VHDL- und mixed-mode Netzlistensimulation

Werkzeuge : CADENCE NCSim
Design-Kits : AMS Hit-Kit
designSetup : `ldv ams`

A. Mäder

Diese Anleitung beschreibt die grundlegenden Schritte, um innerhalb einer VHDL-Testumgebung Gatternetzlisten, als Ausgabe eines Syntheseprozesses, 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 einer VHDL-Netzliste, allgemein
- Mixed-mode Simulation einer Verilog-Netzliste, für den AMS $0,35\,\mu m$ Prozess mit den CADENCE NC-Simulatoren (*N*ative *C*ompiled)

Voraussetzung
Im Folgenden wird immer eine Hierarchie vorausgesetzt, wie sie in Abbildung 1 skizziert ist. Ausgangspunkt ist eine (hierarchische) VHDL-Beschreibung einer Schaltung, `entity` ⟨*topLevel*⟩, die in einer Testumgebung simuliert wurde: `entity` ⟨*testEnv*⟩, `architecture` ⟨*testbench*⟩ zusammen in einer Datei ⟨*testEnv*⟩`.vhd`. Bei der anschließenden Synthese wurde eine hierarchische Netzliste erzeugt, wie in „VHDL-Synthese“ beschrieben, und in einer einzigen Datei ⟨*netlist*⟩`.vhd/.v` gesichert. Durch entsprechende Konfiguration der bestehenden Testumgebung soll diese Netzliste jetzt simuliert werden.

Zeitverhalten und Backannotation
Egal ob VHDL- oder Verilog-Netzlisten und -Gattermodelle simuliert werden, bei der Simulation muss der Designer wissen, welche Zeitmodelle in den herstellerspezifischen Zellbibliotheken benutzt werden. Weil bei aktuellen Submikron-Technologien die Leitungsverzögerungen im Bereich der Gatterlaufzeiten liegen, sind Simulationen, die *nur* die Schaltzeiten der Gatter berücksichtigen, viel zu ungenau. Schon vor dem Layout (Platzierung & Verdrahtung) werden deshalb Werkzeuge zur *statischen Timing-Analyse* eingesetzt, die das Schaltverhalten der Gatter abhängig von deren Ausgangslast (nachgeschaltete Eingänge und heuristisch ermittelte Verdrahtungseffekte) modellieren.

Weil für eine Simulation auf Netzlistenebene ohnehin eine Backannotation – Import des (extern) berechneten Zeitverhaltens – stattfindet, enthalten die meisten Gatterbibliotheken nur *unit-Delay* Modelle, bei denen alle Gatter die gleiche, konstante Verzögerungszeit besitzen. Die dazu notwendigen Schritte sind in dem Abschnitt zur mixed-mode Simulation skizziert.

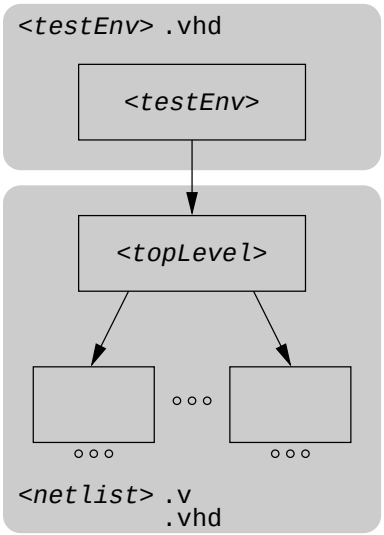

Abbildung 1: Hierarchie und Dateistruktur

VHDL-Simulation, allgemein

Die Schritte dieses Abschnitts gelten nicht für bestimmte Simulatoren, sondern 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. FPGA-Typ sind die passenden Bibliotheken einzutragen.[1] Der Verweis auf die Bibliothek wird dabei ersetzt:
`library IEEE;` ⇒ `library IEEE,` ⟨*refLib1*⟩`,` ⟨*refLib2*⟩`... ;`

Design-Kit	Zellbibliotheken
AMS	`c35_corelib`, `c35_iolib_4m`, `c35_iolibv5_4m` ...
ALTERA	`cyclone`, `stratix`, `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 werden. Werden im Code die arithmetischen Arraytypen `SIGNED` und `UNSIGNED` benutzt, ist die Referenz auf den „offiziellen" Standard `ieee.numeric_std` zu ändern.

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

In der synthetisierten Datei sind, außer den Netzlisten (den `architecture`s), auch die zugehörigen `entity`-Deklarationen enthalten. Um später in der VHDL-Testumgebung frei zwischen den Architekturen *vor* und *nach* der Synthese auswählen zu können, dürfen Entity-Deklarationen der Verhaltensmodelle nicht in der Netzlistendatei wiederholt werden, da die meisten VHDL-Simulatoren Zeitmarken zur Konsistenzsicherung mitführen. Deshalb sollten *erst* die Entity und *dann* alle zugehörigen Architekturen analysiert werden. Beispielsweise können Deklarationen eigener Entities in ⟨*netlist*⟩`.vhd` auskommentiert werden — zusätzliche Entities, wie DesignWare Komponenten, müssen natürlich hier stehen bleiben.

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 ist sie direkt in der Datei der Testumgebung enthalten.

> ⟨*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. VHDL-Simulation der synthetisierten Netzliste

   Anschließend kann wie gewohnt simuliert werden: Codeanalyse der Quelldateien, Elaboration und Simulation.

Simulator		Analyse	Elaboration	Simulation
CADENCE	NC-Sim	`ncvhdl`	`ncelab`	`ncsim`
SYNOPSYS	Scirocco	`vhdlan`	`scs`	`scsim`, `scirocco`
	VSS	`vhdlan`		`vhdlsim`, `vhdldbx`
MENTOR GRAPHICS	ModelSim	`vcom`		`vsim`

Mixed-mode Simulation

Wegen der höheren Simulationsgeschwindigkeit von Verilog-Netzlisten empfiehlt es sich auf Gatterebene Verilog als Hardwarebeschreibungssprache einzusetzen.[2] 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, 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.[3]

Backannotation vorbereiten

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 Timing-Analyse 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.

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

[3] **Achtung:** auch wenn das prinzipielle Vorgehen für andere Fertigungsprozesse ähnlich ist, so lassen sich die Schritte in der Regel nicht übertragen. Eigenschaften der Zellbibliothek legen fest, ob eine Timing-Analyse überhaupt möglich ist und mit welchen Technologieparametern.

1. SDF-Datei erzeugen

Das Syntheseskript `ams_synopsys` hat schon eine Steuerdatei zur Timing-Analyse mit den Gatterbibliotheken erzeugt. In dieser Datei `pearl.cmd` müssen jetzt noch die Namen der Verilog-, der SDF-Datei und der Bezeichner der top-level Entity angepasst werden:

> ⟨*edit*⟩ `pearl.cmd` [shell]

```
...                                                                    keine Änderungen
ReadVerilog  ⟨netlist⟩.v                                               Verilog-Datei
TopLevelCell ⟨topLevel⟩                                                Entity-Name
# or
# ReadSPFNetlist TOPLEVEL.spf

EstimateWireloads -rc -topology balanced -group sub_micron -name 10k
# or
# ReadSPF -bus_delimiter <> TOPLEVEL.spf

WriteSDFDelays -version 2.1 -precision 3 -ns -no_negative_delays ⟨netlist⟩.sdf
Quit
```

Anschließend wird die Timing-Analyse mit dem fertigen Skript gestartet. Dabei werden die Prozessbedingungen eingestellt und der Einfluss des Layouts, beziehungsweise der Leitungskapazitäten, geschätzt.

> `pearlCell pearl.cmd` [shell]

2. 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]

3. SDF-Steuerdatei anlegen

Vor der Simulation muss die, vom Syntheseskript generierte, Steuerdatei der SDF-Backannotation noch angepasst werden. Sie enthält die Information, 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 vorbereiten

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.

4. Konfiguration für mixed-mode Simulation schreiben

Wie bei der reinen VHDL-Simulation wird die Netzliste über eine Konfiguration an die instanziierte Komponente gebunden. 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⟩;
```

5. 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
...
```

6. 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 -sdf_cmd_file netlist.sdf.cmd ⟨*configuration*⟩ [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 -sdf_cmd_file ...`

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

> ncsim -gui ⟨*configuration*⟩ [shell]