Combining SMT with Theorem Proving for AMS Verification

The best of both worlds

Yan Peng & Mark Greenstreet

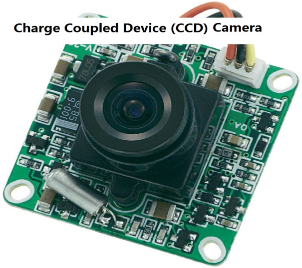
University of British Columbia
Vancouver, BC

NASA Formal Methods Symposium, April 29, 2015

Outline

- AMS verification
  - AMS designs are ubiquitous
  - Motivation
  - Contributions
- Integrating SMT with theorem proving
- Proving global convergence for a digital PLL
- Conclusion

AMS designs are ubiquitous



Motivation

- Different design models and methods, thus more complex to model.
- Time-scales vary widely from sub-picosecond to milliseconds or seconds, makes it harder to simulate.
- Abstractions on time-scales can hide bugs in the implementation.

~~Simulation?~~ → Formal methods → Analytical approach

- Circuits are intended to be correct
- Verify the intuitive argument

Motivation - the best of both worlds

- AMS design verification requires huge amounts of arithmetic reasoning and reasoning about sequences which requires induction.
- SMT and theorem proving are complimentary to each other:
  - SMT - Excellent performance in linear and non-linear arithmetic reasoning.
  - Theorem proving - Strong support for induction and systematic model & proof management.
- We are using ACL2 and Z3 as our prototyping tools.

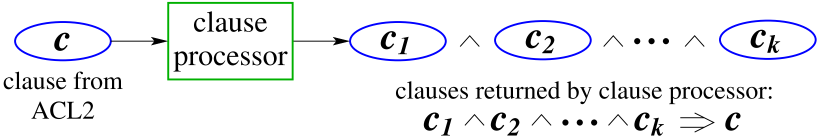
Contributions

- *We demonstrate the value of combining SMT with theorem proving for cyber-physical system verification with a focus on utilizing the non-linear arithmetic capabilities.*
- The first integration of an SMT solver into the ACL2 theorem prover.
- A software architecture for integrating a SMT solver with a theorem prover that addresses many technical challenges.
- A reusable recurrence model for a state-of-the art digital PLL.
- A proof of global convergence for the digital PLL.

Outline

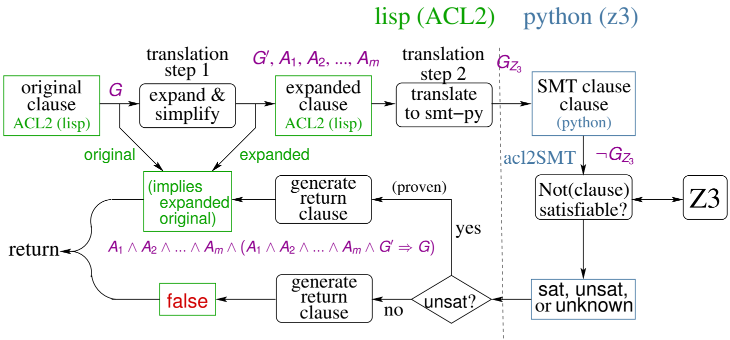
- Characterize AMS verification problems
- Integrating SMT with theorem proving
  - Architecture
  - Technical issues
  - What's trusted?
- Proving global convergence for a digital PLL
- Conclusion

Clause processors in ACL2



- A clause processor takes a goal and decomposes it into a conjunction of subgoals. Each subgoal is a called a clause.
- ACL2 supports two kinds of clause processors: verified and trusted.
  - verified - the correctness of the clause processor is proven within ACL2.
  - trusted - the results of the clause processor are accepted without proof.
- We integrate Z3 into ACL2 as a trusted clause processor.

