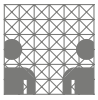
VHDL-Synthese

Werkzeuge : Synopsys Design-Vision
Design-Kits : AMS Hit-Kit
edaSetup : `syn ams`

Andreas Mäder



Diese Anleitung beschreibt die Synthese mit den SYNOPSYS Werkzeugen: 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: „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 SYNOPSYS „HDL Compiler for VHDL User Guide“ beschrieben.

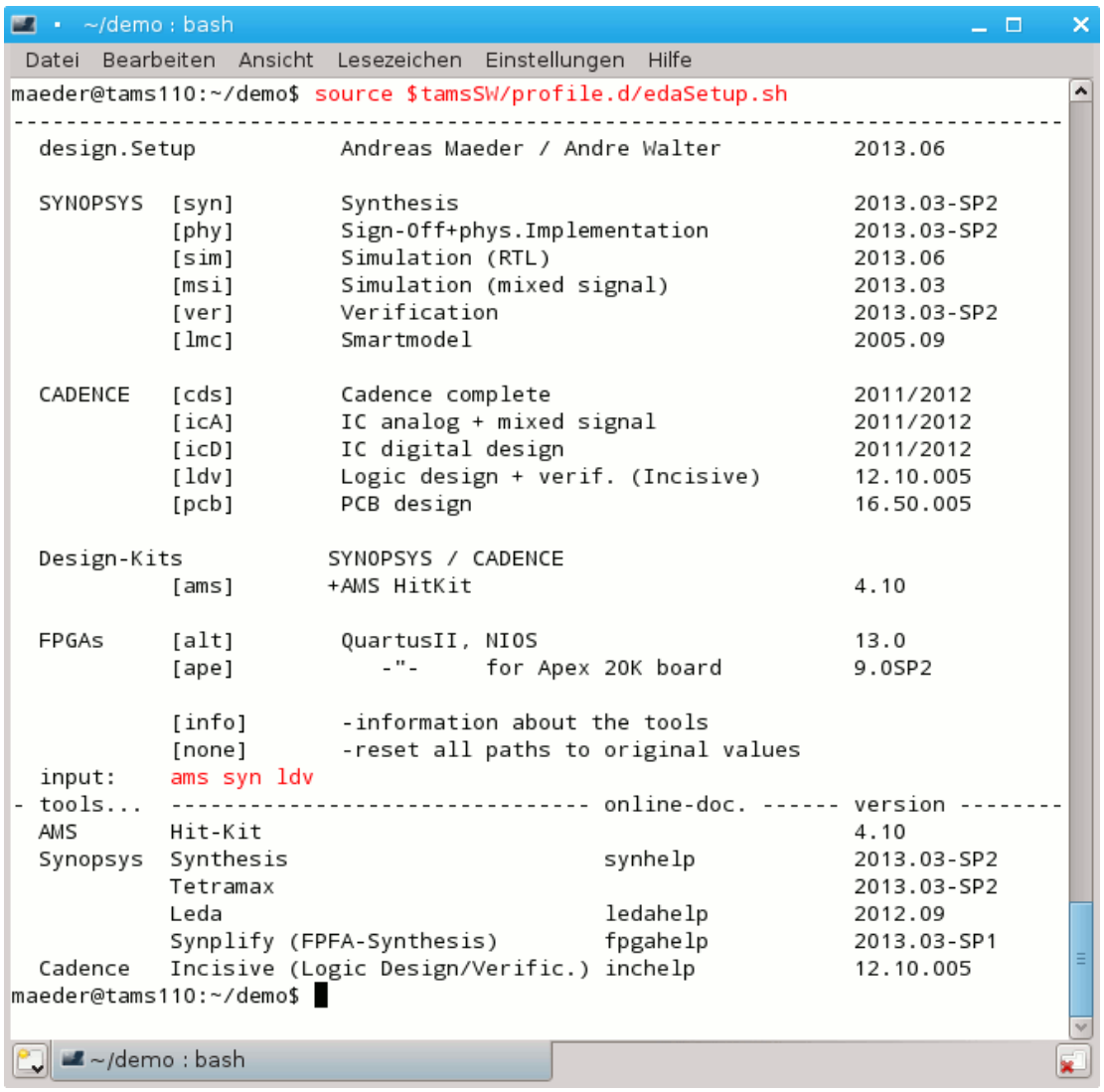
Der Syntheseprozess lässt sich in folgende Schritte unterteilen:

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

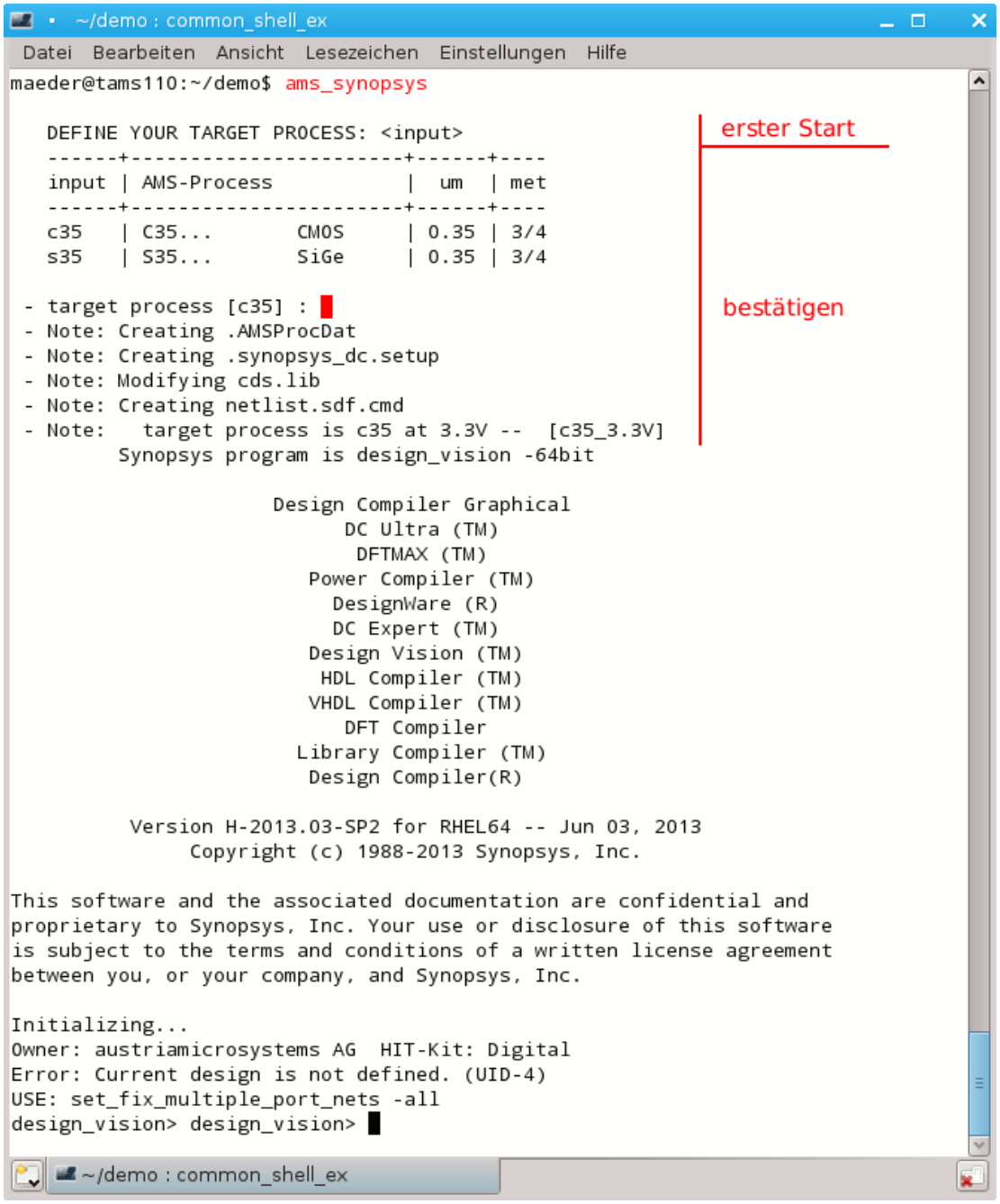
Arbeitsschritte

VHDL Einlesen

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

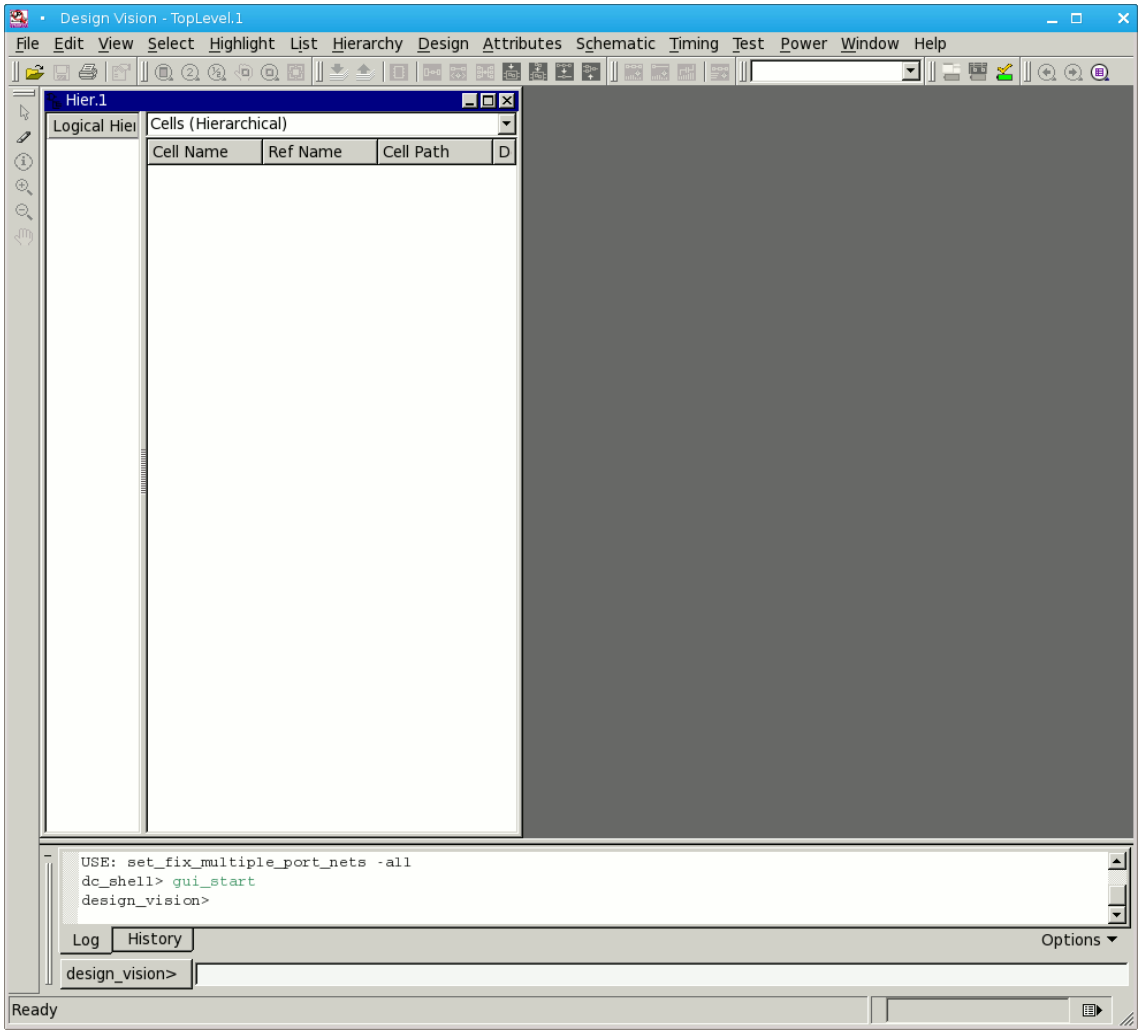


2. Start des Systems:



Der Befehl startet ein Skript, dass beim ersten Aufruf nach dem AMS-Prozess fragt und die passenden Initialisierungsdateien für die Synthese und nachfolgende Schritte erzeugt. Für die $0,35\,\mu m$ 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.

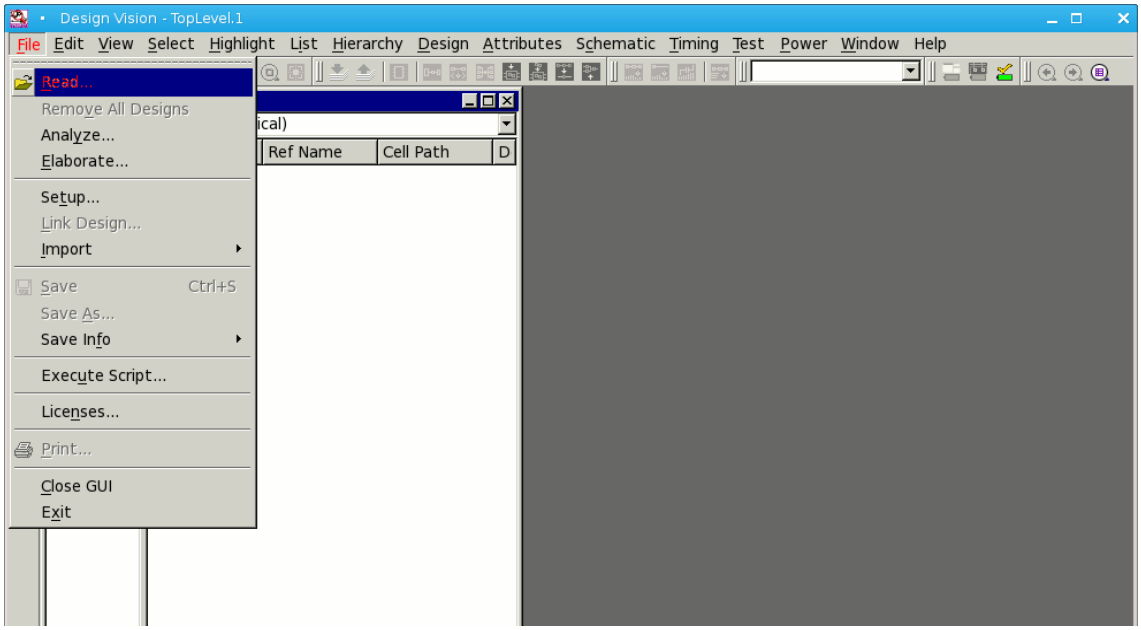
3. Einlesen der Quelldateien
   Bei der Aufbereitung der Daten für die Synthese werden zwei Schritte unterschieden: Bei der *Analyse* wird der VHDL-Code auf die Synthetisierbarkeit untersucht und es werden systeminterne Templatedateien erzeugt. Die anschließende *Elaboration* generiert dann daraus die 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:

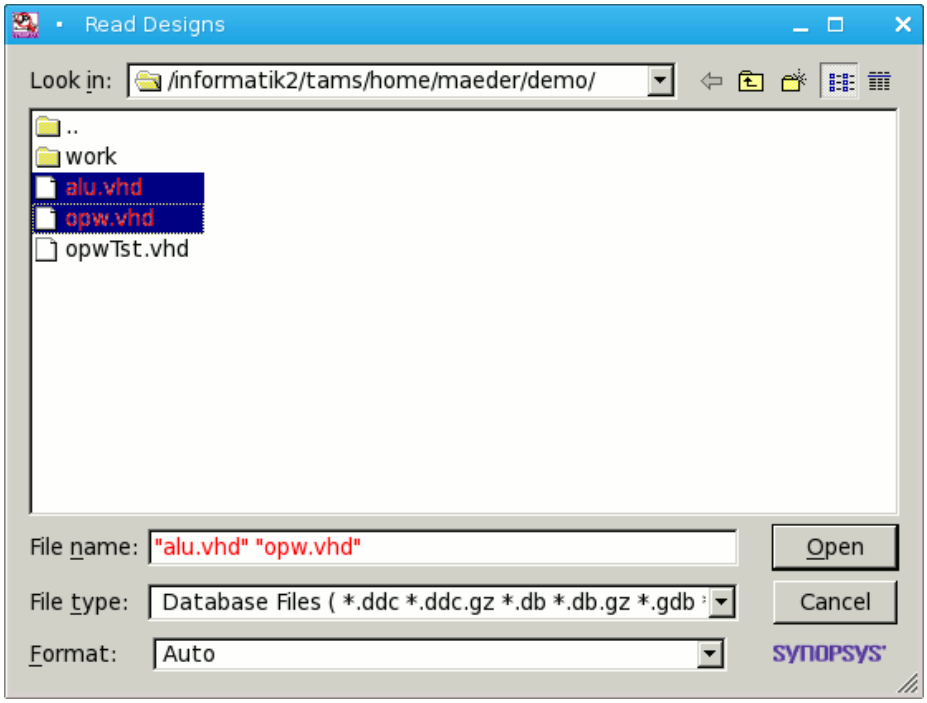
   - zu einer Entity mehrere Architekturen existieren und man explizit eine auswählen möchte — Ansonsten gilt die zeitliche Reihenfolge bei der Codeanalyse.
   - eine Entity `generic`-Parameter besitzt an die Werte übergeben werden sollen.

