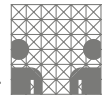


VHDL-Synthese

Werkzeuge : Synopsys Design-Vision
Design-Kits : AMS Hit-Kit
edaSetup : syn_ams

Andreas Mäder



Diese Anleitung beschreibt die Synthese mit den SYNOPSIS Werkzeugen: Wegen der vielfältigen Möglichkeiten in den Syntheseprozess einzugreifen, können hier nur die einfachsten Einstellungen vorgestellt werden. Genauere Informationen finden sich in der SYNOPSIS Online-Dokumentation unter: „Design Compiler“

Design-Flow

Voraussetzung für die Synthese ist eine korrekt simulierte (hierarchische) VHDL-Beschreibung. Die Kodierung von *synthesegerechtem VHDL* ist in „VHDL Kompakt“ und in dem SYNOPSIS „HDL Compiler for VHDL User Guide“ beschrieben.

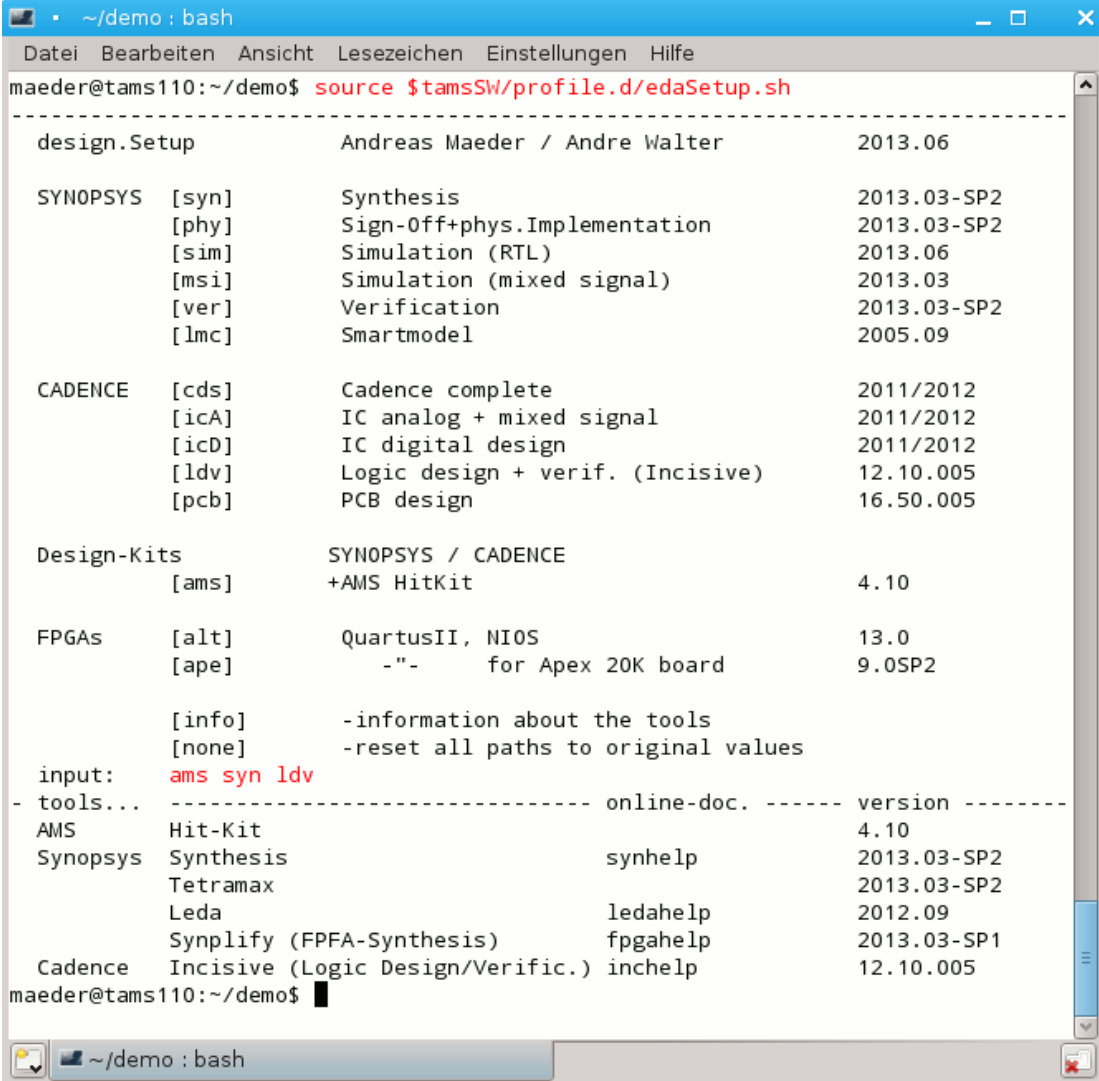
Der Syntheseprozess lässt sich in folgende Schritte unterteilen:

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

Arbeitsschritte

VHDL Einlesen

1. (meist schon vor der Simulation ldv) Initialisierung der Shell:



```

~/demo : bash
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
maeder@tams110:~/demo$ source $tamsSW/profile.d/edaSetup.sh
-----
design.Setup          Andreas Maeder / Andre Walter          2013.06

SYNOPSYS  [syn]      Synthesis                            2013.03-SP2
          [phy]      Sign-Off+phys.Implementation         2013.03-SP2
          [sim]      Simulation (RTL)                     2013.06
          [msi]      Simulation (mixed signal)            2013.03
          [ver]      Verification                         2013.03-SP2
          [lmc]      Smartmodel                           2005.09

CADENCE   [cds]      Cadence complete                     2011/2012
          [icA]      IC analog + mixed signal             2011/2012
          [icD]      IC digital design                    2011/2012
          [ldv]      Logic design + verif. (Incisive)     12.10.005
          [pcb]      PCB design                           16.50.005

Design-Kits          SYNOPSYS / CADENCE
          [ams]      +AMS HitKit                          4.10

FPGAs   [alt]      QuartusII, NIOS                      13.0
          [ape]      "-- for Apex 20K board               9.0SP2

          [info]      -information about the tools
          [none]     -reset all paths to original values

input:  ams syn ldv
tools... ----- online-doc. ----- version -----
AMS     Hit-Kit                               4.10
Synopsys Synthesis                          synhelp    2013.03-SP2
        Tetramax                            2013.03-SP2
        Leda                                ledahelp   2012.09
        Synplify (FPFA-Synthesis)          fpgahelp  2013.03-SP1
Cadence Incisive (Logic Design/Verific.) inchelp   12.10.005
maeder@tams110:~/demo$

```

2. Start des Systems:

```

~/demo : common_shell_ex
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
maeder@tams110:~/demo$ ams_synopsys

DEFINE YOUR TARGET PROCESS: <input>
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
input | AMS-Process          | um | met
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
c35   | C35...               | CMOS | 0.35 | 3/4
s35   | S35...               | SiGe  | 0.35 | 3/4

- target process [c35] : █
- Note: Creating .AMSProcDat
- Note: Creating .synopsys_dc.setup
- Note: Modifying cds.lib
- Note: Creating netlist.sdf.cmd
- Note: target process is c35 at 3.3V -- [c35_3.3V]
      Synopsys program is design_vision -64bit

          Design Compiler Graphical
            DC Ultra (TM)
            DFTMAX (TM)
          Power Compiler (TM)
            DesignWare (R)
            DC Expert (TM)
          Design Vision (TM)
            HDL Compiler (TM)
            VHDL Compiler (TM)
            DFT Compiler
          Library Compiler (TM)
          Design Compiler(R)

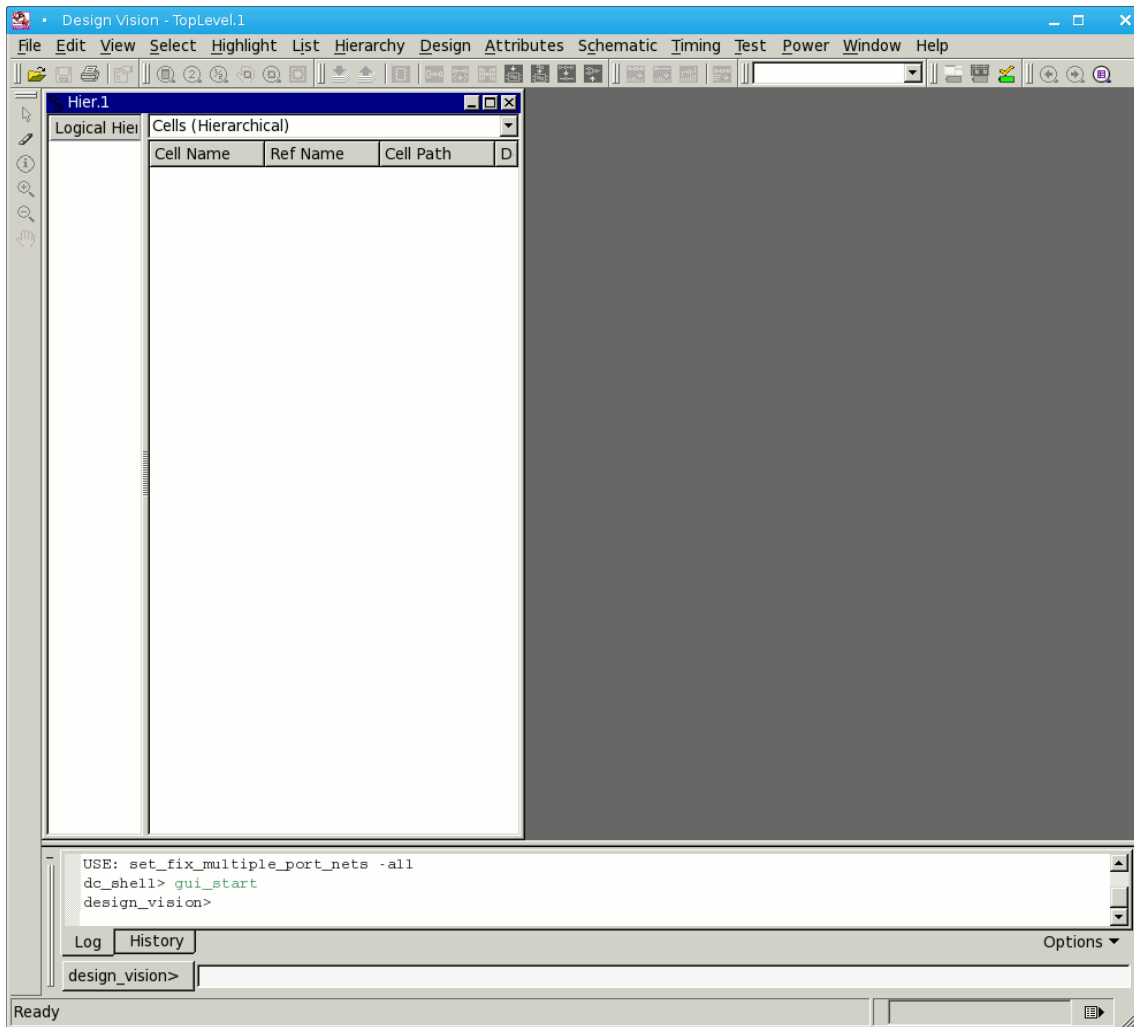
          Version H-2013.03-SP2 for RHEL64 -- Jun 03, 2013
          Copyright (c) 1988-2013 Synopsys, Inc.

This software and the associated documentation are confidential and
proprietary to Synopsys, Inc. Your use or disclosure of this software
is subject to the terms and conditions of a written license agreement
between you, or your company, and Synopsys, Inc.

Initializing...
Owner: austriamicrosystems AG HIT-Kit: Digital
Error: Current design is not defined. (UID-4)
USE: set_fix_multiple_port_nets -all
design_vision> design_vision> █
  
```

Der Befehl startet ein Skript, das beim ersten Aufruf nach dem AMS-Prozess fragt und die passenden Initialisierungsdateien für die Synthese und nachfolgende Schritte erzeugt. Für die 0,35 μm Bibliotheken ist die Voreinstellung c35 zu bestätigen.

Anschließend starten die Synthesewerkzeuge, wobei in der Shell die Kommandozeilenversion dc_shell läuft, die dann die grafische Benutzeroberfläche startet:



Tipp: Das Kommandozeileninterface (mit TCL-Syntax) der GUI oder der Synthese-Shell wird, beispielsweise bei großen Entwürfen, für eine Skript-gesteuerte Batch-Verarbeitung genutzt. In der Datei `command.log` sind die eingegebenen Befehle mitprotokolliert. Zur Erstellung solcher Skripte kann man die Logdatei als Vorlage nutzen.

3. Einlesen der Quelldateien

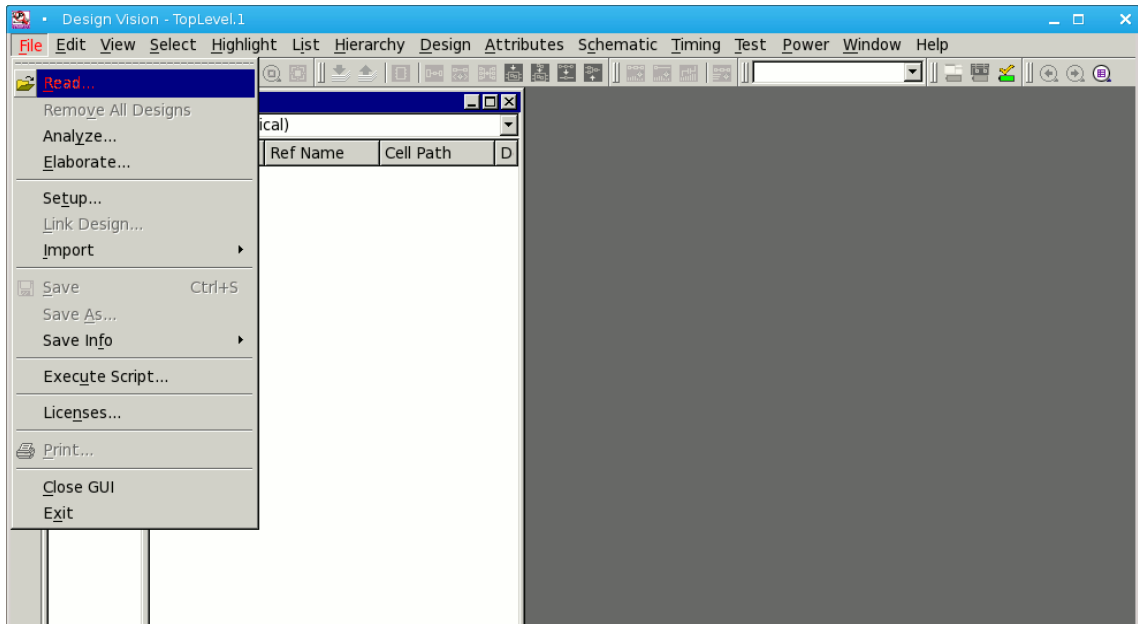
Bei der Aufbereitung der Daten für die Synthese werden zwei Schritte unterschieden: Bei der *analyse* wird der VHDL-Code auf die Synthetisierbarkeit untersucht und es werden systeminterne Templatedateien erzeugt. Die anschließende *elaboration* generiert dann daraus die Datenstrukturen der Synthese.