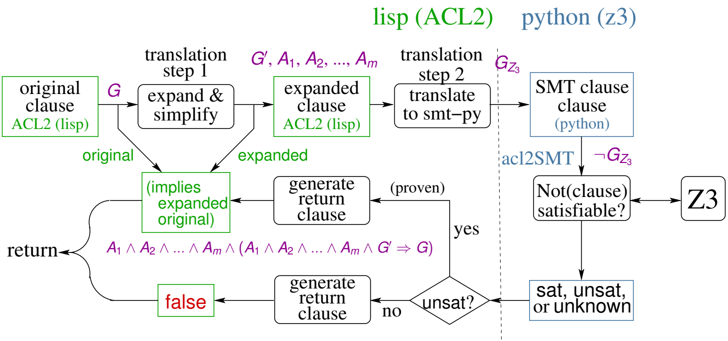
Architecture of `Smtlink`



$(\bigwedge_{i=1}^{m} A_i)$	; each A_i verified by ACL2
$((\bigwedge_{i=1}^{m} A_i) \wedge G') \Rightarrow G$	; verified by ACL2
$G_{Z_3} \Rightarrow G'$	; we trust translation step 2
G_{Z_3}	; verified by Z3

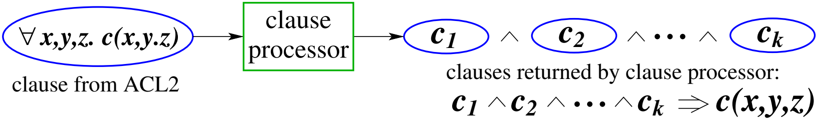
$$\frac{}{G}$$

Architecture of `Smtlink`



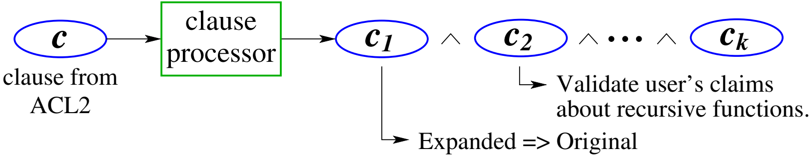
- All methods of the underlying SMT solver are invoked through methods of an object called acl2SMT.
- This architecture is generic enough to be combined with other SMT solvers by extending this class.

Technical issues: reals vs. rationals



- Challenge: ACL2 has rationals and Z3 has reals.
  - In ACL2, $\neg\exists x.\ x^2 = 2$ is a theorem.
  - In Z3, $\exists x.\ x^2 = 2$ is a theorem.
- Solution: only use Z3 to prove propositions where all variables are universally quantified.
  - E.g. we don't support `defun-sk`, `exists`, `forall`, etc.
  - This is enforced syntactically in our clause processor.

Techinical issues: user defined functions

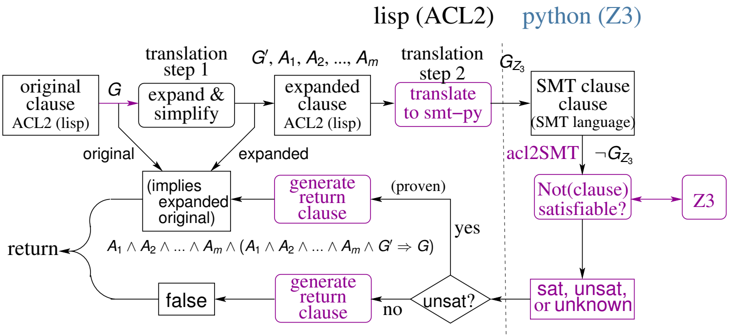


- Challenge:
  - ACL2 supports arbitrary lisp functions.
  - Z3 functions are more like macros (no recursion).
- Solution:
  - Set up translation for a basic set of functions.
  - Expand non-recursive functions.
  - Expand recursive functions to bounded depth.
  - Deeper calls are declared to return an arbitrary value of the appropriate type.
  - Expansion done on ACL2's representation: can verify correctness.

Other issues:

- Claims can contain non-polynomial terms.
  - Replace offensive subexpression with a variable.
  - User adds constraints about these variables.
  - These constraints are returned as clauses for ACL2 to prove.
- ACL2 may need hints to discharge clauses returned from the clause processor.
  - Solution: nested hints.
  - These hints tell the clause processor what hints to attach to returned clauses.
- These features provides a very flexible back-and-forth between induction proofs in ACL2 and handling the details of the algebra with Z3.

