

Anforderungen an Testwerkzeuge der nächsten Generation

ATE-Plattform-neutrale, zyklische Timing- und Pattern-Daten sind Voraussetzung zur Bewältigung aktueller Designs

Für viele komplexe Halbleiterbauelemente sind die Testkosten auf knapp 50 Prozent der gesamten Herstellkosten gestiegen. Ein großer Anteil der Testkosten entsteht durch den hohen Zeit- und Arbeitsaufwand, den die Testprogrammerstellung beansprucht. Es gibt viele zeitsparende Methoden und Testentwicklungswerkzeuge – sowohl kundeneigene als auch kommerzielle –, um die Funktions- und Scanpattern in Testprogramme für speziell ausgewählte automatische Testsystemplattformen zu übersetzen. Dennoch versagen diese Werkzeuge und Methoden immer mehr im Hinblick auf die steigende Integrationsdichte und -komplexität der Bauelemente und die neuen Anforderungen in der Produktion.

Um die wachsenden Testkosten zu reduzieren, müssen Testwerkzeuge der nächsten Generation, die sich aus der Funktionalität und Charakteristik des zu prüfenden Bauelementes (DUT) definieren, zwischen Testplattformen austauschbar sein. Die Industrie benötigt dazu automatisierte Testprogrammentwicklungswerkzeuge, die einen zweiteiligen Ansatz unterstützen. Im ersten Schritt werden dabei zyklische Timing- und Testpattern erzeugt, die testplattformneutral sind. Erst im zweiten Schritt werden diese neutralen Muster automatisch in Formate übersetzt, die mit spezifischen Testplattformen kompatibel sind. Der Schlüssel, um diese Ziele zu erreichen, liegt in der Fähigkeit, das Timing automatisch zu erkennen und die Simulationsabweichungen zu korrigieren. Voraussetzung dafür ist, dass diese Werkzeuge die Vorgaben der IC-Entwicklungswerkzeuge (Electronic Design Automation Software) erfassen, in standardisierte Testsprachen umwandeln und sowohl

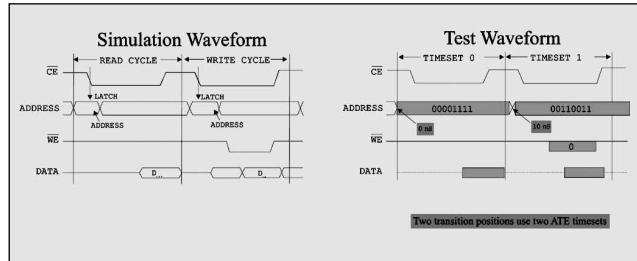


Abb. 1: Unverbundene Testvorgänge verbrauchen die limitierten Zeitvorgabe-Ressourcen des Testsystems

Funktionsmuster als auch Scanpattern unterstützen.

Dieser Artikel wird die Anforderungen an Entwicklungsingenieure beschreiben, die aus der Komplexität moderner Bauelemente resultieren, und eine ‚offene Testentwicklungsumgebung‘ skizzieren, die helfen kann, die Markteinführung zu beschleunigen. Zuletzt wird er Bereiche aufzeigen, in denen die Industrie noch zusätzliche Arbeit und Ressourcen investieren muss.

Der traditionelle Prozess

Design-to-Test wird häufig als der unkomplizierte Prozess gesehen, der er einmal war. Viele Jahre lang benutzte die Mehrzahl der Testingenieure funktionelle Pattern-Reihen, um Bauelemente auf ihren ausgewählten Testplattformen zu prüfen. Diese Pattern-Reihen waren entweder von Hand erstellt oder von funktionellen Simulations-Logdateien übersetzt. De-facto-Industriestandards wurden für Prozesse und Testsprachen festgelegt, die ereignisbezogene Freiformsimulations in Timing- und Pattern-Reihen umwandeln, die in der hochsynchronisierten Welt von zyklusbasierenden Testsystemen benutzt wurden. Dieser Prozess nennt sich Zyklisierung.

Jede Testsystemplattform hat eine Syntax für die Programmierung und Regeln, die nur auf die jeweilige Plattform anzuwenden sind. Testprogramme müssen nach diesen Regeln erstellt werden, um den Schwingungsverlauf zu generieren, der das Verhalten des DUT (zu prüfendes Bauelement) richtig beschreibt.

Das Testprogramm weist Testpins, Formate, Spannung, Timing, Pattern und andere Testerressourcen zu, um das zu prüfende Bauelement vollkommen abdecken zu können.

