A Summary of Intrinsic Partitioning Verification

David Greve, Raymond Richards, Matthew Wilding

Rockwell Collins Advanced Technology Center
Cedar Rapids, IA
{dagreve,rjricha1,mmwildin}@rockwellcollins.com

Abstract

Successful formal methods applications have four characteristics: intrinsically important applications, concise correctness theorems, validated models, and proof automation. We describe a recently-completed verification of a microprocessor's intrinsic partitioning mechanism in those terms.

What Makes for a Good Application of Formal Methods?

Formal methods is the application of mathematical reasoning to establish properties about digital systems. Formal methods can be applied in many different ways with many different notations and tools. They can deal with system models that describe the lowest level of implementation or the most abstract requirements, with properties to be proved that may be comprehensive descriptions of "correctness" or minor aspects that indicate good system development.

Despite the wide range of formal methods applications, we observe that successful formal methods projects share four characteristics.

1. *The target being analyzed is intrinsically important.*
   Formal methods can provide a high level of certainty about a target, but the extra assurance must be worth the effort that formal verification usually entails. Three applications of formal methods that we consider successful are Microsoft's SLAM project [Ball2004], AMD's floating-point verification [Russinoff2000], and Rockwell Collins' requirements validation [Miller2004]. The SLAM project aims to reduce crashes of Microsoft's Windows OS by proving important device driver behaviors. AMD's floating-point work seeks to eliminate errors in the floating-point units on AMD's x86 microprocessors. Rockwell Collins is applying model-checking to help validate requirements for safety-critical systems. Each of these applications of formal methods is solving a problem that is important enough to justify an extra effort.

2. *The target's desired behavior has a concise and understandable formalization.*
   An important indicator of successful formal methods application is the degree to which the description of the needed property is compelling. A proved theorem only increases assurance about a target of evaluation if we trust in the formalization of the desired

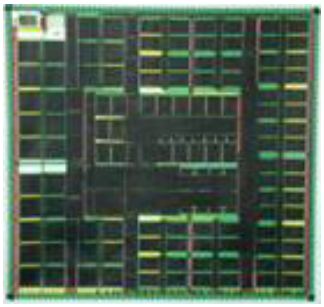
property. The SLAM project proves many, small theorems about proper device driver behavior that are drawn directly from coding guidelines of how device drivers are supposed to be implemented. AMD's floating-point work uses a formalization of the IEEE specification for floating-point operations. The Rockwell Collins requirements analysis work uses the "shall" statements of requirements documents as theorems to prove about the system model. In each of these projects the theorems are tied to a meaningful, compelling property that when formalized is far simpler than the model of the target of evaluation.

3. *The formal model of the target is validated against the "real" target.*
   A mathematical proof is about some formal description of the target, so for a proof to be of practical significance the model of the target must be validated against the actual implementation. Microsoft's approach in the SLAM project is to build a tool that automatically derives structure from the actual code that they analyze. The AMD floating-point work does something similar – a translator from the RTL implementation of the floating-point unit produces ACL2 models automatically that are amenable to analysis. The Rockwell Collins requirements work faces a particular challenge since the requirements presented are informal, but techniques such as executing the requirements models and proving properties that relate the model to the original informal description are used to establish this link. In each of these cases there is a strong connection between the target of evaluation and the formal model being reasoned about.

4. *Proof development and checking is tool-supported.*
   The complexity of proofs about interesting applications makes machine-checking a practical necessity. A highly automated methodology for guiding the proof generator/checker to proofs in the domain of interest is typically required, and much of the creative engineering effort in formal verification involves developing this proof architecture. In Microsoft's case, each property is cast as a formula suitable for model-checking, which provides a largely automatic checking procedure. AMD has developed libraries of properties of machine arithmetic and an approach for applying them automatically using the ACL2 theorem prover. The Rockwell Collins requirements work focuses on finite state models in order to be able to apply a model-checker to establish the theorems. In each case there is significant automation that is enabled by a carefully considered proof architecture

