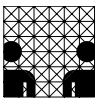
VHDL-Synthese

Werkzeuge : SYNOPSYS Design Analyzer
Design-Kits : AMS Hit-Kit, ES2 SSK, ALTERA, XILINX
designSetup : `syn ams|es2|alt|xil`



A. Mäder

Hier wird der Umgang mit dem SYNOPSYS Design Analyzer beschrieben, wobei die graphische Oberfläche `design_analyzer` eingesetzt wird. Dahinter verbirgt sich das Kommandozeileninterface `dc_shell`, das meist im Batch-Betrieb, bei großen Designs, benutzt wird.

Tipp: In der Datei `command.log` sind die eingegebenen Befehle mitprotokolliert. Für die Erstellung einer Batch-Datei sollte man sich diese Datei ansehen und entsprechend modifizieren. Wegen der vielfältigen Möglichkeiten in den Syntheseprozess einzugreifen, können hier nur die einfachsten Einstellungen vorgestellt werden. Genauere Informationen finden sich in der SYNOPSYS Online-Dokumentation unter: Synthesis Tools – Core Synthesis – „Design Compiler Reference Manual", bzw. „Design Analyzer Reference Manual".

FPGA-Synthese
Ursprünglich bezog sich diese Beschreibung nur auf die Arbeit mit Standardzell Design-Kits, wurde aber wegen des zunehmenden Einsatzes von FPGA-Designs entsprechend erweitert. Dokumentation zu FPGA-Designs findet sich unter:

SYNOPSYS	OnlineDoc.	FPGA Compiler II – User Guide
XILINX	OnlineDoc.	Xilinx/Synopsys Interface Guide
ALTERA	`quartus`-Help	Using Other EDA Design Entry/Synthesis Tools

Für die Synthese von FPGAs sollte, wegen der sehr einfachen Handhabung und guten Schnittstellen zu ALTERA und XILINX, der FPGA-Compiler `fc2` von SYNOPSYS benutzt werden. Die nachfolgend beschriebenen Arbeitsschritte gelten *nur* für den `design_analyzer`!

Legende
Für die Beschreibung der einzelnen Schritte wird die folgende Symbolik benutzt. Rechtsbündig steht jeweils der Titel des Fensters, in dem die Aktion durchzuführen ist, hier: [Xwin].

$>$ `command` [Xwin]
Eingabe von `command` an die Unix-Shell.

$\triangleright$ `command` [Xwin]
Eingabe von `command` an das SYNOPSYS-Kommandointerface.

$\equiv$ `item1` = `input1` [Xwin]
`item2` = `input2`
Ausfüllen von Optionsfenstern: für das Feld `item` wird der Wert `input` entweder über die Tastatur eingegeben oder aus einer Liste von Möglichkeiten ausgewählt, anschließend wird die Eingabe durch `OK`, bzw. `Apply` ausgeführt.

$\square$ `menu - choice` [Xwin]
Auswahl von `choice` aus dem Menü `menu`.

$\uparrow_x$ `object` [Xwin]
Auswahl von `object` mit der Maus (Maustaste: x).

Design-Flow

Voraussetzung sei eine korrekt simulierte (hierarchische) VHDL-Beschreibung. Die Kodierung von *synthesegerechtem VHDL* ist in der „VHDL Kurzbeschreibung“ und in dem SYNOPSYS „HDL Compiler for VHDL Reference Manual“ beschrieben.

Der Syntheseprozess lässt sich in folgende Schritte unterteilen, wobei mit • gekennzeichnete Schritte optional sind:

1. Einlesen der VHDL-Quelldatei(en)
- Behandlung der Hierarchie
2. Randbedingungen für die Synthese festlegen
3. Synthese der Schaltung
4. Ausgabedateien für Simulation und Back-End Programme schreiben

Arbeitsschritte

VHDL Einlesen

1. Start des Systems

   Die Anpassung der SYNOPSYS Werkzeuge an die Design-Kits geschieht über Initialisierungdateien im Login-, bzw. lokalen Verzeichnis – die Dateien für die Simulatoren sind zur Vollständigkeit angegeben, werden aber nicht benutzt.

