Making our Way through the End-of-the Roadmap Maze

Panel Position Statement

Sharad Malik
Princeton University

As we look forward over the next 10-15 years, there are several recognized roadblocks on the silicon roadmap with very few pathways left to navigate between them. The major new challenges dealing with complexity and uncertainty are:

- Compute performance needs to come from greater concurrency rather than increasing the clock period: The limitation on continuing to increase the clock period comes from the need to stay within the power budget envelope. However, this requires building up expertise in programming highly concurrent architectures, with concurrency at possibly different levels of granularity. This is a challenge since most applications/algorithms tend to have mostly sequential flow, perhaps a consequence of the sequential nature of human thought. Further, attempts to efficiently program parallel machines have not been very successful in the past. We could attempt to redouble our efforts here, and develop new algorithms for the automatic extraction of parallelism, but limited success with this in the past should give us some pause in our predictions of success here.
- Increasing dynamic errors will need dynamic error checking and recovery mechanisms: A variety of new modes of operational failures appear on the horizon that may be triggered by thermal conditions, aging, or energized particle hits (soft-errors). These are all expected to have greater impact with finer device geometries. Given the dynamic nature of these failures, there is no alternative but to deal with them at runtime through dynamic error detection and recovery. Related to this is the issue of dealing with increasing process variations for succeeding technology generations. Given the greater range of variations, we can no longer count on worst-case design to result in expected design improvements. Various possibilities for "better than worst-case design" are being explored which must allow for some form of occasional failure that must be recovered from. Again this points to dynamic checking of and recovery from failures. Pursuing this direction will require significant research in developing failure models as well as dynamic checking and recovery techniques at different levels of design granularity. Given the lack of access to cutting edge fabrication facilities, any modeling efforts will need to be conducted in collaboration with industrial partners. While this is challenging, given our past success in modeling at all levels, the chances of success are high here. The big question here is if we will be able to derive these models well in time to drive the checking and recovery techniques that depend on them. In terms of developing checking and recovery techniques, again this direction looks very promising. We have a long successful history of developing fault-tolerant computing techniques and much of that can be retooled to this new context as defined by the failure models.
- Runaway design complexity will need radical new bridges to cross the verification chasm: There is a clear awareness of the rapidly growing gap between the rates of growth of our ability to design and our ability to verify these designs. A simple argument shows us why this gap will grow to a chasm unless radical new steps are taken to address this. Unlike area, performance or power, hardware design complexity cannot be quantified precisely. Probably the closest acceptable measure for design complexity is the number of states in the system. The number of states is exponential in the number of state bits; which in turn tends to be proportional to the number of transistors in the design. This, thanks to Moore's Law has roughly doubled every few years or so. Thus, the exponential growth rate due to Moore's Law combined with the exponential dependence of the number of states on the number of state bits provides for a first order estimate of the growth rate of design complexity that is doubly exponential over time. However, at the same time, the exponential growth rate due to Moore's Law has led to faster computation that we can bring to bear in terms of simulation cycles and

speed of formal verification algorithms. This still leaves us with a complexity growth rate that is increasing exponentially compared to our ability to deal with it.

This exponential gap between the growing complexity and our ability to deal with it is somewhat corroborated by some field data in the context of microprocessors that shows an increase of design bugs that is linear in the number of transistors and thus exponential over time. This increase in complexity and the corresponding bug rate have led to an exponential increase in the verification effort, as measured by both the number of simulation vectors, as well as the number of engineer years. This effort and cost continues to be justified given the high cost of post-manufacture bugs. A post-manufacture, but pre-deployment bug may result in one or more respins of silicon and each respin is estimated to run at several millions of dollars in mask and other costs. A further cost of delays due to logic bugs discovered late in the design and manufacturing process is the lost market opportunity. Significant delays in product deployment may wipe out large profit margins or even lead to cancellation of projects. Post-deployment bugs may be even more costly. The Pentium FDIV bug reportedly cost Intel several hundred million dollars.

Given the large costs of and increasing number of bugs, verification efforts on design projects will continue to grow to cope with this situation. However, an exponential growth in verification costs is untenable even in the near future. In fact, the limited data on increasing number of respins due to logic bugs is indicative of the growing number of bugs that continue to stay in designs even late in the process, despite large verification efforts. Such occurrences will only increase with time. I claim that soon we will need to reconcile ourselves to the fact that hardware, like software will be shipped with bugs. This may seem inconceivable for hardware for at least a couple of reasons. Software bugs result in failures relatively sporadically, given that software tends to be extensively tested at speed. In comparison, hardware validation is predominantly through simulation, which is done at a tiny fraction of the speed of eventual deployment. Thus, bugs in hardware may manifest themselves relatively often when run at speed. This is unacceptable. Second, software bugs can generally be dealt with by resetting the system to some safe state. It is unclear how to do robust, practical and reliable recovery for hardware. Clearly, some creative ideas are needed to deal with this scenario.

One possible solution here is to deploy runtime validation techniques that complement pre-silicon verification. Given that we will need runtime error checking and recovery to support dynamic operational failures, it makes sense to leverage these techniques to support functional failures also. However, unlike operational failure modes, we do not have a good understanding of functional failure modes, i.e. what should we be checking for? Similarly, recovery may be more complicated in this context, since functional failures cannot be dealt with simple redundancy techniques. The lack of experience in this overall direction of research makes this less predictable than the case for operational errors.

Another possible attack on dealing with the verification chasm is to have few hardware parts and achieve product differentiation through software. This is not appealing for a couple of reasons. First, without hardware differentiation we will be paying a high silicon and power cost. Second, given the concern with programming highly concurrent processors, we are depending on a solution that may not deliver. Thus, the more optimistic scenario is to try and piggyback the solutions for dynamic operation failures. The likelihood of success there is high; we just need to understand their combination with the functional failure context.