 825 –830.
- [15] M. D. Nguyen, M. Wedler, D. Stoffel, and W. Kunz, “Formal Hardware/Software Co-Verification by Interval Property Checking with Abstraction,” in *Proceedings of the 48th Design Automation Conference*, ser. DAC ’11. New York, NY, USA: ACM, 2011, pp. 510–515.
- [16] D. Große, U. Kühne, and R. Drechsler, “HW/SW co-verification of embedded systems using bounded model checking,” in *GLSVLSI ’06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*, 2006, pp. 43–48.
- [17] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, “Symbolic model checking using SAT procedures instead of BDDs,” in *Proc. International Design Automation Conference (DAC)*, June 1999, pp. 317–320

# State of the Art: Bounded Model Checking (BMC)

E.g.,

- M. D. Nguyen, M. Wedler, D. Stoffel, and W. Kunz, “Formal Hardware/Software Co-Verification by Interval Property Checking with Abstraction” DAC 2011.
- D. Große, U. Kühne, and R. Drechsler, “HW/SW co-verification of embedded systems using bounded model checking” in *GLSVLSI ’*2006.

Basic approach:

Unroll CPU + software in memory

⇒ program flow (instruction sequencing) as given by the program is represented only *implicitly* (by PC + memory)

⇒ program computation also is represented only *implicitly* (by hardware implementation of CPU)

⇒ High computational complexity for realistic designs

# State of the Art: Symbolic Execution

E.g.,

- C. S. Păsăreanu and W. Visser, “A survey of new trends in symbolic execution for software testing and analysis” *Int. J. Softw. Tools Technol. Transf.*, 2009.
- T. Arons, E. Elster, S. Ozer, J. Shalev, and E. Singerman, “Efficient symbolic simulation of low level software” *DATE* 2008

Basic approach

- Enumerate program execution paths and check conditions for verification along these paths by specialized verification algorithms
- Symbolic formulas explicitly represent all possible input scenarios (along path)

⇒ program flow is enumerated *explicitly* (by path enumeration)

⇒ program computation is represented *explicitly* (through symbolic formulas)

⇒ high complexity of symbolic formulas

# State of the Art: CBMC and related work

E.g.,

- E. Clarke, D. Kroening, and K. Yorav, “Behavioral consistency of C and Verilog programs using Bounded Model Checking” DAC 2003

## Basic approach

- Build a SAT formula for the program computation based on CFG unrolling
- SAT formula contains information about control flow only implicitly in terms of functional dependencies between variables of the formula
- *HW-independent* paradigm (high-level programming language)

# State of the Art: CBMC and related work

E.g.,

- E. Clarke, D. Kroening, and K. Yorav, “Behavioral consistency of C and Verilog programs using Bounded Model Checking” DAC 2003

⇒ **Extension of CBMC for HW-dependent software is difficult!**

The computational model (SAT formula):