Read — Analyse und Elaboration in einem Schritt

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

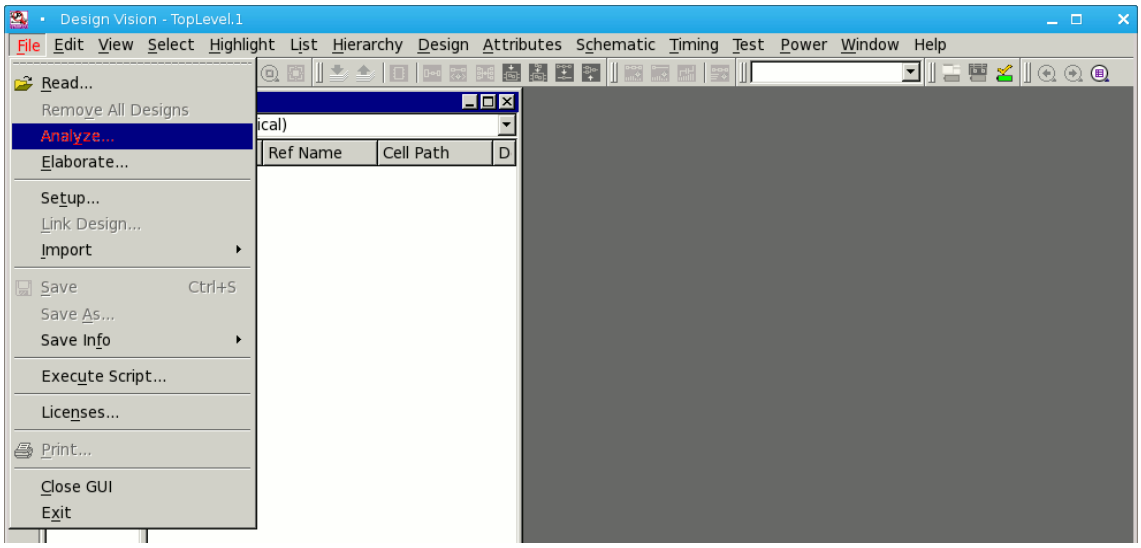


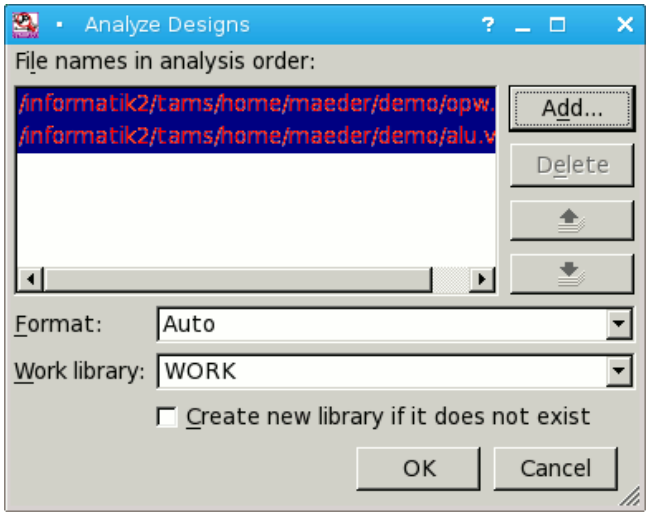
Die Abarbeitungsreihenfolge der einzelnen Dateien ist beliebig, es muss nur gewährleistet sein, dass vor der Synthese alle Quelldateien in der Datenbasis vorhanden sind:

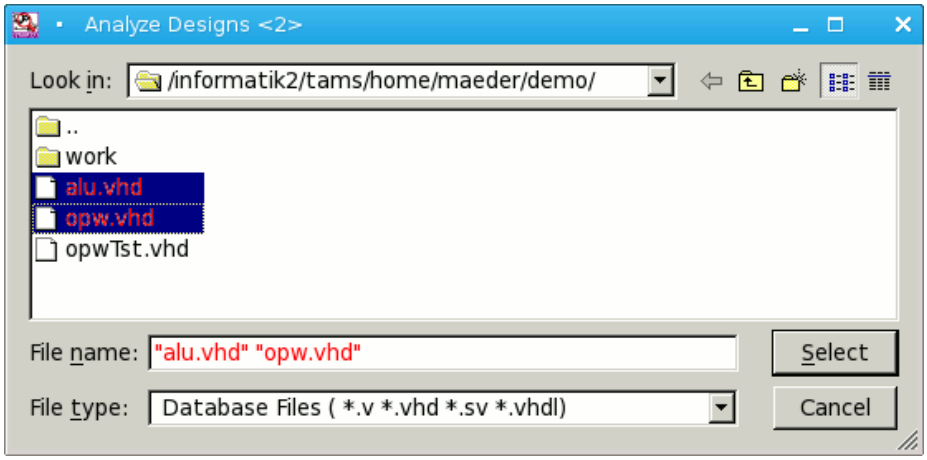


Analyse und Elaboration getrennt

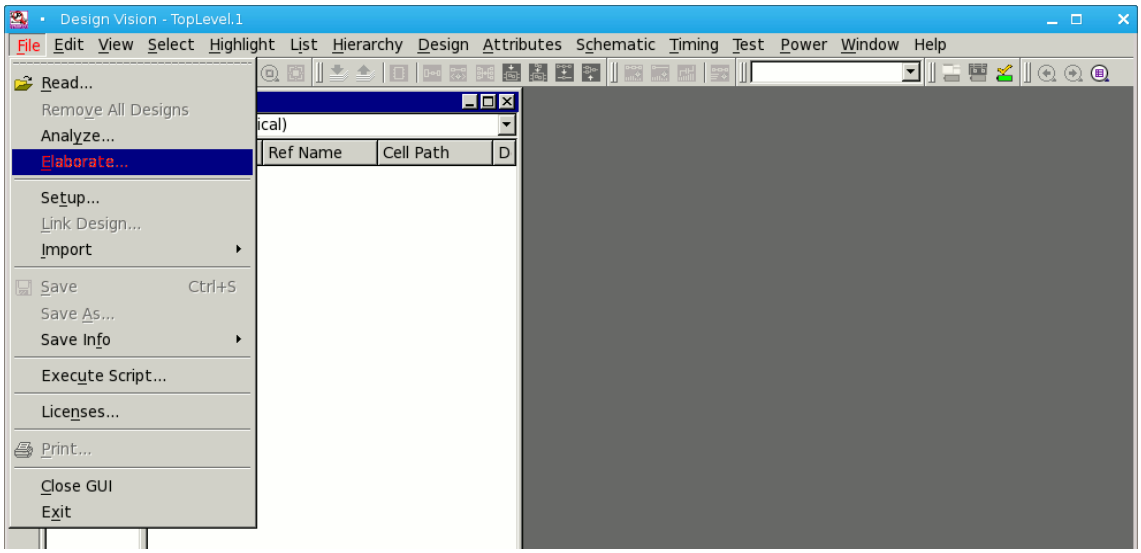
Wenn `generic`-Parameter an VHDL-Entities zu übergeben sind oder alternative Architekturen zur Auswahl stehen, dann müssen Analyse und Elaboration getrennt werden. Zuerst werden alle Dateien der Hierarchie (in beliebiger Reihenfolge) analysiert:

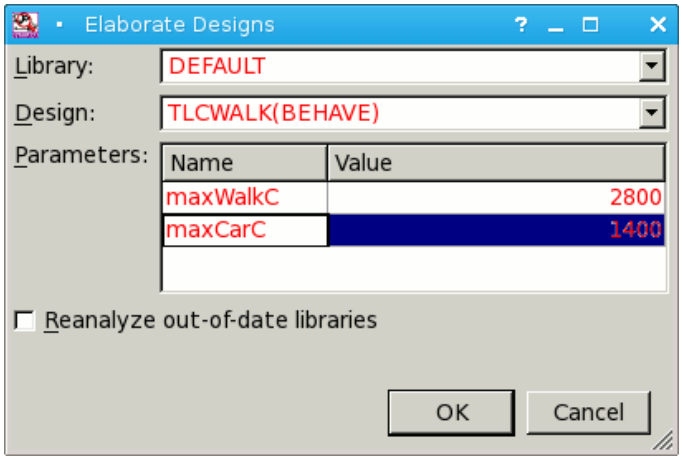






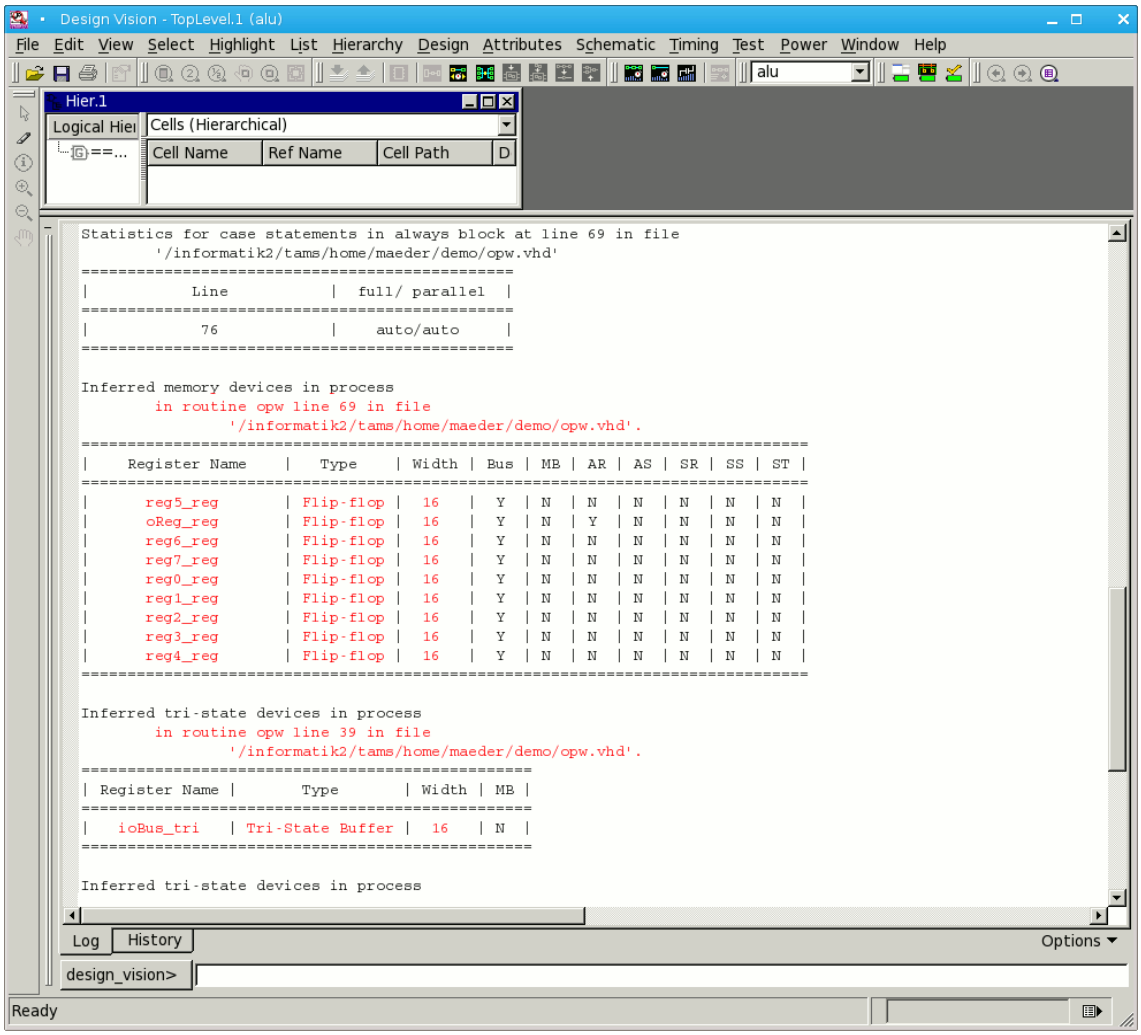
Nach Abschluss der Analysephase wird die Hierarchie, ausgehend von dem gewählten Entwurf, durch Elaboration abgearbeitet. Architekturen werden über ihren Bezeichner, nach folgendem Schema ⟨*entity*⟩(⟨*architecture*⟩), unterschieden. Das Beispiel zeigt auch, wie `generic`-Parameter (Beispiel: Ampelschaltung) 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 Tristate-Treiber durch welche Zeilen des VHDL-Codes impliziert werden. Die Ausgabe sieht dabei folgendermaßen aus:



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 Wertebereichseinschränkung 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

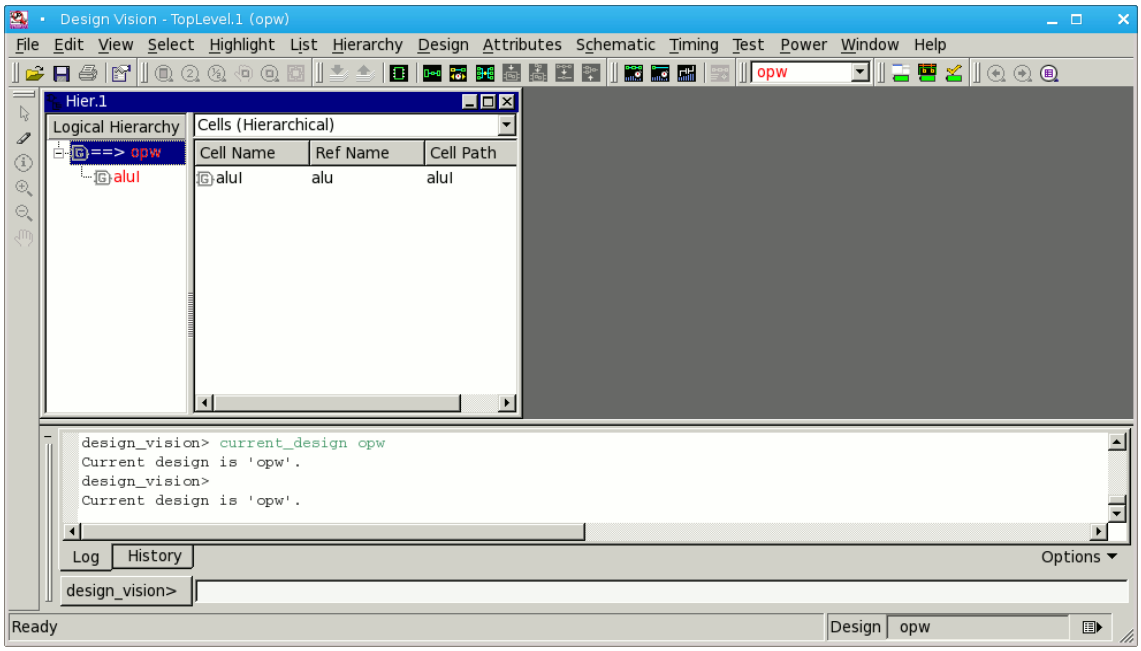
oder eine Zuweisung die ggf. überschrieben wird, muss als *„Default“* im sequentiellen Prozess *vor* der Verzweigung stehen.