`~/.synopsys_dc.setup`	Synthese	setup -Benutzer
`./.synopsys_dc.setup`	–"–	setup -lokal
`~/.synopsys_vss.setup`	Sim. VSS	setup -Benutzer
`./.synopsys_vss.setup`	–"–	setup -lokal
`~/synopsys_sim.setup`	Sim. Scirocco	setup -Benutzer
`./synopsys_sim.setup`	–"–	setup -lokal

AMS -Design-Kit

> `ams_synopsys` [xterm]

Achtung: Der Befehl startet ein Skript, dass beim ersten Aufruf (pro Directory) nach dem Prozess fragt, dementsprechend ist kein Start im Hintergrund möglich. Die (derzeit) installierten Technologien werden beim Start aufgelistet.

Beispielsweise ist für den Entwurf mit den 0.35 μm Standardzellbibliotheken `csx` (Voreinstellung) einzugeben.

ES 2 -Design-Kit

> `es2_design_analyzer` [xterm]

Achtung: Der Befehl startet ein Skript, dass beim ersten Aufruf (pro Directory) nach dem Prozess -`ecpd..`- fragt, dementsprechend ist kein Start im Hintergrund möglich. Der voreingestellte Wert (`ecpd07`) ist zu bestätigen.

ALTERA -FPGA-Kit

> `cp $MAX2_HOME/synopsys/config/.synopsys_vss.setup .` [xterm]

> `cp $MAX2_HOME/synopsys/config/.synopsys_dc.setup .` [xterm]

> `vi .synopsys_dc.setup` [xterm]
In der Setup-Datei ist `flex10k` als Zielarchitektur voreingestellt. Möchte man auf andere Bausteine abbilden, dann ist die Datei entsprechend anzupassen.

> `design_analyzer` [xterm]

XILINX -FPGA-Kit

> `cp $XILINX/synopsys/examples/template.synopsys_vss.setup .synopsys_vss.setup` [xterm]

> `cp $XILINX/synopsys/examples/template.synopsys_dc.setup_dc .synopsys_dc.setup` [xterm]

> `synlibs -dc` ⟨*XilinxTarget*⟩ `>> .synopsys_dc.setup` [0]xterm
Der Befehl `synlibs` erzeugt die Einträge für die gewählte Bausteinfamilie. Die möglichen Einträge für ⟨*XilinxTarget*⟩ kann man sich mit `synlibs -h` auflisten lassen.

> `design_analyzer` [xterm]

2. Einlesen der Quelldateien
Bei der Aufbereitung der Daten für die Synthese werden zwei Schritte unterschieden:

Analyse: Der VHDL-Code wird auf die Synthetisierbarkeit untersucht und es werden Eingabedateien – in Form von Templates – erzeugt.
□ `File - Analyze...` [Synopsys Design Analyzer]

Elaboration: Aus den Templates werden die Datenstrukturen der Synthese aufgebaut.
□ `File - Elaborate...` [Synopsys Design Analyzer]

- Beide Schritte können auch gemeinsam ausgeführt werden.
□ `File - Read...` [Synopsys Design Analyzer]

Das weitere Vorgehen richtet sich nach der Art der VHDL-Beschreibung und der bisherigen Vorgehensweise, die drei möglichen Fälle sind als (a)...(c) beschrieben.

(a) Wenn die VHDL-Dateien bei der Codeanalyse der Simulation schon entsprechend bearbeitet wurden (> `vhdlan -spc_elab` ⟨*vhdlFile*⟩), beziehungsweise wenn die Analyse abgeschlossen ist – nach (b) –, wird die Hierarchie durch Elaboration abgearbeitet:

□ `File - Elaborate...` [Synopsys Design Analyzer]

≡ `Library` = `WORK` [Elaborate Design]
`Design` = ⟨*topEntityId*⟩`(`⟨*archId*⟩`)`
`Parameters` = ⟨*paraId*⟩`=`⟨*paraVal*⟩ `...`
Aus den Templates in `WORK` werden die Elemente der Synthese erzeugt. Die Hierarchie wird dabei komplett abgearbeitet, es muss nur die oberste Hierarchieebene angegeben werden. Für eventuell vorhandene `generic`-Parameter (z.B. für Bitbreiten, Anzahl von Instanzen...), sind passende Werte unter `Parameters` einzutragen.

