Simulation von Gatternetzlisten – VHDL und Mixed-mode

Werkzeuge : Cadence NCSim
Design-Kits : AMS Hit-Kit
edaSetup : `ldv ams`

Andreas Mäder

Diese Anleitung beschreibt die grundlegenden Schritte, um mit einer VHDL-Testumgebung Gatternetzlisten, die von einem Synthesewerkzeug erzeugt worden sind, zu simulieren.

Vor dem Layout prüfen derartige Simulationen die Funktionalität der Schaltung unter Berücksichtigung der Gatterverzögerungszeiten. Zudem ist so auch gewährleistet, dass die Synthese „richtige“ Ergebnisse geliefert hat und keine Programm- oder Bedienungsfehler aufgetreten sind.

Nach dem Layout, der Platzierung und Verdrahtung, können zusätzlich die Leitungslaufzeiten in der Netzlistensimulation berücksichtigt werden. Bei Standardzellentwürfen ist eine abschließende Simulation mit „Backannotation“ unerlässlich, bevor das ASIC zur Fertigung freigegeben werden kann.

Da das spezielle Vorgehen von der Gatterbibliothek und den verwendeten Werkzeugen (Simulator, Timing-Analyzer, Layoutprogramme) abhängig ist, werden im folgenden zwei prototypische Abläufe vorgestellt:

- Simulation von VHDL-Netzlisten: allgemeine Hinweise
- Mixed-mode Simulation von Verilog-Netzlisten. Das Beispiel beschreibt speziell das Vorgehen, bei Benutzung der CADENCE NC-Simulatoren und den Zellbibliotheken des AMS $0,35\,\mu m$ Prozesses und ergänzt damit die Anleitung zur *„VHDL-Synthese“*.

Voraussetzung

Im Folgenden wird immer eine Hierarchie vorausgesetzt, wie sie in Abbildung 1 skizziert ist. Zu implementieren ist eine (hierarchische) VHDL-Beschreibung `entity` ⟨*`topLevel`*⟩, die innerhalb einer Testumgebung simuliert wurde: `entity` ⟨*`testEnv`*⟩, `architecture` ⟨*`testbench`*⟩ = Datei ⟨*`testEnv`*⟩`.vhd`. Durch die Synthese wurde eine hierarchische Netzliste erzeugt, siehe *„VHDL-Synthese“*, und in einer einzigen Datei ⟨*`netlist`*⟩`.vhd/.v` gesichert. Als Konfiguration der vorhandenen Testumgebung soll diese Netzliste jetzt simuliert werden.

Zeitverhalten und Backannotation

Bei aktuellen Submikron-Technologien liegen die Leitungsverzögerungen teilweise deutlich über den eigentlichen Gatterlaufzeiten. Um dies bei der Simulation von Netzlisten zu berücksichtigen, kann man

vor dem Layout die Leitungsverzögerungen mit heuristischen Modellen (Anzahl der Gatter, Verbindungsstruktur, etc. → geschätzte Verdrahtungslängen + Ausgangslast) als *statische Timinganalyse* ermitteln.

nach dem Layout aus der Topologie Verzögerungsmodelle (RC-Modelle) extrahieren und so sehr genau die Effekte durch Leitungslaufzeiten für eine *„Backannotation“* berechnen.

Hier wird nur ein vergleichsweise einfaches heuristisches Modell berücksichtigt, das eine durch den Syntheseprozess erzeugt SDF-Datei (*S*tandard *D*elay *F*ormat) benutzt.[1] Das Verfahren ist in dem zweiten Abschnitt zur mixed-mode Simulation skizziert.

[1] Die genaue Bedienung der hochspezialisierten Werkzeuge zu Timinganalyse und Backannotation würde den Rahmen so eines „Kochrezepts“ sprengen.