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

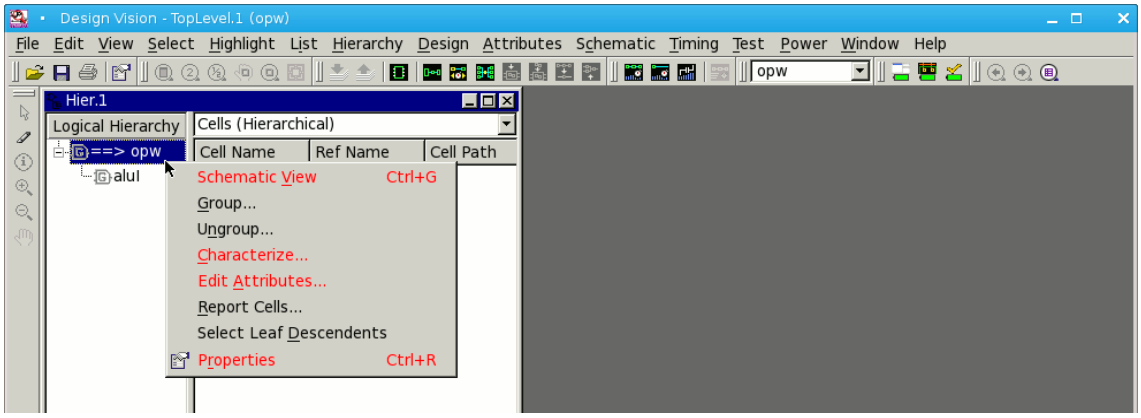
Anmerkung: bei den synchronen Entwürfen die wir beschreiben, kann man davon ausgehen, dass Latches nicht vorkommen (sollten) und immer *„Falsch“* sind!

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:

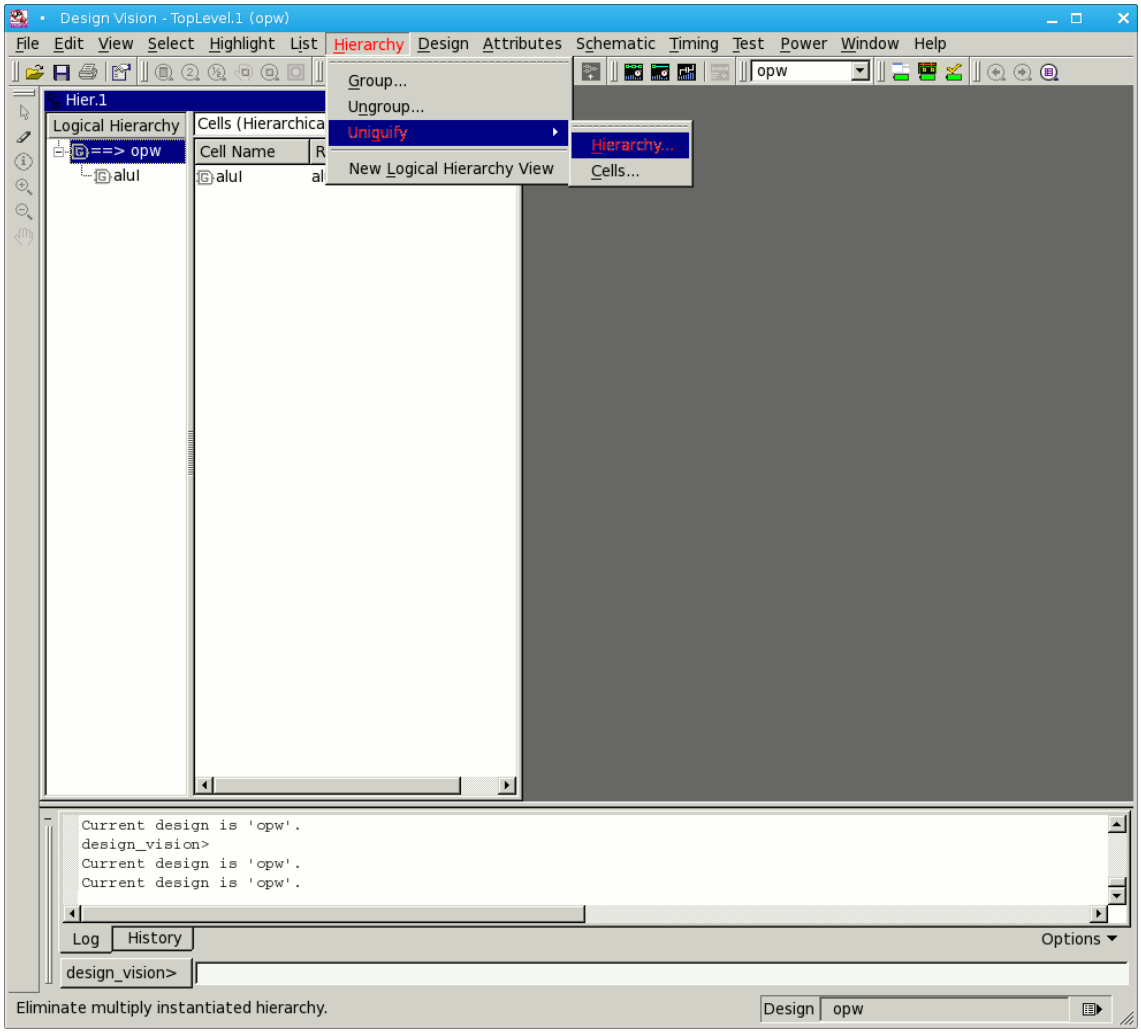


Achtung: Die folgenden Schritte sind nur notwendig, wenn innerhalb der Hierarchie Entities mehrfach instanziiert werden, andernfalls kann direkt mit der Eingabe der Syntheserandbedingungen ab Punkt 7 (Seite 13) begonnen werden!

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 Multiplizierer erhalten aus dem übergeordneten Design eines digitalen Filters jeweils einen konstanten Faktor (konstante Input-Ports).

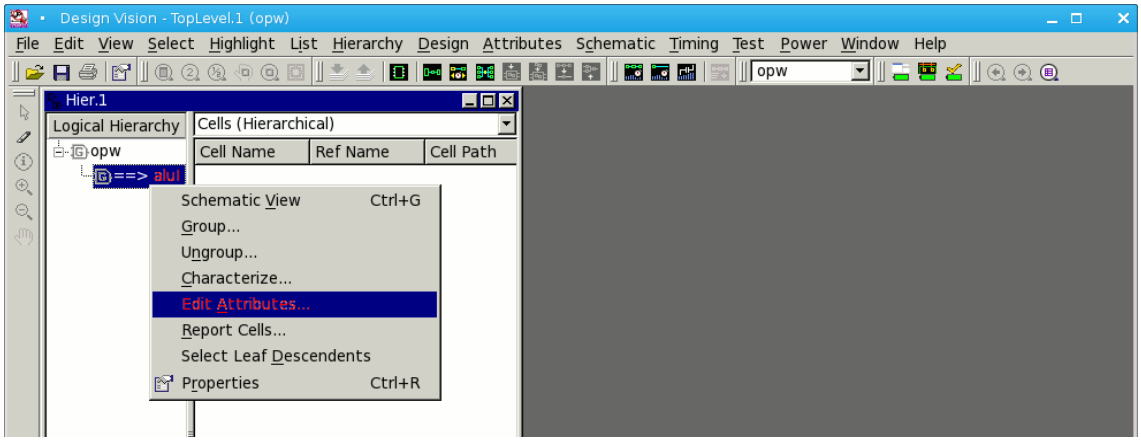
   Ausgehend von der top-level Entity wird die Hierarchie durchlaufen und mehrfach vorhandene Elemente werden unterscheidbar gemacht, um sie bei der später folgenden Synthese individuell bearbeiten zu können:

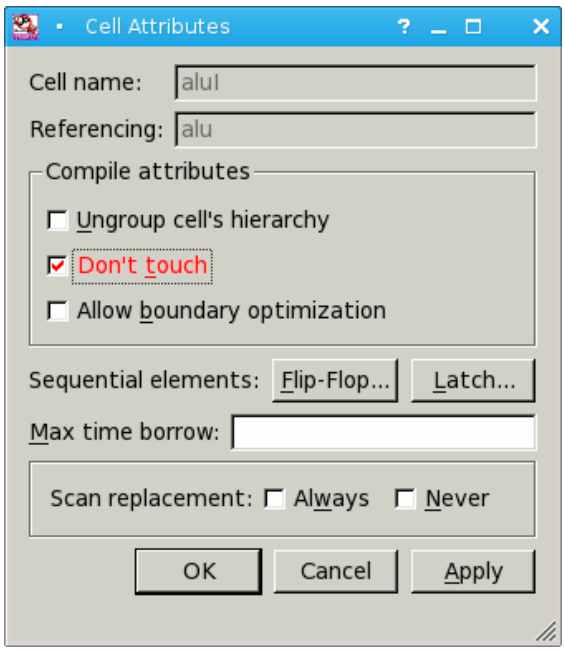


6. (Instanzen mehrmals vorhanden) Gleichartige Behandlung / mehrere Syntheseschritte
   Werden mehrfach referenzierte Elemente der Hierarchie immer in *gleicher Weise* benutzt, so sollten sie (aus Effizienzgründen) nur einmal synthetisiert werden. Beispiele: identische Recheneinheiten eines systolischen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





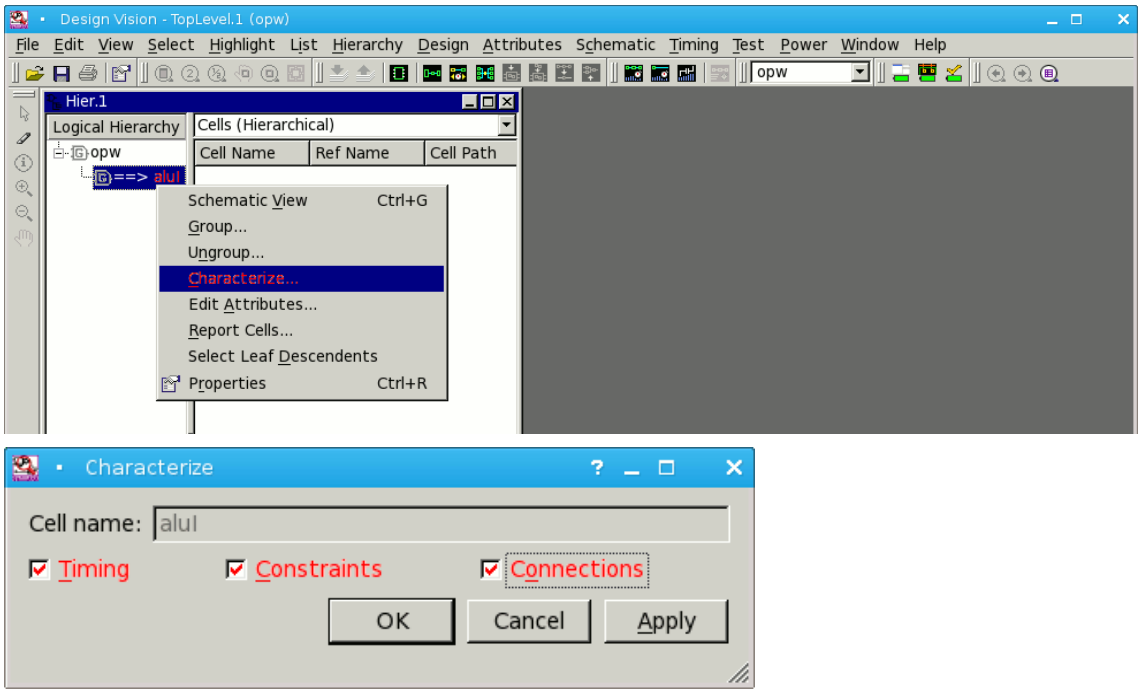
Anschließend wird eine top-down Synthese, wie ab Seite 13, beschrieben duchgeführt:

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).

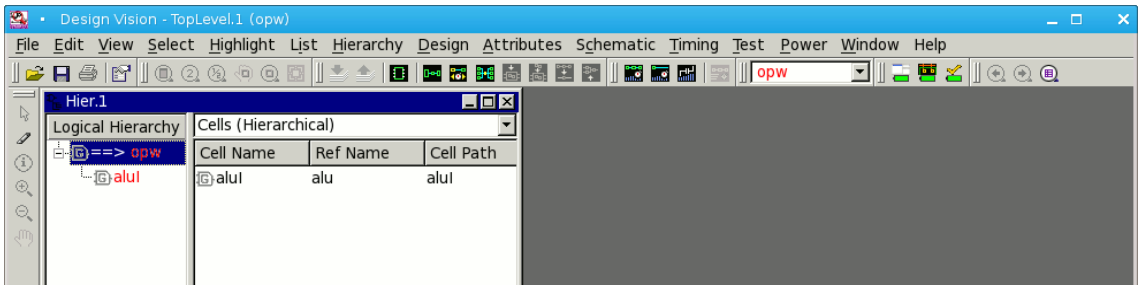
Synthesebedingungen festlegen

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 SYNOPSYS Dokumentation verwiesen. Im folgenden werden drei „einfache“ Möglichkeiten vorgestellt, die auch miteinander beliebig kombiniert werden können.