What's trusted?



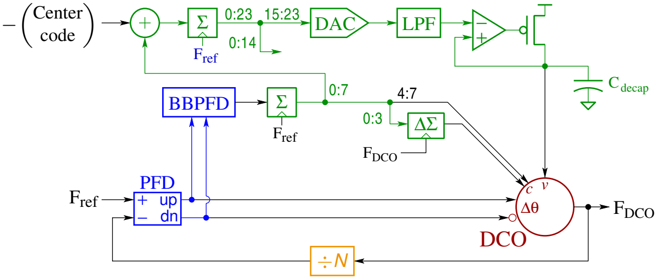
	translation	others	expansion & simplification
LOC(fraction)	656(39%)	453(27%)	584(34%)

- Translation code is straight forward and easy to check.
- Others are mostly boilerplate code for integrating general clause processors.

Outline

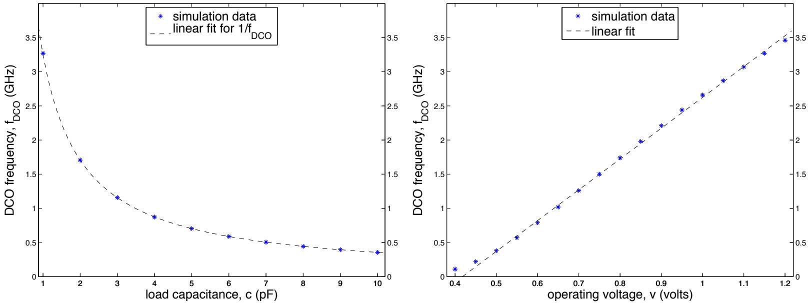
- Characterize AMS verification problems
- Integrating SMT with theorem proving
- Proving global convergence for a digital PLL
  - The digital phase-locked loop
  - Modeling the digital PLL
  - Prove global convergence
- Conclusion

A state-of-the-art Digital PLL [CNA10]



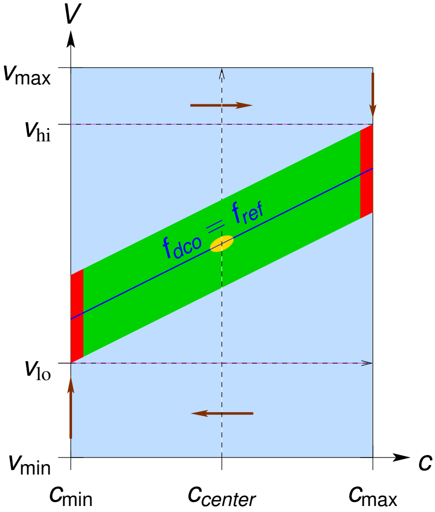
- A *PLL* outputs a signal with a frequency that's N times of the input signal. The output should also aligns the input in phase.
- Three state variables:
  - capacitance setting (digital)
  - supply voltage (linear),
  - phase correction (time-difference of digital transitions).

Modeling the digital PLL



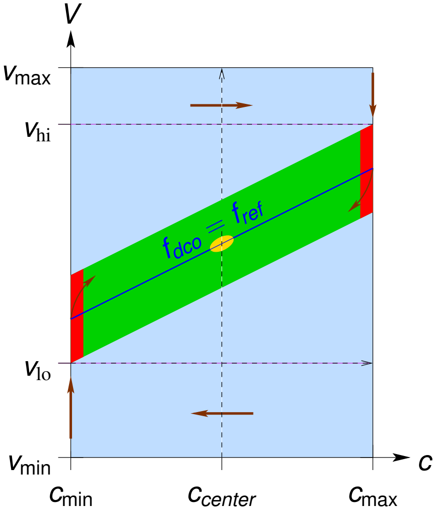
- From Spectre simulation, $f_{dco}(c, v) \approx \frac{1+\alpha v}{1+\beta c} f_0$.
- We use recurrence function to model the circuit behaviour: $[c(i+1), v(i+1), \phi(i+1)] = next(c(i), v(i), \phi(i))$.