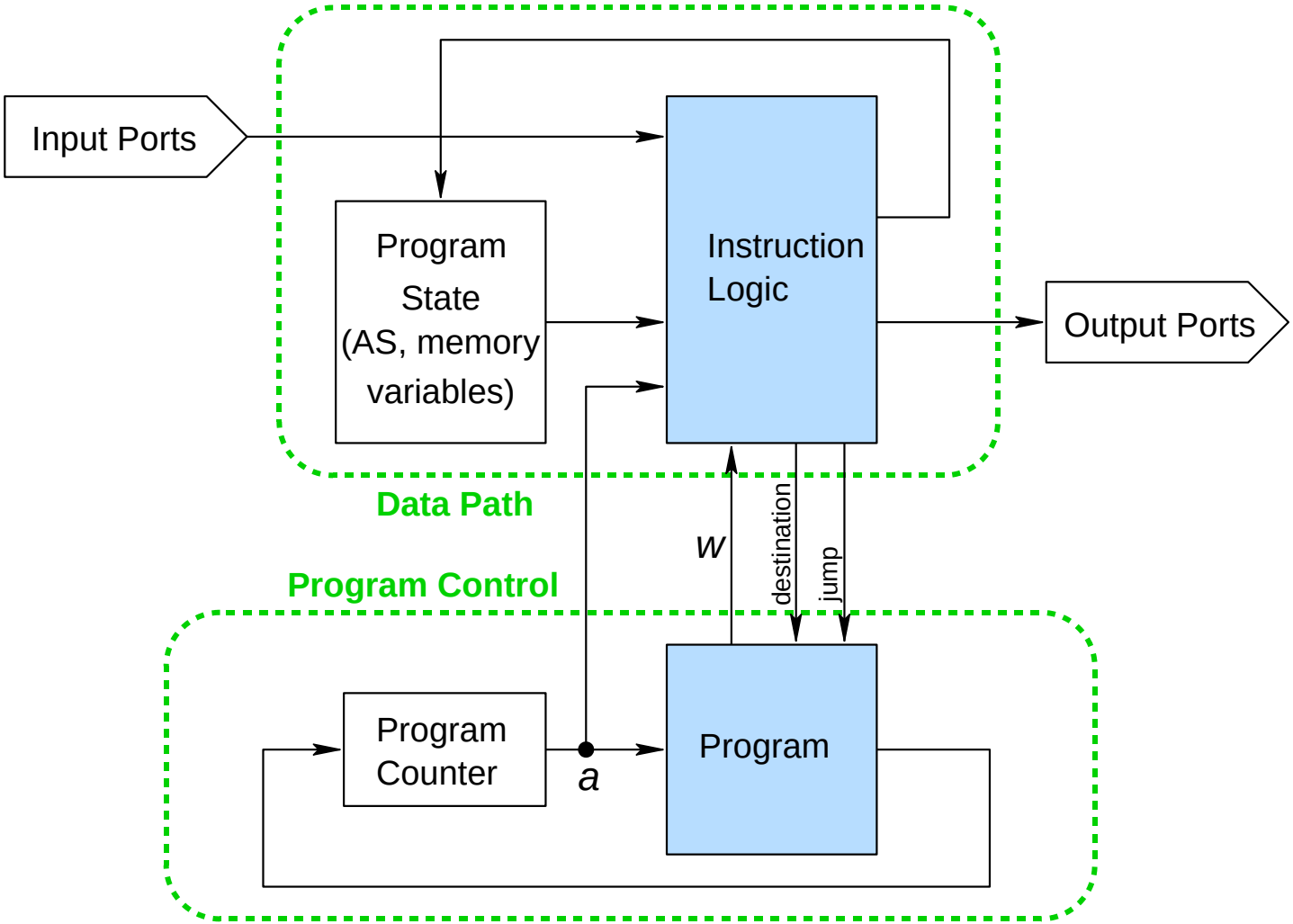
- lacks information about temporal relationships between statements/instructions as is needed when relating program behavior to HW periphery
- cannot be integrated with environment hardware model
- is not compositional