Bei älteren Halbleitergenerationen war dieser Prozess mit Hilfe einer Vielzahl von Scripts, Werkzeugen und günstigen Softwarepaketen relativ einfach und zuverlässig zu realisieren.

In der Welt der komplexen Bauelemente mit vielen Funktionskernen muss sich die Design-to-Test-Software allerdings mit einer Vielzahl neuer Testanforderungen auseinandersetzen:

Input-Unterstützung in multiplen EDA-Formaten

Der Testprogrammentwicklungsprozess beginnt oft mit der Datenausgabe der Simulationssoftware. Bei einem SoC-Baustein (System-on-Chip) können für verschiedene Funktionskerne viele Simulationsquellen in unterschiedlichen Formaten zum Einsatz kommen. Eine offene Testprogrammentwicklungsumgebung muss sich diesen Formaten anpassen, um die Flexibilität der Design- oder Testteams nicht einzuschränken. Der Prozess der Überführung des Bauelements aus der Entwicklung in die Produktion beginnt damit, die Eingangssprache zu analysieren und in eine einheitliche Sprache zu übersetzen. Dadurch wird der Zyklisierungsschritt von Simulatorformaten unabhängig.

Entschlüsselung und Timing-Optimierung bei mehreren Funktionskernen

Der wichtigste Grund für das Versagen alter Testmethoden ist im Bereich der Analyse von kritischen Timing-Parametern in den eingehenden Simulationsergebnissen und deren Konvertierung in zyklische ATE-Formate mit

► Autor
 HAU LAM ist Product Marketing Manager bei Credence Systems Corporation; 215 Fourir Ave Fremont, CA 94539 USA
 Fon: +1/510-657-7400
 Fax: +1/510-623-2524
 E-Mail: marcom@credence.com

Anpassungen, die den ATE-Regeln entsprechen und Zeitrissen vermeiden, zu suchen. Dieser Prozess wurde immer teurer und führte zu Einbußen in der Messgenauigkeit. Vor nur wenigen Jahren – als Projektmanager noch sechs bis acht Monate für Design, Verifikation und Test brauchen durften – konnten Zeitspezifikationen von Mikroprozessoren und intelligenten Funktionskernen, verschiedenen Bustypen und anderen Bauelementen oft mit vorhandenen Zeitspezifikationen und durch einen überschaubaren iterativen Prozess zwischen Design und Test aufgebaut werden. Heute, wo zwischen Verifizierung und Test durchschnittlich nicht mehr als zwei bis acht Wochen verstreichen dürfen, ist dieser Ansatz nicht mehr akzeptabel.

Funktionskerne sind mit verschiedenen Teststrategien eingebunden – einige über Test-Pattern, aber die meisten mit vielen Funktions- und/oder Scan-Pattern. Diese müssen in zwei Arten vorliegen, weil SoC-Bauelemente die physikalischen Grenzen von Testsystemen oft erreichen oder sogar überschreiten können: Einmal in zyklischer Form und für die Testressourcen der Testsystemplattform und Produktionstestzeit optimiert. Optimie-

rung bedarf der Aufsummierung der Zeit über Hunderte von Funktionen eines Kernes und evtl. über Hunderte von Kernen. Optimierung bedeutet auch die Anwendung verschiedener Techniken zur Komprimierung der Pattern.

Im Falle von SoCs, die dem neuen Teststandard IEEE P1500 für eingebettete Kerne entsprechen, müssen die Testwerkzeuge zusätzlich diesen Standard unterstützen.

Handhabung von Teststrategien von Hochgeschwindigkeits- und Mixed-Signal-Applikationen

Die meisten Telekommunikations- und Netzwerkbauelemente haben Hochgeschwindigkeitspins von mehr als 2 GHz. Nur wenige Testsystemplattformen können diese Pinzahlen handhaben. Eine logische BIST-Technologie (built-in self-test) kommt Geschwindigkeitsanforderungen entgegen, analoge BIST-Technologie ermöglicht dem digitalen Testsystem, analoge Komponenten, wie z.B. PLL (phase-lock loop) und Konverter (A/D und D/A) zu testen. Dies hilft, die Kosten von teu-

ren Mixed-Signal-Testsystemen in Grenzen zu halten, aber es fügt dem Testentwicklungsprozess eine weitere Anforderung hinzu. Die neue Testentwicklungsumgebung muss Werkzeuge umfassen, die Pattern dieser logischen und analogen BIST-Simulationen integrieren.