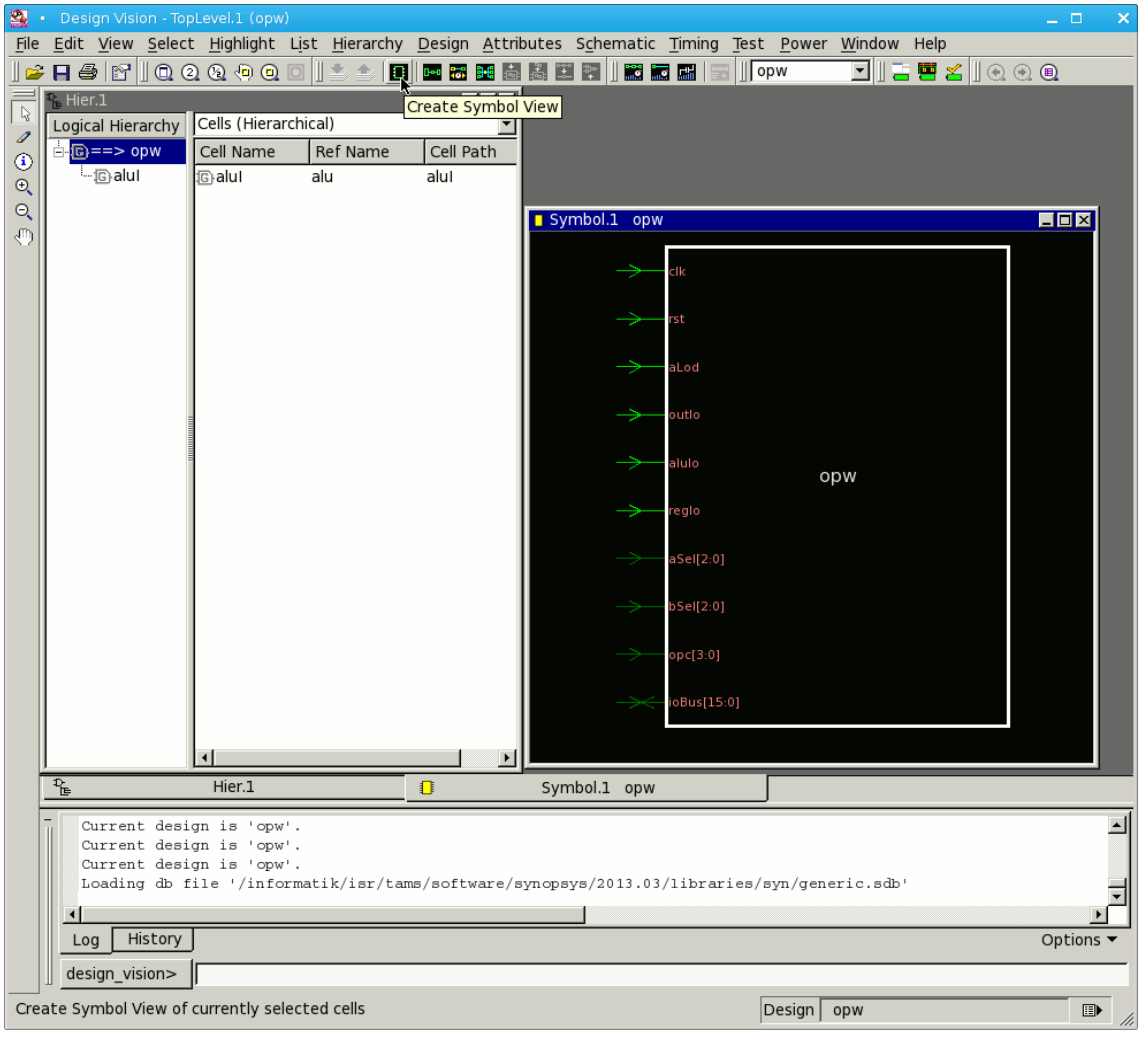
Tipp: für „optimale“ Syntheseergebnisse ist es besser mit realistischen Werten für Fläche bzw. Geschwindigkeit zu synthetisieren und diese Randbedingungen über mehrere Syntheseläufe zu verschärfen.

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



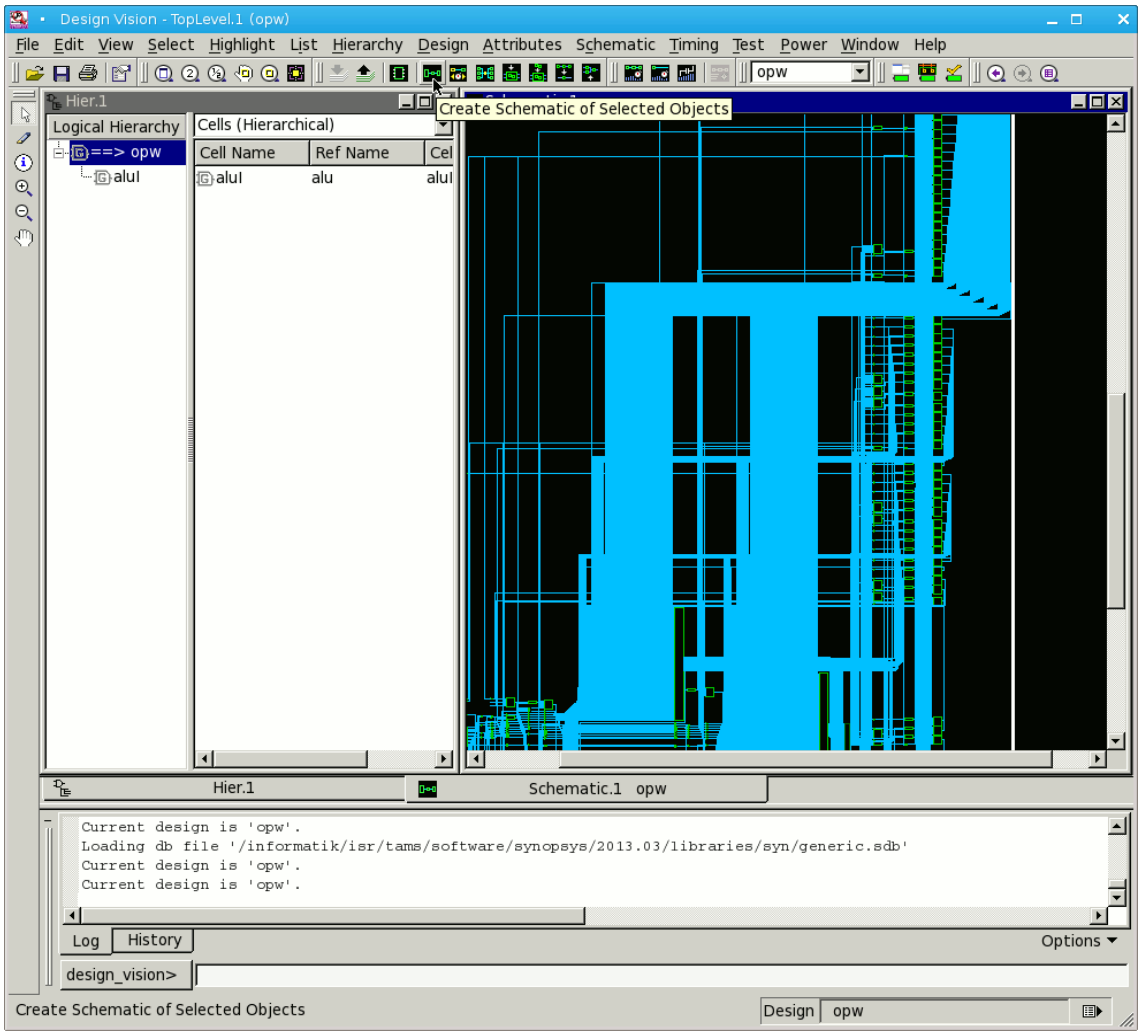
Sollen später Taktfrequenzen und Zeitbedingungen festgelegt werden, dann sollte man ein Symbol und ein Schematic erzeugen, um dort durch Selektion mit der Maus Signale und Ports auszuwählen.

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.

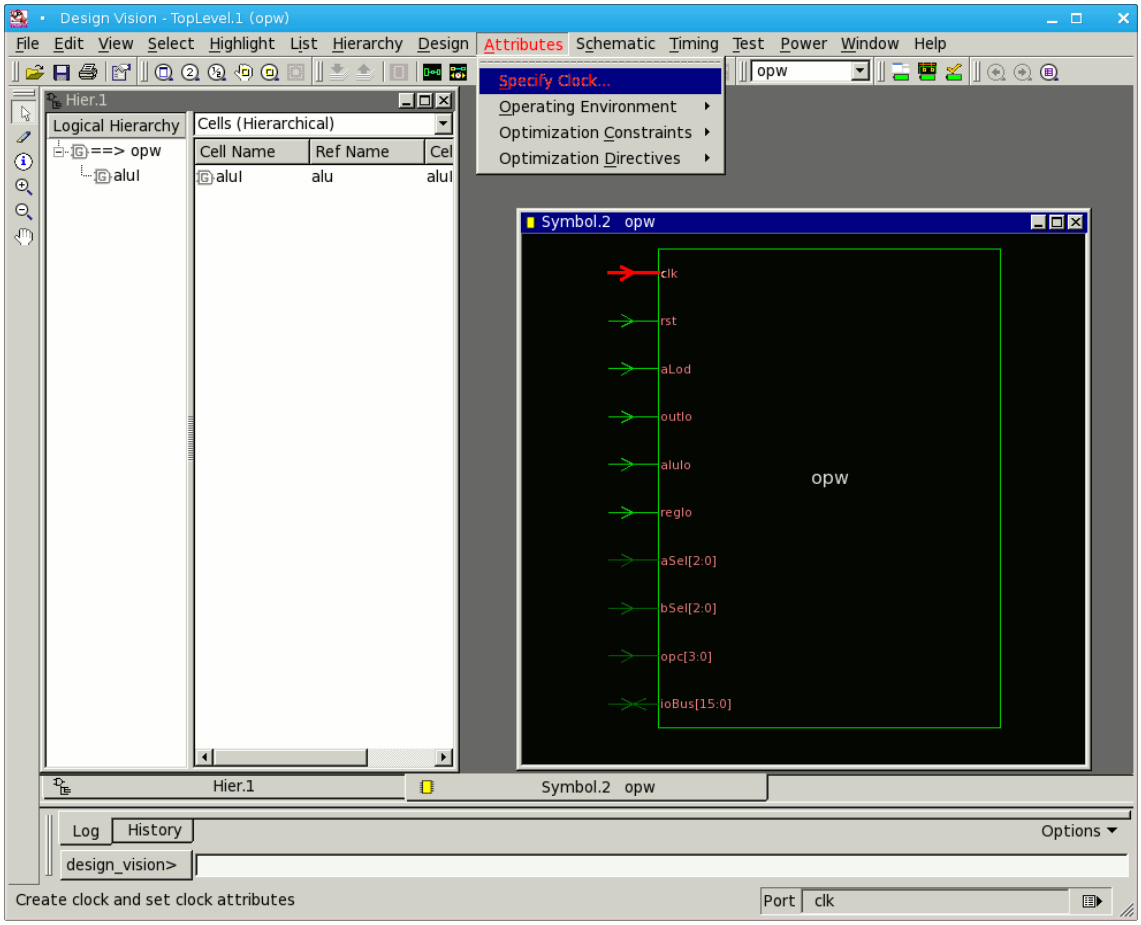


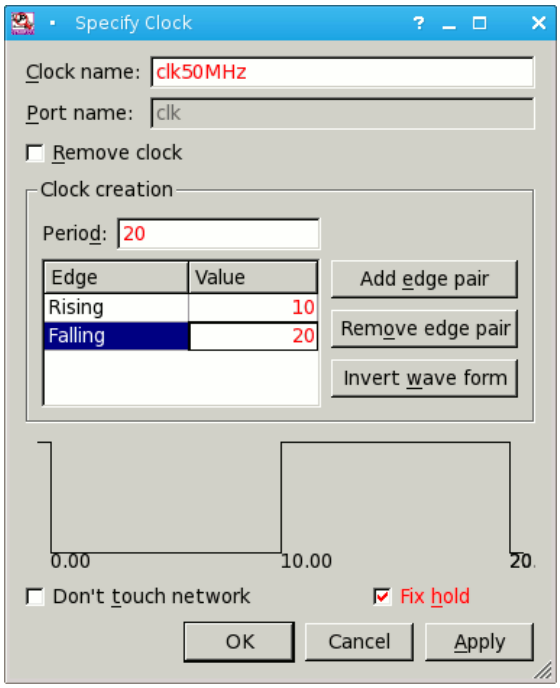
8. Taktfrequenz(en) festlegen
   In der Regel enthalten die Schaltungen Taktleitungen, deren Taktschema (Frequenz, Phasenlage zueinander) für die Synthese unbedingt angeben werden sollte. Bei Entwürfen ohne explizite Taktleitungen (Schaltnetzen) kann man stattdessen Verzögerungszeiten zwischen Ein- und 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.

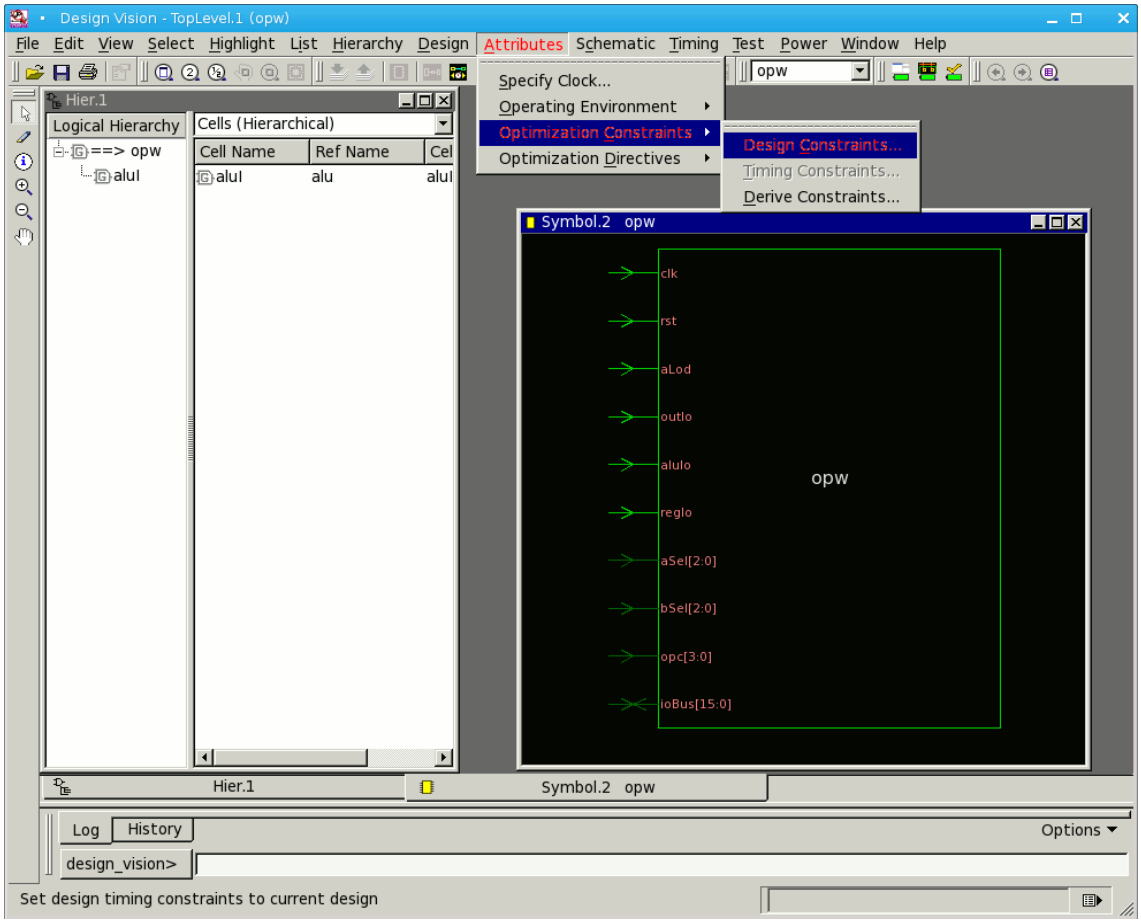
[1] hier nicht weiter beschrieben

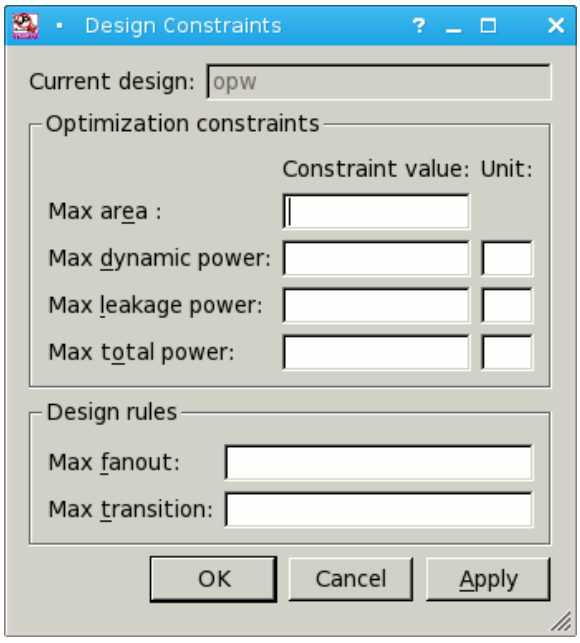
Dazu sind in dem Symbol des top-level Entwurfs die Taktleitungen zu selektieren und dann das Taktschema zu definieren:



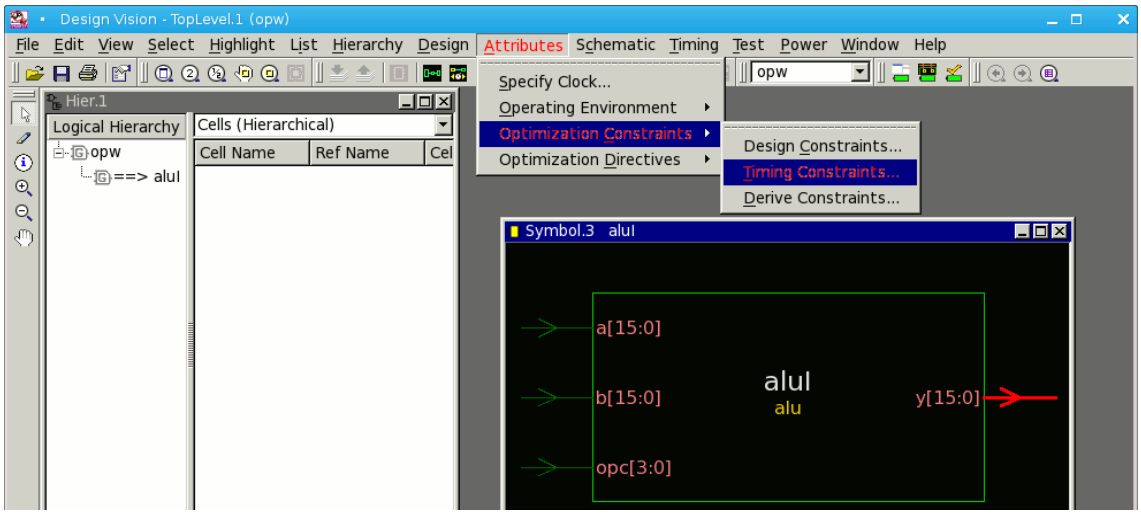


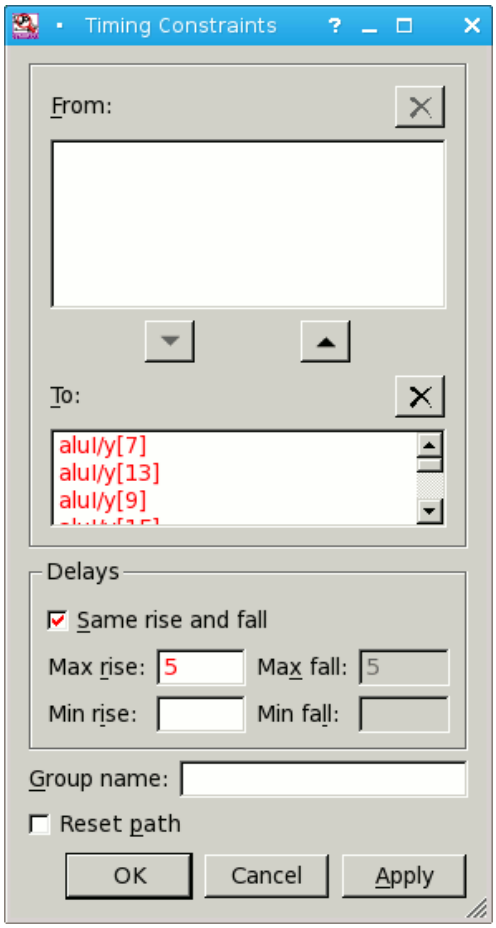
9. (optional) Flächenvorgaben — werden in der Regel nicht weiter benötigt, da als Voreinstellung möglichst kleine, kompakte Netzlisten synthetisiert werden:



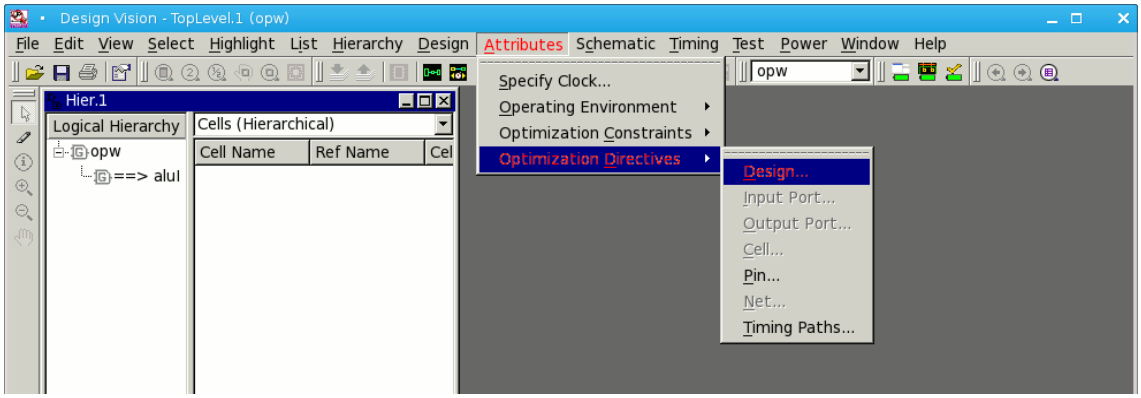


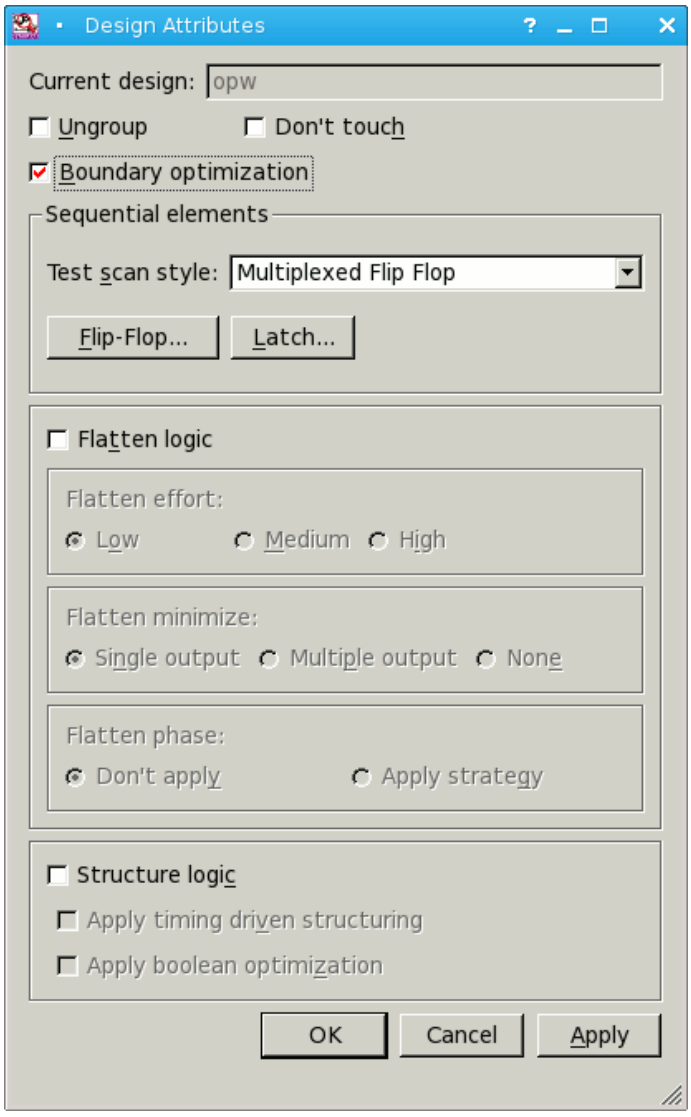
10. (optional) Timingvorgaben — werden in der Regel nicht benötigt, da das Zeitverhalten über die Taktung (s.o.) definiert ist. Das Timing kann zwischen beliebigen Stellen der Netzliste explizit angegeben werden; dabei sind sowohl minimale als auch maximale Laufzeiten möglich. Die Anfangs- oder Endpunkte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



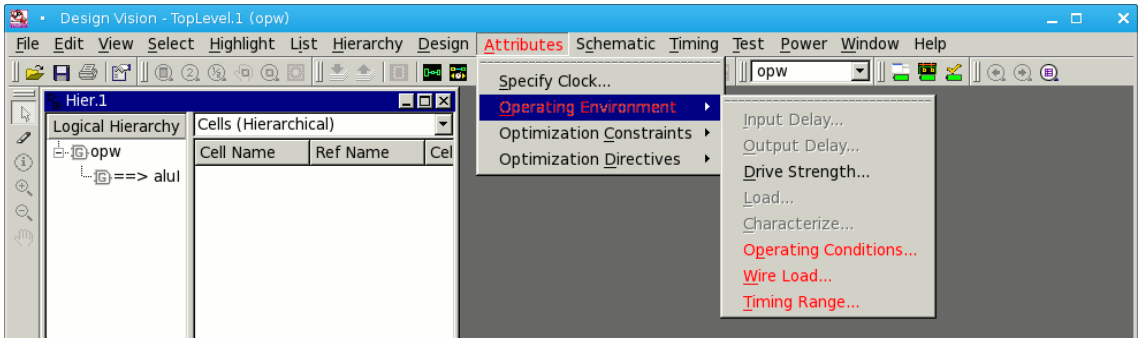


11. (optional) Syntheseattribute — sind sinnvoll voreingestellt und sollten nur in Ausnahmefällen (siehe SYNOPSYS Dokumentation) geändert werden:

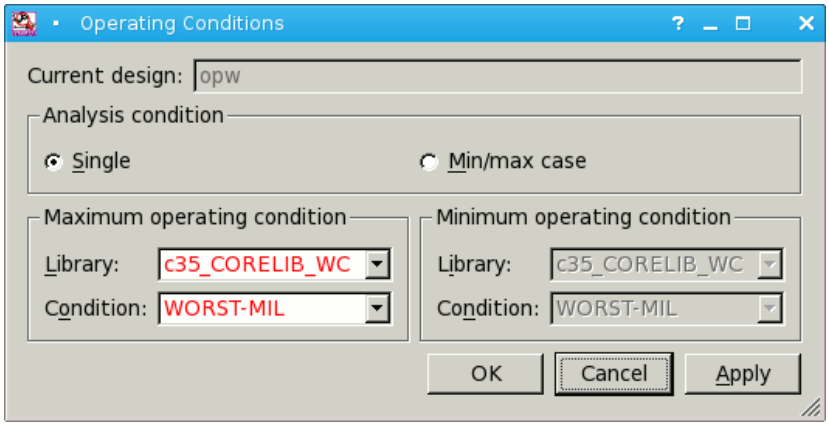


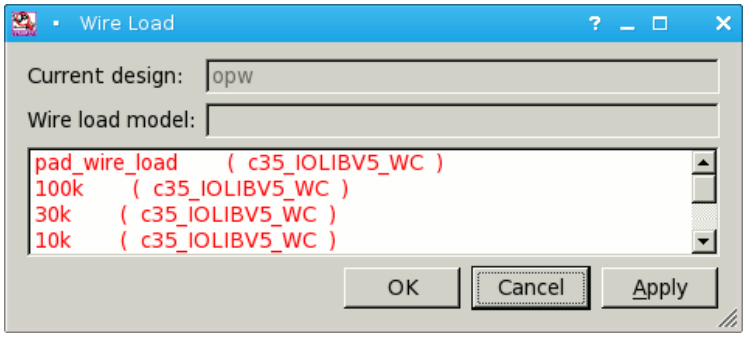


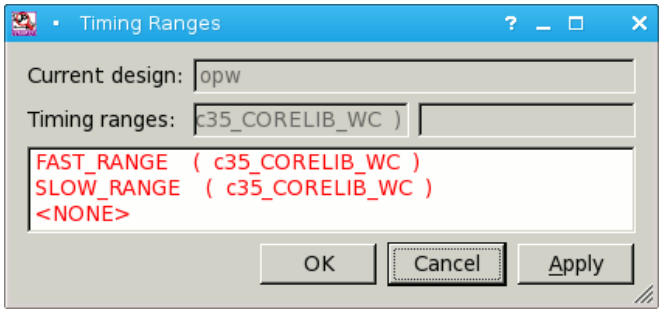
12. (optional) Operationsbedingungen einstellen
Für die später folgenden Schritte (Synthese und Timinganalyse) können die Zeitmodelle der Gatter und externe Lasten festgelegt werden:



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, `IND`ustrial und `MIL`itary.



