Automated Test-Trace Inspection for Microcontroller Binary Code

Thomas Reinbacher[1], Jörg Brauer[2], Daniel Schachinger[1], Andreas Steininger[1], and Stefan Kowalewski[2]

[1] Institute of Computer Engineering
Vienna University of Technology, Austria

[2] Embedded Software Laboratory
RWTH Aachen University, Germany

Binary Code Verification

1. Embedded software mostly not in plain ANSI C
   - side effects, embedded assembler, direct hardware register access
2. Who verified your compiler?
   - GCC 4.3.5 has 8M loc (2.5M C, 1.5M C++, 1.5M Java, 60k ASM ...)
   - Good SW has about 1 error in 250 loc $\rightarrow \frac{8M}{250} = 33k$ flaws
   - Proving correctness of the compiler is typically infeasible
   - Even translation validation is hard (and not really widespread in industry)
3. No source code required (closed source libraries)
4. Binary code is as close as possible to the actual execution

Runtime Verification (RV)

Testing is based on a guess & check paradigm

- Guess a configuration of the program's input
- Check the result of an individual test run

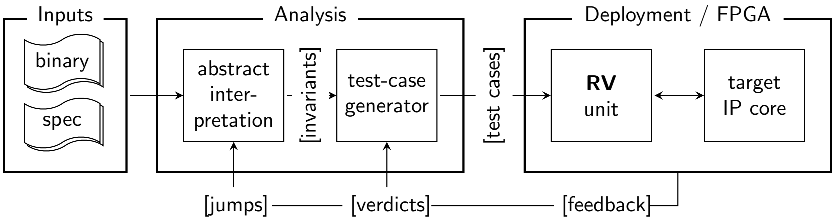
RV bridges the gap between rigorous software verification and dynamic testing

Intuition for our approach

- Use formal methods to derive a set of test cases (guess)
- Use RV to check validity of test cases during execution (check)

CevTes Approach

1. Use AI to derive an over-approximation of the reachable states
2. Find program locations where the specification is violated
3. Backward analysis derives counterexamples (test cases)
4. Interface a hardware unit attached to the SUT to replay a CE and automatically identify spurious warnings



So far

Pure hardware approach [FMICS'11] for a *static* setting ✓

1. Non instrumenting approach
2. Specification in ptLTL, auto-generate observers [Havelund&Rosu; TACAS'02]
3. Observers are generated as VHDL entities
4. Logic synthesis tool translates VHDL entities into a netlist
5. Observer runs in parallel to microcontroller (both on FPGA)

However

- In a *dynamic* setting, specifications may change frequently
- A single run of the synthesis tool may take several minutes ☹

$\leadsto$ Evaluate ptLTL formulae with a μCPU in SW μMonitor
$\leadsto$ Evaluate atomic propositions in HW atChecker

ptLTL Observer

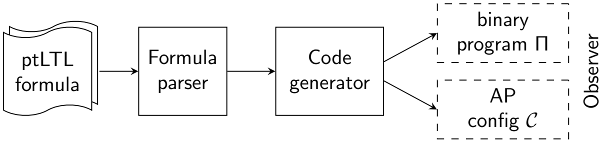
Past time LTL

$$\begin{aligned}\psi \quad ::= \quad & \mathit{true} \mid \mathit{false} \mid AP \mid \neg\psi \mid \psi \bullet \psi \\ & \odot\psi \mid \diamond\psi \mid \boxdot\psi \mid \psi\ S_s\ \psi \mid \psi\ S_w\ \psi\end{aligned}$$

Monitoring operators

$$\psi \quad ::= \quad \uparrow\psi \mid \downarrow\psi \mid [\psi,\psi)_s \mid [\psi,\psi)_w$$

Approach



Synthesize

1. A binary program Π for the μMonitor $\rightsquigarrow$ evaluates ptLTL
2. A configuration $\mathcal{C}$ for the atChecker $\rightsquigarrow$ evaluates *AP*