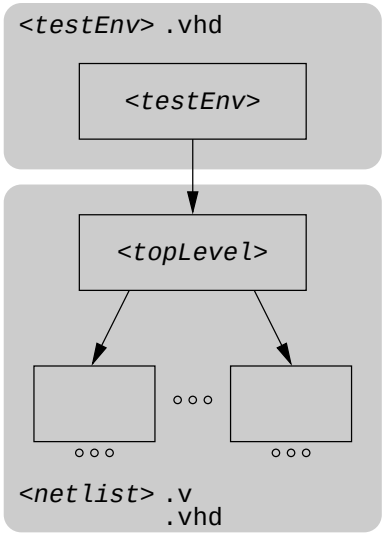

Abbildung 1: Hierarchie und Dateistruktur

VHDL-Simulation, allgemein

Die Schritte dieses Abschnitts gelten unabhängig von bestimmten Simulatoren und beziehen sich allgemein auf die Konfiguration von VHDL-Hierarchien. Für die hier installierten Standardzell-Entwurfsumgebungen wird eine mixed-mode Simulation empfohlen, wie sie im zweiten Teil, ab Seite 4, beschrieben ist.

1. Netzliste nachbearbeiten

Vor der Simulation sind noch einige Änderungen in der Netzlistendatei notwendig; dazu kann ein beliebiger Texteditor benutzt werden.

> ⟨*edit*⟩ ⟨*netlist*⟩`.vhd` [shell]

Zuerst sind die „logischen Namen“ der Zellbibliotheken zu ergänzen: die Synthese erzeugt zwar Verweise auf die IEEE-Bibliotheken, Referenzen auf die Zellbibliothek fehlen aber noch. Abhängig von dem Prozess, bzw. dem FPGA-Typ sind die passenden Bibliotheken einzutragen.[2] Der Verweis auf die Bibliothek wird dabei ersetzt:
`library IEEE;` ⇒ `library IEEE,` ⟨*refLib1*⟩`,` ⟨*refLib2*⟩`... ;`

Design-Kit	Zellbibliotheken
AMS	`c35_corelib`, `c35_iolib`, `c35_iolibv5` ...
ALTERA	`altera_mf`, `cycloneiii`, `apex20ke` ...
XILINX	`unisims`, `logiblox`, `xc9000` ...

Die Synthese erzeugt ein zusätzliches Package `CONV_PACK_`⟨*topLevel*⟩ mit eigenen Deklarationen. In der Regel wird dieses Package nicht benötigt und kann, zusammen mit den Verweisen darauf (`use work.CONV_PACK_`⟨*topLevel*⟩`all;`) entfernt wer-

[2] Wegen der ständigen Aktualisierung der Design- und FPGA-Kits, sollte man sich die jeweilige Dokumentation ansehen, bzw. bei mir nachfragen.

den. Werden im Code die arithmetischen Arraytypen `signed` und `unsigned` benutzt, ist die Referenz auf den „offiziellen“ Standard `ieee.numeric_std` zu ändern.

In der synthetisierten Datei sind, außer den Netzlisten (den `architectures`), auch die zugehörigen `entity`-Deklarationen enthalten. Um in der VHDL-Testumgebung problemlos zwischen den Architekturen *vor* und *nach* der Synthese auswählen zu können, sollten diese Deklarationen eigener Entities in ⟨*netlist*⟩`.vhd` auskommentiert werden;[3] zusätzliche Entities (z.B. DesignWare Komponenten) bleiben stehen.

2. Konfiguration der Testumgebung

Für die Testumgebung wird jetzt eine neue Konfiguration erzeugt, die die synthetisierte Netzliste als instanziierte Komponente einbindet. Die Konfiguration kann, als eigenständige Entwurfseinheit, in einer extra Datei stehen, meist werden Konfigurationen direkt in der Datei mit der Testumgebung deklariert.

> ⟨*edit*⟩ ⟨*testEnv*⟩`.vhd` [shell]

Die Bezeichner der synthetisierten Netzliste entsprechen folgendem Schema:

- die Namen der `entity`s ändern sich nicht (s.o.).
- auch die Label der Instanzen, für hierarchische Konfigurationen, bleiben wie im ursprünglichen VHDL-Code.
- Bezeichnern der `architecture`s wird `SYN_` vorangestellt.

Die Default-Konfiguration, bei der die gesamte Hierarchie durch die Syntheseausgabe ersetzt wird, sieht dann folgendermaßen aus:

```
configuration ⟨configId⟩ of ⟨testEnv⟩ is
  for ⟨testbench⟩
    for ⟨instLabel⟩: ⟨topComponent⟩ use entity work.⟨topLevel⟩(SYN_⟨archId⟩);
    end for;
  end for;
end configuration ⟨configId⟩;
```

Sollen Verhaltensmodelle und synthetisierte Netzlisten gemischt werden, dann muss die Konfiguration hierarchisch erweitert werden, beispielsweise als:

```
configuration ⟨configId⟩ of ⟨testEnv⟩ is
  for ⟨testbench⟩
    for ⟨instLabel⟩: ⟨topComponent⟩ use entity work.⟨topLevel⟩(SYN_⟨archId⟩);
      for SYN_⟨archId⟩
        for ⟨i1Label⟩: ⟨component1⟩ use entity work.⟨entity1Id⟩(SYN_⟨arch1Id⟩);
        end for;
        for ⟨i2Label⟩: ⟨component2⟩ use entity work.⟨entity2Id⟩(⟨arch2Id⟩);
        end for;
        ...              weitere Konfigurationen der zweiten Hierarchieebene
      end for;
    end for;
  end for;
end configuration ⟨configId⟩;
```

[3] So vermeidet man Probleme mit den Zeitstempeln der EDA-Werkzeuge.

3. VHDL-Simulation der synthetisierten Netzliste

   Anschließend kann wie gewohnt simuliert werden: Codeanalyse der Quelldateien, Elaboration der Schaltung und Simulation (der Konfiguration).

Simulator		Analyse	Elaboration	Simulation
CADENCE	NC-Sim	`ncvhdl`	`ncelab`	`ncsim`
SYNOPSYS	Scirocco	`vhdlan`	`vcs`	`simv`

Mixed-mode Simulation

Bei der Simulation aus Gatterebene werden meist Verilog-Netzlisten benutzt, da die Simulationsgeschwindigkeit oft schneller ist als bei VHDL-Code.[4] Insbesondere für abschließende Tests vor der Fertigung (*golden Simulation*) fordern viele Hersteller explizit Verilog Simulationen mit Backannotation.

Gemischte Simulationen verbinden (komfortable) VHDL-Testumgebungen mit (schnellen) Verilog-Netzlisten. Der hier vorgestellte Ablauf benutzt den NC-Simulator von CADENCE, der mehrere Hardwarebeschreibungssprachen (VHDL, Verilog und SystemC) verarbeitet; die HDLs (*H*ardware *D*escription *L*anguage) werden nur bei der Codeanalyse unterschiedlich behandelt. Als Zellbibliothek wird der AMS $0,35\,\mu m$ Prozess `c35b4` benutzt.[5]

Timinganalyse

Während früher die Zellbibliotheken – dies gilt sowohl für VHDL-, als auch für Verilog-Modelle – Gatterverzögerungszeiten enthielten, benutzen inzwischen fast alle ASIC-Hersteller *unit-Delay* Modelle mit konstanten Verzögerungszeiten für alle Gatter: 1 ns, 10 ps… Der Grund dafür ist, dass das Zeitverhalten parametrisierbar sein muss, um

- den Bereich der Fertigungsstreuungen (Min.-/Typ.-/Max.-Delay) abzudecken.
- verschiedene Betriebsbedingungen (Temperatur, Spannung…) zu berücksichtigen.
- Ausgangslasten durch Fan-Out und durch geschätzte oder aus dem Layout extrahierte Leitungslängen zu simulieren.

Anstatt die Parameter in der Simulation nur aus Bibliotheken auszuwählen, werden deshalb externe Programme zu statischen Timinganalyse eingesetzt, die die Verzögerungszeiten für die gewünschten Parameter berechnen und eine SDF-Datei (*S*tandard *D*elay *F*ormat) ausgeben. In der Simulation ersetzt diese (standardisierte) Datei dann die unit-Delay Zeiten.

- SDF-Datei

  Bei dem Syntheseprozess, wie er in *„VHDL-Synthese“* beschrieben ist, wurde schon eine SDF-Datei ⟨*netlist*⟩`.sdf` erzeugt.

[4] Weil Verilog als Sprache älter als VHDL ist, sind, insbesondere auf Gatterebene, die Simulationsalgorithmen stärker optimiert.

[5] **Achtung:** auch wenn das prinzipielle Vorgehen für andere Fertigungsprozesse oder FPGA-Entwürfe ähnlich ist, so lassen sich die Schritte in der Regel nicht übertragen. Je nach Hersteller (≡ verwendeter Zellbibliothek), unterscheiden sich die Schritte bei der Timinganalyse voneinander.