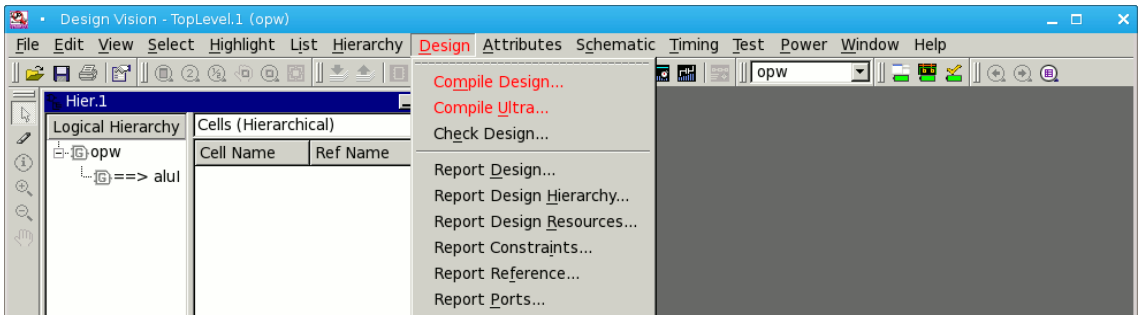


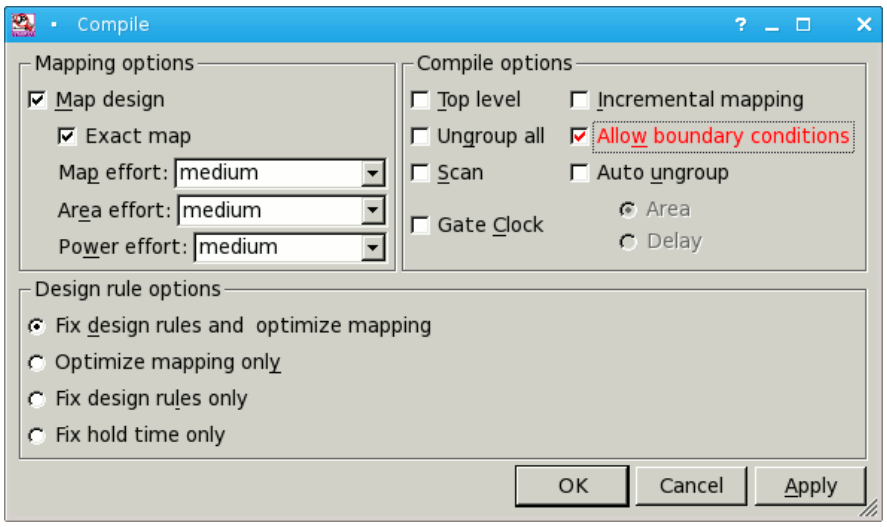


Synthese der Gatternetzliste

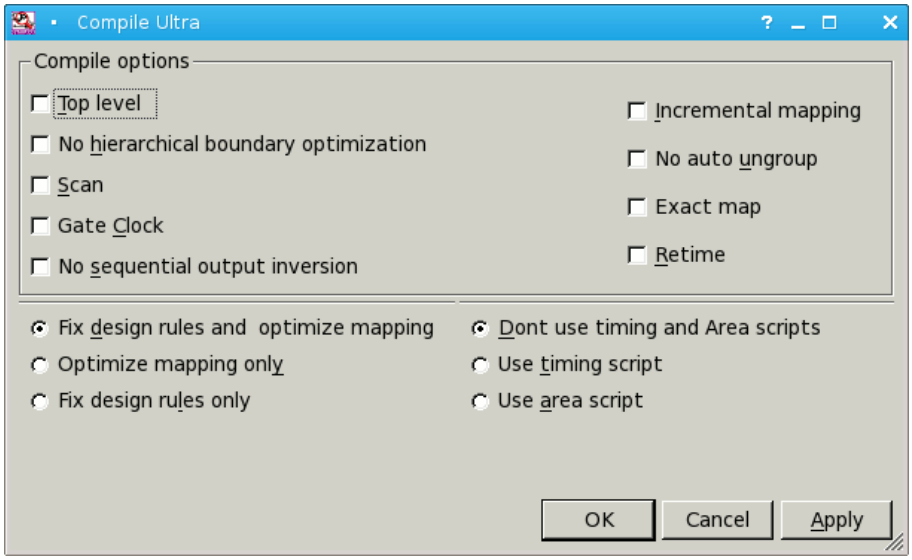
13. Hardwaresynthese und Abbildung auf die Zellbibliothek
Ausgehend von dem aktuellen (Teil-) Entwurf, durchläuft die Synthese die gesamte Hierarchie — die Ausnahme bilden *Don't touch*-Attribute, siehe „Behandlung der Hierarchie“, ab Seite 9.

Für die Synthese stehen zwei verschiedene Programmkomponenten zur Verfügung:

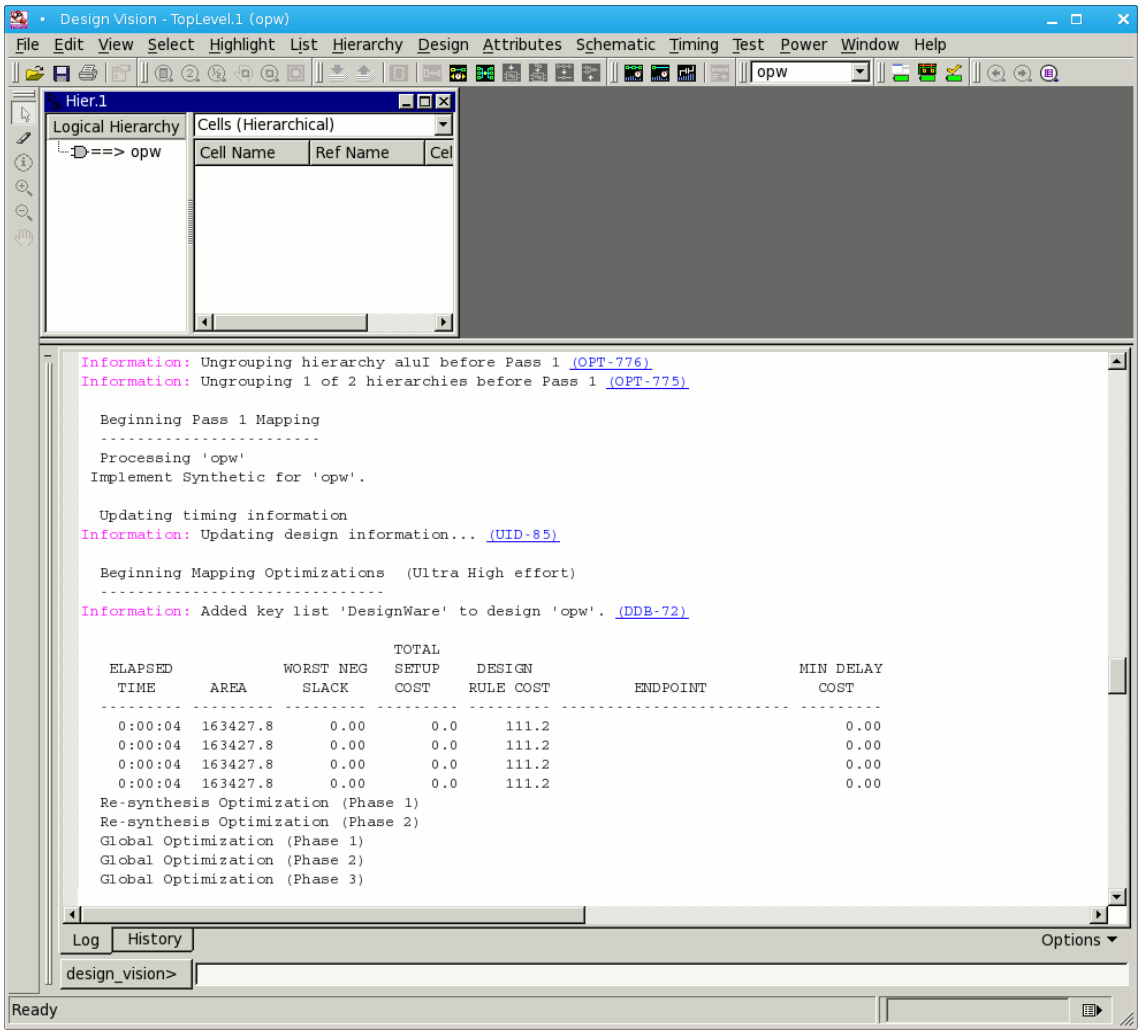




oder



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



14. Bewertung der Syntheseergebnisse
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.

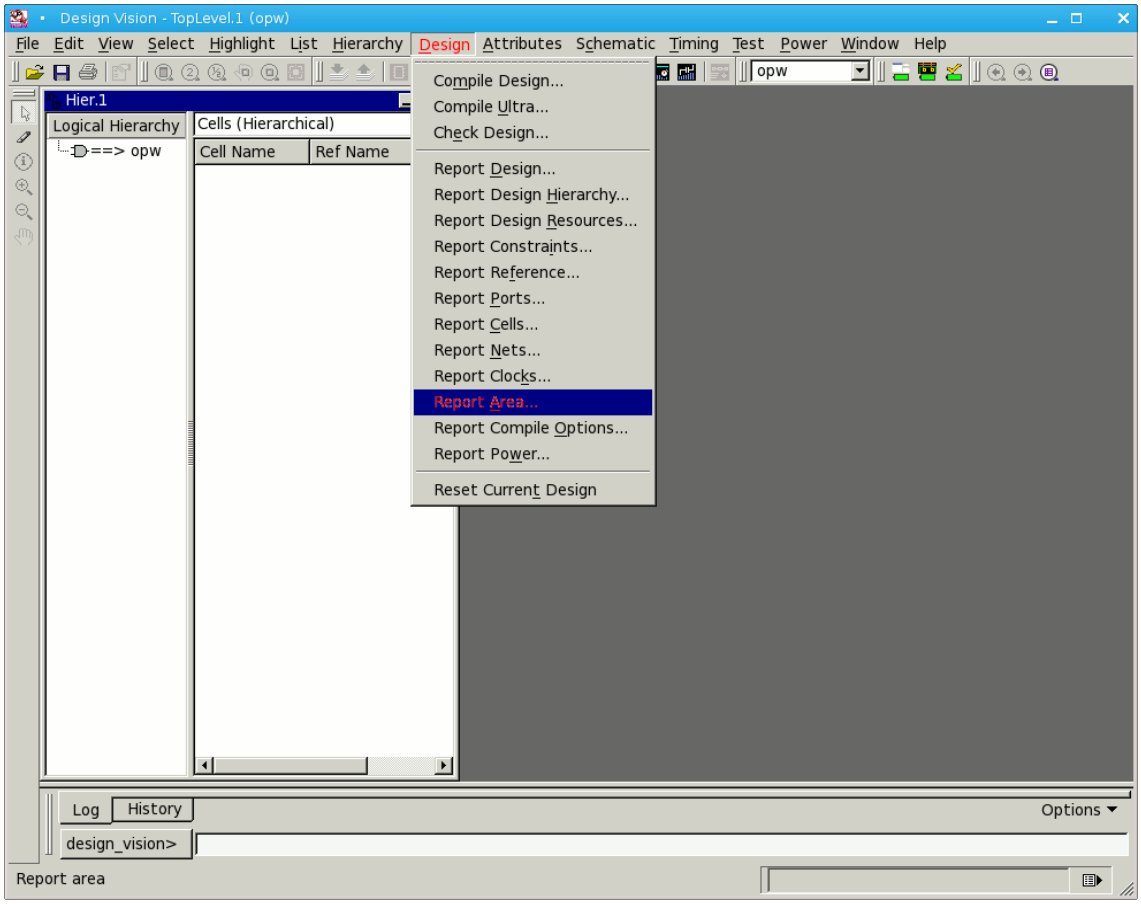
Hier werden nur einige der umfangreichen Analysemöglichkeiten des Synthesewerkzeugs vorgestellt. Die bei der Timinganalyse ausgegebene Information bezieht sich immer auf das unter Punkt 12, Seite 19 festgelegte Zeitmodell. Durch Auswahl anderer Operationsbedingungen können „worst-case“ und „best-case“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

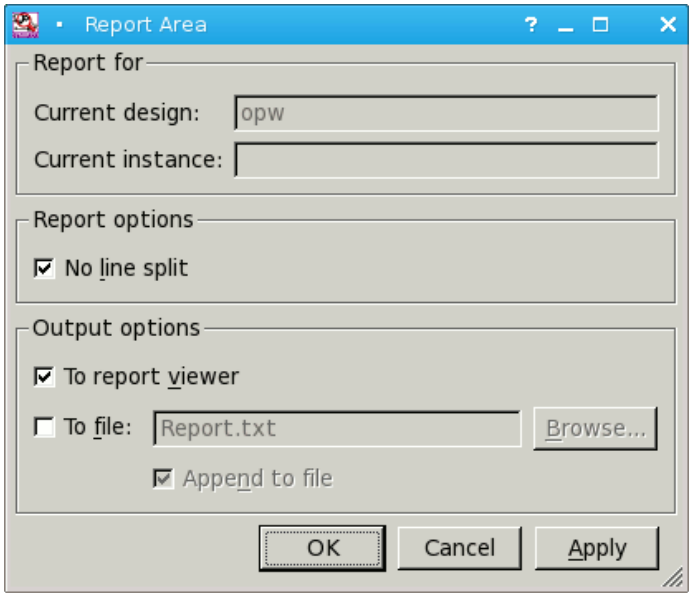
Kontrolle des Schematic und der Hierarchie

