AMS verification with Theorem Proving and SMT

Yan Peng and Mark R. Greenstreet
University of British Columbia
{yanpeng,mrg}@cs.ubc.ca

AMS designs present new challenges for design verification because the analog blocks operate on time-scales that require pico-second scale time steps for accurate, transistor-level simulations, whereas digital adaptation loops may require times of microseconds to several milliseconds to converge. Several authors have proposed addressing these challenges by using linear models with their closed-form solutions for the analog components [10] or "continuization" approaches where discrete time dynamics are over-approximated by differential inclusions [3, 4, 15]. In our present work, we consider direct analysis of the recurrences that arise from the discrete-time, digital part of the AMS circuit. A similar approach was taken in [2] where they used Mathematica to analyse "symbolic recurrence equations" to show stability of an idealized Delta-Sigma ADC.

We observe that reachability tools like SpaceEx [8] and Coho [17, 16] or SMT solvers such as Z3 [12] and iSAT [7] typically cannot verify properties of interest to practicing designers using realistic models in a fully automatic fashion. Instead, these tools are used to prove simpler properties, essentially lemmas, that are then combined, often by human reasoning, to produce the complete result. Alternatively, one can use models that are oversimplified, or verify properties that match the tool rather than the designer's concerns. To overcome these limitations, we view the automatic tools as lemma provers, and seek to combine their results in a systematic fashion. In our current research, we use the ACL2 theorem prover [14] and the Z3 SMT solver [12]. Our choices of these tools was purely pragmatic. ACL2 has a well-documented interface for connecting external decision procedures [13], and Z3 is a state-of-the-art SMT solver with support for non-linear arithmetic and a Python API that is a convenient target for a clause translator.

Combining ACL2 and Z3

ACL2 supports extensions by means of "clause processors". Such a processor is a lisp function that takes an ACL2 clause as an argument, and returns a list of clauses as a result, with the interpretation that the conjunction of the returned clauses implies the original clause. ACL2 defines two types of clause processors: "verified" processors that have been proven correct with in the ACL2 logic; and "trusted" that are declared by the user to be correct, but not proven with ACL2. ACL2 tags any theorem whose proof depends (transitively) on a trusted clause processor with a "trust tag" for that clause processor. Effectively, the soundness of the trusted clause processor becomes a hypothesis of the theorem. We integrate Z3 into ACL2 as a trusted clause processor.

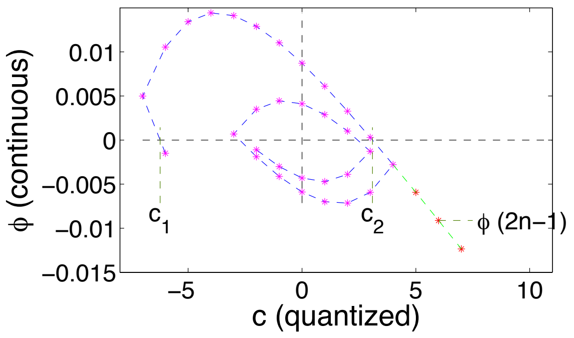
Our translator is syntax-directed and straightforward. We require the clause to be proven by Z3 to be an implication whose antecedent is a conjunction of type-predicates: these declare the types of free-variables appearing in the consequent. Our translator expands user-defined functions in the expression, and writes the negation of the flattened expression using the Z3py API (Python). If Z3 reports that this negated formula is unsatisfiable, then the original formula holds for all values of the free variables. Our clause processor then returns a clause of the form "the flattened formula implies the original formula" which is readily discharged by the ACL2 core. With this approach, we rely on the soundness of Z3 and the translation to Z3py and ACL2 automatically verifies the correctness of the clause flattener for each clause that we prove.

If Z3 finds a satisfying assignment to the negated formula, our clause processor simply returns the original formula; in other words, we make no claims of the validity of the counter-example. This is because Z3 provides theories over *reals*, booleans, and integers, whereas the ACL2 formula was quantified over *rationals*, booleans, and integers. Thus, a counter-example to the Z3 formula may not be a valid counter-example to the original ACL2 formula. On the other hand, if a formula that is universally quantified over the reals is valid, then so is the corresponding formula for rationals. We plan to extend our approach to allow ACL2 to use counter-examples from Z3 when all numbers in the assignment are rational.

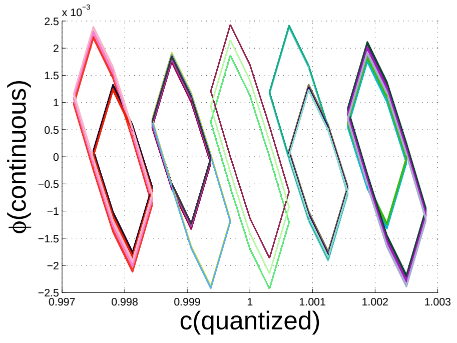
Status and Results

This is work in progress. We are using the digital-PLL from [5] as our initial example. At this point we have a proof in ACL2 that goes as far as the final induction argument to prove that the final region of convergence (the limit cycles, see Figure $1(a)$) is small. In the figure, c is the control capacitance for the digitally-controlled ring oscillator (DCO), and ϕ is the phase offset between the DCO and the reference. This last proof involve proving some rather tedious inequalities. ACL2 automatically generates the required inequalities, thus avoiding the need for manual derivations. While ACL2's rewrite engine cannot discharge them, we have shown that they are easily handled by Z3. We are in the process of debugging the function expansion code for our interface and will certainly have it completed in the next week or two.

The combination of theorem proving and SMT solving opens up other paths for AMS circuit analysis. By directly reasoning about the discrete dynamics, we can prove tighter bounds for the limit behaviours than obtained by using continuous over-approximations (e.g. [15]). We are encouraged that the proof is based on the intuition from the continuized version of the design: we use the Ricatti equation for the continuous model to propose a Lyapunov function that can be used to show convergence of the discrete time model. Much of this process is automated by ACL2 and Z3.



(a) an illustration of trajectory leading to a limit cycle



(b) a set of limit cycles

Figure 1: limit cycle behaviour

The SMT approach also enables analysis of properties of the design that are obscured by approximate models. For example, using the SMT solver, we have shown that the DPLL has multiple, stable limit cycles, and different limit cycles have slightly different jitter properties (See Figure $1(b)$). The exciting aspect of our approach is that we can make queries of the form: "Is the limit cycle unique?" or "Does there exist a limit cycle with particular properties?"

We are using the DPLL as a starting example. We plan to apply the method to other AMS designs such as the switching power-supply designs from [10], the Delta-Sigma ADCs from [6, 1], and the anaesthesia viability problem from [9].

Will AMS designers use a theorem prover?

No. We recognize that very few people are both experienced AMS designers and have the formal mathematical logic background to use an automated theorem prover such as ACL2. We are using the combination of ACL2 and Z3 as a prototyping platform to explore the opportunities for working directly with the discrete time recurrences of AMS circuits rather than with continuous time approximations. As noted above, the initial results are promising. As we gain experience with this approach, we expect to identify commonly used reasoning techniques (such at the Ricatti equation method described above) that could form the bases for automated tools. Similar methods have been used for the combination of SMT solvers and theorem provers for software verification [11].

References

[1] G. Al-Sammane, M. H. Z. Z. J. Dong, and S. Tahar. Towards assertion based verification of analog and mixed signal designs using PSL. In *Forum on Specification and Design Languages (FDL07)*, 2007.

[2] G. Al-Sammane, M. H. Zaki, and S. Tahar. A symbolic methodology for the verification of analog and mixed-signal designs. In *Proceedings of the 10th Design Automation and Test Europe Conference (DATE'2007)*, pages 249–254, Apr. 2007.

[3] M. Althoff, A. Rajhans, et al. Formal verification of phase-locked loops using reachability analysis and continuization. In *Proceedings of the 2011 International Conference on Computer Aided Design*, pages 659–666, Nov. 2011.

[4] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi. Formal verification of phase-locked loops using reachability analysis and continuization. *Communications of the ACM*, 56(10):97–104, Oct. 2013.

[5] J. Crossley, E. Naviasky, and E. Alon. An energy-efficient ring-oscillator digital PLL. In *Proceedings of the Custom Integrated Circuits Conference (CICC'2010)*, Sept. 2010.

[6] T. Dang, A. Donzé, and O. Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *Proceedings of the* 5^{th} *International Conference on Formal Methods in Computer Aided Design*, pages 21–36, Nov. 2004.

[7] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[8] G. Frehse, C. L. Guernic, et al. SpaceEx: Scalable verification of hybrid systems. In *Proceedings of the* 23^{rd} *Conference on Computer Aided Verification*, 2011.

[9] V. Gan, G. A. Dumont, and I. M. Mitchell. Benchmark problem: A PK/PD model and safety constraints for anesthesia delivery. Presented at Applied Verification for Continuous and Hybrid Systems (ARCH), Apr. 2014.

[10] J. E. Jang, M. Park, and J. Kim. An event-driven simulation methodology for integrated switching power supplies in SystemVerilog. In *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–7, May 2013.

[11] K. Leino and M. Rustan. Automating theorem proving with smt. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 2–16. Springer Berlin Heidelberg, 2013.

[12] L. Moura and N. Bjørner. Z3: An efficient SMT solver. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin Heidelberg, 2008.

[13] S. Ray. Connecting external deduction tools with ACL2. In *Scalable Techniques for Formal Verification*, pages 195–216. Springer US, 2010.

[14] A. Slobodova, J. Davis, S. Swords, and W. Hunt, Jr. A flexible formal verification framework for industrial scale validation. In *Formal Methods and Models for Codesign (MEMOCODE), 2011 9th IEEE/ACM International Conference on*, pages 89–97, 2011.

[15] J. Wei, Y. Peng, G. Yu, and M. Greenstreet. Verifying global convergence for a digital phase-locked loop. In *Proceedings of the 13th Conference on Formal Methods in Computer Aided Design (FMCAD'2013)*, pages 113–120, Oct 2013.

[16] C. Yan. *Reachability Analysis Based Circuit-Level Formal Verification*. PhD thesis, The University of British Columbia, 2011.

[17] C. Yan and M. R. Greenstreet. Circuit level verification of a high-speed toggle. In *Proceedings of Seventh Conference on Formal Methods in Computer Aided Design (FMCAD)*, Austin, Nov. 2007.