Zeit zum Handeln

Steigende Designkomplexität erfordert Wechsel der Verifikationsmethodologie

Verifikation wird im zunehmenden Maße der Engpass, um elektronische Produkte mit hoher Qualität auf den Markt zu bringen. Genauer betrachtet ist die Situation noch prekärer. Der Verifikationsprozess ist bei hochkomplexen Designs nur noch bedingt kontrollierbar. Damit beansprucht Verifikation nicht nur mehr Zeit, vielmehr muss ihre Effektivität bezweifelt werden. BRIAN BAILEY, DR. CHRISTOPH SUEHNEL

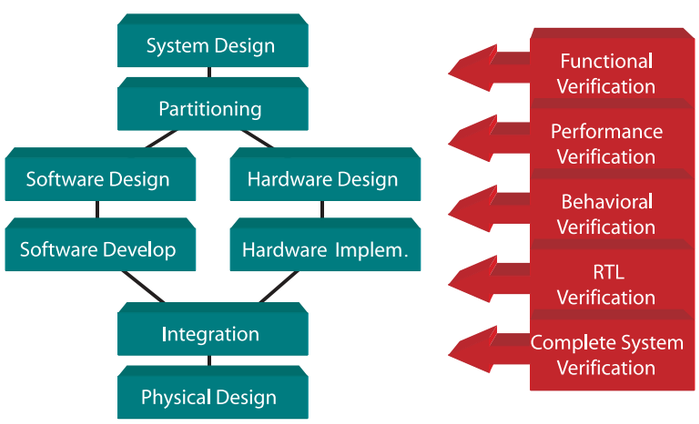

Abb. 1: Design-Flow

Im Folgendem werden Verifikations-Probleme betrachtet und einige von ihnen genauer untersucht, um herauszufinden, wie mit bestimmten Vorgehensweisen die gegenwärtige Situation in diesem Bereich verbessert werden kann. Um zu verstehen, welche Änderungen in einer bestehenden Verifikationsmethodologie vorgenommen werden müssen, ist es wichtig das Problems bis zu seiner Wurzel zurückzuverfolgen. Hier gibt es jedoch nicht nur eine Wurzel, sondern mehrere. Jede einzelne führt uns zu einem anderen Teil der Lösung.
Eine naheliegende Ursache für Probleme ist die Komplexität des Designs. Die Beziehung zwischen der Designgröße und der Verifikations-Komplexität wird von vielen Faktoren beeinflusst, die von zwei Extremen begrenzt werden. Betrachtet man ein Design und weist diesem die Verifikationskomplexität 1 zu, dann verdoppelt sich die Verifikationskomplexität, wenn das gleiche Design parallel dazu geschaltet wird. Das ist darauf zurückzuführen, dass die Tiefe der Verifikationsvektoren etwa gleich bleibt, aber die Breite sich verdoppelt. Ist der zweite Block jedoch in Reihe zum ersten geschaltet, dann vervierfacht sich die Verifikationskomplexität. In der Realität tendieren Designs mehr zum zweiten Extrem. Es gibt jedoch Techniken, mit denen die Situation in Richtung zum ersteren Fall beeinflusst werden kann.
Ein weiterer Faktor neben der Designkomplexität, der oft übersehen wird, ist der Softwareanteil. Letztendlich muss das gesamte System und nicht nur die Hardwarefunktionalität verifiziert werden. Das bürdet dem Verifikationsprozess zusätzliche Lasten auf. Die Berücksichtigung des Softwareaspekts wird dadurch wesentlich erschwert, dass die Unterschiede zwischen Hardware- und Softwaredesign erheblich sind. Das betrifft nicht nur die Werkzeuge und Methodologien , sondern auch ihre kulturellen and Unterschiede und Abweichungen im Management. Systeme mit großem Anteil an benutzer-definierter Software sind im allgemeinen in hohem Grade konfigurierbar. Dadurch werden zusätzliche Herausforderungen an die Verifikation gestellt.