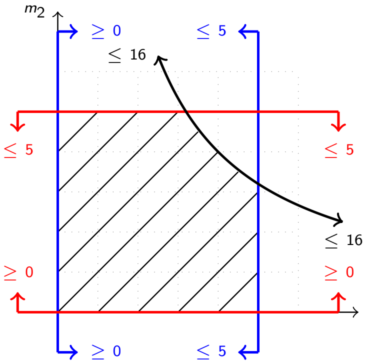
Evaluate Atomic Propositions TVPI constraints

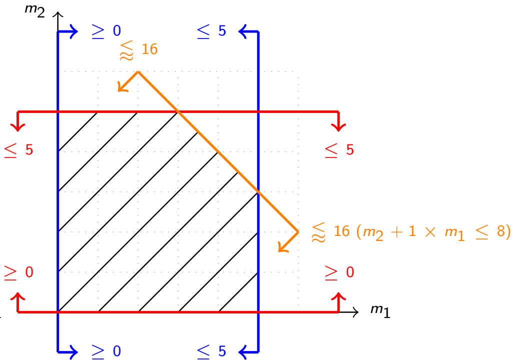
AP are a form of two-variable-per-inequality constraints

$$\alpha \cdot m_1 + \beta \cdot m_2 \bowtie C$$

where:

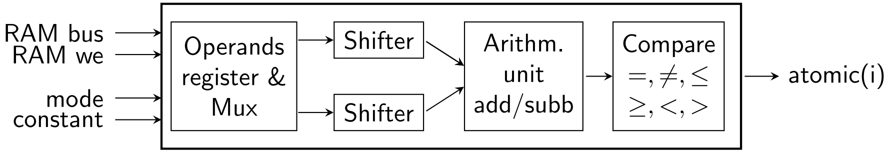
- $\alpha, \beta \in \{0, \pm 2^n \mid n \in \mathbb{N}\}$ m_1, m_2 are locations within RAM
- $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$ $C \in \mathbb{Z}$ is a constant



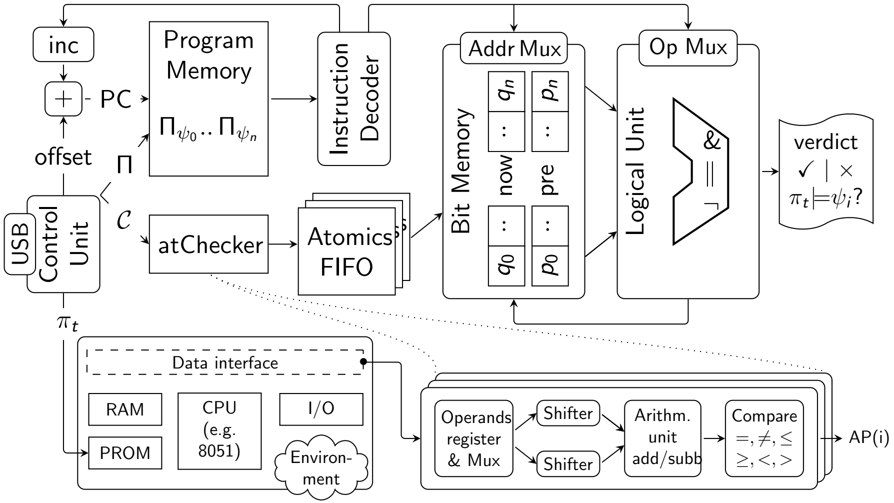


Evaluation of TVPI constraints results in fairly efficient HW units

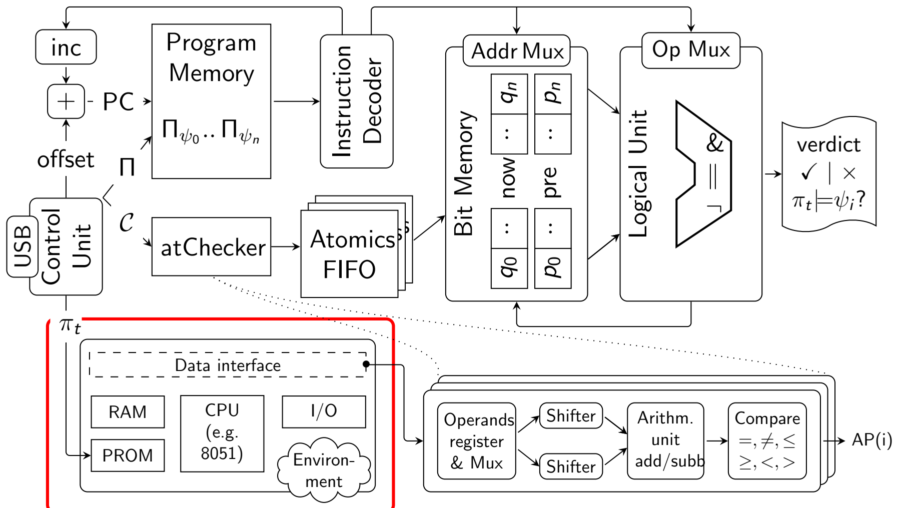
- Ripple carry adder: $\mathbf{Add}(\langle a \rangle, \langle b \rangle, c)$
- Subtraction of $\langle a \rangle - \langle b \rangle$ is equivalent to $\mathbf{Add}(\langle a \rangle, \langle \overline{b} \rangle, 1)$
- Relational operators are similar
- Sensing the memory interface of the SUT (target microcontroller)



System Overview

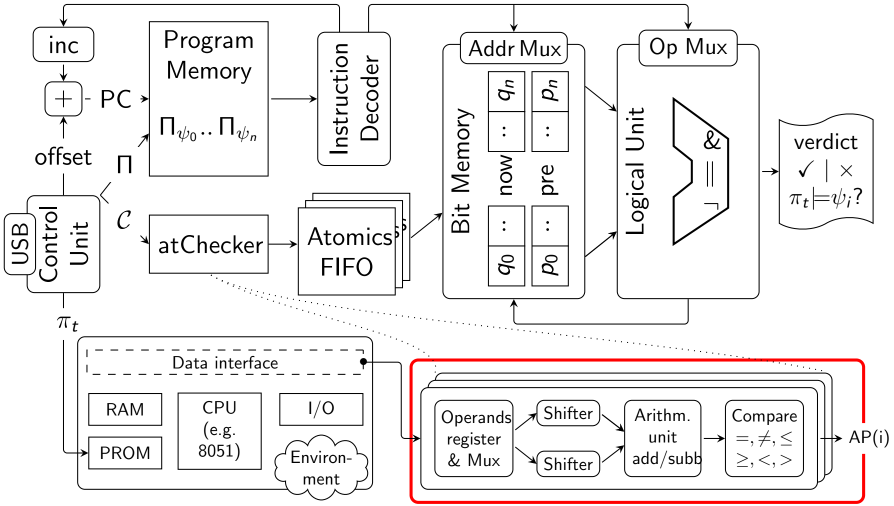


System Overview



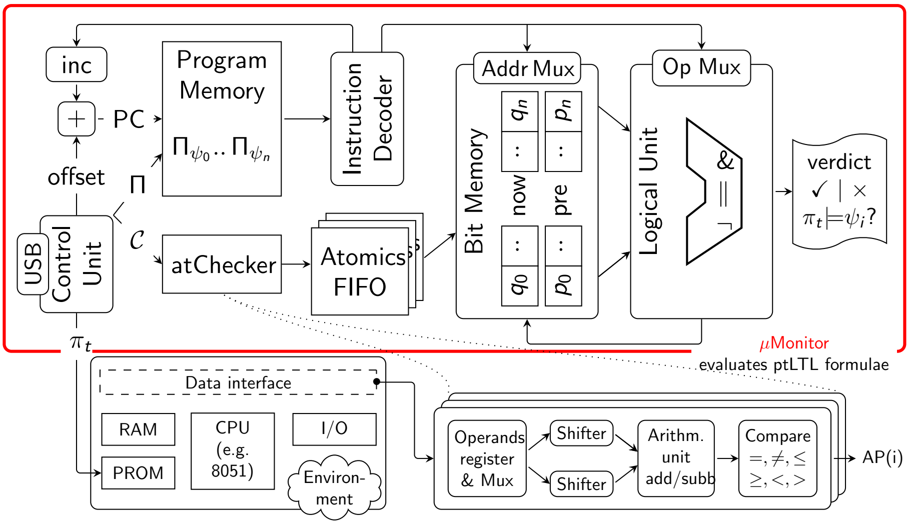
Intel MCS-51 IP Core
runs SUT natively

System Overview



atChecker
evaluates AP (TVPI constraints)