Simulationen von BIST-Schaltkreisen generieren umfangreiche Ausgangsdateien – manche überschreiten sogar die 2-Gbyte-Dateisystemgrenze von 32-Bit-Betriebssystemen und können damit die Ablaufzeiten erheblich erhöhen. Folglich muss eine adäquate Testentwicklungsumgebung nicht nur große Dateien lesen können, sondern dies auch in einem akzeptablen Zeitrahmen erledigen.

Re-Targeting für verschiedene ATE-Plattformen

Müssen sich Testteams auf unterschiedlichen Testplattformen bewegen, wird der Zyklisierungsprozess schwieriger. Testentwicklungswerkzeuge müssen durch die Anwendung von stabilen ATE-Re-targeting-Funktionen flexibel gestaltet sein. Unabhängig davon, ob die Testabteilung nun aus einer einzigen Person besteht, ausgelagert ist, oder aus Spezialistentteams mit mehreren Testsystemen im Haus zusammengesetzt ist, ist entscheidend, dass der in die Testprogrammierstellung investierte Zeit- und Finanzaufwand nicht durch die Testsystemwahl oder den Wechsel der Testsystemplattform verdoppelt wird. Stellt das Bauelement ein Nachfolgeprodukt eines anderen dar, sollte auch der Testentwicklungsprozess nicht wieder ganz von vorne beginnen müssen; es sollte möglich sein, bestehende Timing- und Test-Pattern in wenigen Schritten anzupassen. Der Zyklisierungsschritt sollte bei korrekter Ausführung nicht testerabhängig sein und auch nicht wiederholt werden müssen. Würden Testentwicklungswerkzeuge der nächsten Generation lieferantenneutrale, zyklische Daten liefern, wäre die Testsystemanpassung ein einfacher Vektorübersetzungsprozess.

Unterstützung verschiedener EDA-Formate in der offenen Testentwicklungsumgebung

Bei der Handhabung der Simulatorergebnisse verschiedener EDA-Werkzeuge beginnt der Prozess vom Design zum Test mit einem Normalisierungsschritt. Die offene Testumgebung analysiert die Sprache aller Eingangsformate und übersetzt sie in eine gängige Sprache, die von jeder Testsystemplattform erkannt werden kann. Auf diese Weise wird der nächste Schritt, der Zyklisierungsschritt, von Simulatorformaten unabhängig.

- ▶ Wenn der Benutzer das Timing spezifizieren will, muss er nur die wichtigen Bau-teilsignale spezifizieren, die er oft im Datenblatt des Designs findet. Das Testentwicklungswerkzeug wird den Rest, durch Erkennen des unspezifischen Signaltimings im Eingangs-Simulationsfile, ausfüllen.
- ▶ Shape promotion: Wenn alle kompatiblen Signalverläufe in der Hash-Tabelle kombiniert sind, kann der Anwender die Resource Testsystem optimal nutzen. Das begünstigt die bestmögliche Testzeit und ermöglicht dem Anwender größere Testprogramme auf einem physikalisch limitierten Testsystem zu realisieren.
- ▶ Schrittweise Timing-Verbesserung: Bei der Arbeit mit vielen Simulationsausgangsdateien vom gleichen Bauelement wird der Zyklisierungsschritt das Timing schrittweise sammeln, sodass einfaches Timing auf funktionelle Pattern oder Patternanhäufungen wieder angewendet werden kann. Solange z.B. Simulationsgruppen mit Randausreißern innerhalb der vom Anwender definierten Toleranzgrenze liegen, werden sie automatisch angepasst.

Patternkompression zur Anpassung an den ATE- Pattern-Speicher

Wie schon erwähnt, wächst die Größe der Bauelemente-Pattern, weil die Chips mehrere Funktionskerne enthalten und weil BIST-Schaltkreise erstaunlich große funktionelle Simulationsausgangsdateien erzeugen können (obwohl ein großer Prozentsatz der BIST-Pattern tatsächlich ungenutzte Zyklen beinhalten, die auf die eingebauten Schaltkreise warten, um ihre Transaktionen zu beenden). Das offene Testentwicklungswerkzeug wird zwei Arten der Patternkomprimierung brauchen: Eines für den Datentransfer und die Ablage dieser großen Datei und eines um letztere an den Patternspeicher der Testumgebung anzupassen.