The proof - the high level description



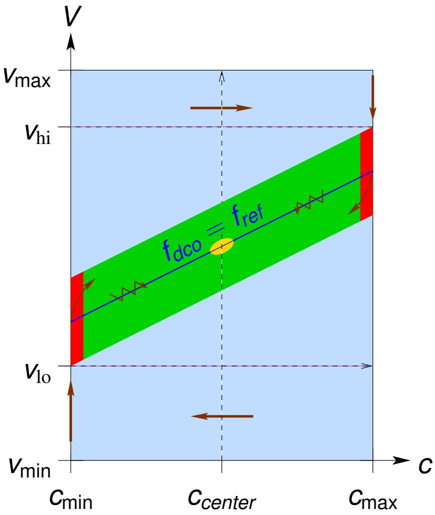
- Initial to wall, Z3
- Climb the wall, Z3

The proof - the high level description



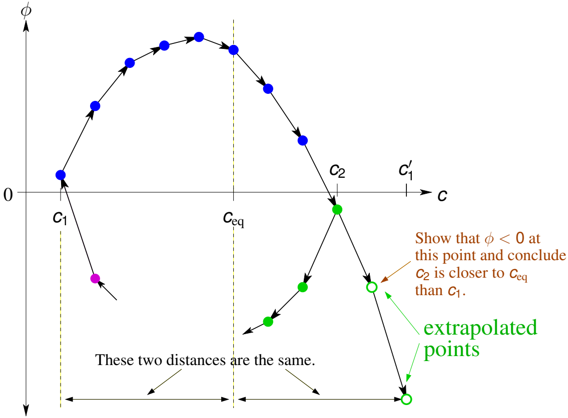
- Initial to wall, Z3
- Climb the wall, Z3
- Leave the wall, bounded model checking, Z3
  - Unwind the recurrence for bounded depth. Z3 shows that all points have left the wall.

The proof - the high level description



- Initial to wall, Z3
- Climb the wall, Z3
- Leave the wall, unbounded model checking, Z3
- Spirals along the $f_{dco} = f_{ref}$ line to the middle yellow region, the most technical theorem, ACL2

The proof - the main theorem



- When we encounter heavy non-linear arithmetic reasoning, we use `Smtlink`.
- `Smtlink` solves the the key polynomial inequality that sets the foundation for further inequalities to hold.

Some statistics

- 13 page long hand-written proof.
- 75 lemmas, 10 of which were discharged using the SMT solver.
- Of those ten, one was the key, polynomial inequality from the manual proof.
- ACL2 completes the proof in a few minutes running on a laptop computer.
- We found one error in the process of transcribing the hand-written proof to ACL2.

Conclusion

- We built a sound and extensible integration of an SMT solver into a theorem prover.
- We demonstrated the effectiveness of the approach by proving global convergence for a state-of-the-art AMS design.
- Benefits we can get from analytical approach:
  - Ranges for initial states, parameters and etc.
  - Proofs are easy to extend. (E.g. insert uncertainty into the model, or minor revision on the model)

Conclusion

Future work:

- Extend our integration and contribute to the ACL2 community.
  - Integrate all code into ACL2.
  - Fully exploit `Smtlink` to shorten my proof.
  - Automatically generated hints.
  - Checked counterexample reports.
- Automate AMS proofs.
- Example problems from other physical domains: medical control systems, machine learning problems and etc.

References

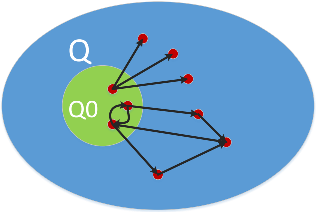
Ghiath Al-Sammane, Mohamed H Zaki, and Sofiène Tahar, *A symbolic methodology for the verification of analog and mixed signal designs*, Proceedings of the conference on Design, automation and test in Europe, EDA Consortium, 2007, pp. 249–254.