Beide Schritte können auch gemeinsam ausgeführt werden. Ob eine Trennung von Analyse und Elaboration notwendig ist, hängt von der Art der VHDL-Beschreibung und der bisherigen Vorgehensweise ab. Sie zwingend notwendig wenn:

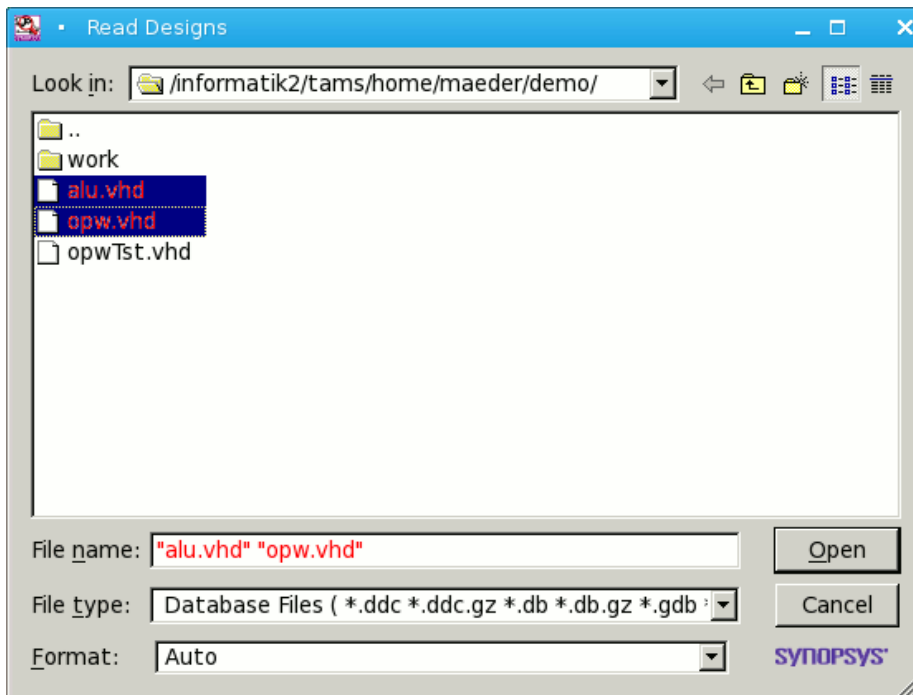
- zu einer Entity mehrere Architekturen existieren und man explizit eine auswählen möchte — Ansonsten gilt die zeitliche Reihenfolge bei der Codeanalyse.
- eine Entity `generic`-Parameter besitzt an die Werte übergeben werden sollen.

Read — Analyse und Elaboration in einem Schritt

In dem einfachen Fall können die Entities der Hierarchie direkt eingelesen werden:

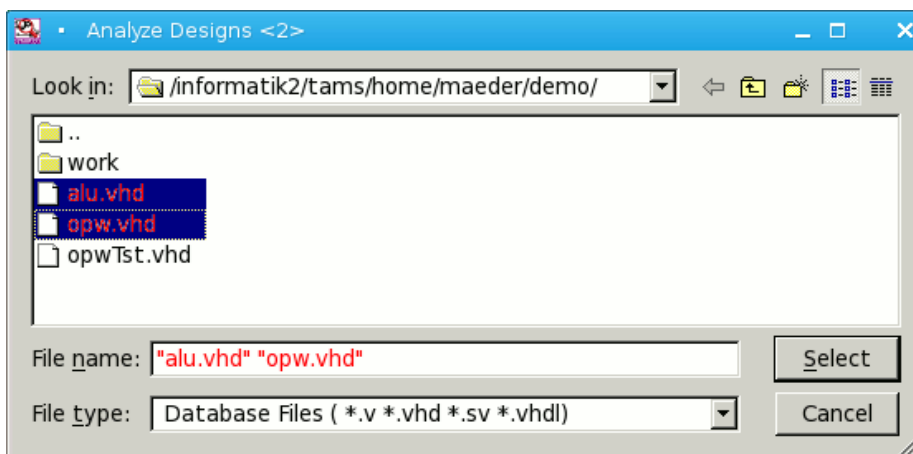
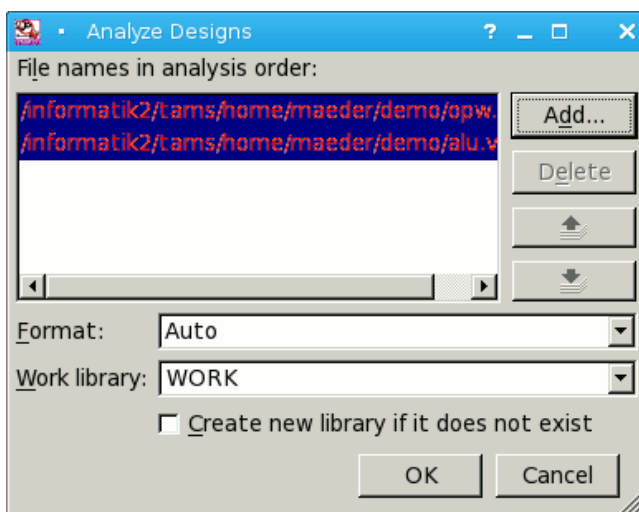
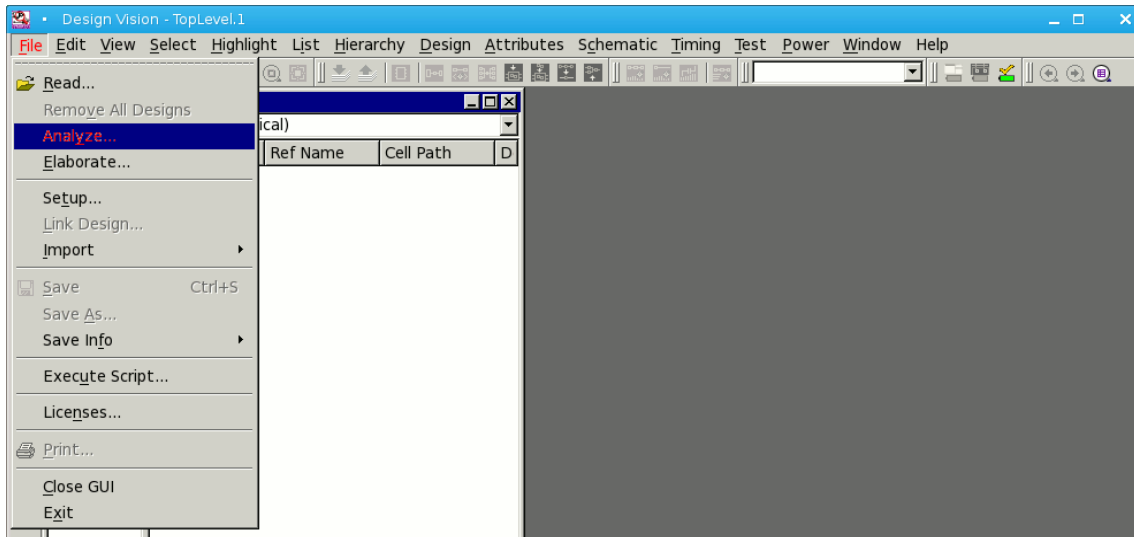


Die Abarbeitungsreihenfolge der einzelnen Dateien ist beliebig, es muss nur gewährleistet sein, dass vor der Synthese alle Quelldateien in der Datenbasis vorhanden sind:

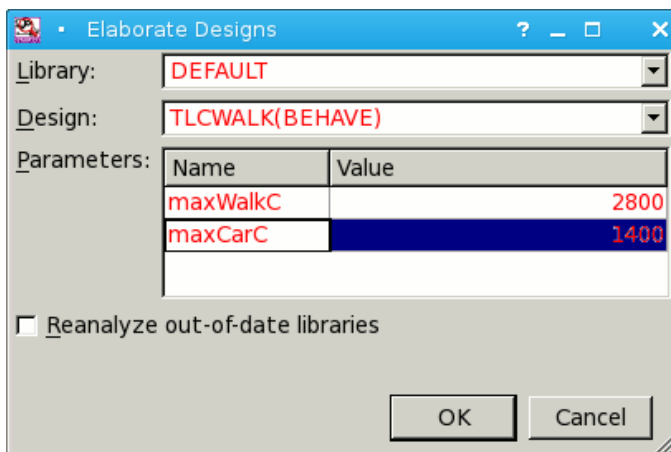
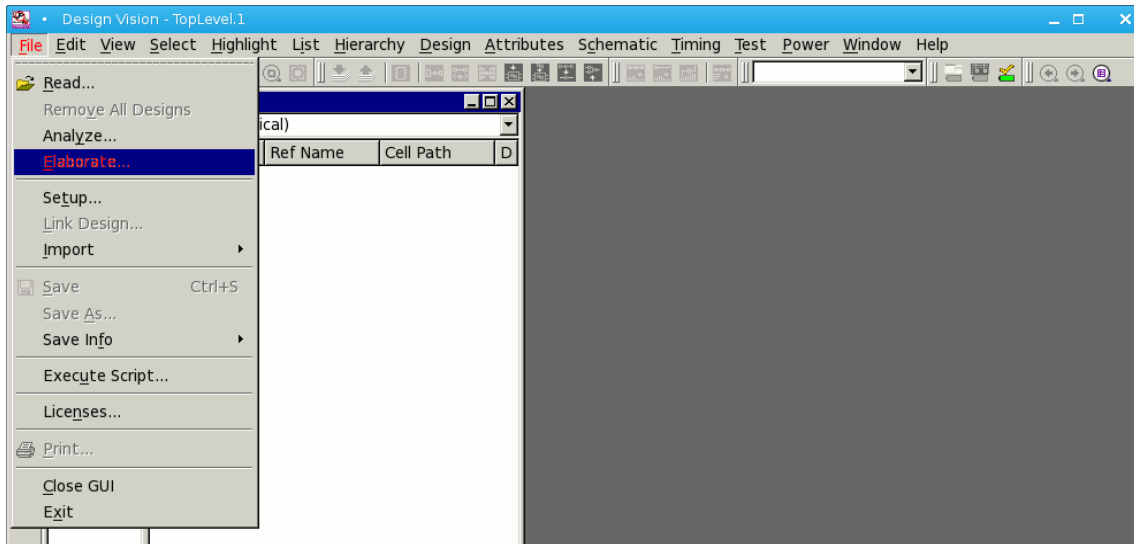


Analyse und Elaboration getrennt

Wenn generic-Parameter an VHDL-Entities zu übergeben sind oder alternative Architekturen zur Auswahl stehen, dann müssen Analyse und Elaboration getrennt werden. Zuerst werden alle Dateien der Hierarchie (in beliebiger Reihenfolge) analysiert:



Nach Abschluss der Analysephase wird die Hierarchie, ausgehend von dem gewählten Entwurf, durch Elaboration abgearbeitet. Architekturen werden über ihren Bezeichner, nach folgendem Schema $\langle entity \rangle (\langle architecture \rangle)$, unterschieden. Das Beispiel zeigt auch, wie generic-Parameter (Beispiel: Ampelschaltung) spezifiziert werden:



4. Kontrolle der Daten

Ist der VHDL-Code syntaktisch korrekt und *synthetisierbar*, dann werden Objekte für die Synthese erzeugt, andernfalls enthält die Log-Datei entsprechende Fehlermeldungen.

In dieser Datei wird unter anderem ausgegeben, welche speichernden Elemente (Flip-flops/Latches) und Tri-state-Treiber durch welche Zeilen des VHDL-Codes impliziert werden. Die Ausgabe sieht dabei folgendermaßen aus:

```

Design Vision - TopLevel.1 (alu)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power Window Help
alu
Hier.1
Logical Hierarchy Cells (Hierarchical)
Cell Name Ref Name Cell Path D
-----
Statistics for case statements in always block at line 69 in file
'/informatik2/tams/home/maeder/demo/opw.vhd'
-----
| Line | full/ parallel |
-----
| 76 | auto/auto |
-----

Inferred memory devices in process
in routine opw line 69 in file
'/informatik2/tams/home/maeder/demo/opw.vhd' .
-----
| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
-----
| reg5_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| oReg_reg | Flip-flop | 16 | Y | N | Y | N | N | N | N |
| reg6_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg7_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg0_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg1_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg2_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg3_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
| reg4_reg | Flip-flop | 16 | Y | N | N | N | N | N | N |
-----

Inferred tri-state devices in process
in routine opw line 39 in file
'/informatik2/tams/home/maeder/demo/opw.vhd' .
-----
| Register Name | Type | Width | MB |
-----
| ioBus_tri | Tri-State Buffer | 16 | N |
-----

Inferred tri-state devices in process
Log History Options
design_vision>
Ready

```

Achtung: es ist darauf zu achten, dass diese Elemente auch wirklich so vorgesehen waren und nicht die Folge einer *ungeschickten* VHDL-Beschreibung sind. Typische Fehlerquellen dabei sind:

- Numerische Signale oder Variablen haben bei der Deklaration keine Wertebereichs-einschränkung erhalten. Für die meisten Datentypen werden dann 32-bit Wortbreite angenommen.
- Obwohl nur das Verhalten eines Schaltnetzes beschrieben werden soll, wurden Latches für Signale eingefügt. Dies passiert, wenn im VHDL-Code bedingte Signalzuweisungen stehen. Entweder finden Zuweisungen in allen möglichen Fällen statt

oder eine Zuweisung die ggf. überschrieben wird, muss als „Default“ im sequentiellen Prozess *vor* der Verzweigung stehen.

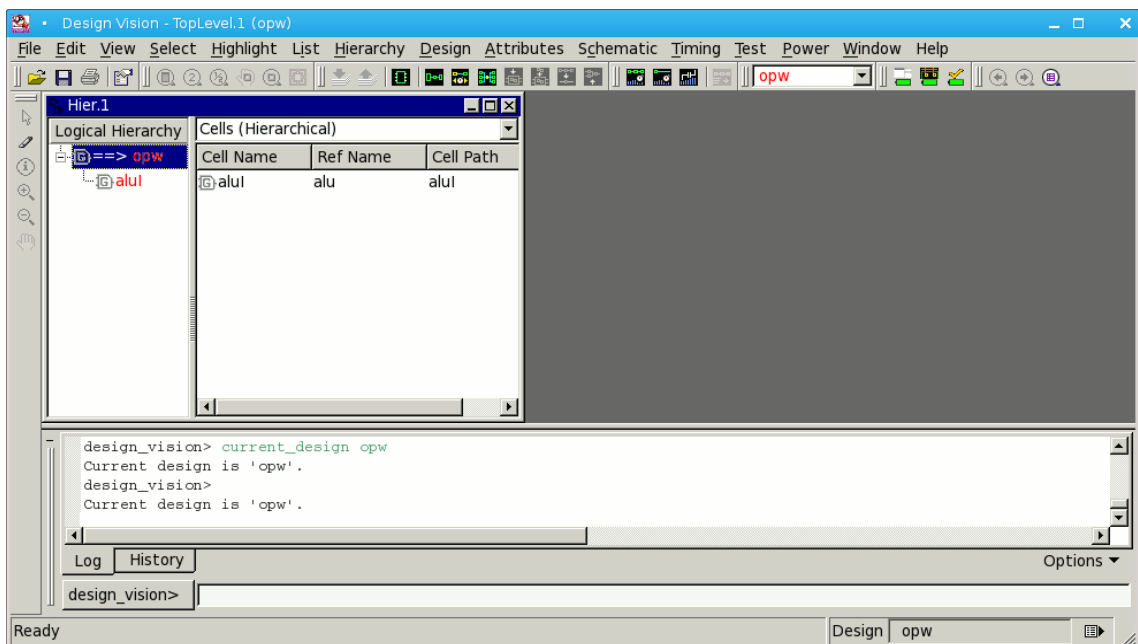
Die VHDL-Beschreibung ist dann abzuändern damit nicht unnötige Hardware generiert wird — im Falle von Latches wird meist auch die Funktion der Schaltung fehlerhaft!

Anmerkung: bei den synchronen Entwürfen die wir beschreiben, kann man davon ausgehen, dass Latches nicht vorkommen (sollten) und immer „Falsch“ sind!