# Our Approach

New hardware-dependent computational model

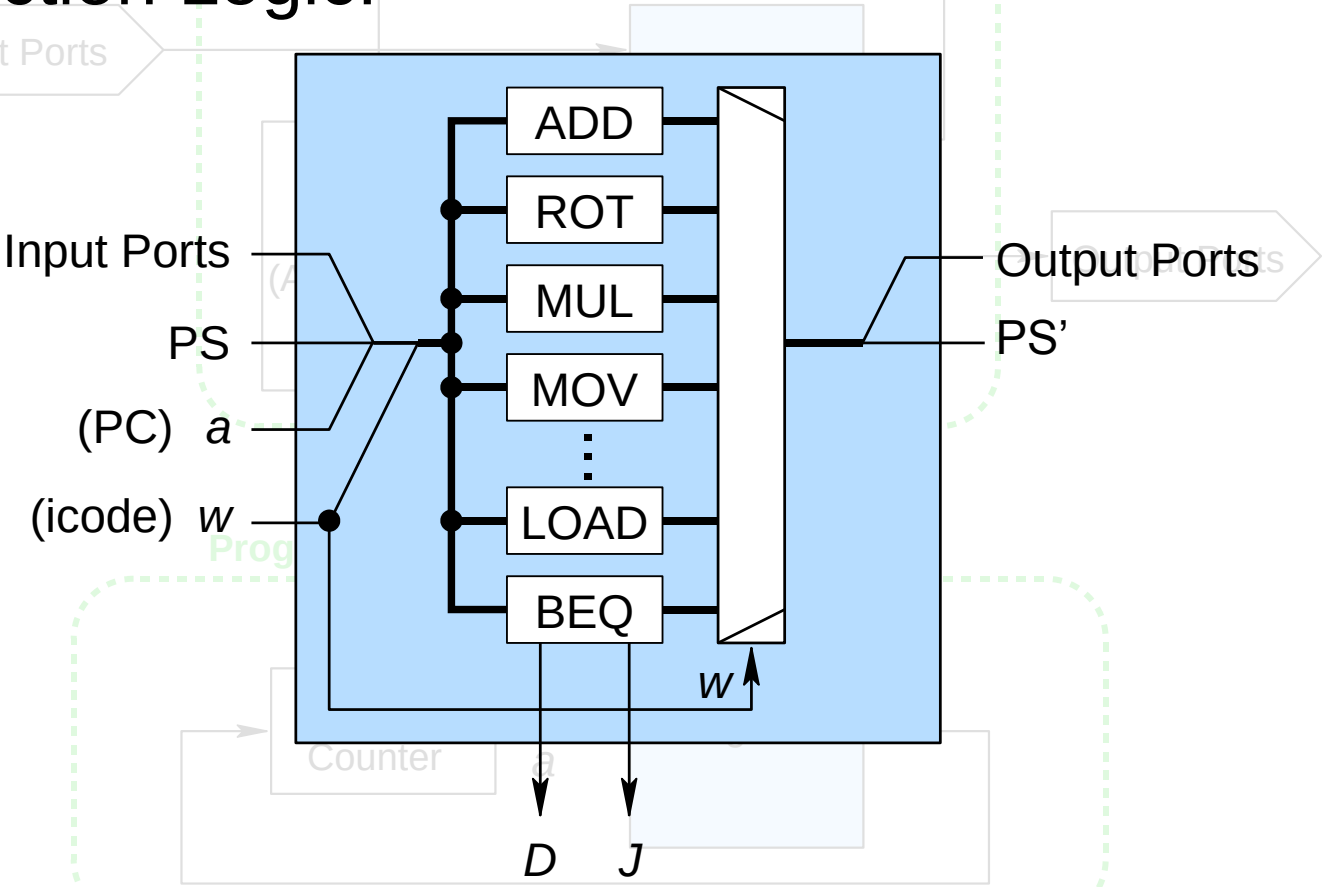
- Program computation should be represented implicitly
  - ⇒ compact model
- Program flow should be represented explicitly
  - ⇒ facilitates SAT reasoning (since global information about global execution paths is explicitly available)
- Representation of the temporal dependencies between instructions and I/O
  - ⇒ enables HW-dependent verification
  - ⇒ enables compositionality of model

# Basis: Abstract HW/SW Model

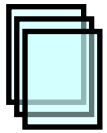


# Basis: Abstract HW/SW Model

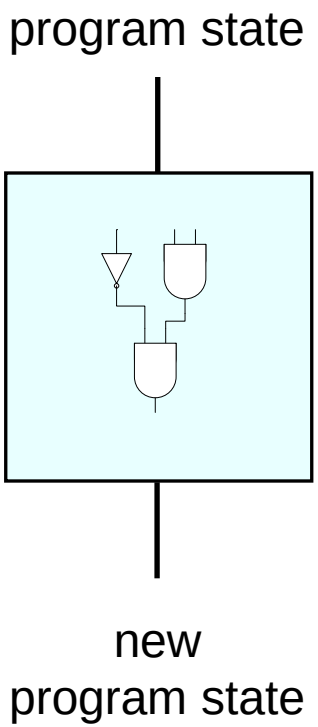
- Instruction Logic:



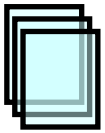
# Instruction Cell



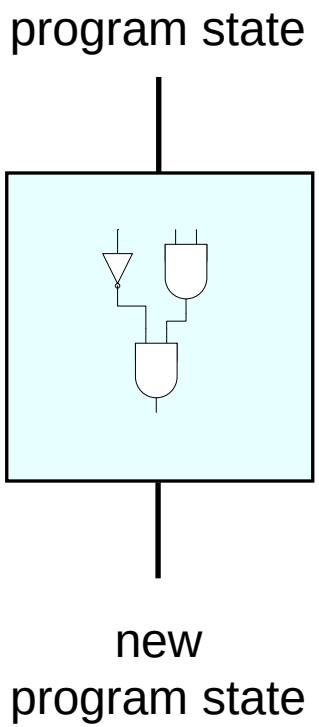
- Abstract model for a CPU instruction
- Hardware-dependent
- Describes the modification of the program state
- Can be formally verified against RTL implementation of CPU



# Program State



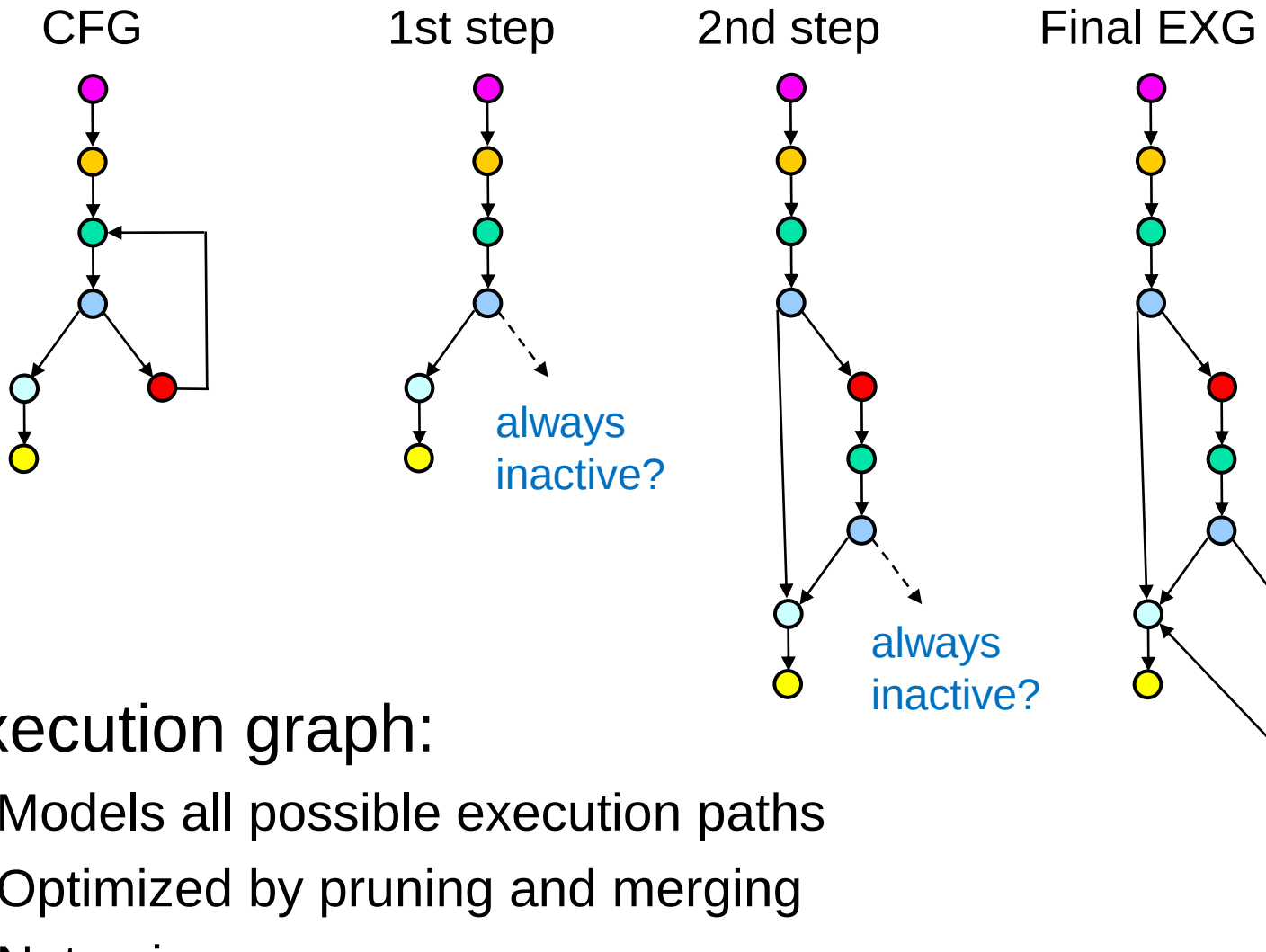
- Registers
- Control signals in the CPU
- Program data in RAM
- “Active” signal (program flow variable)



