**Abstract**

ET: Enhanced tool (solution strategy)

As far as I can tell, the authors didn’t extend their solution strategy (as discussed in “Why ET?”, below), so I have modified Newman’s pro-forma abstract somewhat.

The effectiveness of <formal methods> in supporting the design of <reliable software> has been demonstrated. Additional validation is described for the design of <shared memory bus protocols> based on <Symbolic Model Verifier (SMV), a model checker that relies on binary decision diagrams (BDDs) for the verification of a computational tree logic (CTL) model of the protocol>. Examples are provided confirming the effectiveness of its support for <reliable software> design.

**Why ET?**

The authors spend a good deal of their time on the actual application of their analysis strategy (that is, on their analysis of the FutureBus+ specification). Although they never really seem to bring it up to the surface, however, the most important undercurrent within the paper is the sense, “Look, we used formal methods on something practical and it worked out. So we think formal methods are tractable as a discipline—not to mention useful. So believe us, formal methods are a worthwhile and promising area of study.” Maybe I’m reading too much into it, but the main value of the paper didn’t seem so much to be their analysis of the FutureBus+ protocol (valuable though it was!). Instead, I think the main contribution was further external validation of their analysis approach.

**Question - [Method/means of development]**

How can we ensure that an implementation is correct?

**Results - [Procedure / technique]**

In previous work, the authors were largely responsible for transforming the above question into the following: How can we verify that an encoding of hardware/software matches a model of the specification? The reframing of the question leads to the concept of formally representing the specification (for instance, using CTL) and then algorithmically verifying that the implementation can only get into states consistent with the specification. Around this reframing grew up a whole research area of model checking algorithms.

The authors had already created one algorithm, which they called SMV. The problem, as I see it right now, is that formal methods were still deemed impractical or not useful. We already heard what Perlis and friends wrote about formal methods. Even the authors of the current paper note that model checking of some protocols can rapidly lead to over $10^{50}$ states, and they needed to make around a dozen simplifications of the FutureBus+ protocol in order to reduce it to a tractable size. Thus, one yawning gap seemed to be the external validation of this software development procedure/technique. The current paper seems to address this gap.

**Validation - [Experience]**

The authors took a real IEEE bus protocol specification (FutureBus+), made a number of simplifications to it, and then created a formal model in a syntax compatible with their SMV tool. They then created an “implementation” in a hardware description language and checked whether the implementation matched the protocol. They uncovered errors not only in the implementation but also in the protocol. The tractability and utility in this instance suggests that formal methods might be worth their salt.