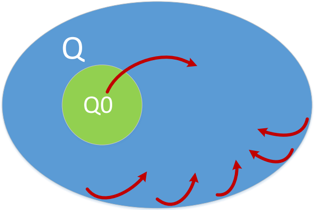
J. Crossley, E. Naviasky, and E. Alon, *An energy-efficient ring-oscillator digital pll*, Custom Integrated Circuits Conference (CICC), 2010 IEEE, 2010, pp. 1–4.

Leonardo De Moura and Nikolaj Bjørner, *Z3: an efficient smt solver*, Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems (Berlin, Heidelberg), TACAS'08/ETAPS'08, Springer-Verlag, 2008, pp. 337–340.

Matt Kaufmann and J. S. Moore, *An industrial strength theorem prover for a logic based on common lisp*, IEEE Trans. Softw. Eng. **23** (1997), no. 4, 203–213.

Formally characterize AMS verification problem





$\forall s(0) \in Q_0 \forall i \geq 0.\ s(i) \in Q$ $\qquad$ $\forall x(0) \in Q_0 \forall t \geq 0.\ x(t) \in Q$

Digital	Analog	AMS
$s(i+1) = next(s(i), in(i))$	$\frac{dx}{dt} = f(x, in, u)$	$\begin{aligned} \frac{dx}{dt} &= f_q(x) \\ q(i+1) &= d(q(i), th(x)) \end{aligned}$

Two features in formal model of AMS designs:

- Large non-linear arithmetic formulas
- Properties for sequences of states

SMT&Theorem proving

	SMT	**Theorem proving**
What	Satisfiability Modulo Theory	Computer aided theorem proving
Strength	Powerful (non)linear arithmetic solver and others	1. Systematic proof management 2. Induction proofs
Weakness	1. Lack of induction 2. Lemmas don't connect	Manual and tedious proofs
Tool	Z3[DMB08]	ACL2[KM97]

Geometric series theorem

Theorem (Geometric Sum)

Suppose $r \in \mathbb{R}$, $n \in \mathbb{N}$, $r > 0$ and $r \neq 1$. Then,

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

Setup	LOC	# of theorems	runtime(s)	code time
raw Z3(can't)	-	-	-	-
raw ACL2(proved)	169	19	0.14	2 days
arithmetic-5(proved)	29	1	0.15	10 min
ACL2 & Z3(proved)	72	2	0.06	20 min

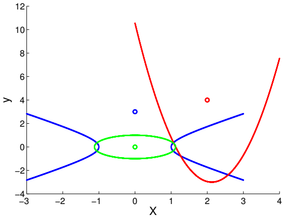
Polynomial inequalities

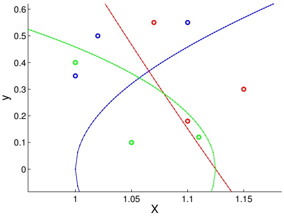
Theorem (Polynomial inequality)

Suppose $x \in \mathbb{R}$ and $y \in \mathbb{R}$, then

$$\begin{aligned} 1.125x^2 + y^2 &\le 1 \\ x^2 - y^2 &\le 1 \\ 3(x - 2.125)^2 - 3 &\le y \end{aligned}$$

does not have a solution.





Polynomial inequalities

Theorem (Polynomial inequality)

Suppose $x \in \mathbb{R}$ and $y \in \mathbb{R}$, then

$$
\begin{aligned}
1.125x^2 + y^2 &\leq 1 \\
x^2 - y^2 &\leq 1 \\
3(x - 2.125)^2 - 3 &\leq y
\end{aligned}
$$

does not have a solution.

Setup	LOC	# of theorems	runtime(s)	code time
raw Z3(proved)	27	1	0.0004	10 min
raw ACL2(failed)	40	-	-	10 min
arithmetic-5(failed)	41	-	-	10 min
ACL2 & Z3(proved)	59	1	0.02	10 min

Technical issues: typed vs. untyped

- This is not a theorem in ACL2:

```
(defthm not-really-a-theorem
  (iff (equal x y) (zerop (- x y))) )
```

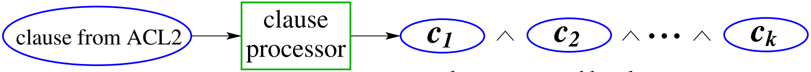
- Here is a counter-example:

Expression	Value
`(- 'dog (list "hello", 2, 'world))`	`0`
`(zerop (- 'dog (list "hello", 2, 'world)))`	`t`
`(equal 'dog (list "hello", 2, 'world))`	`nil`
`(iff (equal 'dog (list "hello", 2, 'world)) (zerop (- 'dog (list "hello", 2, 'world))))`	`nil`

- But this is a theorem:

```
(defthm this-is-a-theorem
  (implies (and (rationalp x) (rationalp y))
           (iff (equal x y) (zerop (- x y))) ))
```

Technical issues: typed vs. untyped




```
(implies (and (rationalp x)
              (rationalp y)
              (rationalp z))
         (c x y z))
```

clauses returned by clause processor:

$c_1 \wedge c_2 \wedge \cdots \wedge c_k \Rightarrow$ **c(x,y,z)**

- Solution: user adds type assertions to antecedent.
  - These are almost always needed anyways.
  - This requirement is not a significant burden for the user.

Technical issues: principle for ensuring soundness

- T_is are the type predicates; h_js are the “other” hypothesis; C is the conclusion of the theorem. The ACL2 theorem is defined as:

$$\forall x_1, x_2, ..., x_m \in U. \quad \left(\bigwedge_{i=1}^{m} T_i(x_i) \wedge \bigwedge_{j=1}^{n} h_j(x) \right) \Rightarrow C(x) \quad (1)$$

- S_1, S_2, ..., S_m are the SMT sorts corresponding to the type recognizers T_1, T_2, ..., T_m; $\tilde{h}_j(x)$ is the translation of $h(x)$; and $\tilde{C}(x)$ is the translation of $C(x)$. The corresponding SMT theorem is defined as:

$$\forall x_1 \in S_1, x_2 \in S_2, ..., x_m \in S_m. \quad \left(\bigwedge_{j=1}^{n} \tilde{h}_j(x) \right) \Rightarrow \tilde{C}(x) \quad (2)$$

Technical issues: principle for ensuring soundness

Soundness is ensured if:

- $\forall x_i \in U.\ \ T_i(x_i) \Rightarrow x_i \in S_i$
- $\forall x_1, x_2, ..., x_m \in U.\ \ (\bigwedge_{i=1}^{m} T_i(x_i)) \Rightarrow (h_j(x) \Rightarrow \tilde{h}_j(x))$
- $\forall x_1, x_2, ..., x_m \in U.\ \ (\bigwedge_{i=1}^{m} T_i(x_i)) \Rightarrow (\tilde{C}(x) \Rightarrow C(x))$

For `Smtlink` construction, that means:

- Types as translated by `Smtlink` must be no stronger than those of the ACL2 theorem.
- Hypotheses must be no stronger than those of the ACL2 theorem.
- The conclusion must be at least as strong.

Modeling the digital PLL

$$
\begin{aligned}
c(i+1) &= \text{saturate}(c(i) + g_c\,\text{sgn}(\phi), c_{\min}, c_{\max}) \\
v(i+1) &= \text{saturate}(v(i) + g_v(c_{center} - c(i)), v_{\min}, v_{\max}) \\
\phi(i+1) &= \text{wrap}(\phi(i) + (f_{dco}(c(i), v(i)) - f_{ref}) - g_\phi \phi(i)) \\
f_{dco}(c, v) &= \tfrac{1+\alpha v}{1+\beta c} f_0 \\
\text{saturate}(x, lo, hi) &= \min(\max(x, lo), hi) \\
\text{wrap}(\phi) &= \text{wrap}(\phi + 1), \qquad \text{if } \phi \leq -1 \\
&= \phi, \qquad \text{if } -1 < \phi < 1 \\
&= \text{wrap}(\phi - 1), \qquad \text{if } 1 \leq \phi
\end{aligned}
$$

- By simulation we get the model for f_{DCO}.
- This approach is similar to the one proposed in [ASZT07].