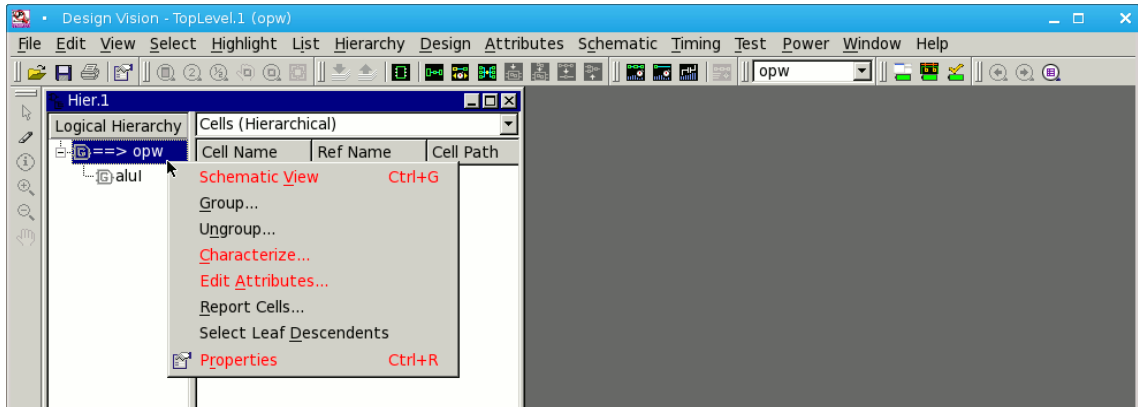
Behandlung der Hierarchie

5. (optional) Kontrolle der Hierarchie

Nach Auswahl eines (Teil-) Entwurfs wird dessen Hierarchie in dem Hierarchiebrowser des Synthesewerkzeugs dargestellt:



Neben der grafischen Darstellung der Hierarchie erlaubt dieses Werkzeug auch die weitere Handhabung der Instanzen, die Festlegung von Attributen für die Synthese etc. Die rechten Maustaste öffnet dazu ein kontextsensitives Menü:

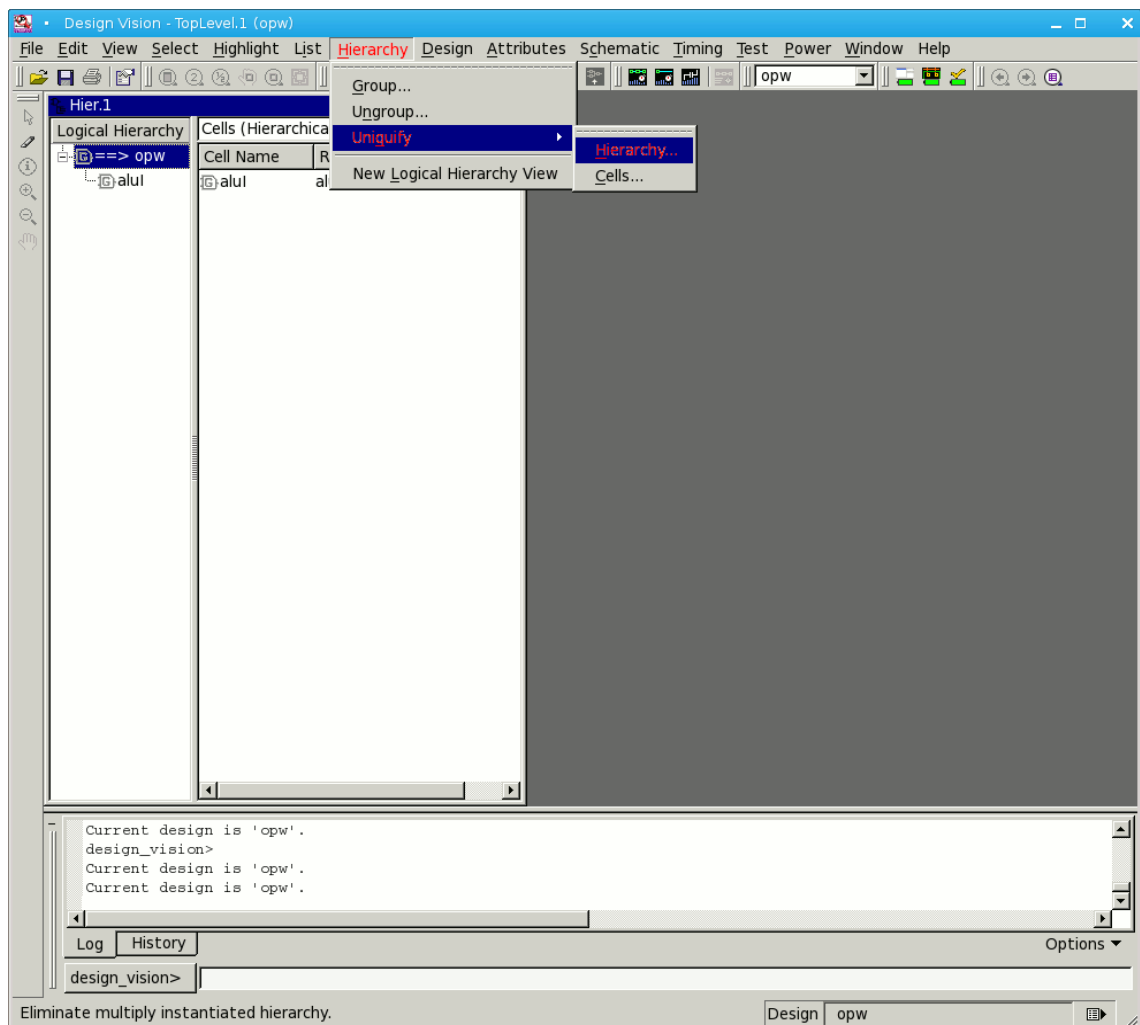


Achtung: Die folgenden Schritte sind nur notwendig, wenn innerhalb der Hierarchie Entities mehrfach instanziiert werden, andernfalls kann direkt mit der Eingabe der Syntheserandbedingungen ab Punkt 7 (Seite 13) begonnen werden!

Normalerweise wird die Hierarchie während der Synthese automatisch traversiert und alle instanziierten Entities bearbeitet. Für Teile der Hierarchie die mehrfach vorkommen, sind zwei Vorgehensweisen möglich: unterschiedliche (individualisierte) oder gleichartige Behandlung einzelner Instanzen.

6. (Instanzen mehrmals vorhanden) Unterschiedliche Behandlung / ein Syntheselauf
Diese Strategie ist anzuwenden, wenn Instanzen in *unterschiedlicher Weise* mehrfach benutzt werden, z.B. durch andere Generics, nicht benutzte Ausgänge oder konstante Eingangsbelegungen. Beispiele: konfigurierbare FIFOs mit unterschiedlicher Wortlänge und -Breite (Generics), die Instanzen eines Multiplizierers erhalten aus dem übergeordneten Design eines digitalen Filters jeweils einen konstanten Faktor (konstante Input-Ports).

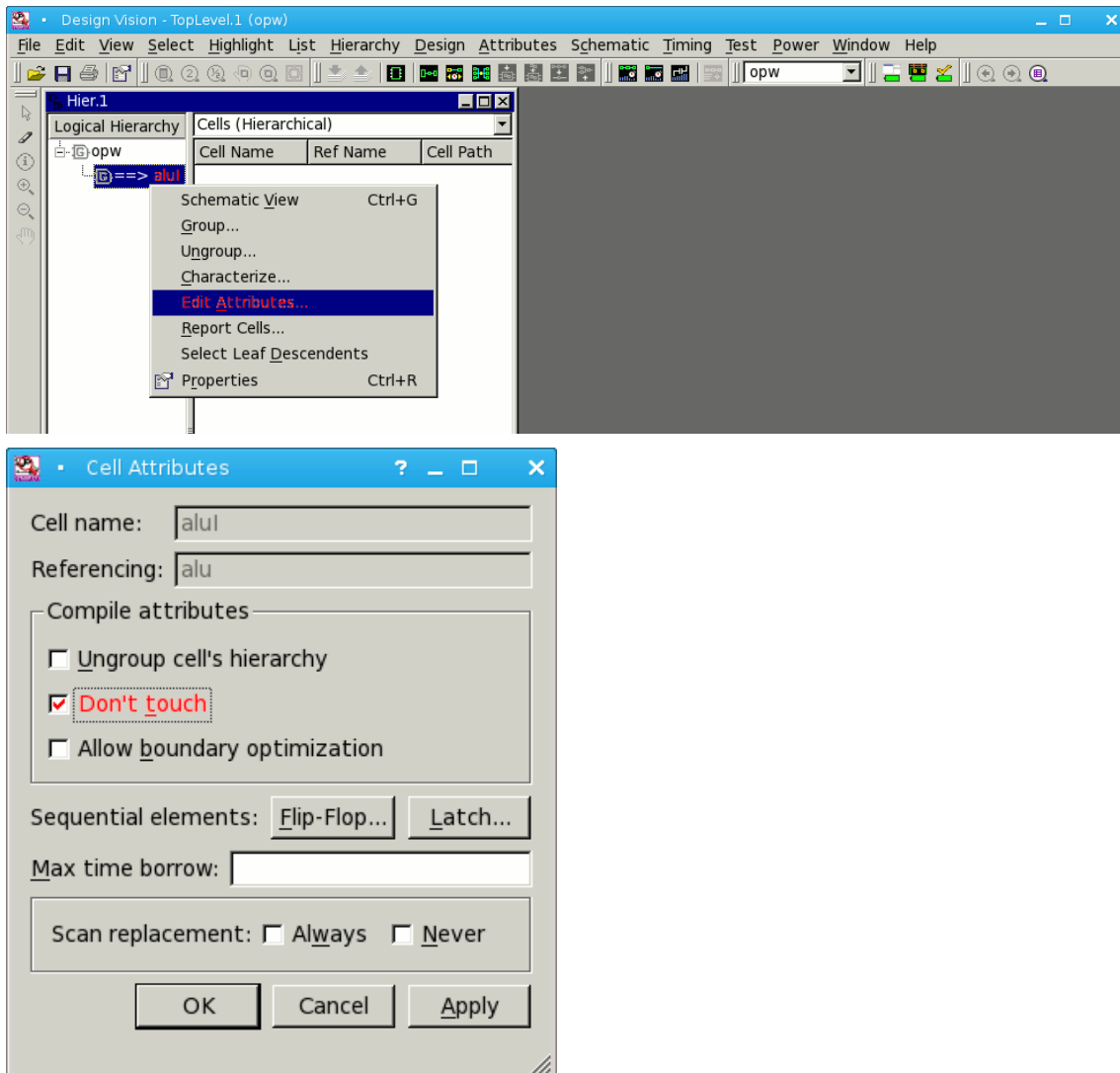
Ausgehend von der top-level Entity wird die Hierarchie durchlaufen und mehrfach vorhandene Elemente werden unterscheidbar gemacht, um sie bei der später folgenden Synthese individuell bearbeiten zu können:



6. (Instanzen mehrmals vorhanden) Gleichartige Behandlung / mehrere Syntheseschritte
Werden mehrfach referenzierte Elemente der Hierarchie immer in *gleicher Weise* benutzt, so sollten sie (aus Effizienzgründen) nur einmal synthetisiert werden. Beispiele: identische Recheneinheiten eines systolischen Arrays, Multipliziererinstanzen eines programmierbaren Filter (Faktoren frei wählbar).

Tipp: Bei sehr großen (Teil-) Entwürfen ist es auch sinnvoll die Synthese in kleinere „Portionen“ zu unterteilen, um die Programmlaufzeit und den Speicherbedarf geringer zu halten. In diesen Fällen wird auch die folgende Strategie der getrennten Synthese einzelner Teile eingesetzt.

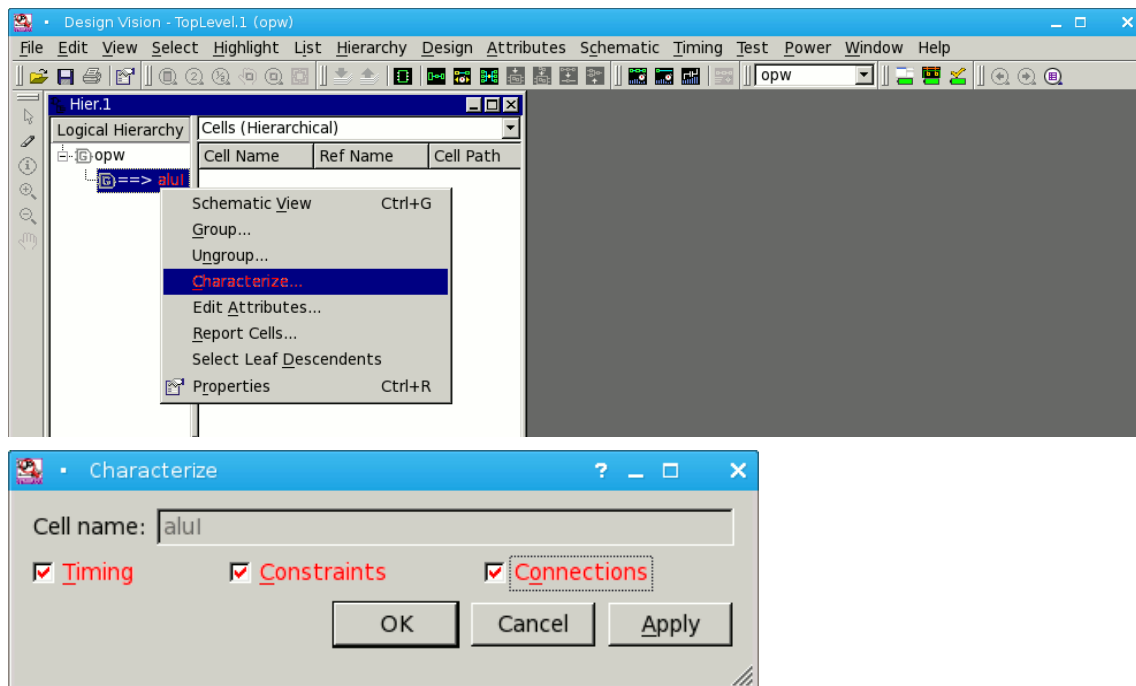
Dazu werden zuerst diejenigen Teile der Synthesehierarchie, die bei einem top-down Vorgehen nicht berücksichtigt werden sollen, mit dem Attribut `Don't touch` gekennzeichnet. Dies kann sowohl für Instanzen – hier immer „Cell“ genannt – als auch für Entities erfolgen. Die hier skizzierte Vorgehensweise behandelt einzelne Instanzen:



Anschließend wird eine top-down Synthese, wie ab Seite 13, beschrieben durchgeführt:

7. Top-level Entwurf Auswählen
8. Taktfrequenzen festlegen
- 9.–11. Synthesevorgaben machen
12. Operationsbedingungen einstellen
13. Synthese der Gatternetzliste

Die Randbedingungen des top-level Designs (Taktrate, Timing, Flächenvorgaben. . .) werden danach auf noch nicht synthetisierte Teilentwürfe propagiert:



Dann müssen diese Teilentwürfe, wie ab Punkt 13, Seite 20 beschrieben, synthetisiert werden. Nachdem *alle* Teile verarbeitet wurden, können die Ergebnisse ausgewertet (Seite 21) und die Daten ausgegeben werden (Seite 28).