(b) Besitzen die VHDL-Entities (beliebig in der Hierarchie) `generic`-Parameter, dann müssen Analyse und Elaboration getrennt ausgeführt werden. Die Analyse ist für *alle* Dateien der Hierarchie (in beliebiger Reihenfolge) durchzuführen:

□ `File - Analyze...` [Synopsys Design Analyzer]

≡ `File Names(s)` = ⟨*vhdlFileLis*⟩ [Analyze File]
`File Format` = `VHDL`
`Library` = `WORK`

Anschließend wird wie unter (a) beschrieben die Elaboration gestartet.

(c) Trifft keiner der oben beschriebene Sonderfälle zu, so können die Entities der Hierarchie direkt eingelesen werden.

□ `File - Read...` [Synopsys Design Analyzer]

≡ `File Names(s)` = ⟨*vhdlFileLis*⟩ [Read File]
`File Format` = `VHDL`
Die Abarbeitungsreihenfolge der einzelnen Dateien ist beliebig, es muss nur gewährleistet sein, dass vor der Synthese alle Quelldateien in der SYNOPSYS Datenbasis vorhanden sind.

Achtung: Beim Einlesen – Schritt (a) oder (c) – wird eine log-Datei ([VHDL]) ausgegeben; traten kein Fehler auf, so wurden die entsprechenden Objekte erzeugt. In dieser Datei wird unter anderem ausgegeben, welche speichernden Elemente (Flipflops) durch welche Zeilen des VHDL-Codes impliziert werden. Die Ausgabe sieht dabei folgendermaßen aus:

```
Inferred memory devices in process '⟨processId⟩'
    in routine ⟨entityId⟩ line ⟨lineNr⟩ in file
    '⟨vhdlFile⟩'
=================================================...
|    Register Name     |      Type       |  Width  | ...
=================================================...
| ⟨signal/varId⟩_reg   | Flip-flop|Latch | ⟨bitNr⟩ | ...
| ...
=================================================...
```

Dabei ist darauf zu achten, dass diese Elemente auch wirklich vom Designer so vorgesehen waren und nicht die Folge einer *ungeschickten* VHDL-Beschreibung sind. Solche möglichen Fehlerquellen können sein:

- Signale oder Variable vom Typ `integer` haben keine Wertebereichseinschränkung bei der Deklaration erhalten. Entsprechend dem Datentyp werden 32-bit breite Register erzeugt.
- Obwohl nur das Verhalten eines Schaltnetzes beschrieben werden soll, wurden Latches für Signale eingefügt. In diesem Fall werden Signalzuweisungen im VHDL-Code von Bedingungen abhängig gemacht. Dann müssen entweder in allen möglichen Verzweigungen Signalzuweisungen vorkommen oder eine *Default-Zuweisung* muss im sequentiellen Prozess *vor* der Verzweigung stehen.

Gegebenenfalls ist die VHDL-Beschreibung noch abzuändern damit nicht unnötige Hardware generiert wird — im Falle von Latches wird meist auch die Funktion der Schaltung fehlerhaft!

Behandlung der Hierarchie
optional: Die folgenden Schritte sind nur notwendig, wenn innerhalb der Hierarchie Entities mehrfach instantiiert werden, andernfalls kann direkt mit Punkt 7 begonnen werden!

Normalerweise wird die Hierarchie während der Synthese automatisch traversiert und alle instantiierten Entities bearbeitet. Werden dabei Teile der Hierarchie mehrfach instantiiert, so kann dieser Fall mit folgenden Strategien bearbeitet werden.

Unterschiedliche Behandlung von Instanzen Diese Methode ist anzuwenden, wenn Instanzen in *unterschiedlicher* Weise mehrfach benutzt werden oder an den *Eingangsports mit Konstanten* versehen sind — Beispiel: Die Entity eines Multiplizierers, wird in dem übergeordneten Design eines Filters, mit Konstanten für einen Faktor, mehrmals instantiiert.

3. Identifikation von Instanzen

$\uparrow_l$ ⟨`topDesign`⟩ [Synopsys Design Analyzer]

□ `Edit - Uniquify - Hierarchy` [Synopsys Design Analyzer]
Elemente, die in der Hierarchie mehrfach vorkommen, werden eindeutig (entsprechend oft) erzeugt. Später bei der Synthese werden sie dann individuell bearbeitet.

