The top- 10 reasons you don't design for test

At a panel session on the acceptance barriers confronting design for test and built-in self test (BIST) Richard Sedmak president of Self-Test Services ((215) 628-9700), presented a list of reasons designers don't design for testability. With apologies to David Letterman we have adapted that list here.

10) There is no push-button answer to designing for testability. And everyone knows how much engineers like to push buttons.

9) Test requirements are usually poorly defined. Failure of the marketing people to put a specification for testability into the statement of work makes It easy for you to meet it.

8) Little or no communication occurs between the design manufacturing and service organizations. When you don't know the sort of problems that arise after your designs reach the production floor and ultimately, the customer, you can't improve subsequent iterations.

7) Companies don't do a good job of tracking manufacturing defects and field failures. You aren't the only one who doesn't know what happens once designs leave your hands.

6) Your company has no Iife-cycle cost-of-test model. Because the company has never tracked the impact of failure to test over the life of a product, the company can't make informed tradeoffs about the up-front cost of designing for lost versus the back-end cost of ignoring it.

5) The testing crisis within your organization hasn't reached a critical level. Your company hasn't yet had to recall a high-impact product and placate angry customers because of a design or component problem that test failed to catch.

4) Management has no real commitment to test. Oh sure, everybody says that test is important. But how do they spend their money? Has your company developed life-cycle cost-of-test models? Does the testability of your projects influence your raises and promotions?

3) Schedules and budgets make no allowance for increased testability. Because there are no real transparent methods, making a design testable takes time and costs money.

2) Adding testability steals precious nanoseconds from performance and demands high real-estate penalties, This is probably the most common excuse to avoid testability and the most specious. Most designers who do design for test say performance and area impacts aren't design killers. All paths are not critical. You can provide control to and observation of nodes near, but not on, the critical path. Real-estate costs are a function of your chosen testability scheme, the complexity of your design, and the technology you choose for building it. If You're using 50% of a large gate array, for example, adding scan-based testability will lower yield and will appear to cost you pennies. Ultimately, though, you'll save money through reduced failures in test or in the field. In contrast, if you've decided to implement a register-oriented design in a small, highly utilized gate array, adding scan could force you into a larger array and cost substantially more.

1) You are rarely rewarded if you do design for test and seldom penalized if you don't. If adding testability forces you to slip your schedule, are you praised for adding test or punished for slipping the schedule? Do your gate budgets include an allowance for test? If adding test forces your design into a larger gate array, would your company add the test logic, remove some of the design's function, or keep all the function and shoe-horn a little bit of test logic into the smaller array? Are you promoted based on how easy your designs are to test or on how well your devices meet performance specifications?