Brian Bailey ist Chief Technologist, Mentor Graphics, Consulting Division

Dr. Christoph Suehnel ist Program Manager bei Mentor Graphics, Consulting Division und Member IEEE

Beobachtbarkeit und Steuerbarkeit sind für die Verifikation von wesentlicher Bedeutung. Unter Steuerbarkeit versteht man das Maß, mit dem ein Design in jeden gewünschten Zustand durch manipulieren der Primäreingänge, bringen kann. Die Steuerbarkeit kann durch bestimmte Aktionen, wie das Vorladen von Memories oder das Setzen von Flipflops positiv beeinflusst werden. Leider sind solche Praktiken meist auf die Systeminitialisierung beschränkt. Zwischen der Verifikationskomplexität und der Steuerbarkeit gibt es einen unmittelbaren Zusammenhang. Die Beobachtbarkeit ist ein Maß dafür, in wieweit der Zustand eines Systems durch Betrachtung der primären Ausgänge bestimmt werden kann. Die Beobachtbarkeit eines Designs hat einen wesentlichen Einfluss darauf, ob ein existierender Fehler während der Verifikation entdeckt werden kann. Ist die Beobachtbarkeit eines Systems gering, dann wirkt ein Test, mit dem ein bestimmtes Ergebnis erzielt werden soll, mehr oder weniger zufällig. Dadurch können viele Fehler die vorhanden sind, unentdeckt bleiben, wenn der Test nicht richtig aufgebaut ist. Abhilfe kann in diesem Falle die sogenannte ‚White Box'-Verifikation schaffen.

Alle Probleme, die bisher diskutiert wurden, stehen im unmittelbaren Zusammenhang mit der Komplexität und Vielfalt des Designs. Das ist aber nur ein Aspekt des gesamten Verifikationsproblems. Sogar wenn der Entwickler das Design perfekt ausgeführt hat, können Fehler durch unvollständige und ungenaue Spezifikationen eintreten. Zum Einen kann es sein, dass der Ingenieur die Spezifikation nicht genau interpretiert hat und zum Anderen, dass die Spezifikation selbst nicht korrekt war. Die Überprüfung dieser Situation wird meist als Validierung bezeichnet. Sie muss aber ebenso im Entwurfsprozess berücksichtigt werden. Die meisten Firmen führen die Validierung heute als letzten Schritt im gesamten Verifikations-Flow aus. Das ist natürlich im höchsten Grade ineffizient.

Alle diese Probleme sollten in einen Testplan adressiert werden. Dieser legt die Vorgehensweise fest, mit der die kritischen Bereiche des Designs identifiziert werden können und definiert die Verifikationsziele.

Design- und Verifikations-funktionen

Wie Abbildung 1 zeigt, beginnt der Entwurfsprozess mit dem Systemdesign, das entweder aus einer Textbeschreibung oder einer ausführbare Spezifikation besteht. Im zweiten Schritt erfolgt die Partitionierung des Systems in Hardware- und Software. D.h. die Systemarchitektur wird definiert. Dabei wird auch festgelegt, in welcher Weise diese beiden Systemkomponenten miteinander kommunizieren. Dieser Teil des Entwurfsprozesses ist im hohen Grade iterativ. Nachdem die Spezifikationen für die Hardware- und Softwarefunktionalität erstellt wurden, beginnen die beiden Entwicklungsteams (Hardware, Software) mit der Implementierung der jeweiligen Funktionalität. Bevor die Layoutgenerierung beginnen kann, muss die Integration von Hardware- und Softwareanteil erfolgen. Abbildung 1 ignoriert die Iterationsschleifen in diesem Flow, um die Darstellung einfach zu halten.

Der Verifikations-Flow läuft parallel zum Design-Flow. Auf der Ebene der ausführbaren Spezifikation liegt der Fokus auf funktioneller Verifikation. Nach der Partitionierung in High-level-Hardware- und Softwarefunktionalitäten richtet sich die Aufmerksamkeit auf die daraus resultierenden Architekturen und die Systemperformance. Wenn die Partitionierung einmal festgelegt ist, rückt die Verifikation des funktionellen Verhaltens in den Mittelpunkt (behavioral verification). Nachdem Hardware- und Softwaredesign abgeschlossen sind, werden sie einzeln verifiziert und nach ihrer Integration gegeneinander getestet.

Falls Firmen keinen formalisierten Design-Flow, wie oben gezeigt, haben, so müssen sie doch sicherstellen, dass alle Verifikationsaspekte berücksichtigt werden. Wichtig für eine effiziente Arbeitsweise ist, dass die einzelnen Aspekte in der richtigen Reihenfolge getestet werden. Zum Beispiel kann die funktionelle Verifikation nicht einfach weggelassen werden, weil kein High-level-Modell zur Verfügung steht. Die funktionelle Verifikation auf einen späteren Zeitpunkt im Design-Flow zu verschieben ist weniger effektiv, weil dann genauere Modelle verwendet werden müssen als eigentlich nötig sind.

Verifikationskomponenten

Die Defizite moderner Simulatoren bestehen nicht darin, dass Kapazität oder Geschwindigkeit zu gering sind, problematisch ist vielmehr die Vielfalt von Aufgaben, die sie abdecken müssen, einschließlich Stimuli-Generierung, Response-Vergleich, Coverage-Berechnung sowie Management der enormen Datenmengen, die mit der Ausführung des Testplans verbunden sind. Die Komplexität der Verifikationsaufgaben hat dazu geführt, dass einzelne, kleinere Firmen exzellente Point-Tools für die jeweiligen speziellen Teile des Verifkationspuzzles auf den Markt gebracht haben.

Einige dieser Point-Tools können direkt über ein API-Interface an den Logiksimulator angekoppelt werden. Abbildung 2 zeigt die Hauptteile einer Verifikationsumgebung. Mit Ausnahme des Blockes ‚Equivalence Checking' sind sie nicht speziell bezogen auf einen bestimmten Abstraktionslevel der Gesamtverifikation, sondern sind gleichermaßen für alle Level anwendbar.

Dreh und Angelpunkt ist die ‚Execution Engine'. Darunter versteht man Methoden wie Simulation, Emulation, Co-Simulation und sogar ‚Rapid Prototyping' oder Testchips. Es ist die Stelle an der das Design entweder in virtueller (Design-Daten, Modelle) oder realer Form (FPGA, Testchips) existiert und auf das Stimuli angewendet werden. Das Design selbst kann aus Designdaten (z.B. RTL-Beschreibung), die von den Entwicklern erzeugt wurden und IP-Blöcken, die von anderen Designs oder Designgruppen wie auch externen IP-Providern übernommen werden, bestehen.

Im Stimulus-Block erfolgt die Generierung und Anwendung von Information auf die Primäreingänge des Designs. Das können einfach Pattern sein, wie auch hochkomplexe Modelle, die mnemonische Kommandos in Sequenzen von Ereignissen umsetzen, die auf das Design angewendet werden. Diese Art von Modell wird üblicherweise als Transaktor bezeichnet.

Falls die Stimuli von einem Transaktor kommen, ist ein Transaktionsgenerator nötig. Das kann z.B. ein File-Reader sein, der eine Transaktion aus einem File liest und diese an den Transaktor weiterreicht. Oder es ist ein Zufalls-Transaktionsgenerator. Auf diesem Gebiet kommen zunehmend mehr Tools auf den Markt. Äquivalent zum Stimulus-Block ist der Response-Block. Dieser speichert die durch die Stimuli erzeugten Ergebnisse des Designs. In seiner einfachsten Form ist das Ergebnis ein Patternvektor, der in einem File abgelegt wird. Es kann jedoch auch ein Transaktor sein, der zum Beispiel einen Protokolltest ausführt und Transaktionen an seinem Ausgang erzeugt, wenn bestimmte Pattern am Eingang anliegen.

Der Checker-Block ist der Teil in einem Verifikations-Flow, der heute bei der Implementierung einer Testbench den höchsten Aufwand erfordert. Die gesamte Testbench besteht aus den Blöcken Stimulus, Generator, Response, Checker und Predictor. Häufig sind Checker und Predictor in einem Block zusammengefasst. Sie bekommen ihre Daten direkt vom Generator. Diese Strategie ist weit verbreitet. Leider erlaubt sie im Allgemeinen nicht die Wiederverwendung von Testbench-Blöcken. Daher ist ihre Anwendung nicht empfehlenswert.

Die Frage, wann ist die Verifikation abgeschlossen, ist von großer Bedeutung. Leider gibt es diese Frage keine einfache Antwort. Das liegt daran das die meisten Verifikationsmethoden über keine Metrik verfügen, die Auskunft über den Verifikationsfortschritt geben. Ausnahmen sind die coverage-basierenden Verfahren (Line-Coverage, funktionale Coverage).

Der Coverage-Block kann dazu verwendet werden, um den Stimuli-Generator zu beeinflussen, sodass Stimuli für spezielle Corner-Cases erzeugt werden können. Durch Auswertung der Ergebnisse der Corner-Case-Verifikation kön-

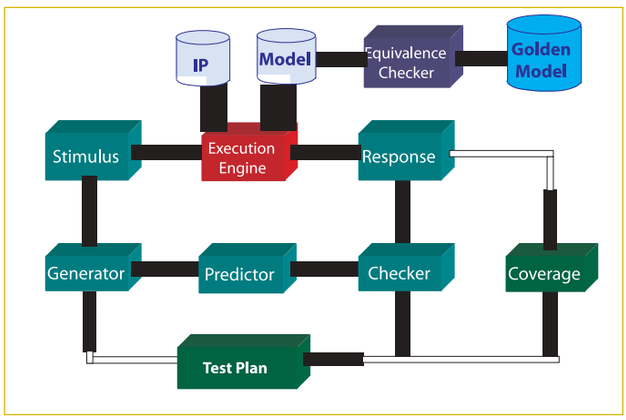

Abb. 2: Verifikation-Management

nen funktionelle Probleme identifiziert und damit auch eliminiert werden. Ebenso wichtig ist aber der Nachweis der geforderten Funktionalität für das Design. Ohne einen vollständigen Verifikations- oder Testplan kann das nicht gewährleistet werden. Die Risiko, das wichtige und kritische Funktionalität nicht abgeprüft wird ist ohne einen solchen Plan groß.

Formale Verifikation hat eine lange Zeit mehr versprochen als geleistet. Jetzt beginnt sie das zu liefern, was sie bisher für sich in Anspruch genommen hat. ‚Equivalence Checking' ist die am weitesten entwickelte Funktion. Sie erlaubt, dass ein Design gegen ein Referenz-Model getestet werden kann. Das Referenz-Modell ist häufig ein RTL-Modell. Es wird mit einem Gate-Level-Modell verglichen, das durch Synthese erzeugt wurde. Auch modifizierte Netzlisten (z. B. nach der Scan-Insertion) können gegen ein RTL-Referenz-Modell getestet werden. Formale Verifikation, gekoppelt mit Statischer-Timing-Analyse (einer weiteren formalen Methode) kann die Notwendigkeit einer Gate-Level-Simulation vollständig ablösen. Weiterhin gibt es statische Methoden, die – obwohl sie nicht zu den formalen Methoden gehören – dazu beitragen können, Probleme zu identifizieren, bevor eine funktionale Verifikation ausgeführt wurde. Diese Tools werden häufig als LINT-Tools bezeichnet, und leisten im allgemeinen viel mehr als einfache Syntax-Checks. Sie decken State-Machine-Verifikation und ähnliche Gebiete ab. Tools zum Model-Checking stecken immer noch in den Kinderschuhen. Man versteht darunter den Vergleich von High-level-Beschreibungen gegeneinander. In letzter Zeit sind aber auch hier einige wesentliche Fortschritte gemacht wurden.

Schließlich muss die gesamte Verifikationsumgebung verwaltet werden. Welche Testcases untersuchen welchen Teil des Designs? Sind alle anwendbaren Testcases wirklich ausgeführt worden? Welche Version der Testcases, des Designs und der Tools wurden benutzt? Die Menge der Information, die während der Verifikation erzeugt werden, wird zunehmend größer. Ohne ein komplette und effektives Verifikations-Management besteht die Gefahr, dass der gesamte Verifikationsprozess uneffektiv bleibt.

Ausblick

Das Gebiet der Verifikation hat sich in den letzten Jahren explosionsartig erweitert. Nur sehr wenige Unternehmen haben die Zeit und Ressourcen gehabt, um in ein umfassendes Review ihrer internen Verifikationsmethodologie zu investieren. Bestenfalls wurde über die Einführung neuer Strategien nachgedacht. Jetzt ist die Zeit, um neue Verifikationsstrategien zu implementieren, bevor es zu spät ist.

more@click DV52252