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

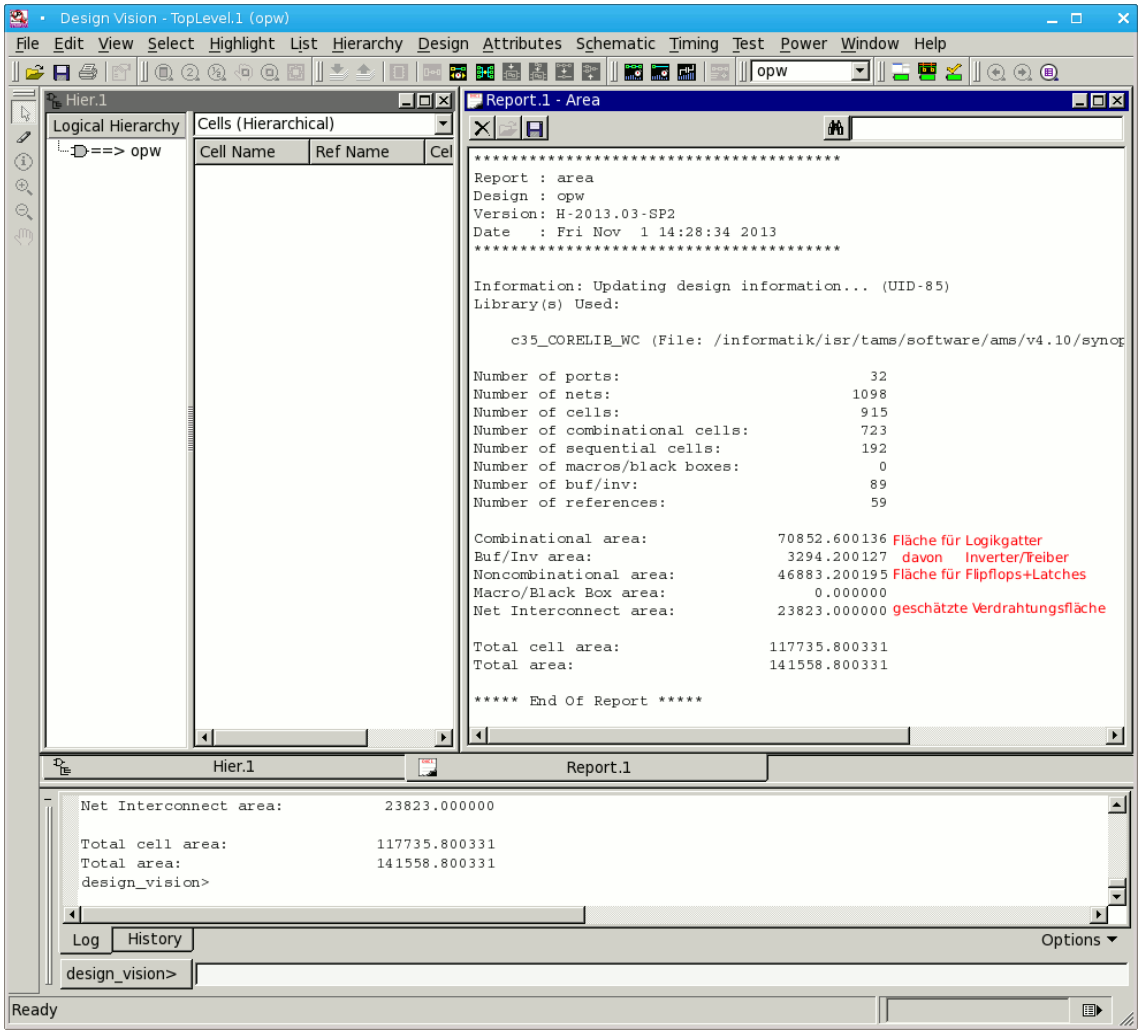
Ausgabe von Statistiken — Flächenbedarf

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



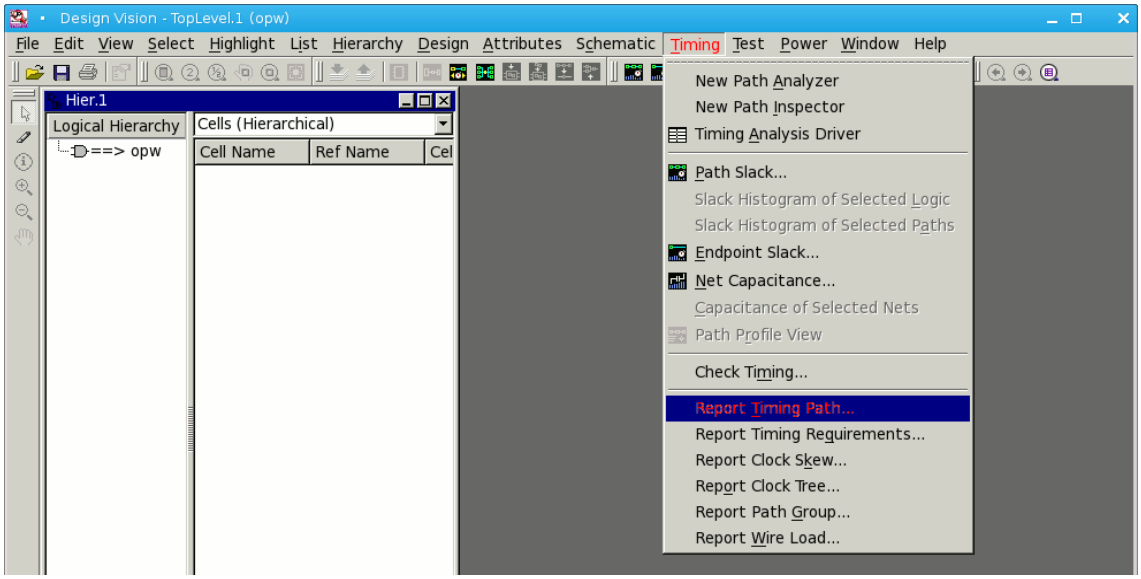


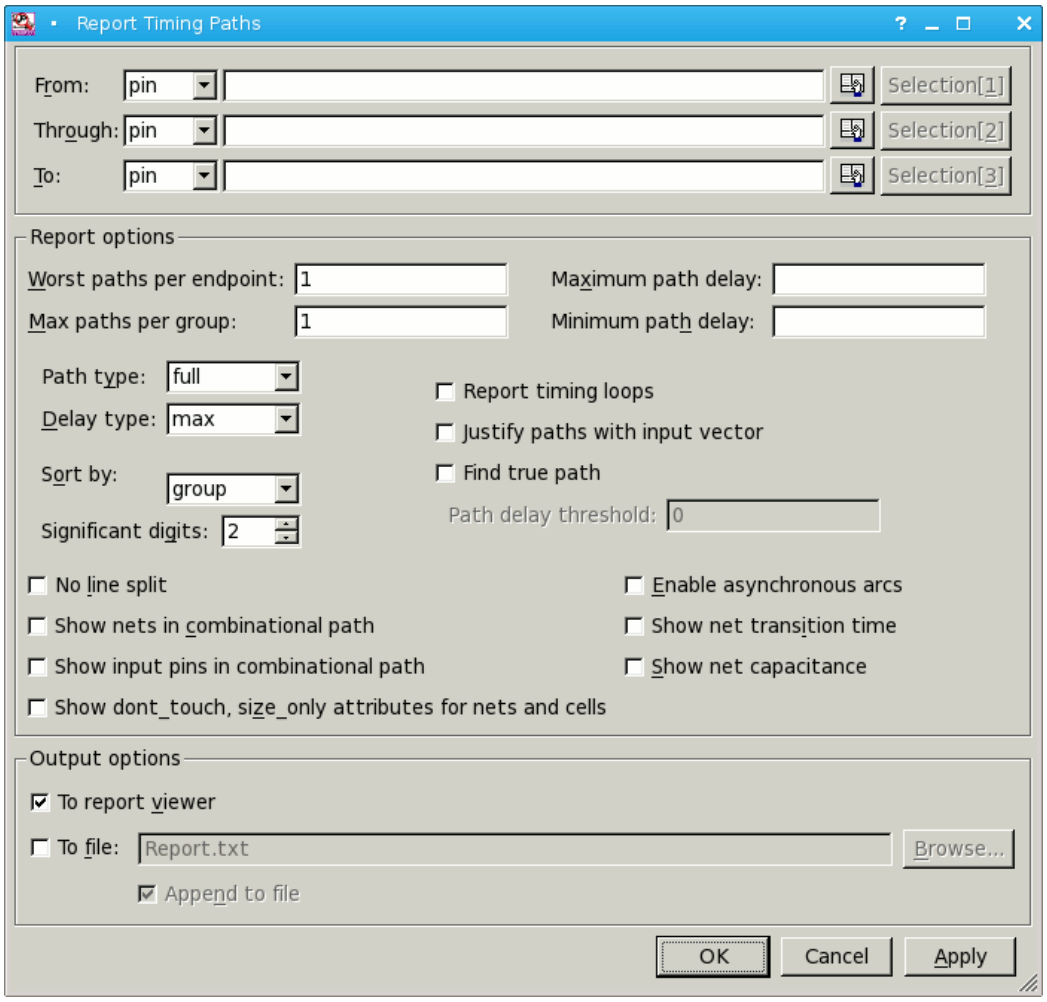
Die Ausgabe, hier in der Log-Datei, enthält folgende Flächenmaße $[\mu m^2]$:



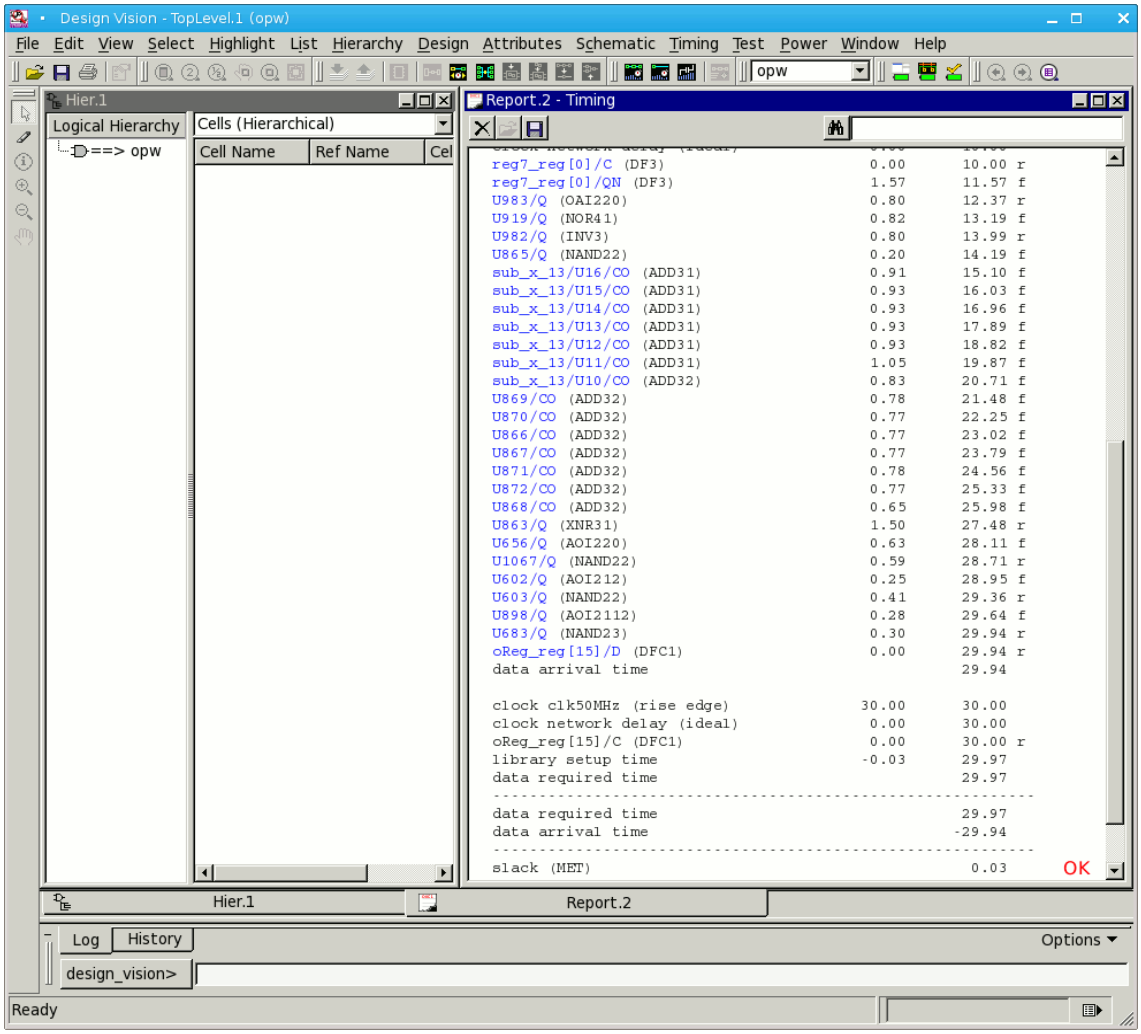
Timing der Schaltung

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

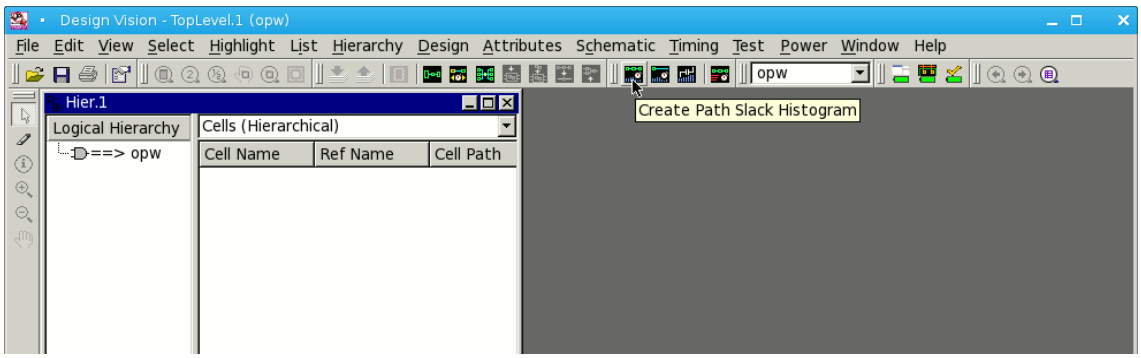


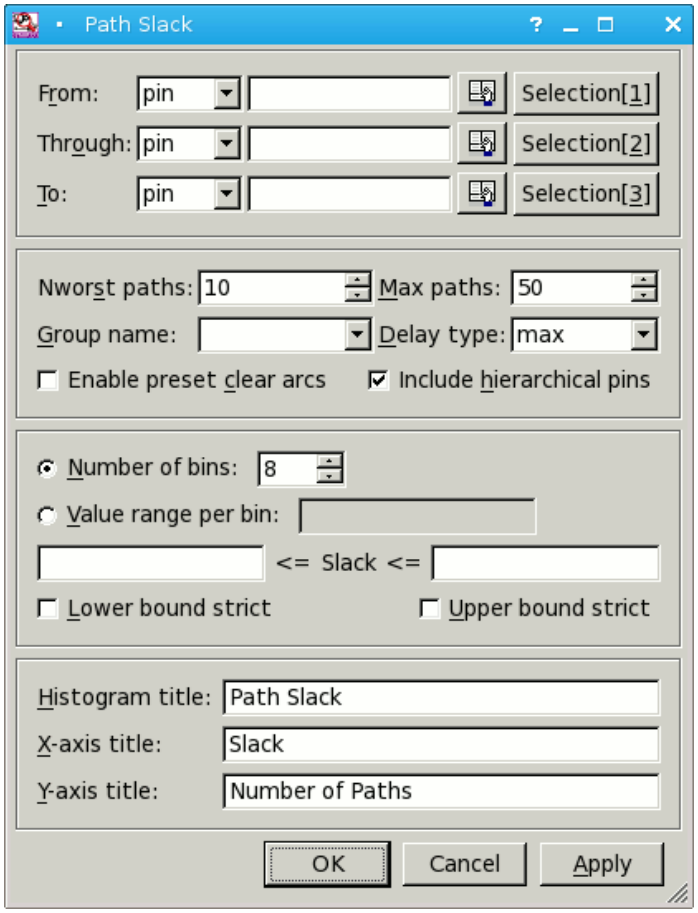


Die Ausgabe des Timing-Reports bezieht sich auf den kritischen (längsten) Pfad. Das Timing wurde hier, was auch sonst meistens der Fall sein dürfte, durch den Chiptakt spezifiziert:

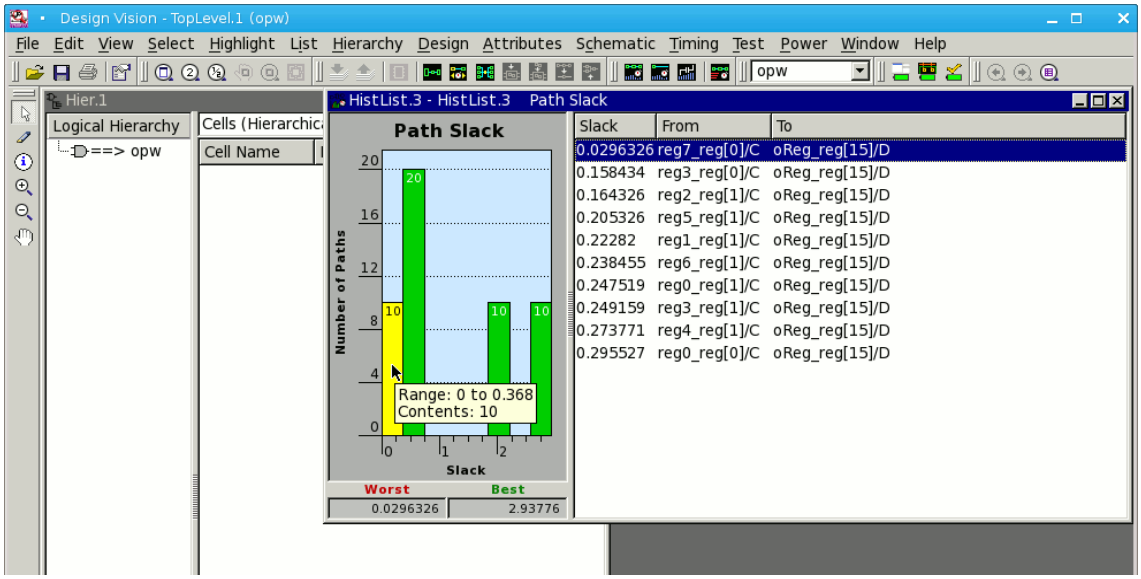


Für alle Pfade mit expliziten Zeitbedingungen kann deren Verteilung als *„path slack histogram“* angezeigt werden.