AAMP7 Intrinsic Partitioning

The AAMP7 is a microprocessor designed for use in embedded systems. It is the latest generation of the AAMP microprocessor family, members of which have been employed in systems with security-critical or safety-critical requirements. The AAMP7 provides a novel architectural feature, *intrinsic partitioning*, that enables the microprocessor to enforce an explicit communication policy between applications. This mechanism simplifies the construction of systems composed of several applications that operate on the microprocessor concurrently. Assuming that the AAMP7 can be trusted to enforce the communication policy, applications that reside on the AAMP7 can be developed and verified without regard to the behavior of applications that reside in other partitions. Intrinsic partitioning is



particularly valuable in realms where much of the cost of developing a system lies in verification and certification. The AAMP7's intrinsic partitioning mechanism is an implementation of what is called a *separation kernel* in the world of secure applications and what is known as a *partitioning system* in the world of safety-critical applications.

We used ACL2 to show that the AAMP7 microprocessor's intrinsic partitioning works as expected. We modeled the implementation of the AAMP7 with respect to partitioning in great detail in a formal language and proved that a formalization of the needed separation property holds of our AAMP7 model. The model is very detailed and corresponds directly to the microcode of the microprocessor.

One of our goals in this work is to meet formal methods requirements for certification standards such as the Common Criteria [CC], which requires that a low-level model

> … provide a description of how each module is expected to be implemented from a design perspective. [CC, part 3 paragraph 354]

The use of ACL2 to meet high-assurance Common Criteria requirements is discussed in [Richards2004]. One interpretation of the requirement for low-level design models is that the low-level design model be sufficiently detailed and concrete so that an implementation can be derived from them with no further design decisions. Because there are no design decisions remaining, one can easily validate the model against the implementation. Note that this low level of abstraction of a model can make a proof about it challenging. In the case of the AAMP7 proofs, a particular challenge in developing the proofs was that the low-level design models memory as a linear address space with only read and write primitives. Although this enables straightforward validation of the model against the actual microcode implementation, it also requires the development of a proof methodology that supports reasoning about operations that are modeled on a linear address space.

We briefly describe the formal verification of the AAMP7 intrinsic partitioning mechanism in terms of the four aspects of formal methods.

1. The AAMP7's intrinsic partitioning mechanism is important

The goal in building a partitioning mechanism is to limit what must be evaluated in a verification or certification context. For example, secure systems can be developed that use partitions to enforce separation between processes at different security levels. A small, trusted "separation kernel" mediates all communication between partitions thereby assuring that nonauthorized communication does not occur. Assuming that the partitioning system is implemented properly and that the communication policy between partitions is loaded correctly, there is no need to evaluate the applications running in different partitions to show that the communication policy is enforced. Safety-critical applications can also exploit intrinsic partitioning: by hosting different applications in separate partitions it is possible to architect a system so that applications need be evaluated at only the needed level of rigor. This system architecting philosophy is described by John Rushby in [Rushby1981, Rushby1999].

The correct implementation of the partitioning mechanism is of course vital to assure the correctness of a larger system that depends upon it. Furthermore, some of the initial applications of the AAMP7 are security applications that are architected to exploit intrinsic partitioning and require stringent evaluation of all mechanisms being relied upon to separate data at different classification levels.

2. The AAMP7's intrinsic partitioning has a concise formalization.

Space partitioning is the crucial property that we are interested in showing about the AAMP7. Figure 1 presents a formalization of separation of state between partitions. (See [Greve2003b for a detailed description of this conjecture, including a description of each of the functions.)

```
(let ((segs (intersect (dia seg) (segs (current st1)))))
    (implies
     (and
      (equal (selectlist segs st1) (selectlist segs st2))
      (equal (current st1) (current st2))
      (equal (select seg st1) (select seg st2)))
     (equal
      (select seg (next st1))
      (select seg (next st2)))))
```

Figure 1. Separation kernel space separation theorem [Greve03b]

Unlike the other formal methods projects we describe above, there are no requirements documents or IEEE specifications that capture the notion of separation we require for verifying AAMP7 intrinsic partitioning. Rather, we present a theorem we think is appropriate and argue that it makes for a good specification of separation. This is done in [Greve2003b], which presents a proof about a firewall that is implemented with a separation kernel assumed to have this property. The firewall that is implemented using a separation kernel is modeled in ACL2 and proved correct, relying only on the assumed property about the underlying kernel. The fact that the separation kernel correctness theorem is all that is relied upon upon to establish the correctness of the larger system suggests that this correctness theorem captures an important aspect the separation kernel's behavior.