- Anschließend werden die Schritte des nächsten Abschnitts, ab Punkt 8, ausgeführt.

Gleichartige Behandlung von Instanzen Werden mehrfach referenzierte Entities in der Hierarchie in immer *gleicher Weise* benutzt, so sollten sie (aus Effizienzgründen) nur einmal synthetisiert werden — Beispiel: Die Entity einer Registerbank, wird in dem übergeordneten Design eines Operationswerks mehrfach benutzt.

Tipp: Bei *sehr großen (Sub-)Designs* ist es auch sinnvoll die Synthese in kleinere „Portionen" zu unterteilen, um die Programmlaufzeit und den Speicherbedarf geringer zu halten.

3. Festlegung der „Synthesehierarchie"
Neben der hier vorgestellten Methode Entities komplett zu übergehen, können auch gezielt einzelne Instanzen aus dem top-down Prozess herausgenommen werden — das genaue Vorgehen ist der Online-Dokumentation zu entnehmen.

$\uparrow_l$ ⟨`subDesign`⟩ [Synopsys Design Analyzer]
Selektion der Entity, die beim Traversieren der Hierarchie von dem Syntheseprozess ausgenommen wird.

□ `Attributes - Optimization Directives - Design...` [Synopsys Design Analyzer]

≡ `Don't touch = on` [Design Attributes]
Nach dem Setzen des Attributs werden alle Instanzen, die in der Hierarchie gefunden werden, von der Synthese ausgeschlossen.

Diese Schritte sind für alle referenzierten Teile der Schaltung, die einzeln behandelt werden sollen, zu wiederholen.

4. top-level Synthese — Schritte:

7 top-level Design selektieren
8 Padzellen einfügen
9 Operationsbedingungen einstellen
10 Synthesevorgaben machen (Optimierung)
11 Synthese der Gatternetzliste

5. Constraint Propagation
Randbedingungen des top-level Designs (Clocks, Area, Timing…) können jetzt auf die noch nicht synthetisierten Teile propagiert werden. Dabei ist allerdings zu beachten, dass wegen der Gleichartigkeit aller Instanzen einer Entity, nur ein Satz an Randbedingungen berücksichtigt wird. Es sollte daher gewährleistet sein, dass alle Referenzen unter gleichartigen Randbedingungen stattfinden!

$\uparrow_l$ ⟨*topDesign*⟩ [Synopsys Design Analyzer]
Nach Auswahl des gerade synthetisierten Designs ist die Hierarchie so lange zu traversieren (Anzeigen des Schematic, ggf. Auswahl einer Instanz, Anzeigen des Schematic…), bis die noch nicht synthetisierte Entity instantiiert wird.

$\uparrow_l$ ⟨*instance*⟩ [Synopsys Design Analyzer]
Die Auswahl der „richtigen" Instanz kann man in dem Fenster links unten kontrollieren: `Instance:` ⟨*instId*⟩

□ `Attributes - Operating Environment - Characterize...` [Synopsys Design Analyzer]

≡ `Timing` = `on` [Characterize]
`Constraints` = `on`
`Connections` = `off`
Setzt die Randbedingungen *dieser Instanz* für das Subdesign.

6. Synthese des Subdesigns — Schritte:

7 Subdesign selektieren; hier ist nicht die Instanz, sondern die entsprechende Entity auszuwählen. Dazu muss man in der Hierarchie bis auf die oberste Ebene zurückgehen (Button ⇑).

– die Schritte 8 – 10 wurden implizit unter Punkt 5 ausgeführt.

11 Synthese der Gatternetzliste

• Nachdem *alle* Subdesigns synthetisiert wurden, wird ab Punkt 12 weiter fortgefahren.

Synthesebedingungen festlegen

7. Selektion des top-level Designs (bzw. Subdesigns s.o.)

$\uparrow_l$ ⟨*topDesign*⟩ [Synopsys Design Analyzer]
Das selektierte Design wird schraffiert umrandet dargestellt, alle weiteren Befehle beziehen sich darauf.