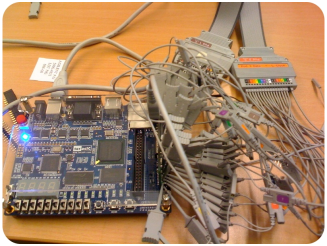
System Overview



Implementation Details

FPGA: Altera Cyclone III EP3C16

Unit	LC	f_{max}
μMonitor	367	145 MHz
atChecker	290	80 MHz
8051 IP-Core	4000	16 MHz



μMonitor

- Instruction set features 16 opcodes to handle full ptLTL
- Each opcode is 3 bytes long

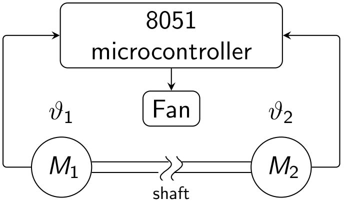
8051 IP-Core

- Unmodified, industrial 8051 IP-Core [www.oreganosystems.at]

Industrial Automation

Digital controller implementation

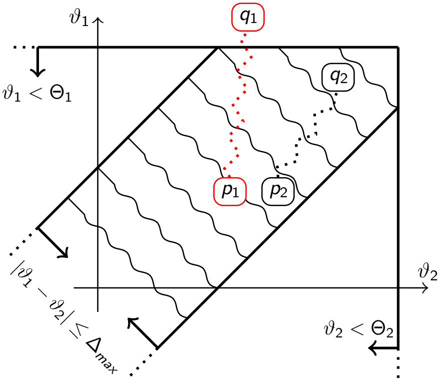
- Temperature control of two DC motors M_1 and M_2
- Motors have max operating temperature, i.e., Θ_1 and Θ_2
- Sanity check $|\vartheta_1 - \vartheta_2| \leq \delta_{max}$
- SUT
  - continuously reads operating temperatures, i.e., ϑ_1 and ϑ_2
  - invokes cooling system when $\vartheta_1 > T_{on_1}$ or $\vartheta_2 > T_{on_2}$



Example — Industrial Automation

```
SINT8 temp1, temp2, fanOn;
 ...
void main(){
  while(true) controlLoop();
}
void controlLoop(){
  if ((temp1>T1)||(temp2>T2)){
    fanOn = controlAlgo();
     ...
    setCooling(fanOn);
  }
   ...
}
void updateTemp(){
  temp1 = readTemp(M1);
  temp2 = readTemp(M2);
}
```

$\sim$ 250 lines of Keil μVision3 C-code



$$\psi: \quad \mathrm{Inv}(|\vartheta_1 - \vartheta_2| \leq \Delta_{max}) \quad \wedge$$
$$\uparrow(\mathrm{fanOn} = \#\mathrm{F\_ON}) \Rightarrow [\vartheta_1 > T_{on_1} \vee \vartheta_2 > T_{on_2}\,;\ \vartheta_1 \geq \Theta_1 \vee \vartheta_2 \geq \Theta_2)_s$$

- atChecker will trace state changes of `fanOn`, `temp1`, `temp2`
- μMonitor will check the validity of ψ

$\boxed{p_2} \rightsquigarrow \boxed{q_2}$ ✓ is a valid wrt. to ψ

$\boxed{p_1} \rightsquigarrow \boxed{q_1}$ ✗ is invalid wrt. to ψ, hence, detected by μMonitor

Conclusion & Future Work

Non-intrusive monitoring framework for ptLTL

- Atomic propositions evaluated in hardware
- Validity of ptLTL specifications determined by a μCPU (μMonitor)
- μMonitor runs in parallel to SUT
- ptLTL specifications are synthesized into μMonitor programs
- SUT is an unmodified off-the-shelf IP core running the binary code under investigation

More recent and future work

- CFG recovery a priori instead of on-the-fly (see EMSOFT'11)
- Checking real-time properties (in progress)
- Orthogonal but related future work: automatic test-case / trace generation

CevTes $\Leftrightarrow$ Framework for Testing / RV of Embedded Software
`[http://ti.tuwien.ac.at/ecs/research/projects/cevtes]`