# *Active* Signal

- Additional 1-bit signal in program state
- Signal is asserted if corresponding program state belongs to active execution path
- Additional logic in instruction cells to handle active signal:
  - Datapath and load/store instructions:
    propagate *active* flag from input state to output state
    (no extra logic required)
  - Branch instructions: propagate *active* flag depending on branch condition from input state to branch target output state. All other output states are set inactive.

# Transform CFG to Execution Graph (EXG)

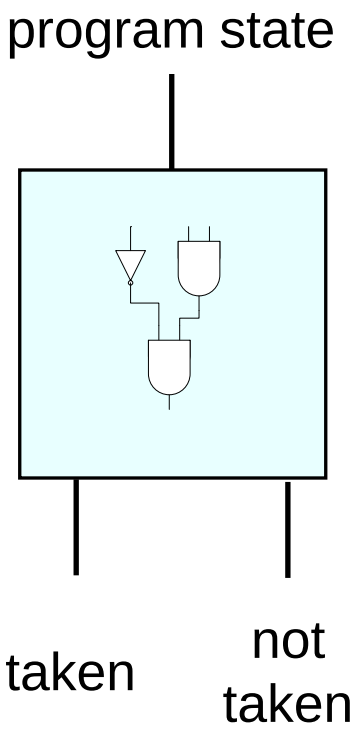



Execution graph:

- Models all possible execution paths
- Optimized by pruning and merging
- Not unique

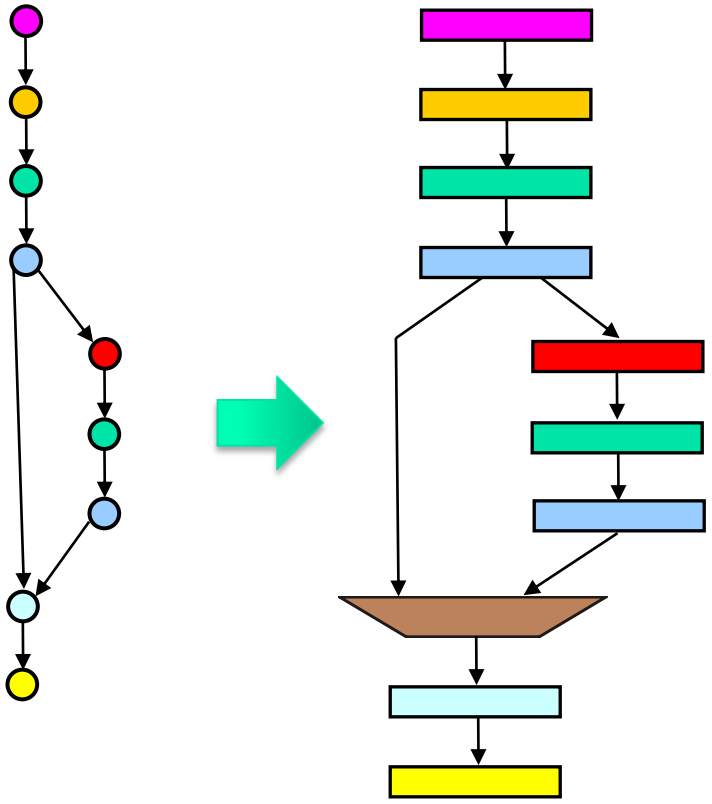
# Verifying if a branch is active

- Branch instruction
  - Propagates “active” signal to exactly one of the branches
- Generated property:
  - Claim: “active” signal is never asserted
  - Property holds: branch never taken
  - Property does not hold: branch is taken in some executions