8. Automatisches Einfügen von Padzellen
optional: Nur bei der Synthese des top-level Elements sinnvoll, wobei dieses dann dem kompletten IC entspricht und als Standardzell- oder FPGA-Entwurf mit back-end Werkzeugen weiterbearbeitet werden soll. In diesem Fall können automatisch Pad-Zellen eingefügt werden — alternativ dazu könnten diese auch „von Hand" in CADENCE DF II hinzugefügt werden.

Eine Nachbearbeitung mit dem CADENCE Schematic-Editor ist in jedem Fall notwendig, da die Spannungsversorgung des ICs in der Synthese noch nicht berücksichtigt werden kann.

Bei FPGA-Entwürfen ist dieser Befehl teilweise auch möglich (XILINX) siehe dazu die entsprechende Dokumentation der Hersteller.

□ `Attributes - Optimization Directives - Design...` [Synopsys Design Analyzer]

≡ `Port is Pad` `= on` [Design Attributes]
Die VHDL-Ports werden als Pads der Schaltung gekennzeichnet.

□ `Edit - Insert Pads...` [Synopsys Design Analyzer]

≡ – bestätigen [Insert Pads]

Anmerkung: gegebenenfalls sollen bestimmte Pads – Unterscheidung nach Treiberleistung, Spannungspegeln etc. – verwendet werden. Beispielsweise zeigen die folgenden Befehle, wie für den ES 2 Prozess CMOS Eingänge eingestellt werden.

□ `Setup - Command Window...` [Synopsys Design Analyzer]
Öffnet ein Eingabefenster der DC-shell.

▷ `set_pad_type -exact LIBIPS8B all_inputs()` [Command Window]
Hier wird der Pad Typ festgelegt. Dies ist sowohl für einzelne Pads als auch für ganze Gruppen möglich (siehe SYNOPSYS Design Analyzer Dokumantation).

9. Operationsbedingungen einstellen
Durch diese Auswahl wird das Timingmodell der Gatter für die folgenden Schritte (Synthese und Timinganalyse) festgelegt. Neben den Ober- und Untergrenzen für die Verzögerungszeiten, wird bei Standardzellen meist auch noch nach Temperaturbereichen (`IND`ustrial und `MIL`itary) unterschieden — wir benutzen `IND`.

□ `Attributes - Operating Environment - Operating Conditions...` [Synopsys ...]

≡ `Design Name` `= ⟨designId⟩` [Operating Conditions]
`Operating Conditions` `= ⟨condition⟩`
Eingabe durch $\uparrow_l$ ⟨*condition*⟩; die Namen sprechen für sich.

10. Randbedingungen für die Optimierung vorgeben
optional: Bei der Realisierung einer Schaltung durch eine Gattenetzliste, gibt es nicht nur eine, sondern beliebig viele Lösungen. Dieser *Suchraum* wird während des Syntheseprozesses nach einer „möglichst guten" Realisierung (bezogen auf eine Bewertungsfunktion) hin untersucht.

Die unterschiedlichen Realisierungen unterscheiden sich hinsichtlich ihres Flächenbedarfs und den Verzögerungszeiten (Geschwindigkeit). Im Allgemeinen sind kleine Lösungen langsam (viele gemeinsame logische Teilausdrücke $\Rightarrow$ große sequentielle Tiefe), während sehr schnelle Realisierungen sehr groß werden.

Wegen der Möglichkeiten den Syntheseprozess zu beeinflussen, sei hier nochmals auf die SYNOPSYS Dokumentation verwiesen. Im folgenden werden drei „einfache" Möglichkeiten vorgestellt, die auch miteinander beliebig kombiniert werden können.
Anmerkung: für „optimale" Syntheseergebnisse ist es besser mit realistischen Werten für Fläche bzw. Geschwindigkeit zu synthetisieren und diese Randbedingungen über mehrere Syntheseläufe zu verschärfen.

□ `Setup - Command Window...` [Synopsys Design Analyzer]
Wenn noch nicht vorhanden, Eingabefenster der DC-shell öffnen.

Fläche Als Voreinstellung werden kleine, kompakte Netzlisten erzeugt.

▷ `set_max_area` ⟨*areaVal*⟩ [Command Window]
Die Fläche wird als Real-Wert angegeben, wobei sich die Maßeinheit nach den Technologiedatei richtet: AMS mil^2
ES2 μm^2

Clock Sind externe Taktfrequenzen der Schaltung (spätere Umgebung) bekannt, sollte man hier entsprechende Vorgaben machen.