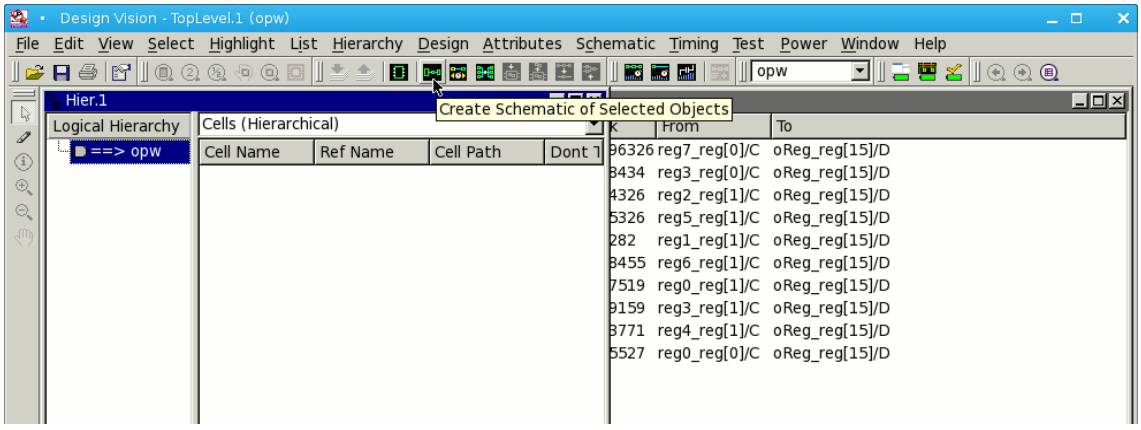


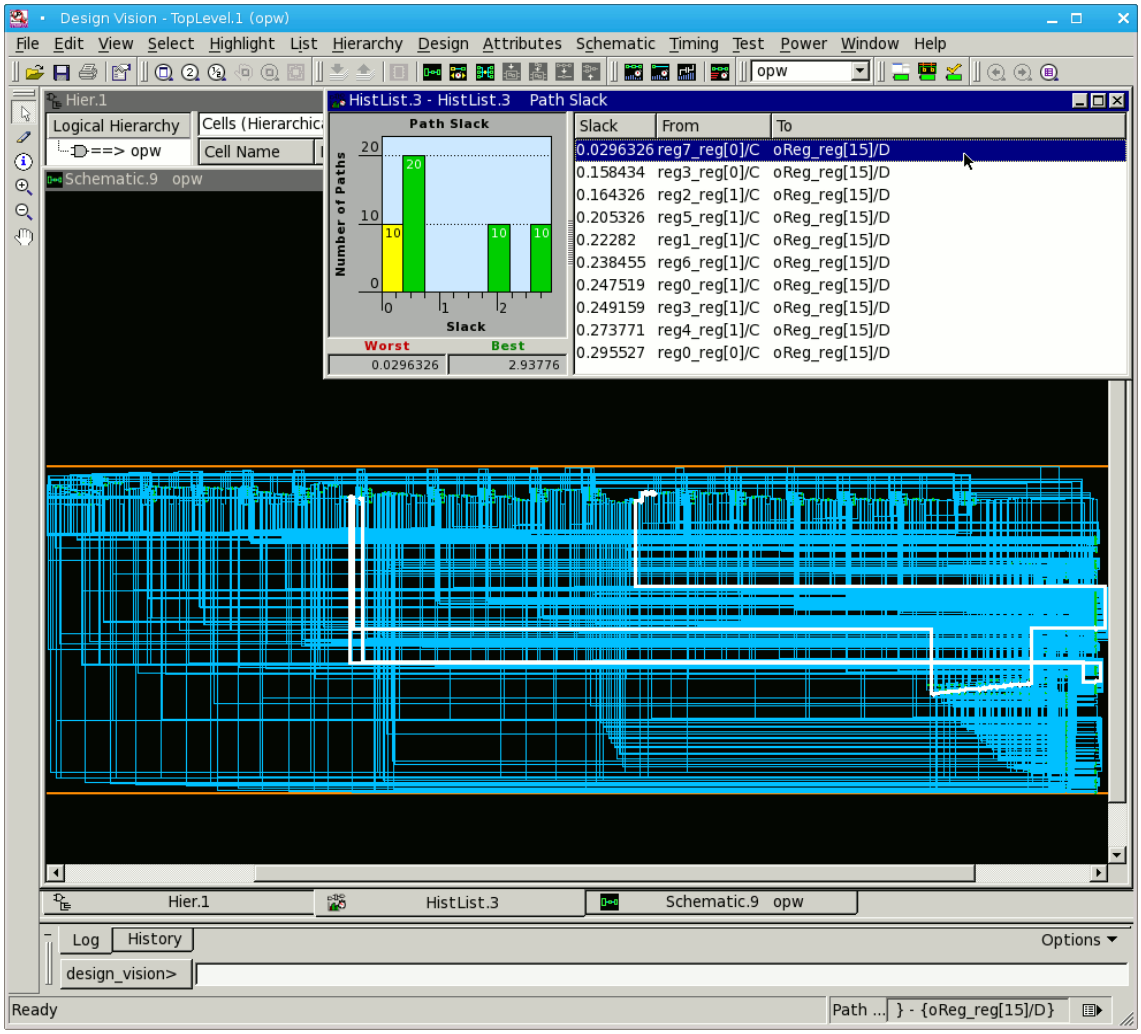


Auch hier sieht man den längsten Pfad…



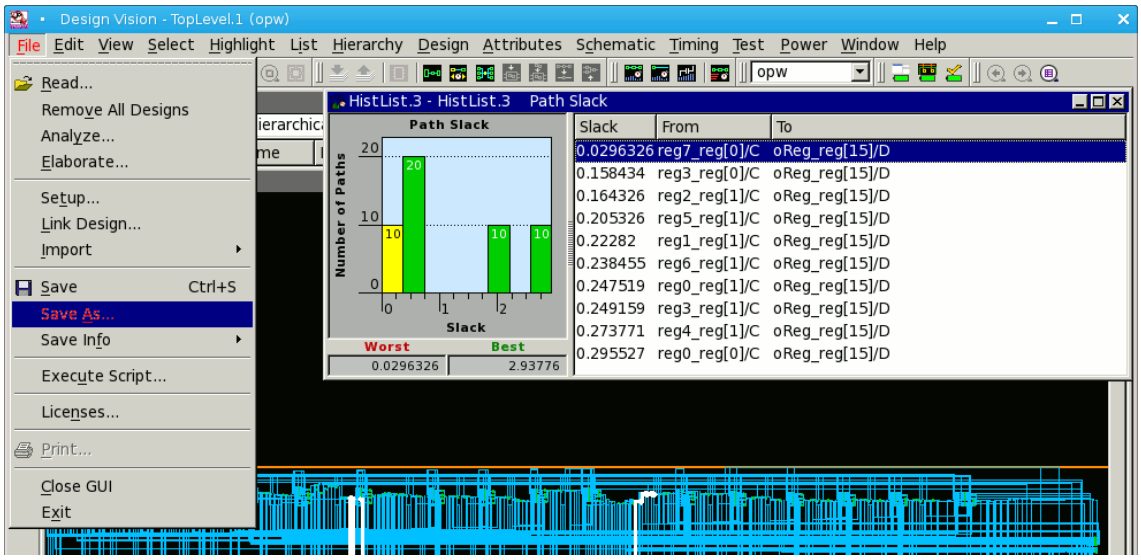
Dieser kritische Pfad, wie auch beliebige Pfade ausgewählter Netze, können im Schematic visualisiert werden:



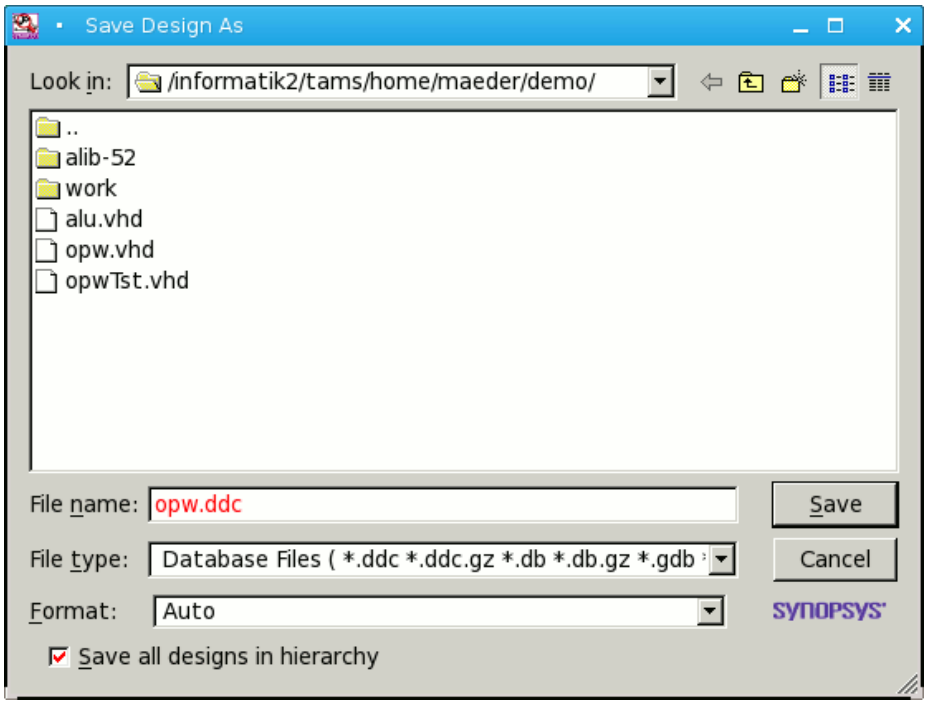


Sichern und Ausgabedateien erzeugen

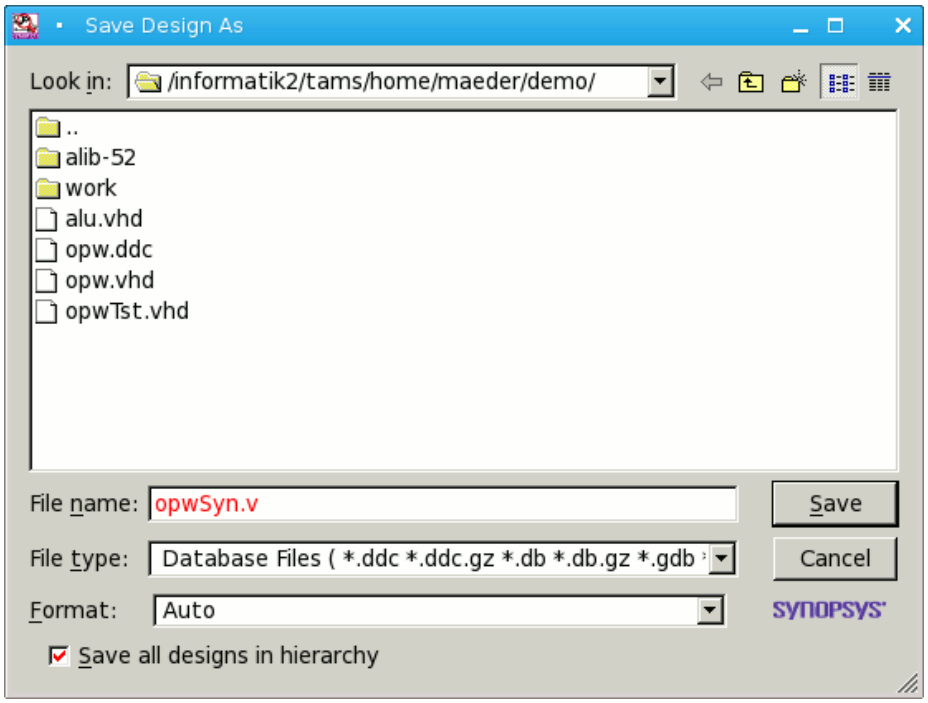
15. Sichern der internen Datenbasis:



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:

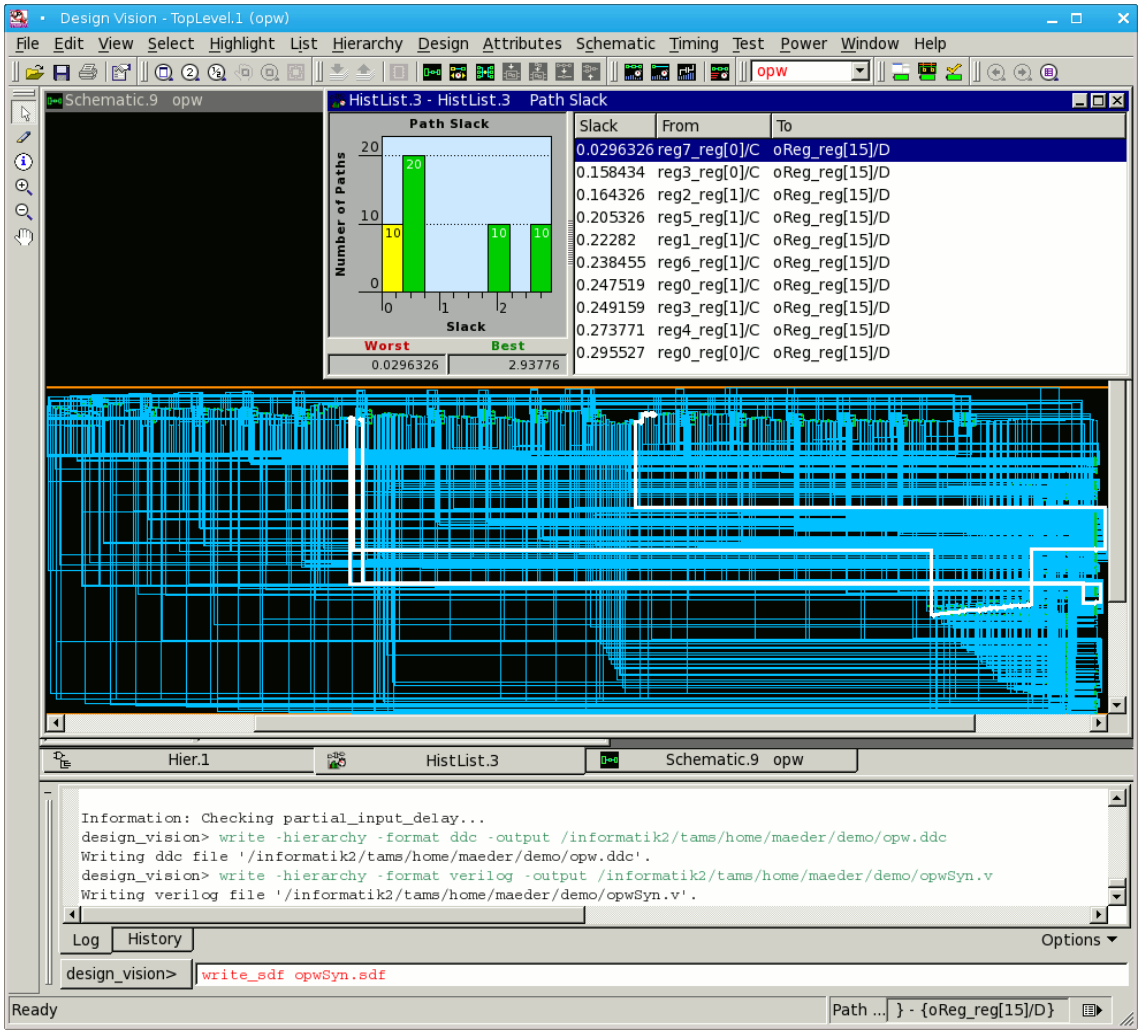


Aus Effizienzgründen wird eine gemischte Simulation von VHDL-Testumgebung und Verilog-Netzliste bevorzugt. Die dazu notwendigen Schritte sind in der extra Beschreibung *„Simulation von Gatternetzlisten – VHDL- und Mixed-mode“* erläutert.

17. SDF-Datei schreiben

Die SDF-Datei (*S*tandard *D*elay *F*ormat) 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`:



18. . . . fertig, Programm beenden: