VHDL- und mixed-mode Netzlistensimulation

Werkzeuge : CADENCE LDV
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.

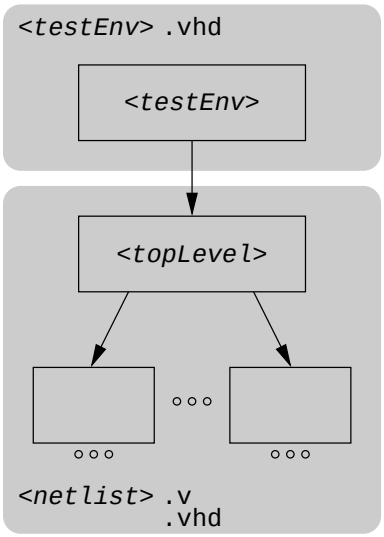
Vor dem Layout prüfen derartige Simulationen die Funktionalität der Schaltung unter Berücksichtigung der Gatterverzögerungszeiten. Zudem ist so auch gewährleistet, dass die Synthese „richtige“ Ergebnisse geliefert hat und keine Programm- oder Bedienungsfehler aufgetreten sind.

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, AMS 0,35 μ-Prozess `csx`, CADENCE LDV

Voraussetzung
Im Folgenden wird immer eine Hierarchie vorausgesetzt, wie sie in Abbildung 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.



VHDL-Simulation, allgemein

Die Schritte dieses Abschnitts beziehen sich nicht auf spezielle Simulatoren, sondern beziehen sich allgemein auf die Konfiguration von VHDL-Hierarchien. Für die derzeit am AB TAMS 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, in dem Beispiel `vi`.

> `vi` ⟨*netlist*⟩`.vhd` [xterm]

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] Die Änderungen können durch Suchen und Ersetzen eingebaut werden: `1,$s/library IEEE;/library IEEE,` ⟨*refLib1*⟩`,` ⟨*refLib2*⟩`... ;/`

Design-Kit	Zellbibliotheken
AMS	`csx_HRDLIB`, `csx_HRDLIB_3B`, `csx_IOLIB_3M`, `csx_IOLIBV5_3M` ...
ES 2	`ecpd07`
ALTERA	`flex10k_ftgs`, `flex8000_ftgs`, `max9000_ftgs`, `max7000_ftgs`,...
XILINX	`unisim`, `simprim`, `logiblox`, `xdw`, `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.

In der von SYNOPSYS erzeugten VHDL-Datei sind, außer den synthetisierten Netzlisten/den `architectures`, 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. Die meisten VHDL-Simulatoren führen Zeitmarken zur Konsistenzsicherung mit, so dass *erst* die Entity und *dann* alle zugehörigen Architekturen analysiert werden sollten. Beispielsweise können in ⟨*netlist*⟩`.vhd` enthaltene Deklarationen eigener Entities auskommentiert werden — zusätzliche Entities, wie DesignWare Komponenten, müssen natürlich hier stehen bleiben.

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

2. Konfiguration der Testumgebung
In einer Konfiguration wird jetzt die „neue“ synthetisierte Netzliste als Architektur der in der Testumgebung instanziierten Komponente eingebunden. Die Konfiguration als eigenständige Entwurfseinheit kann in einer extra Datei stehen, meist ist sie aber in direkt in der Datei der Testumgebung enthalten.

> `vi` ⟨*testEnv*⟩`.vhd` [xterm]

Eine neue Konfiguration der Testumgebung instanziiert dann die synthetisierte Architektur. Die Bezeichner der Netzliste werden wie folgt gebildet:

- 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 – die gesamte Hierarchie wird durch die Syntheseausgabe ersetzt – 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
SYNOPSYS	VSS	`vhdlan`		`vhdlsim`, `vhdldbx`
	Cyclone	`vhdlan`	`cylab`	`cysim`
	Scirocco	`vhdlan`	`scs`	`scsim`
CADENCE	Leapfrog	`cv`	`ev`	`sv`
	NC-Sim	`ncvhdl`	`ncelab`	`ncsim`
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 beide HDLs gleichermaßn verarbeitet. Eine Unterscheidung der Eingabesprache geschieht nur durch getrennte Programme zur Codeanalyse. Als Zellbibliothek wird der AMS 0,35μ Prozess `csx` benutzt.[3]

Backannotation vorbereiten
Während früher die Zellbibliotheken – dies gilt sowohl für VHDL-, als auch für Verilog-Modelle – Gatterverzögerungszeiten enthielten, sind inzwischen bei fast allen Herstellern die Zellen mit *unit-Delay* Modellen beschrieben: jedes Gatter besitzt eine konstante Verzögerungszeit (1 ns, 10 ps…). Der Grund dafür ist, dass die Verzögerungszeiten trotzdem parametrisierbar sein müssen, 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 über Heuristiken auszuwählen, werden deshalb externe Programme zu statischen Timing-Analyse eingesetzt. Sie berechnen die Verzögerungszeiten für die gewünschten Parameter und erzeugen als Ausgabe eine Datei in dem IEEE standardisierten SDF (*S*tandard *D*elay *F*ormat), das dann die unit-Delay Zeiten ersetzt.

1. SDF-Datei erzeugen
Das Programm zur Timing-Analyse wird über die Kommandozeile gesteuert. In dem hier vorgestellten Ablauf werden dabei die Prozessbedingungen eingestellt und anschließend der Einfluss des Layouts, beziehungsweise der Leitungskapazitäten, geschätzt.

```
> pearl                                                              [xterm]
▷ SetProcess      0.74:1.00:1.36                                     [pearl]
▷ SetVoltage      3.6:3.3:3                                          [pearl]
▷ SetTemperature -50:25:125                                          [pearl]
▷ SetLibraryCorner all                                               [pearl]
▷ ReadTLF /local/tams1.1/ams/v3.40/tlf/csx_3.3V/csx_HRDLIB.tlf       [pearl]
▷ ReadTLF /local/tams1.1/ams/v3.40/tlf/csx_3.3V/csx_HRDLIB_3B.tlf    [pearl]
▷ ReadTLF /local/tams1.1/ams/v3.40/tlf/csx_3.3V/csx_IOLIBV5_3M.tlf   [pearl]
▷ ReadTLF /local/tams1.1/ams/v3.40/tlf/csx_3.3V/csx_IOLIB_3M.tlf     [pearl]
```

[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.

▷ `ReadVerilog` ⟨*netlist*⟩`.v` [pearl]

▷ `TopLevelCell` ⟨*topLevel*⟩ [pearl]

▷
```
EstimateWireloads -rc -topology balanced -group sub_micron -name 10k
```

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

▷ `Quit` [pearl]

Alternativ können die Befehle auch in eine Steuerdatei geschrieben und diese dann abgearbeitet werden.

> `pearl` ⟨*commandFile*⟩ [xterm]

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` [xterm]

3. SDF-Steuerdatei anlegen
Für die Simulation wird mit einem Texteditor eine Datei erzeugt, die beschreibt, wie die SDF-Datei heisst und auf welchen Teil der Hierarchie sich die Verzögerungszeiten beziehen. Dazu muss das Label der Instanz ⟨*topLevel*⟩ angegeben werden: ⟨*instLabel*⟩.

> `vi` ⟨*netlist*⟩`.sdf.cmd` [xterm]

```
COMPILED_SDF_FILE = "⟨netlist⟩.sdf.X",
SCOPE = :⟨instLabel⟩,
LOG_FILE = "⟨netlist⟩.sdf.log";
```

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. Bibliotheken einbinden
In die bereits vorhandene Datei `cds.lib`, die Zuordnung der Simulationsbibliotheken zum Dateisystem beschreibt, sind die Gatterbibliotheken einzufügen.[4]

> `cat /local/tams1.1/ams/vhdlLib/ncsim/SunOS/csx.add >>cds.lib` [xterm]

5. Konfiguration für mixed-mode Simulation schreiben
Wei 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.

[4] Der CADENCE NC-Simulator ist auch für Linux verfügbar – `pearl` nicht! –, die vorbereiteten Bibliotheken sind dann unter `...vhdlLib/ncsim/Linux/...` installiert.

> vi ⟨*testEnv*⟩.vhd [xterm]

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

6. Netzliste nachbearbeiten
Vor der Simulation sollten in der Verilog-Datei noch fehlende Zeitskalierungen eingetragen werden; sie sind zwar nicht zwingend notwendig, aber man vermeidet weitere Fehlermeldungen.

> echo ''timescale 10ps/1ps' > tmpFile [xterm]

> cat ⟨*netlist*⟩.v >> tmpFile [xterm]

> mv tmpFile ⟨*netlist*⟩.v [xterm]

7. 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 ⟨*netlist*⟩.v [xterm]

> ncvhdl -v93 ⟨*testEnv*⟩.vhd [xterm]

> ncelab -sdf_cmd_file ⟨*netlist*⟩.sdf.cmd ⟨*configuration*⟩ [xterm]
Dabei auftretende Warnungen „`Too few module port connections.`“, beispielsweise bei Flipflops mit mehreren Ausgängen (z.B. `DFA`), können ignoriert werden.

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

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