1. SDF-Datei compilieren

   Ähnlich der Codeanalyse wird die Datei in ein simulatorinternes Format übersetzt und erzeugt eine Ausgabedatei ⟨*netlist*⟩`.sdf.X`.

   > `ncsdfc` ⟨*netlist*⟩`.sdf` [shell]

2. SDF-Steuerdatei anlegen

   Vor der Simulation muss die, vom Syntheseskript generierte, Steuerdatei der SDF-Backannotation noch angepasst werden. Sie legt fest, wie die SDF-Datei heißt und auf welchen Teil der Hierarchie sich die Verzögerungszeiten beziehen, also welches Label die Instanz ⟨*topLevel*⟩ besitzt: ⟨*instLabel*⟩.

   > ⟨*edit*⟩ `netlist.sdf.cmd` [shell]

   ```
   COMPILED_SDF_FILE = "⟨netlist⟩.sdf.X",        compilierte SDF-Datei
   SCOPE = :⟨instLabel⟩,                          ⟨topLevel⟩-Label
   LOG_FILE = "⟨netlist⟩.sdf.log";                Log-Datei
   ```

Simulation

Im Folgenden werden noch die Schritte gezeigt, um die mixed-mode Simulation zu starten. Die Simulation selbst, Start und Handhabung des Simulators, unterscheidet sich nicht von der gewohnten Vorgehensweise.

– Verilog-Datei

  Die Synthese, in *„VHDL-Synthese“* beschrieben, hat als Ausgabe eine Verilog-Netzliste des Entwurfs in der Datei ⟨*netlist*⟩`.v` geschrieben.

3. VHDL `configuration` für die mixed-mode Simulation erstellen

   Zur Erinnerung: in VHDL können Konfigurationen benutzt werden, um innerhalb von Hierarchien Paare aus Entity+Architecture an Komponenten zu binden. Dadurch können in Testumgebungen Instanzen vor der Synthese (VHDL Code als Register-Transfer Modell) gegen synthetisierte Modelle (Verilog Gatternetzlisten) ausgetauscht werden. Um eine eindeutige Zuordnung zu haben, empfehlen sich explizite `configuration`s.

   Der VHDL-Bezeichner der Architektur ist für die Verilog-Netzlisten fest vorgegeben `module`. Die Port-Kompatibilität von Komponente und top-Level Entity wird vorausgesetzt, so dass bei der Konfiguration keine weiteren Port-Maps oder Typkonvertierungen notwendig sind — wie auch in den VHDL Beispielen Seite 3.

   > ⟨*edit*⟩ ⟨*testEnv*⟩`.vhd` [shell]

   ```
   configuration ⟨configId⟩ of ⟨testEnv⟩ is
     for ⟨testbench⟩
       for ⟨instLabel⟩: ⟨topComponent⟩ use entity work.⟨topLevel⟩(module);
       end for;
     end for;
   end configuration ⟨configId⟩;
   ```

4. Netzliste nachbearbeiten

Vor der Simulation müssen noch fehlende Zeitskalierungen in der Verilog-Datei eingetragen werden.

> ⟨*edit*⟩ ⟨*netlist*⟩.v [shell]

```
` timescale 10ps/1ps                    Zeitskala Einfügen
module ⟨entityId⟩                       vor erstem Module
...
```

5. Codeanalyse und Elaboration

Die Schritte vor der Simulation können zwar auch über die grafische Oberfläche `nclaunch` ausgeführt werden, aber die Festlegung der SDF-Steuerdatei geht mit der Kommandozeile einfacher.

> `ncvlog -linedebug` ⟨*netlist*⟩`.v` [shell]

> `ncvhdl -linedebug -v93` ⟨*testEnv*⟩`.vhd` [shell]

> `ncelab -access rwc -sdf_cmd_file netlist.sdf.cmd` ⟨*configId*⟩ [shell]

Dabei auftretende Warnungen „`Too few module port connections.`“, beispielsweise bei Flipflops mit mehreren Ausgängen (z.B. `DFA`), können ignoriert werden oder lassen sich per Kommandozeilenoption abschalten:
`ncelab -nowarn SDFNCAP -nowarn CUVWSP ...`

Anschließend wird die Simulation wir gewohnt aufgerufen, entweder über die GUI oder in der Kommandozeile.

> `ncsim -gui` ⟨*configId*⟩ [shell]