Die meisten Werkzeuge wurden in letzter Zeit um Algorithmen zur Standard-Komprimierung erweitert, wie z.B. ‚Gnu Zip‘ oder ‚WinZip‘. Designingenieure können große BIST- oder ATPG-Ausgangsfiles vor der Abgabe in die Testabteilung mit einer dieser Methoden komprimieren. Das Testentwicklungswerkzeug muss dann in der Lage sein, diese komprimierten Files für die ‚Design-to-Test‘-Analyse und den Testsystemoutput zu lesen.

Die zweite Art der Patternkomprimierung sind einige traditionelle Techniken, die von den Testsystemherstellern selbst eingebaut werden, bekannt als Wiederholungen (re-

peats), Schleifen (loops) und Unterroutinen (subroutines). Die ‚Repeat‘-Komprimierung ist eine einfache Pattern-Reihe mit testsystem-spezifischen Micro-Codes, die die Anzahl angibt, wie oft das Testsystem die Pattern-Reihe auf das zu prüfende Bauelement anwenden muss. Damit werden oft die ungenutzten Zyklen in den BIST-Pattern komprimiert. Ähnlich wiederholt die ‚Loop‘-Komprimierung eine Anzahl von Pattern-Reihen einige Male. Die ‚Subroutinen‘-Komprimierung ist eine Technik, die vom Testsystemhersteller angewandt wird, um einen Pattern-Block, der eine bestimmte Funktion in einem separaten Testsystem-Speicher erfüllt, abzulegen. Diese speziellen Subroutinen werden immer dann dem Testsystem-Standardspeicher zugeschaltet, wenn sie durch einen testsystem-spezifischen Subroutinen-Micro-Code aufgerufen werden.

Für die meisten SoC-Anwendungen, die sich heute auf dem Markt befinden, müssen die ‚Design-to-Production‘-Werkzeuge automatisch beide Arten der Patternkomprimierung zur Verfügung stellen.

Zusammenführung von BIST-Teststrategien mit Testplattformen

Heute wenden Entwickler Softwarespeicher-Managementtechniken an, um an der physikalischen Grenze von 32 Bit pro File arbeiten zu können. Diese Grenze ist durch das Betriebssystem, auf dem die Software läuft, gegeben. BIST-Designs, die eine IEEE 1149.1-Standardschnittstelle implementieren, sind im Augenblick die Erfolgreichsten. Durch diese Schnittstelle kann die Testentwicklungsoftware jegliche Art von BIST unterstützen: Logic, Memory und Analog. Diese Schnittstelle erlaubt auch einen externen Zugriff auf BIST-Schaltkreise in einem SoC durch lediglich fünf der digitalen Pins eines Testsystems (Abb. 2).

Anpassung und Rückanpassung an ein Testsystem

Beginnt der Testentwicklungsprozess mit der Umwandlung von Simulationsergebnissen in zyklische, testplattformneutrale Daten – mit dem Timing für viele ähnliche Kerne – ist die Anpassung an eine spezifische Testsystemplattform ein Vektorumrechnungsvorgang: die einfache Zuordnung (Mapping) vom Schwingungsverlauf zum Testsystemformat. Damit muss die Anpassung an ein Testsystem selbstverständlich nicht mehr beim Simulationsfile beginnen, sondern in einem zyklus-basierten Zustand.

Künftige Aufgaben

Um die Automatisierung vom Design zum Test vollständig zu realisieren, bleibt noch viel zu tun. Die Auswertung von Test-Pattern und -timing, bevor das erste Silizium zur Verfügung steht, sollte ein in den Testfluss integrierter Prozess sein. Ohne diesen Schritt werden zwangsläufig Fehler während des ohnehin sehr kostenintensiven Testens auftreten – oder schlimmer noch – auf dem Markt.

Es werden mehr Rückmeldungsmechanismen zur Unterstützung der Fehleranalyse benötigt. Kein kommerzielles Design-to-Test-Werkzeug dokumentiert und speichert relevante Informationen, wie es Designdaten in Testdaten umwandelt. Eine solche Art von Mechanismus würde die Fehleranalyse erheblich verbessern.

Wenn ein Mixed-Signal-Bauelement mit dem Testsystem geprüft wird, werden die Mixed-Signal-Funktionen noch immer manuell mit dem digitalen Subsystem initiiert, eingerichtet und synchronisiert. Eine einfache Automatisierung dieser Schritte wäre ein gewaltiger Schritt nach vorne.

Beitrag als PDF im Internet:



How to use

more @ click !

1. www.duv24.net
2. ‚more@click‘-Code eingeben
3. Anbieter kontaktieren – diskutieren – recherchieren