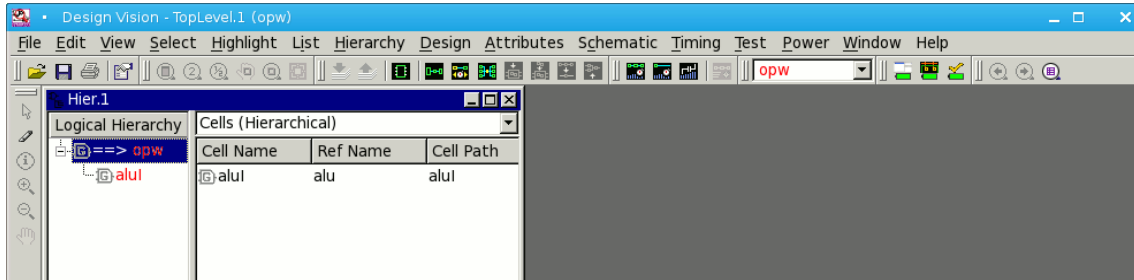
Synthesebedingungen festlegen

Bei der Realisierung einer Schaltung durch eine Gatternetzliste, 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 SYNOPSIS Dokumentation verwiesen. Im folgenden werden drei „einfache“ Möglichkeiten vorgestellt, die auch miteinander beliebig kombiniert werden können.

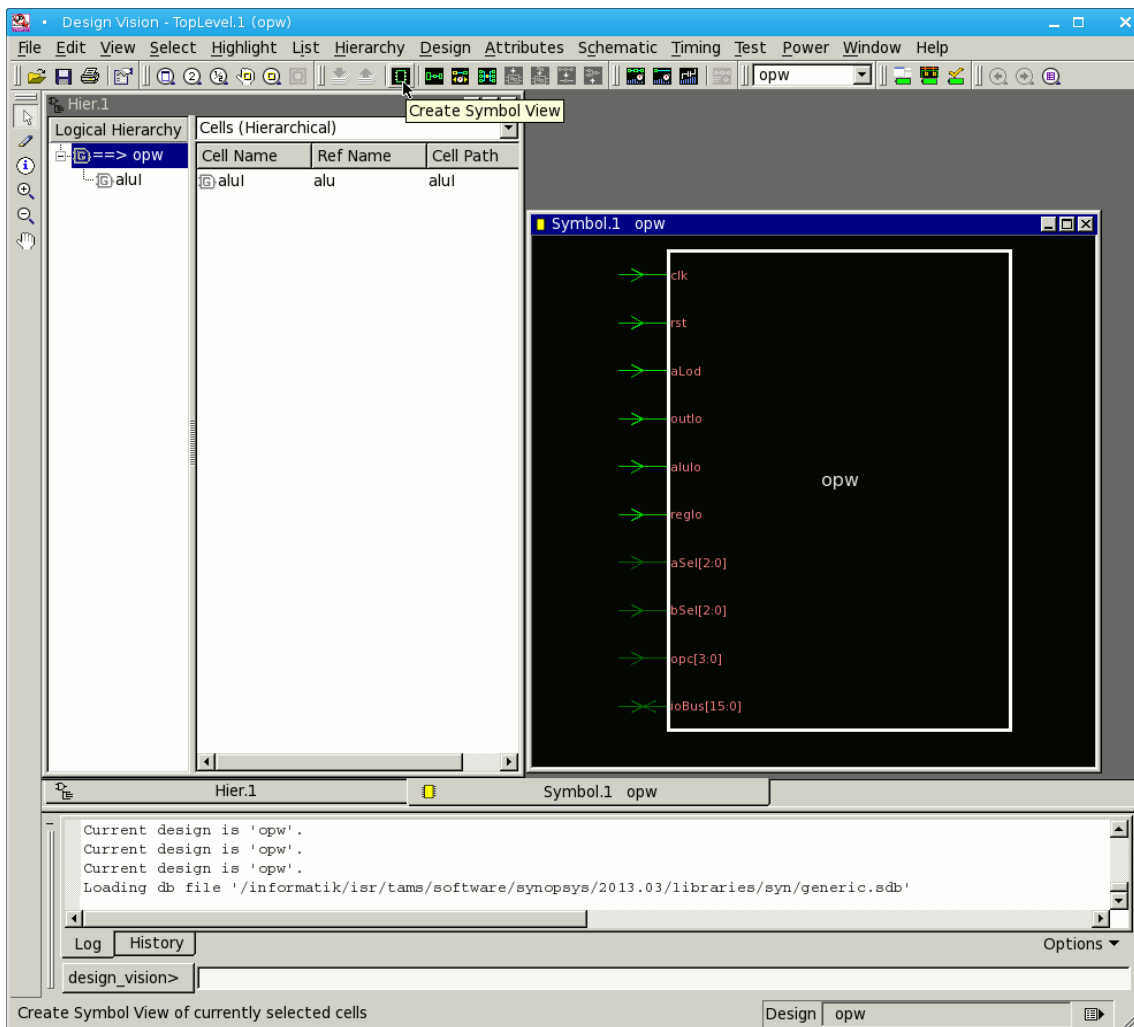
Tipp: für „optimale“ Syntheseergebnisse ist es besser mit realistischen Werten für Fläche bzw. Geschwindigkeit zu synthetisieren und diese Randbedingungen über mehrere Syntheseläufe zu verschärfen.

7. Auswahl des top-level Entwurfs im Menü der GUI, bzw. im Hierarchiebrowser. Alle weiteren Befehle beziehen sich darauf:



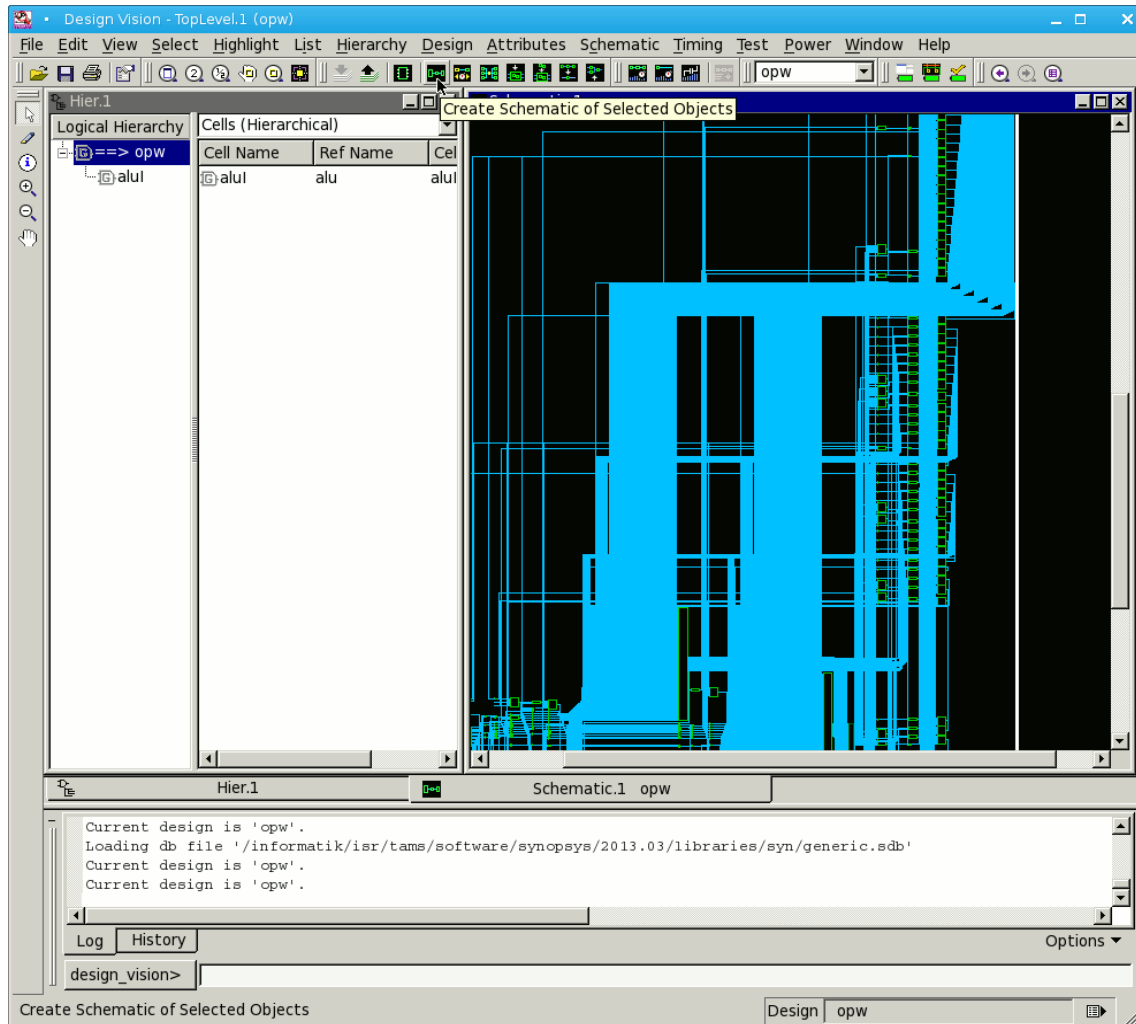
Sollen später Taktfrequenzen und Zeitbedingungen festgelegt werden, dann sollte man ein Symbol und ein Schematic erzeugen, um dort durch Selektion mit der Maus Signale und Ports auszuwählen.

Symbol erzeugen



Schematic erzeugen

Das Schematic der Schaltung enthält *vor* der Synthese nur künstliche Elemente einer internen Bibliothek. Erst *nach* dem Syntheseprozess ist die Netzliste des Schematic aus den Zellen der Gatterbibliothek aufgebaut.



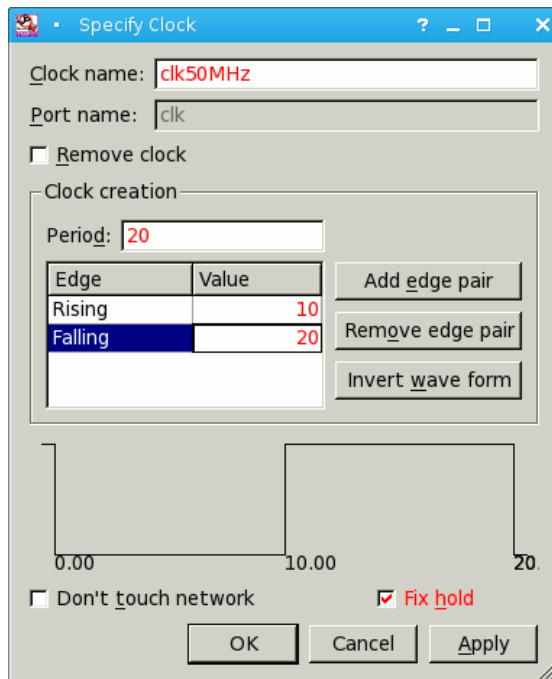
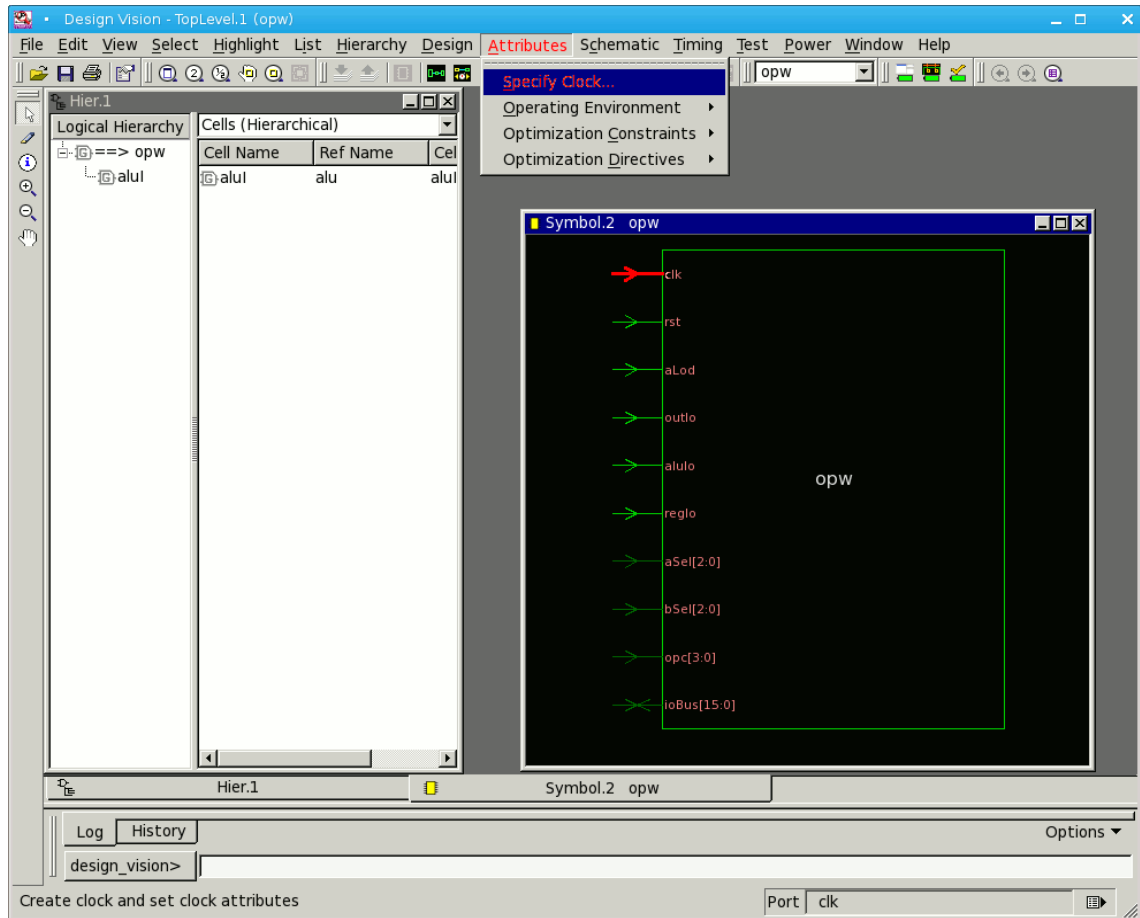
8. Taktfrequenz(en) festlegen

In der Regel enthalten die Schaltungen Taktleitungen, deren Taktschema (Frequenz, Phasenlage zueinander) für die Synthese unbedingt angegeben werden sollte. Bei Entwürfen ohne explizite Taktleitungen (Schaltnetzen) kann man stattdessen Verzögerungszeiten zwischen Ein- und Ausgängen definieren.¹