▷ `create_clock -period` ⟨*clkPeriodVal*⟩ ⟨*clkPortId*⟩ [Command Window]
Die Taktperiode ist in μs anzugeben. Bei der Optimierung wird der Pfad zwischen, bzw. vor, Registern berücksichtigt. Dadurch wird die explizite Angabe von Zeitpfaden (s. Timing) meist überflüssig.

Timing Zwischen beliebigen Stellen der Netzliste kann das Timing explizit angegeben werden; dabei sind sowohl minimale als auch maximale Laufzeiten möglich. Hier wird allerdings nur die Beschränkung für eine möglichst schnelle Schaltung vorgestellt.

▷ `set_max_delay` ⟨*delayVal*⟩ `-to` ⟨*pathLis*⟩ [Command Window]
`set_max_delay` ⟨*delayVal*⟩ `-to {all_outputs() all_registers(-data_pins)}`
Vorgabe der Verzögerungszeit in μs. Bei der Optimierung werden die Pfade zu allen Endpunkten berücksichtigt — in dem Beispiel: die Primärausgänge und Eingänge von Registern.

Synthese der Gatternetzliste

11. Hardwaresynthese und Abbildung auf die Zellbibliothek
Ausgehend von dem selektierten (obersten) Element, läuft die Synthese für die gesamte Hierarchie ab — die Ausnahme bilden `Don't touch`-Attribute, siehe „Behandlung der Hierarchie“.

□ `Tools - Design Optimization...` [Synopsys Design Analyzer]

≡ [Design Optimization]

`Map Design`	`=on`
`Map Effort`	`=Medium`
`Verify Design`	`=off`
`Allow Boundary...`	`=on`
`Execute in`	`=Foreground`

Während des Laufs beachte man die log-Datei ([Compile Log]).

12. Bewertung der Syntheseergebnisse
Entspricht das Ergebnis nicht den Anforderungen, so müssen die Randbedingungen der Synthese (Punkt 10) entsprechend angepasst und ein neuer Syntheselauf (Punkt 11) gestartet werden.

Hier werden nur einige Möglichkeiten von SYNOPSYS Design Analyzer vorgestellt. Die bei der Timinganalyse ausgegebene Information bezieht sich immer auf das unter Punkt 9 festgelegte Zeitmodell. Durch Auswahl anderer Operationsbedingungen können „worst-case“ und „best-case“ Timing der synthetisierten Struktur ermittelt werden.

Ausgabe von Statistiken SYNOPSYS erlaubt die Ausgabe sehr detaillierter Statistiken, sinnvoll erscheinen hier: `Area`, `Timing` und ggf. `Hierarchy`.

□ `Analysis - Report...` [Synopsys Design Analyzer]

↑$_l$ ⟨*reportOptions*⟩ [Report]

Kontrolle des Schematic / Ansehen der Hierarchie

↑$_l$ ⟨*dbObject*⟩ | ⟨*schInst*⟩ | `...` [Synopsys Design Analyzer]

↑$_l$ ⟨*dbView*⟩ Button: linke Seite im Fenster [Synopsys Design Analyzer]

Timing von Pfaden

Kritischer (längster) Pfad

□ `Analysis - Highlight - Critical Path` [Synopsys Design Analyzer]

Längste / Kürzeste Pfade zu Elementen

↑$_l$ ⟨*schInst*⟩ [Synopsys Design Analyzer]

□ `Analysis - Highlight - Max/Min Path - ... Path/Rise/Fall` [Synopsys ...]

Timing einzelner Netze

↑$_l$ ⟨*schNet*⟩ [Synopsys Design Analyzer]

□ `Analysis - Show Timing...` [Synopsys Design Analyzer]

↑$_l$ ⟨*schNet*⟩ Auswahl weiterer Netze... [Synopsys Design Analyzer]

Lasten einzelner Netze

↑$_l$ ⟨*schNet*⟩ [Synopsys Design Analyzer]

□ `Analysis - Show Net Load...` [Synopsys Design Analyzer]

↑$_l$ ⟨*schNet*⟩ weitere Netze... [Synopsys Design Analyzer]

13. Sichern

□ `File - Save` [Synopsys Design Analyzer]

Die interne SYNOPSYS Datenbasis wird ausgehend von dem top-level Element für die gesamte Hierarchie durchlaufen und in Form einzelner `db`-Dateien gesichert.

Sollte das Design später noch einmal bearbeitet werden, so können diese Dateien geladen werden (analog zu Punkt 2).

Ausgabedateien erzeugen

Abhängig vom Simulator, der Zielbibliothek und dem Programm zur Nachbearbeitung – Platzierung und Verdrahtung bei Standardzellen, Mapping bei FPGAs –, werden verschiedene Netzlistenformate als Ausgabe benötigt, wie die Tabelle veranschaulicht.

	AMS - ASIC		**ES 2** - ASIC		**Altera** - FPGA		**Xilinx** - FPGA	
	Simul.	Layout	Simul.	Layout	Simul.	Layout	Simul.	Layout
VHDL	*		*		*		*	
Verilog	*	*	(*)	*	*		*	
EDIF						*		*

Die Simulation der synthetisierten Netzlisten ist in der extra Beschreibung „VHDL- und mixed-mode Netzlistensimulation“ beschrieben — aus Effizienzgründen wird eine gemischte Simulation von VHDL-Testumgebung und Verilog-Netzliste bevorzugt.

14. Namenskonvertierungen durchführen
Bei Verilog- und VHDL-Netzlisten müssen die Bezeichner noch den Namenskonventionen der jeweiligen Sprache angepasst werden.

□ `Setup - Command Window...` [Synopsys Design Analyzer]
Wenn im bisherigen Verlauf noch kein Eingabefenster geöffnet wurde, erhält man damit die Kommandozeile für die folgenden Befehle.

▷ `change_names -hierarchy -rules vhdl` [Command Window]

▷ `change_names -hierarchy -rules verilog` [Command Window]
Je nach Ausgabeformat sind die passenden Regeln anzugeben. Werden beide Netzlistenformate benutzt, VHDL zur Simulation und Verilog zum Datentransfer, dann müssen auch beide Ersetzungen durchgeführt werden *bevor* die Netzlisten erzeugt werden. Der Befehl erzeugt entsprechende Ausgaben in [Command Window].

15. Netzlisten schreiben
Über die Kommandozeile werden jetzt die verschiedenen Netzlistenformate erzeugt. Nur eine einzige Datei wird geschrieben, sie enthält die gesamte Hierarchie des synthetisierten Entwurfs.

VHDL-Netzliste für die Simulation

▷ `write -format vhdl -hierarchy -output` ⟨*netlist*⟩`.vhd` [Command Window]
Achtung: bei dem Dateinamen ⟨*vhdlFile*⟩ muss man aufpassen, dass man bestehende Dateien (z.B.: ⟨*designId*⟩`.vhd`) nicht überschreibt!

Verilog-Netzliste für die Simulation und/oder Standardzell-Layout

▷ `write -format verilog -hierarchy -output` ⟨*netlist*⟩`.v` [Command Window]

EDIF-Netzliste für den Datentransfer zu FPGA Back-End Werkzeugen

▷ `write -format edif -hierarchy -output` ⟨*netlist*⟩`.edf` [Command Window]

Das genaue Vorgehen, um die EDIF-Netzliste mit den FPGA-Programmen weiter zu bearbeiten, ist der jeweiligen Dokumentation zu entnehmen, siehe Seite 1 — hier noch zwei Anmerkungen.

ALTERA hat eine implizite Namenskonvention, beispielsweise bei MAX+Plus II. Danach muss der Dateiname dem Namen der *top-level Entity* entsprechen, um die Datei fehlerfrei einzulesen!

XILINX kann, neben der Netzliste, auch die Randbedingungen des Syntheseprozesses einlesen. Der folgende Befehl erzeugt die Eingabedatei für `dc2ncf`.

▷ `write_script >` ⟨*designId*⟩`.dc` [Command Window]

16. . . . fertig: SYNOPSYS Design Analyzer beenden

□ `File - Quit` [Synopsys Design Analyzer]

≡ – bestätigen [Quit Design Analyzer]