The formalization described in [Greve2003b] can also be conveniently expressed in other notations. Recently, the security policy was recast in the logic of the PVS theorem proving system [Rushby2004].

3. The formal model of the AAMP7 is validated against the "real" AAMP7.

We constructed a low-level design model of the partitioning-relevant operation of AAMP7. That model consists of approximately 3000 lines of ACL2 code. A crucial consideration is how to validate this hand-written model against the actual AAMP7. The AAMP7 is a microcoded microprocessor, and much of the functionality of the machine is encoded in its microcode. "Trusted" microcode is microcode that operates with memory protection turned off, thereby providing access to the datastructures maintained by the AAMP7 to support intrinsic partitioning. All the partitioning-relevant microcode runs in this trusted mode, and the low-level design model of the AAMP7 models all the microcode that implements this functionality.

We conducted a successful code-to-spec review with a National Security Agency evaluation team in March, 2004. This review validated the formal model against the actual AAMP7. We developed a documentation package that was used during this review. The documentation provided included:

- material explaining the semantics of ACL2 and AAMP7 microcode,
- listings of the AAMP7 microcode and the ACL2 low-level model,
- the source code listing of a tool that identifies trusted-mode microcode sequences, and a listing of such sequences in the AAMP7 microcode,
- cross-references between microcode line numbers, addresses, and formal model line numbers, and
- the ACL2-checkable proofs on compact disk.

The low-level design model was written specifically to make this code-to-spec review relatively straightforward. An ACL2 macro allows an imperative-style description that eases comparison with microcode. Also, very importantly, the model is written with the model of memory that the microcode programmer uses when writing microcode. That is, memory has only two operations: read and write. The simplicity of the memory model makes the code-to-spec review easier but adds a great deal of complexity to the proof. Since the proof is machine-checked while the model validation process requires evaluation, this is a good tradeoff. It provides a high level of assurance with a reasonable level of evaluation. Nearly all the time on the project was spent constructing the proofs, but they were evaluated very easily by the evaluators because they could be replayed using ACL2. Most of the evaluation time was spent on the code-to-spec review. Figure 2 presents a fragment of the low-level design model.

```
;=== ADDR: 052F

   (st. ie = nil)
   (Tx = (read32 (vce_reg st) (VCE.VM_Number)))

;=== ADDR: 0530

   (st. Partition = Tx)

;=== ADDR: 0531

   (TimeCount = (read32 (vce_reg st) (VCE.TimeCount)))

;=== ADDR: 0532

   (PSL[0]= TimeCount st)
```

Figure 2. A Fragment of the AAMP7 formal low-level model

4. The AAMP7 proofs are fully checked using ACL2, made possible by a proof architecture and theorem library infrastructure.

Much as large software implementations require an architecture, so too do large proof efforts. Figure 3 shows the final theorem proved about the AAMP7. Note that this is an instance of the separation theorem described in Figure 1, with a few differences. First, functions that describe the communication policy ("`dia`") and the segments associated with a particular partition ("`segs`") are functions of segment name and processor state rather than just segment name. This allows them to "pull" the configuration information out of the processor state. We prove this is

appropriate by proving that these functions are invariant with respect to a step of the low-level design model ("`next`"). Second, we add some assumptions about a secure initial state of the AAMP7. These assumptions guarantee that the AAMP7's state is reasonable – that the datastructures have a reasonable shape, that different datastructures do not overlap, etc.

The proof architecture breaks the proof into three main pieces
1) Proofs validating the correctness theorem (as described in [Greve2003b])
2) Proof that the abstract model meets the security policy
3) Proof that the low-level model corresponds with the abstract model

In addition to libraries provided in the standard ACL2 release, several libraries of ACL2 lemmas were developed for this project. Two of the libraries are released and documented [Smith2004, Greve2003a]. As previously indicated, an important challenge of this project was developing a method for reasoning about read and write operations on a linear address space. An initial version of this problem was posed as a challenge problem in [Greve2002], and a description of the Rockwell Collins approach – called GACC for Generalized Accessor – is outlined in [Greve2004]. It provides a systematic approach for describing datastructures and a template for proving a few helpful facts about each operation.

```
(implies
   (and
    (secure-configuration spex)
    (spex-hyp :any :trusted :raw spex fun::st1)
    (spex-hyp :any :trusted :raw spex fun::st2))
   (implies
    (let
     ((abs::st1 (lift-raw spex fun::st1))
      (abs::st2 (lift-raw spex fun::st2)))
      (and
       (let ((segs (intersection-equal
                (dia-fs seg abs::st1)
                (segs-fs (current abs::st1) abs::st1))))
       (equal (raw-selectlist segs abs::st1)
              (raw-selectlist segs abs::st2)))
       (equal (current abs::st1)(current abs::st2))
       (equal (raw-select seg abs::st1)(raw-select seg abs::st2))))
    (equal
     (raw-select seg (lift-raw spex (fun::next spex fun::st1)))
     (raw-select seg (lift-raw spex (fun::next spex fun::st2))))))
```

Figure 3. AAMP7 intrinsic partitioning separation theorem

Summary

We have completed a substantial ACL2 proof about AAMP7's intrinsic partitioning mechanism. The formal methods artifacts have been successfully evaluated, and are part of a package that is currently undergoing certification. The proofs require about 4 hours to replay using ACL2 2.8. We believe that good applications of formal methods have four characteristics: intrinsically important applications, concise correctness theorems, validated models, and proof automation. The AAMP7 intrinsic partitioning project has each of these characteristics.

Bibliography

[Ball2004] Thomas Ball, Byron Cook, Vladimir Levin, Sriram K. Rajamani, "SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft", Microsoft technical report MSR-TR-2004-08, Jan 2004.

[CC] `http://www.commoncriteriaportal.org/`

[Greve2002] David Greve and Matthew Wilding, "Dynamic Datastructures in ACL2: A Challenge", Nov 2002. `http://hokiepokie.org/docs`

[Greve2003a] David Greve and Matthew Wilding, "Typed ACL2 Records", Fourth International Workshop on the ACL2 Prover and Its Applications (ACL2-2003), Boulder, CO, July 2003.

[Greve2003b] David Greve, Matthew Wilding, and W. Mark Vanfleet, "A Separation Kernel Formal Security Policy", Fourth International Workshop on the ACL2 Prover and Its Applications (ACL2-2003), Boulder, CO, July 2003

[Greve2004] David Greve, "Address Enumeration and Reasoning over Linear Address Spaces", ACL2 Workshop 2004.

[Miller2003] Miller, S. P., Tribble, A. C., and Heimdahl, M. P. E., "Proving the Shalls," 12th International Formal Methods Europe Symposium, Pisa, Italy, September 2003.

[Richards2004] Raymond Richards, David Greve, Matthew Wilding, W. Mark Vanfleet, "The Common Criteria, Formal Methods, and ACL2", ACL2 Workshop 2004.

[Rushby1981] J. Rushby, "Design and Verification of Secure Systems", Proceedings of the Eighth Symposium on Operating Systems Principles, volume 15, December 1981.

[Rushby1999] John Rushby, "Partitioning for Safety and Security: Requirements, Mechanisms, and Assurance", NASA contractor report CR-1999-209347, 1999.

[Rushby2004] John Rushby, "A Separation Kernel Formal Security Policy in PVS", SRI CSL technical note, March 2004.

[Russinoff2000] David Russinoff, "A Case Study in Formal Verification of Register-Transfer Logic with ACL2: The Floating Point Adder of the AMD Athlon Processor", FMCAD 2000.

[Smith04] Eric Smith, Serita Nelesen, David Greve, Matthew Wilding, and Raymond Richards, "An ACL2 Library for Bags (Multisets)", ACL2 Workshop 2004.