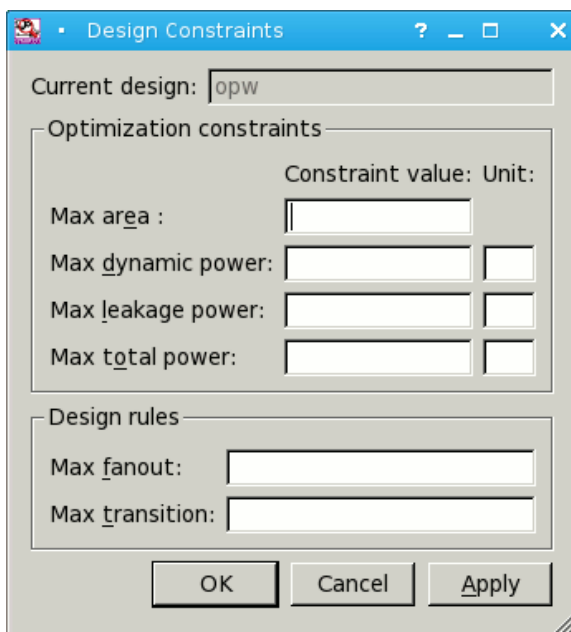
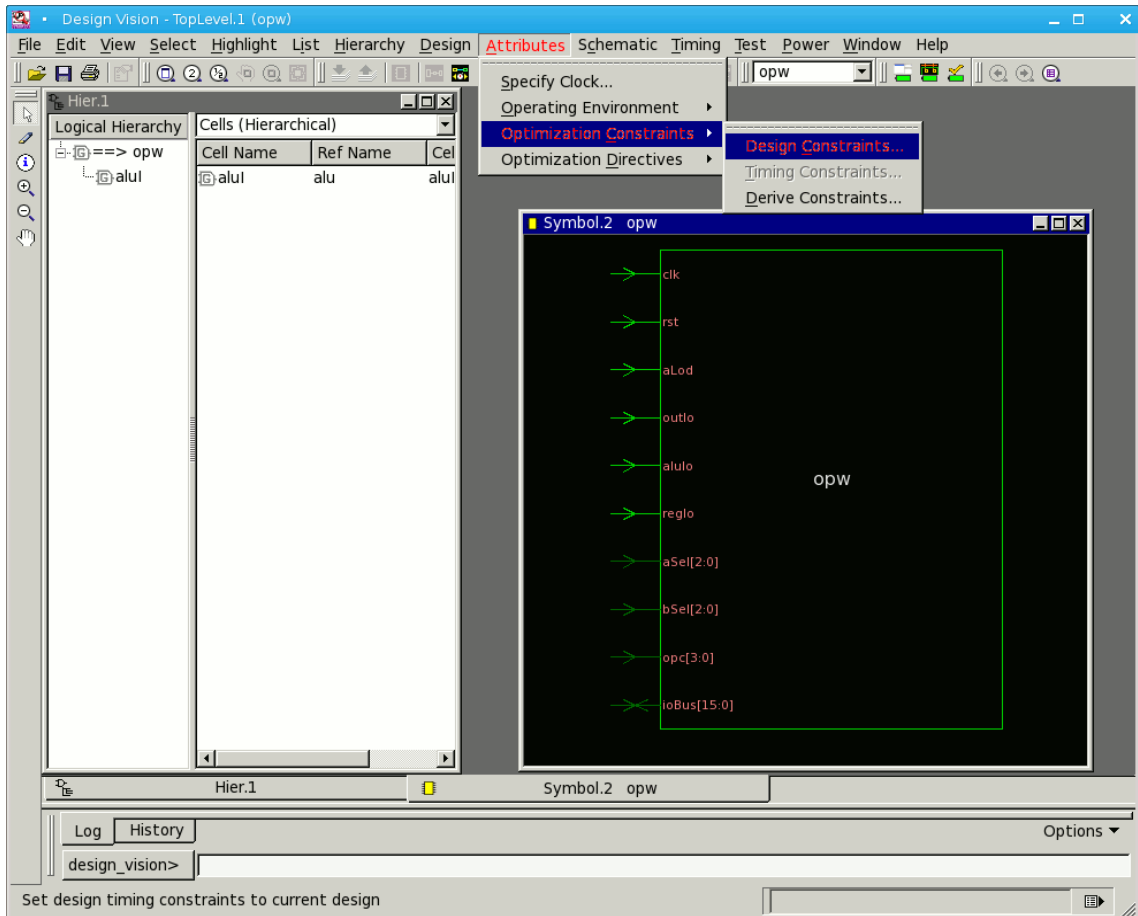
Randbedingungen für das Taktschema stellen die einfachste und sicherste Möglichkeit dar, um Zeitbedingungen in der Synthese zu definieren. Die Taktperiode wird in *ns* angegeben. Bei der Optimierung wird der Pfad zwischen, bzw. vor, Registern berücksichtigt, so dass die explizite Vorgabe von Zeitpfaden (s.u.) überflüssig wird.

¹hier nicht weiter beschrieben

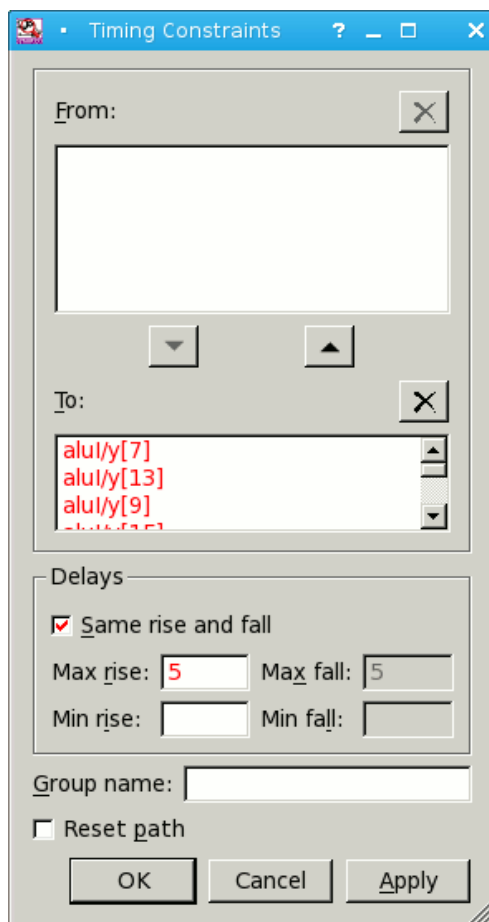
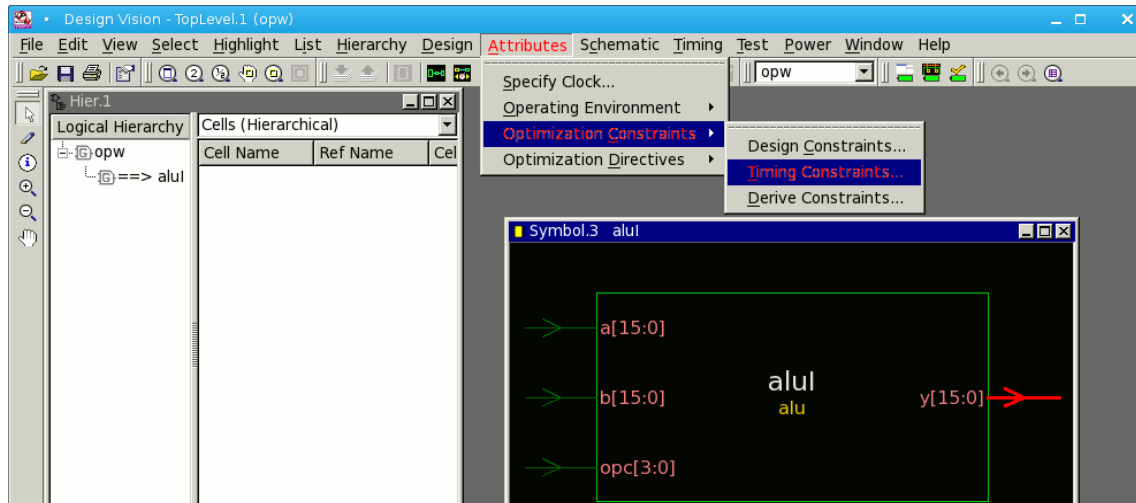
Dazu sind in dem Symbol des top-level Entwurfs die Taktleitungen zu selektieren und dann das Taktschema zu definieren:



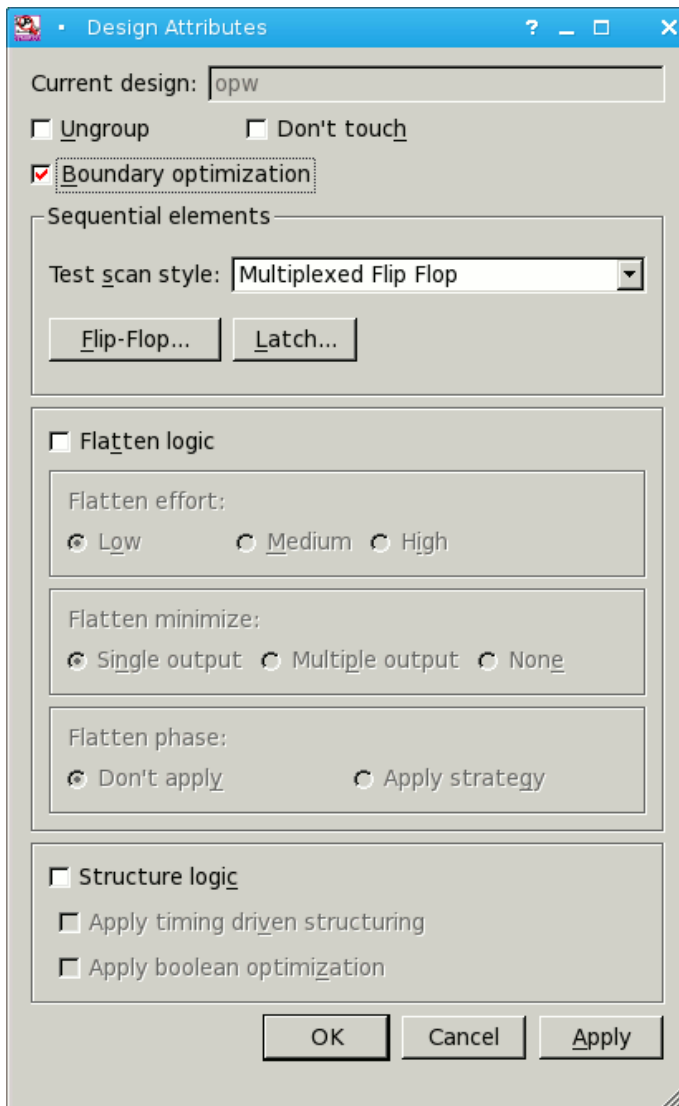
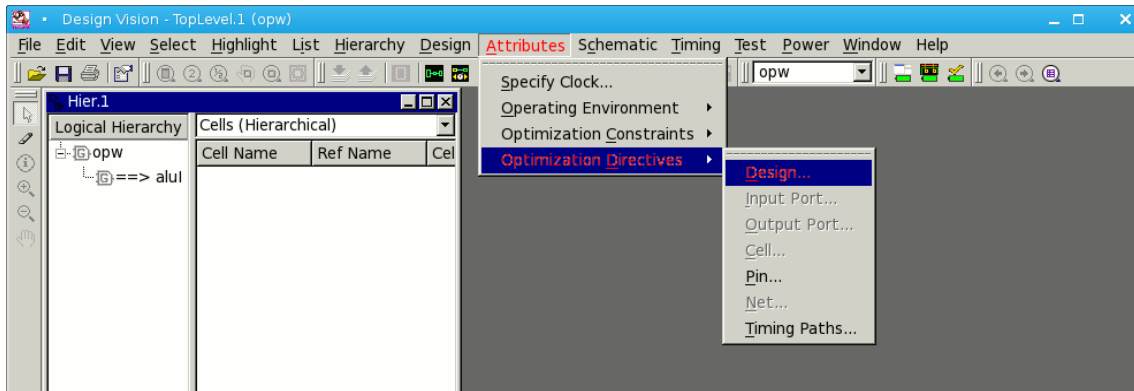
9. (optional) Flächenvorgaben — werden in der Regel nicht weiter benötigt, da als Voreinstellung möglichst kleine, kompakte Netzlisten synthetisiert werden:



10. (optional) Timingvorgaben — werden in der Regel nicht benötigt, da das Zeitverhalten über die Taktung (s.o.) definiert ist. Das Timing kann zwischen beliebigen Stellen der Netzliste explizit angegeben werden; dabei sind sowohl minimale als auch maximale Laufzeiten möglich. Die Anfangs- oder Endpunkte von Pfaden sollten vorher im Schematic, bzw. Symbol selektiert werden:

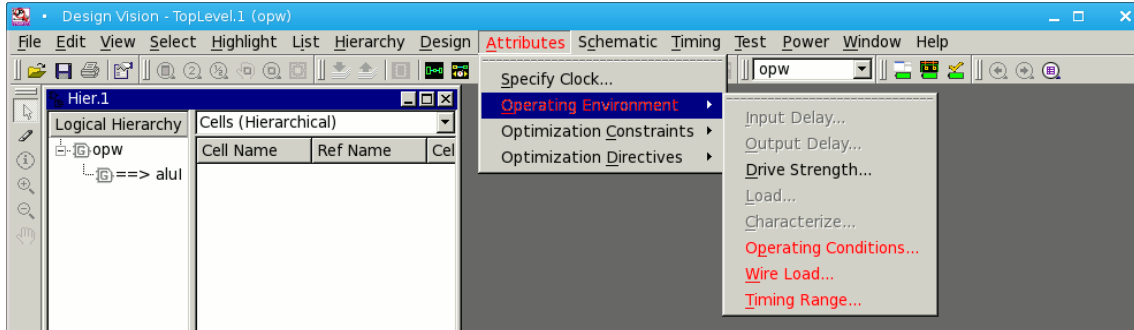


11. (optional) Syntheseattribute — sind sinnvoll voreingestellt und sollten nur in Ausnahmefällen (siehe SYNOPSIS Dokumentation) geändert werden:

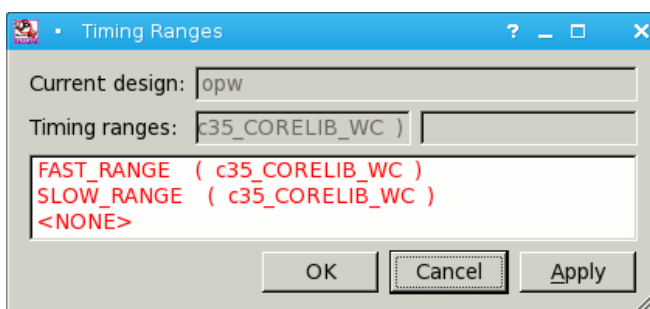
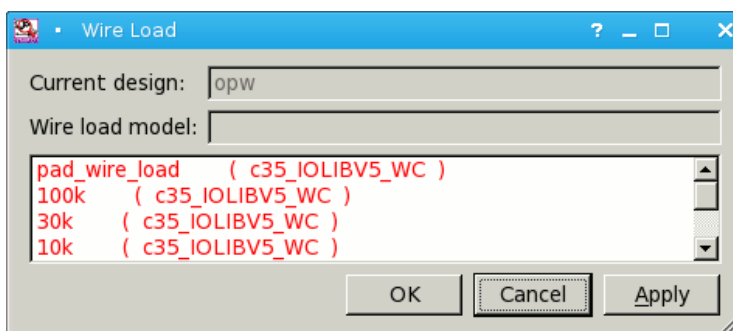
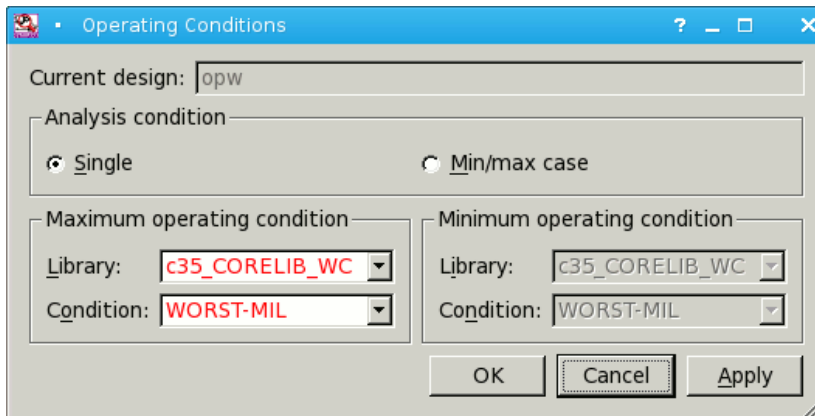


12. (optional) Operationsbedingungen einstellen

Für die später folgenden Schritte (Synthese und Timinganalyse) können die Zeitmodelle der Gatter und externe Lasten festgelegt werden:



Die Standardzellen sind typischerweise für Bereiche (Ober- und Untergrenzen) von Geschwindigkeit, Versorgungsspannung und Temperatur spezifiziert. Je nach Einsatzzweck werden so passende Parametersätze ausgewählt, z.B.: Standard, INDUSTRIAL und MILITARY.

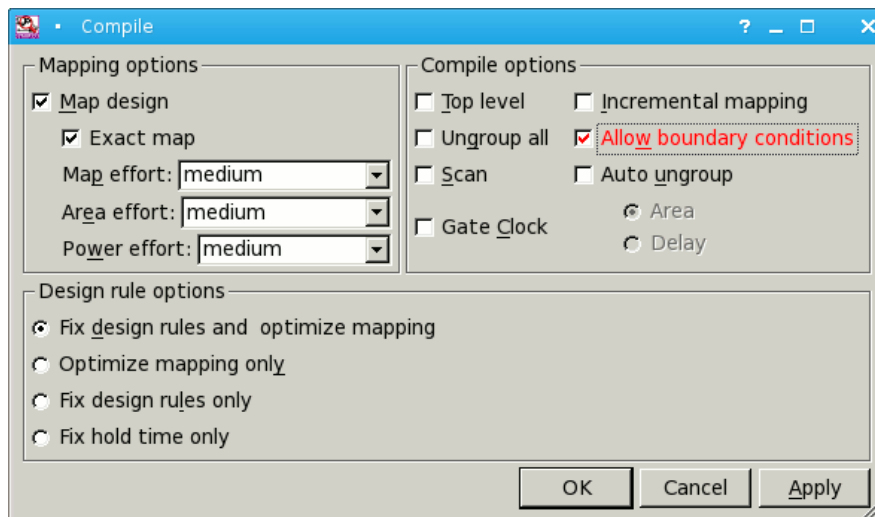
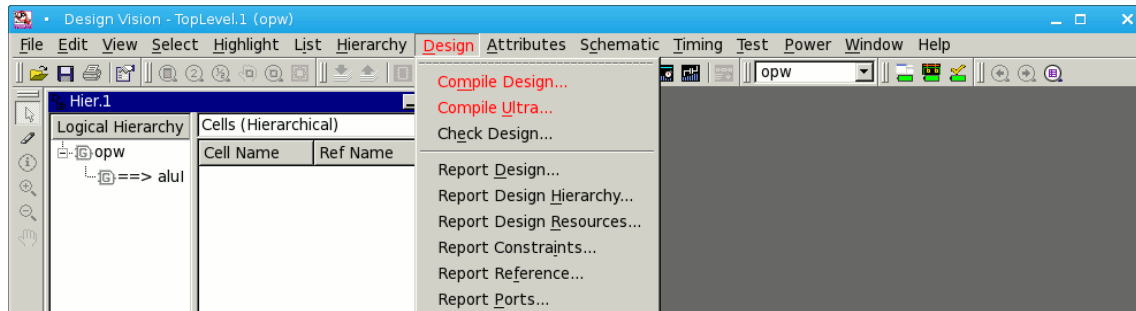


Synthese der Gatternetzliste

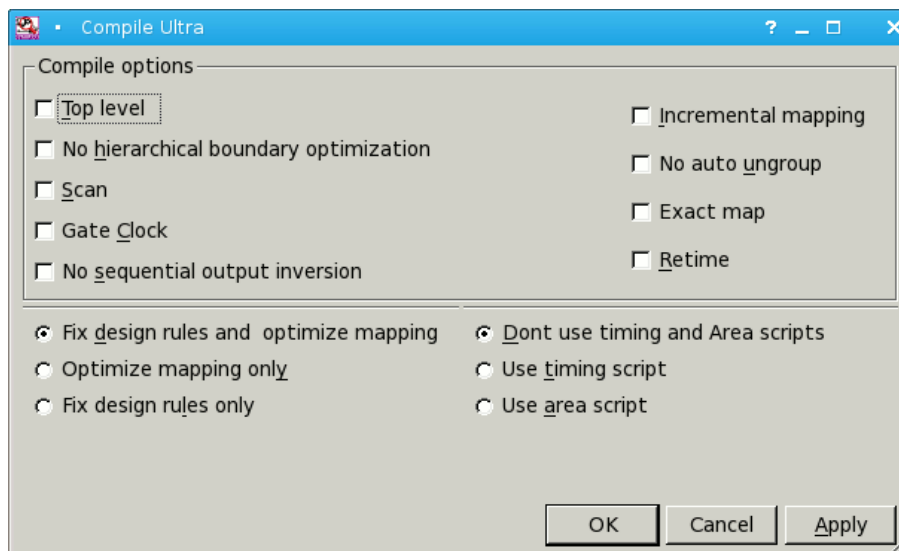
13. Hardwaresynthese und Abbildung auf die Zellbibliothek

Ausgehend von dem aktuellen (Teil-) Entwurf, durchläuft die Synthese die gesamte Hierarchie — die Ausnahme bilden *Don't touch*-Attribute, siehe „Behandlung der Hierarchie“, ab Seite 9.

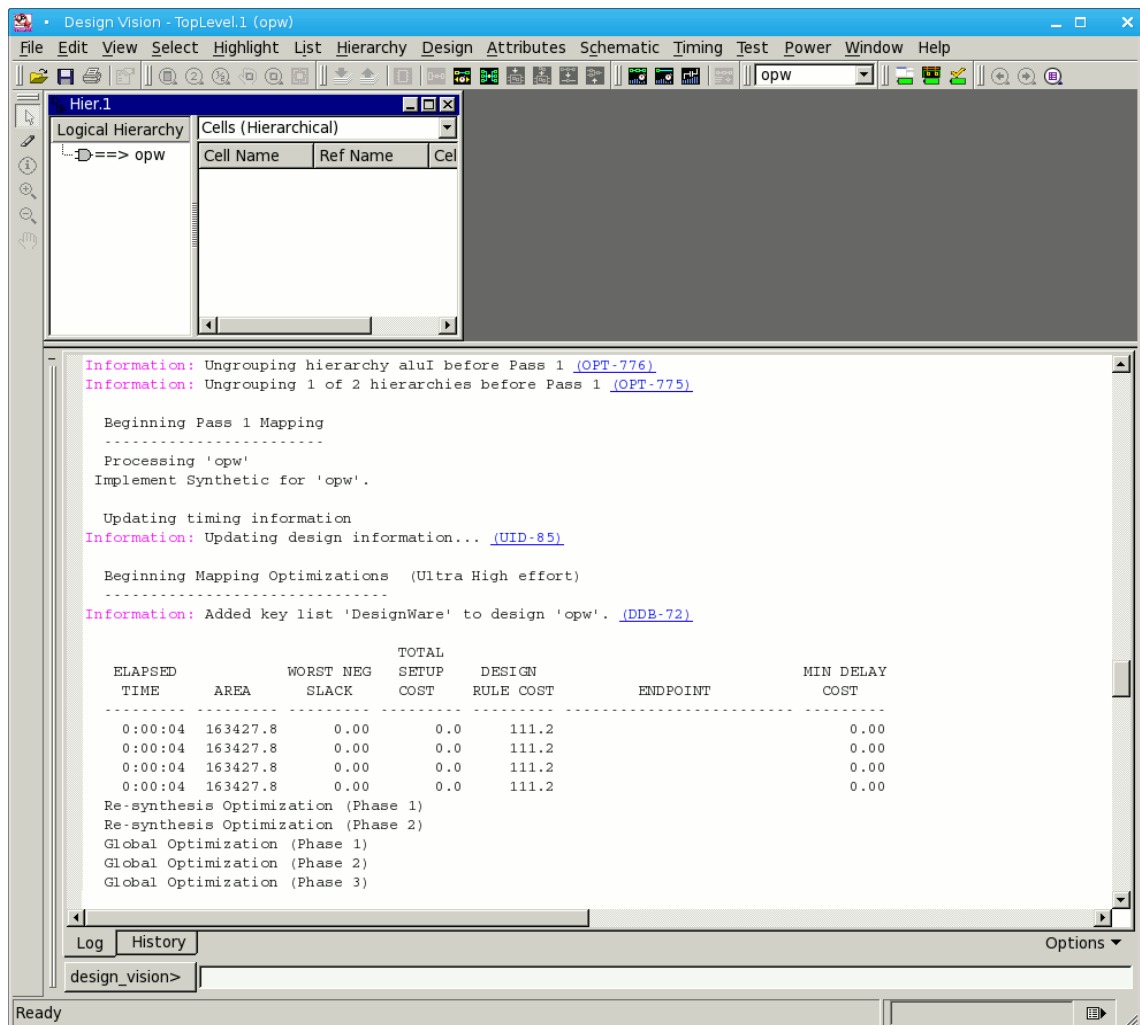
Für die Synthese stehen zwei verschiedene Programmkomponenten zur Verfügung:



oder



Die einzelnen Schritte des Syntheseprozesses werden in der Log-Datei mitprotokolliert:



14. Bewertung der Synthesergebnisse

Entspricht das Ergebnis nicht den Anforderungen, so müssen die Randbedingungen der Synthese (ab Punkt 7, Seite 13) entsprechend angepasst und ein neuer Syntheselauf (Punkt 13, Seite 20) gestartet werden.

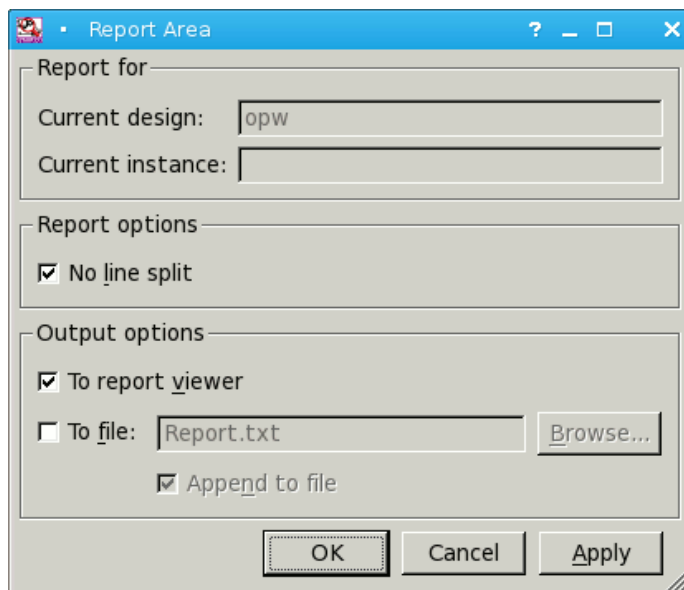
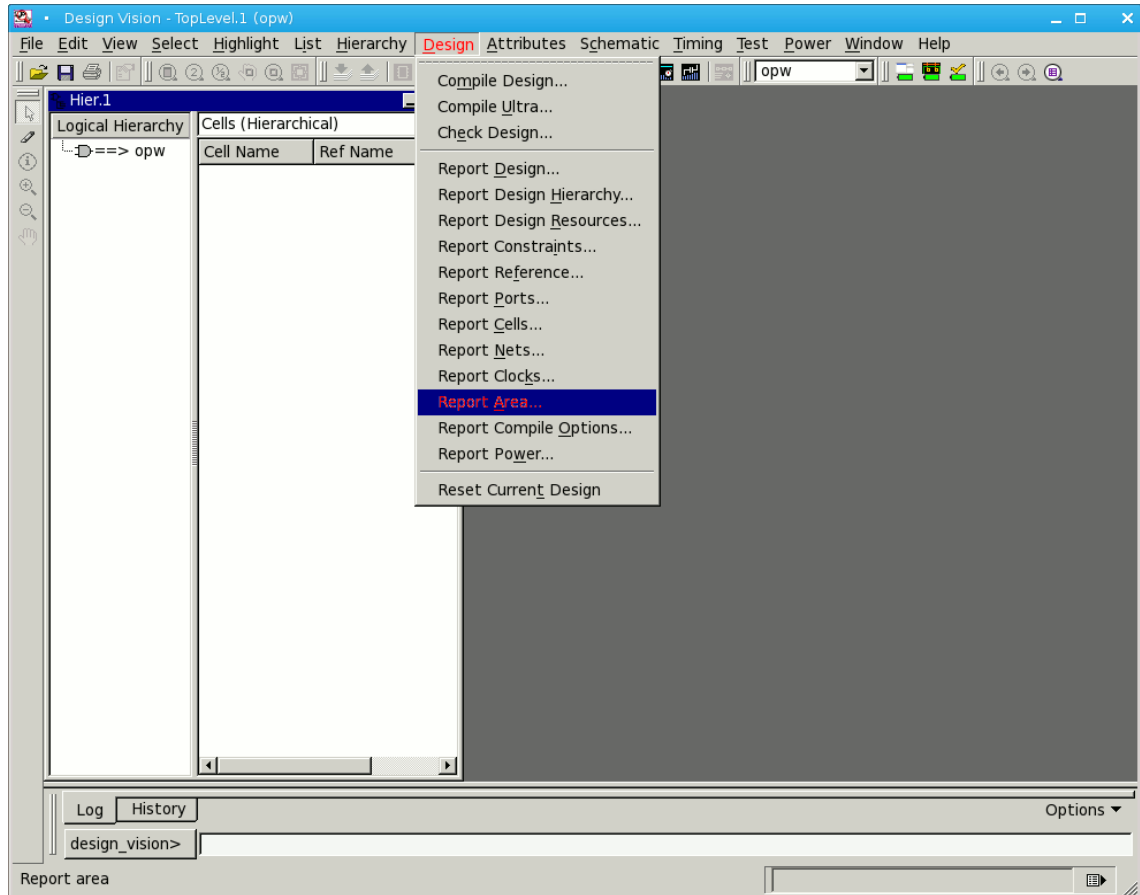
Hier werden nur einige der umfangreichen Analysemöglichkeiten des Synthesewerkzeugs vorgestellt. Die bei der Timinganalyse ausgegebene Information bezieht sich immer auf das unter Punkt 12, Seite 19 festgelegte Zeitmodell. Durch Auswahl anderer Operationsbedingungen können „worst-case“ und „best-case“ Timing der synthetisierten Struktur ermittelt werden.

Kontrolle des Schematic und der Hierarchie

Wie schon zuvor auf Seite 14 gezeigt, kann ein Schematic der synthetisierten Netzliste erzeugt, angesehen und die Hierarchie traversiert werden.

Ausgabe von Statistiken — Flächenbedarf

SYNOPTIS erlaubt die Ausgabe sehr detaillierter Statistiken, wobei besonders die Fläche interessant ist:



Die Ausgabe, hier in der Log-Datei, enthält folgende Flächenmaße [μm^2]:

The screenshot shows the Design Vision software interface. The main window displays a report titled "Report.1 - Area" with the following content:

```

*****
Report : area
Design : opw
Version: H-2013.03-SP2
Date   : Fri Nov 1 14:28:34 2013
*****

Information: Updating design information... (UID-85)
Library(s) Used:

    c35_CORELIB_WC (File: /informatik/isr/tams/software/ams/v4.10/synop

Number of ports:           32
Number of nets:            1098
Number of cells:           915
Number of combinational cells: 723
Number of sequential cells: 192
Number of macros/black boxes: 0
Number of buf/inv:         89
Number of references:      59

Combinational area:        70852.600136 Fläche für Logikgatter
Buf/Inv area:              3294.200127 davon Inverter/Teiber
Noncombinational area:    46883.200195 Fläche für Flipflops+Latches
Macro/Black Box area:     0.000000
Net Interconnect area:    23823.000000 geschätzte Verdrahtungsfläche

Total cell area:          117735.800331
Total area:               141558.800331

***** End Of Report *****

```

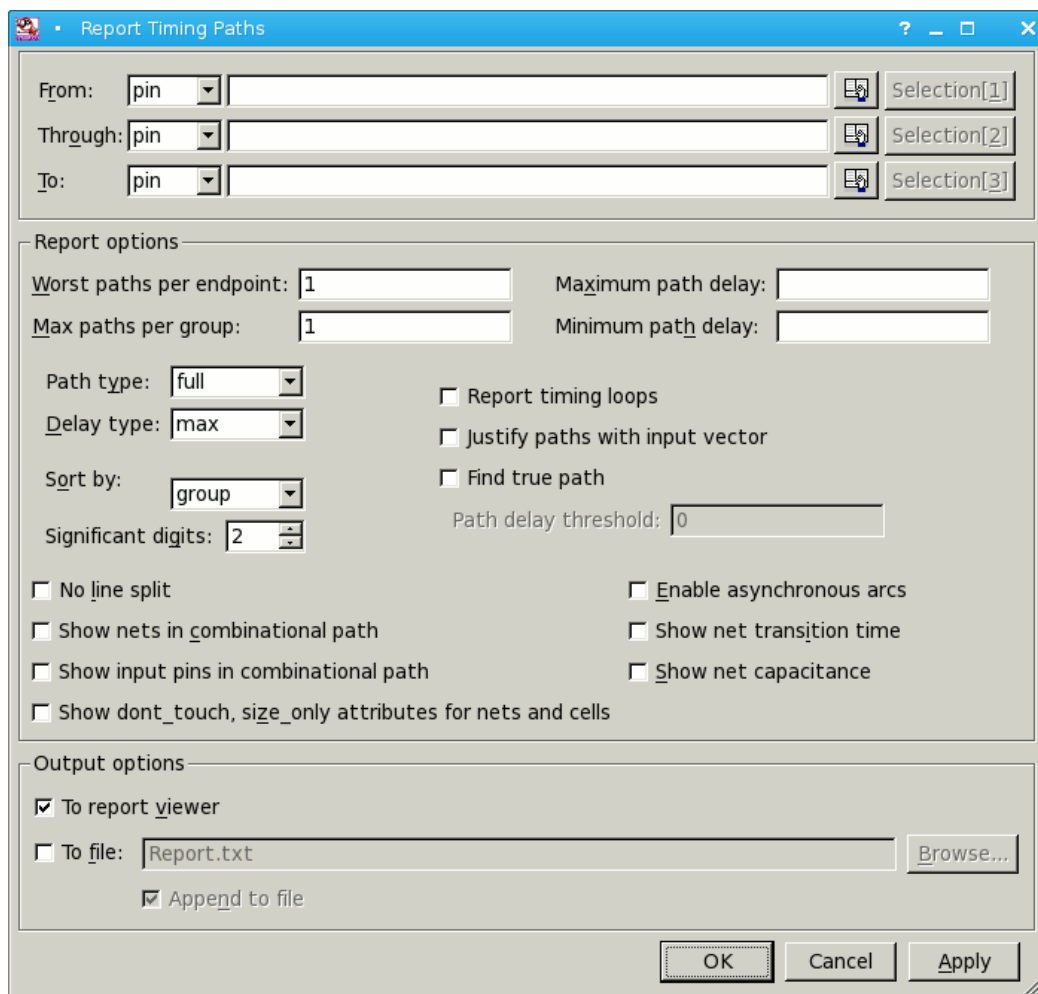
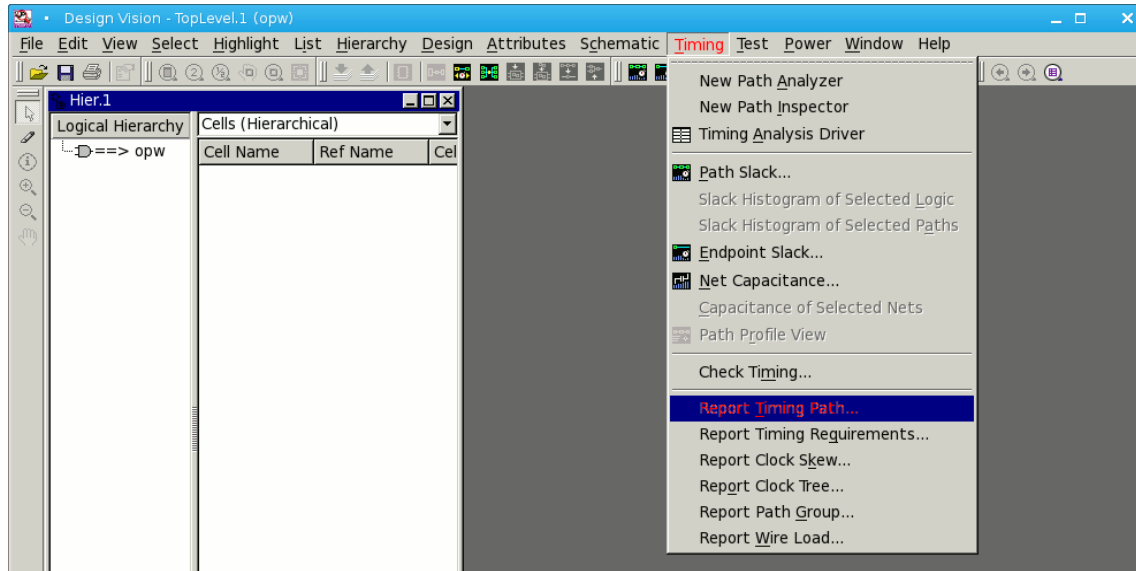
Below the report window, a summary table is visible:

Net Interconnect area:	23823.000000
Total cell area:	117735.800331
Total area:	141558.800331

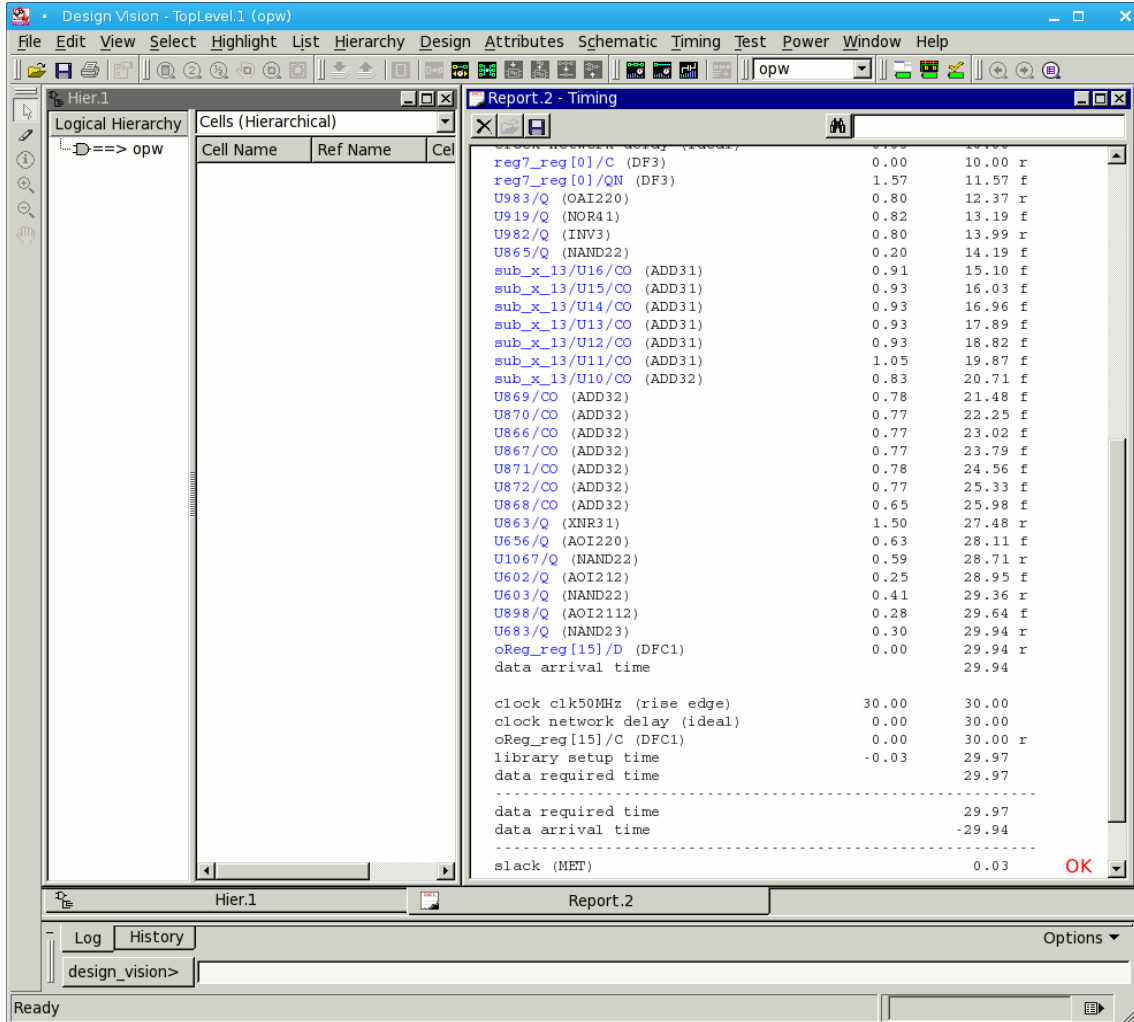
The interface also shows a command prompt at the bottom with the prompt "design_vision>" and a status bar at the very bottom that says "Ready".

Timing der Schaltung

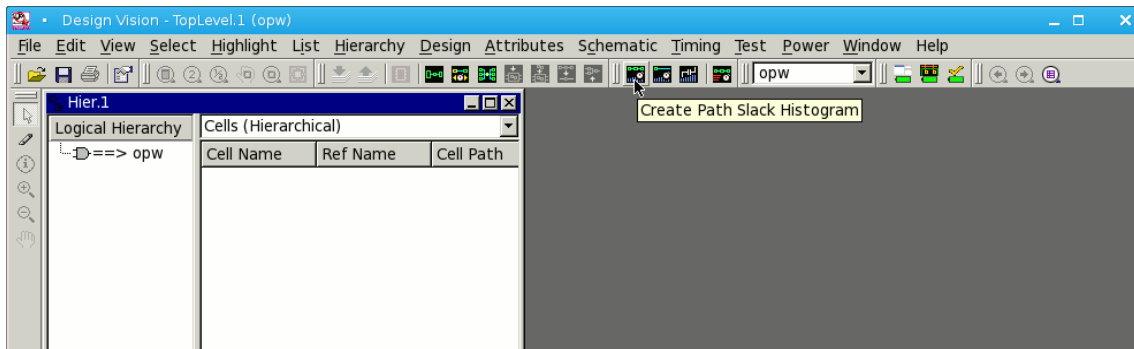
Das gesamte Timing der Schaltung wird ausgegeben, wenn kein Netz explizit ausgewählt wurde, ansonsten wird das Zeitverhalten dieses Netzes ausgegeben:

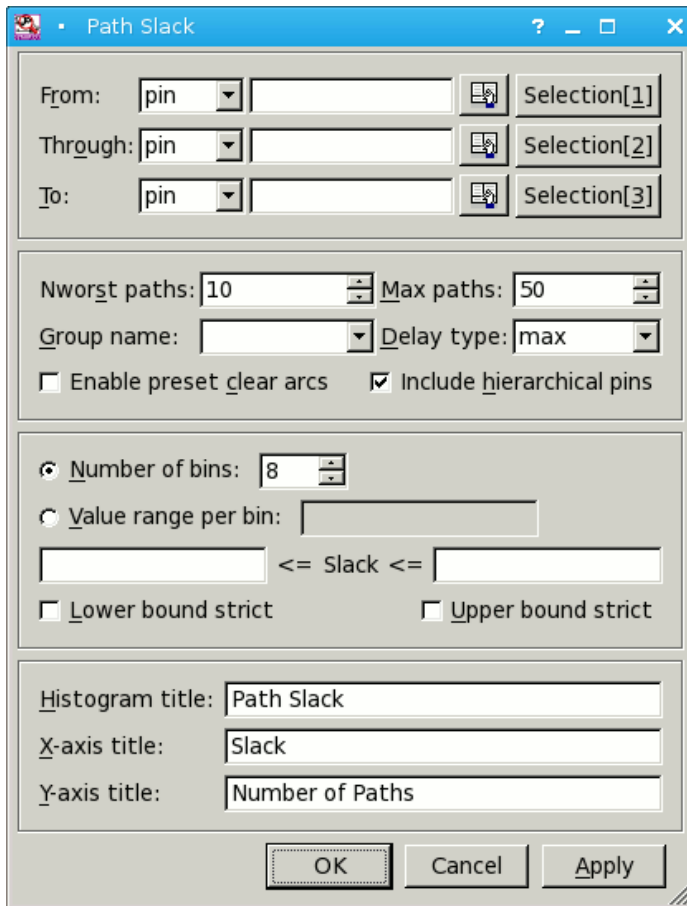


Die Ausgabe des Timing-Reports bezieht sich auf den kritischen (längsten) Pfad. Das Timing wurde hier, was auch sonst meistens der Fall sein dürfte, durch den Chiptakt spezifiziert:

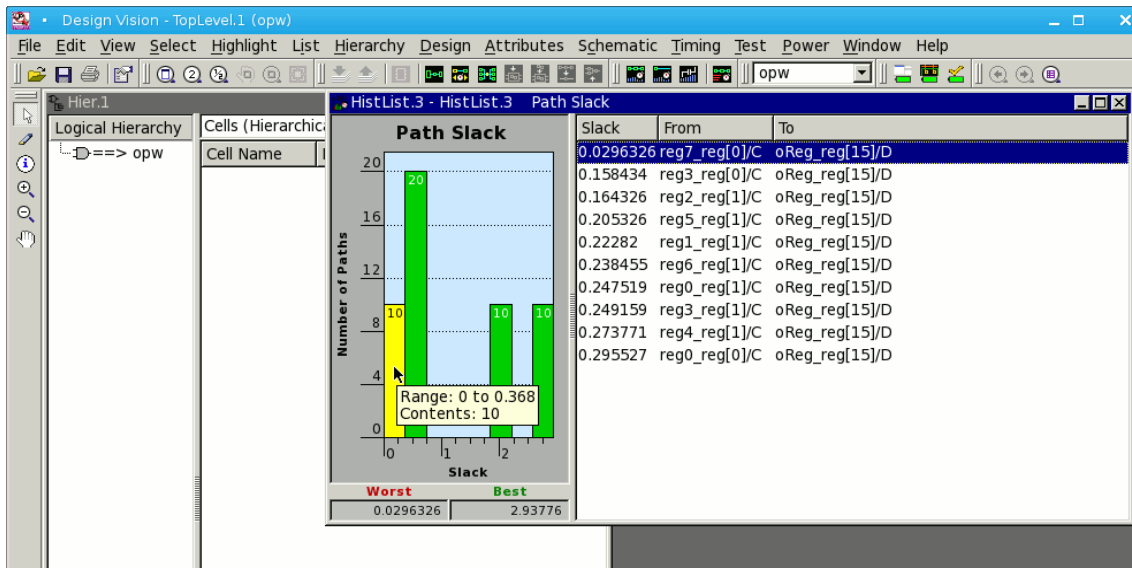


Für alle Pfade mit expliziten Zeitbedingungen kann deren Verteilung als „path slack histogram“ angezeigt werden.





Auch hier sieht man den längsten Pfad...



Dieser kritische Pfad, wie auch beliebige Pfade ausgewählter Netze, können im Schematic visualisiert werden:

The top screenshot shows the 'Create Schematic of Selected Objects' window with the following table:

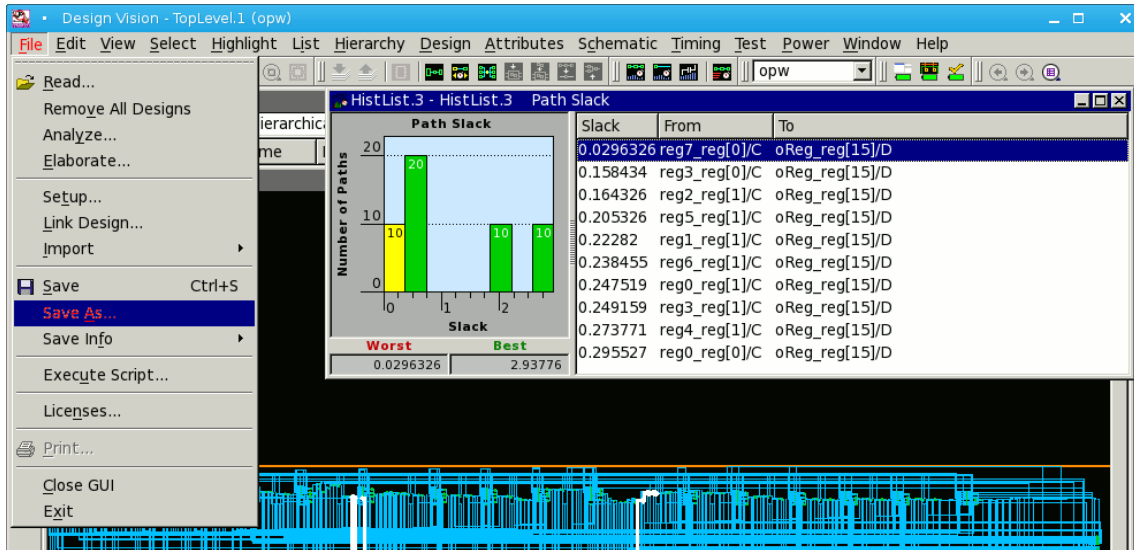
Slack	From	To
96326	reg7_reg(0)/C	oReg_reg[15]/D
8434	reg3_reg(0)/C	oReg_reg[15]/D
4326	reg2_reg(1)/C	oReg_reg[15]/D
5326	reg5_reg(1)/C	oReg_reg[15]/D
282	reg1_reg(1)/C	oReg_reg[15]/D
8455	reg6_reg(1)/C	oReg_reg[15]/D
7519	reg0_reg(1)/C	oReg_reg[15]/D
9159	reg3_reg(1)/C	oReg_reg[15]/D
8771	reg4_reg(1)/C	oReg_reg[15]/D
5527	reg0_reg(0)/C	oReg_reg[15]/D

The bottom screenshot shows the 'HistList.3 - HistList.3 Path Slack' window with a histogram and a detailed path visualization. The histogram shows the number of paths for each slack value (0, 1, 2). The 'Worst' slack is 0.0296326 and the 'Best' is 2.93776. The detailed path visualization shows a circuit schematic with a critical path highlighted in red.

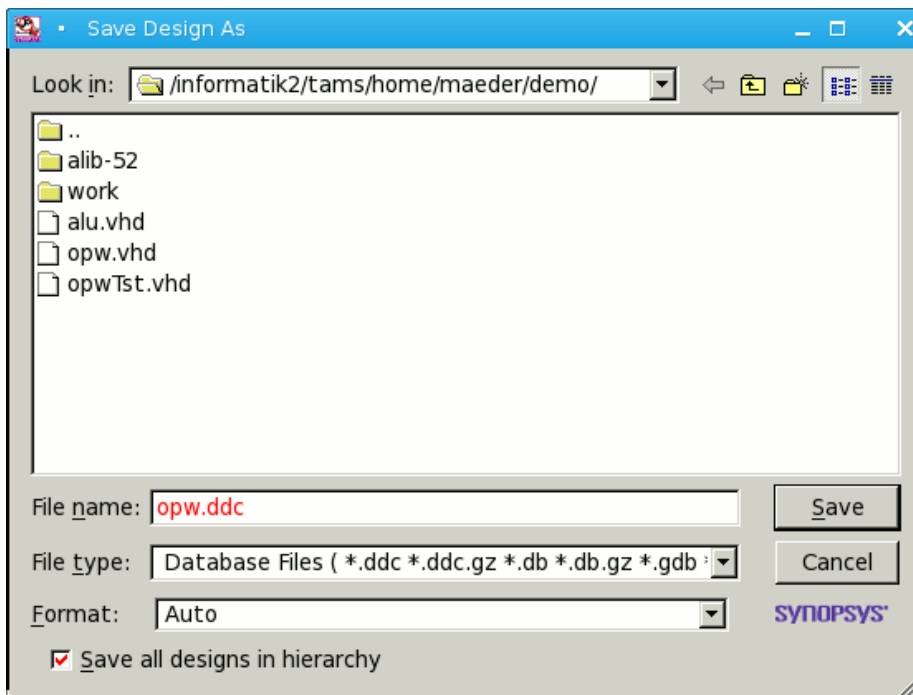
Slack	From	To
0.0296326	reg7_reg(0)/C	oReg_reg[15]/D
0.158434	reg3_reg(0)/C	oReg_reg[15]/D
0.164326	reg2_reg(1)/C	oReg_reg[15]/D
0.205326	reg5_reg(1)/C	oReg_reg[15]/D
0.22282	reg1_reg(1)/C	oReg_reg[15]/D
0.238455	reg6_reg(1)/C	oReg_reg[15]/D
0.247519	reg0_reg(1)/C	oReg_reg[15]/D
0.249159	reg3_reg(1)/C	oReg_reg[15]/D
0.273771	reg4_reg(1)/C	oReg_reg[15]/D
0.295527	reg0_reg(0)/C	oReg_reg[15]/D

Sichern und Ausgabedateien erzeugen

15. Sichern der internen Datenbasis:

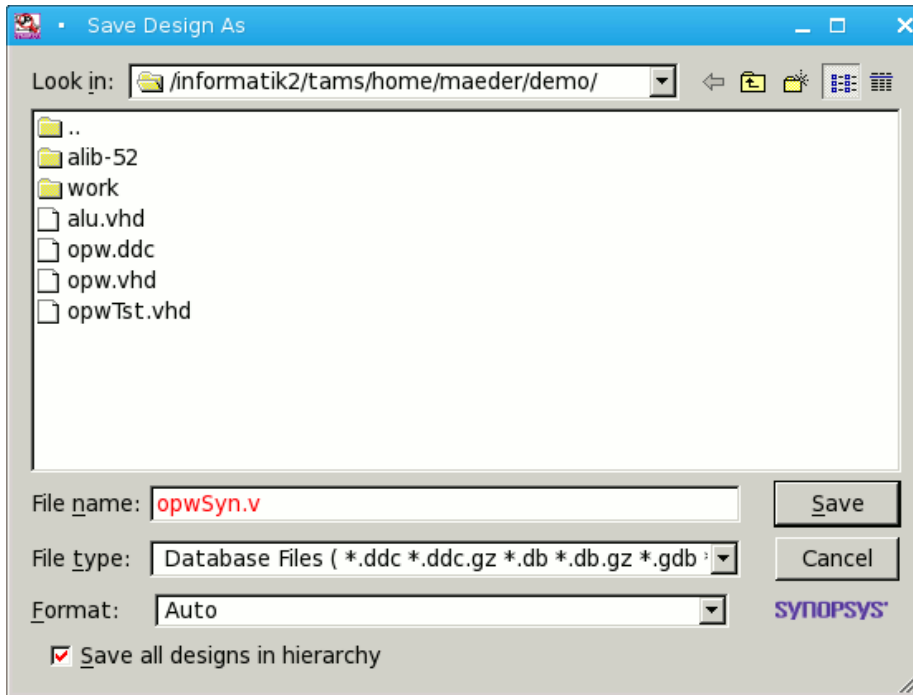


Soll der Entwurf später noch einmal bearbeitet werden, so kann man die hier gesicherte Datei laden (anstatt Schritt 3, Seite 4). Sie beinhaltet neben allen Elementen der Hierarchie auch die spezifischen Einstellungen für Taktsignale, Syntheserandbedingungen, Attribute etc.



16. Verilog Netzliste schreiben

Für die Simulation der synthetisierten Netzliste, aber auch für die folgende Platzierung und Verdrahtung durch ein Standardzell-Backend (CADENCE SoC Encounter), wird eine Datei in der Hardwarebeschreibungssprache Verilog erzeugt:

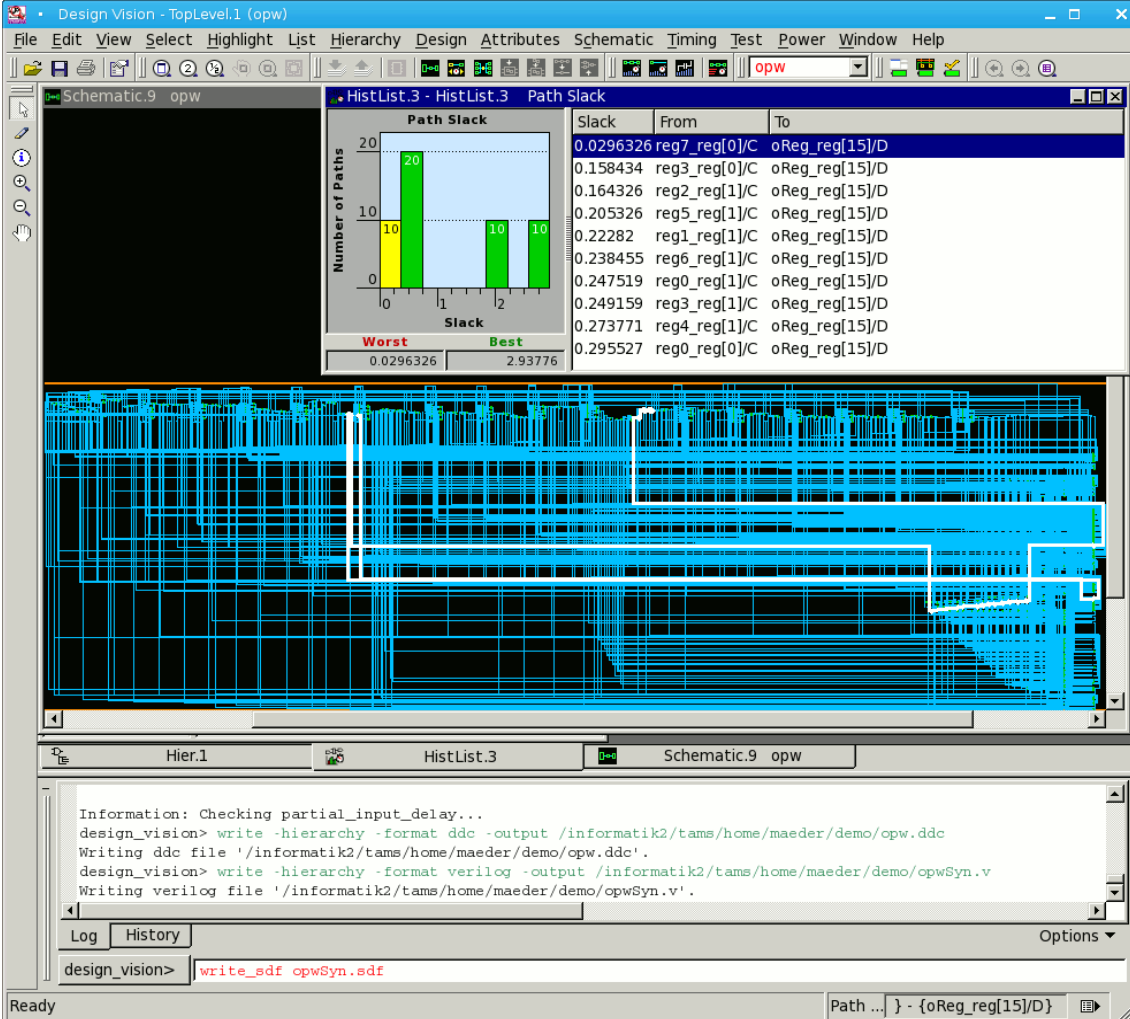


Aus Effizienzgründen wird eine gemischte Simulation von VHDL-Testumgebung und Verilog-Netzliste bevorzugt. Die dazu notwendigen Schritte sind in der extra Beschreibung „Simulation von Gatternetzlisten – VHDL- und Mixed-mode“ erläutert.

17. SDF-Datei schreiben

Die SDF-Datei (Standard Delay Format) beschreibt für die Netzliste die Verzögerungszeiten von Gattern und Leitungen. Da noch kein Layout der Schaltung erstellt wurde, aus dem man genaue Leitungslaufzeiten extrahieren kann, werden diese nur über Heuristiken geschätzt. Um die (Verilog-) Netzliste später mit den Verzögerungszeiten zu simulieren, wird diese Datei benötigt.

Der Befehl wird über die Kommandozeile eingegeben und bezieht sich auf das aktive Design, hier opw:



The screenshot displays the Design Vision interface with the following components:

- Path Slack Histogram:** A bar chart showing the number of paths for different slack values. The x-axis is labeled 'Slack' with values 0, 1, and 2. The y-axis is labeled 'Number of Paths' with values 0, 10, and 20. The bars are colored yellow (10 paths at slack 0), green (20 paths at slack 1), and blue (10 paths at slack 2).
- Path Slack Table:** A table listing paths and their slacks. The columns are 'Slack', 'From', and 'To'. The data is as follows:

Slack	From	To
0.0296326	reg7_reg[0]/C	oReg_reg[15]/D
0.158434	reg3_reg[0]/C	oReg_reg[15]/D
0.164326	reg2_reg[1]/C	oReg_reg[15]/D
0.205326	reg5_reg[1]/C	oReg_reg[15]/D
0.22282	reg1_reg[1]/C	oReg_reg[15]/D
0.238455	reg6_reg[1]/C	oReg_reg[15]/D
0.247519	reg0_reg[1]/C	oReg_reg[15]/D
0.249159	reg3_reg[1]/C	oReg_reg[15]/D
0.273771	reg4_reg[1]/C	oReg_reg[15]/D
0.295527	reg0_reg[0]/C	oReg_reg[15]/D
- Command Line:** The command line shows the following commands:


```
design_vision> write -hierarchy -format ddc -output /informatik2/tams/home/maeder/demo/opw.ddc
Writing ddc file '/informatik2/tams/home/maeder/demo/opw.ddc'.
design_vision> write -hierarchy -format verilog -output /informatik2/tams/home/maeder/demo/opwSyn.v
Writing verilog file '/informatik2/tams/home/maeder/demo/opwSyn.v'.
```
- Command Line Input:** The command line input field contains the command: `write_sdf opwSyn.sdf`

18. ... fertig, Programm beenden:

The screenshot shows the Design Vision interface with the following components:

- File Menu:** Opened, showing options like Read..., Analyze..., Elaborate..., Save, and Exit.
- HistList.3 - HistList.3 Path Slack:** A histogram showing the distribution of path slacks. The x-axis is 'Slack' (0, 1, 2) and the y-axis is 'Number of Paths'. The 'Worst' slack is 0.0296326 and the 'Best' slack is 2.93776.
- Table:** A table listing path slack data.

Slack	From	To
0.0296326	reg7_reg[0]/C	oReg_reg[15]/D
0.158434	reg3_reg[0]/C	oReg_reg[15]/D
0.164326	reg2_reg[1]/C	oReg_reg[15]/D
0.205326	reg5_reg[1]/C	oReg_reg[15]/D
0.22282	reg1_reg[1]/C	oReg_reg[15]/D
0.238455	reg6_reg[1]/C	oReg_reg[15]/D
0.247519	reg0_reg[1]/C	oReg_reg[15]/D
0.249159	reg3_reg[1]/C	oReg_reg[15]/D
0.273771	reg4_reg[1]/C	oReg_reg[15]/D
0.295527	reg0_reg[0]/C	oReg_reg[15]/D
- Timing Diagram:** A complex timing diagram showing multiple signals in blue and white.
- Terminal Window:** Shows the following output:


```

            Writing verilog file '/informatik2/tams/home/maeder/demo/opwSyn.v'.
            design_vision> write_sdf opwSyn.sdf
            Information: Annotated 'cell' delays are assumed to include load delay. (UID-282)
            Information: Writing timing information to file '/informatik2/tams/home/maeder/demo/opwSyn.sdf'. (WT-3)
            design_vision>
            
```