# Program Netlist (PN)

- PN generation
  - Replace every EXG node by an instruction cell
  - HW-dependent description of program computation
  - Insert merge cells
- Merge cell
  - Merges two paths into one
  - Path is selected by incoming *active* signal



# Advantages of PN Model

**Replacing EXG nodes by instruction cells**

- PN is a combinational circuit

  ⇒ Supports SAT-based FV

- CFG-based unrolling

  ⇒ Explicitly represented program flow

- Concatenation of instruction cells

  ⇒ Implicitly represented program computation

# Advantages of PN Model (2)

**Explicit program flow:**

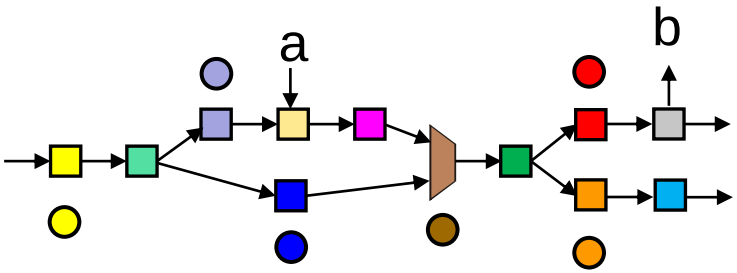
program paths are considered separately

⇒ Simplification of *program computation*

- CPU logic is simplified to single instruction cell
- Machine instruction parameters are constant: individual instruction cell is further simplified by Boolean constraint propagation
- Simplification takes place during PN generation (not during property verification)

# Advantages of PN Model (3)

- *Active* flags:
  - Small set of signals captures global reachability information of entire program
  - Valuation on active signals selects possible execution paths or sets of execution paths
  - Helps SAT solver to take into account reachability without detailed reasoning on all intermediate instructions
- Example:




Property:

A: ● = true

C: if b.active ●

then b=a ●

# Experiment – Software LIN Master (1)

- Industrial automotive design
- 1309 lines of C code
- Send/receive message frames via UART
- Interrupt-driven
- Compiled for SuperH2 based 32-bit CPU, RISC architecture, 5-stage pipeline

# Experiment – Software LIN Master (2)

- Model generation:
  - Interrupt Service Routine (ISR), main program (scheduler) and initialization generated automatically
  - Composed model: assembled manually

| Program component | #instructions CFG | #instructions PN | CPU (s) | mem. (MB) |
|---|---|---|---|---|
| LIN-Init | 225 | 385 | 1.32 | 36 |
| LIN-Scheduler | 85 | 84 | 0.13 | 27 |
| LIN-ISR | 790 | 1138 | 11.00 | 102 |

# Experiment – Software LIN Master (3)

Property verification:

- Composed model: 24001 instructions, 62592 primary inputs
- Verify correct generation of LIN frame
  (header, data, checksum and control signal for UART)
- Commercial HW property checker (OneSpin 360MV)

| **Property** | **CPU(s)** | **MEM(MB)** |
|---|---|---|
| RX frame 4or8 data bytes incl. checksum | 17 | 1641 |
| RX frame 4or8 data bytes wrong checksum | 28 | 1545 |
| TX frame 4or8 data bytes incl. Checksum | 15 | 1584 |
| Wrong PID or not matching ID (8bytes) | 14 | 1566 |

# Conclusion

- Program netlist (PN)
  - New hardware-dependent model for low-level software verification combining the advantages of HW-style bounded model checking and SW-style symbolic execution
  - Automatic PN generation successful for industrial automotive software
- Future Work
  - Equivalence checking
  - Integrate PN with hardware for FV of firmware

# Thank you!

- Questions?

bschmidt@eit.uni-kl.de