Die folgende Beschreibung soll die grundlegenden Schritte zeigen, wie

- eine VHDL-Beschreibung mit dem Synopsys Design-Analyzer synthetisiert und auf die ES 2 Zellbibliotheken abgebildet wird.
- eine VHDL-Ausgabe der synthetisierten Schaltung für die Simulation erzeugt wird und wie sie in der Simulation benutzt werden kann.
- die Synopsys Datenbasis in eine EDIF-Datei für den Datentransfer zu Cadence DF II umgewandelt wird.

Dabei sind jeweils nur die einfachsten Schritte gezeigt — bei Problemfällen, die bei größeren Entwürfen (wahrscheinlich) auftreten werden, sei auf die entsprechenden Manuals verwiesen... Bei der Beschreibung der Benutzereingaben gilt die schon in anderen Anleitungen verwendete Symbolik.

1 Synopsys Design-Analyzer

Voraussetzung sei eine *korrekt* simulierte (hierarchische) VHDL-Beschreibung.

1. Start des Systems

   > `es2_design_analyzer` [xterm]

   Achtung: Der Befehl startet ein Skript, daß beim ersten Aufruf (pro Directory) nach dem Prozeß `-ecpd`..- und den Operationsbedingungen (Temperaturbereich, Simulationslasten...) `-ind` / `mil`- fragt, dementsprechend ist kein Start im Hintergrund möglich. Die voreingestellte Werte sind zu bestätigen.

2. Vorbereitung für parametrisierbare Entities (Templates)[1]
   Enthält die VHDL-Beschreibung generische Elemente (`entity` mit `generic`), so werden diese nach dem Einlesen der VHDL-Beschreibung noch nicht als Objekte der Synopsys Datenbasis erzeugt. Erst später bei Abarbeitung der Hierarchie werden DB-Elemente, entsprechend den spezifizierten `generic`-Werten, erzeugt — bzw. durch expliziten Aufruf von: ☐ `File - Elaborate` [Synopsys Design Analyzer]

   ☐ `Setup - Variables...` [Synopsys Design Analyzer]

   ≡ `Variable Names:` [Variables]
   `hdlin_auto_save_templates = "true"`

3. Eingabefenster der DC-shell öffnen — später folgen hier einige Eingaben

   ☐ `Setup - Command Window...` [Synopsys Design Analyzer]

[1] optional: nur für VHDL-Entities mit `generic`

4. Einlesen der VHDL Quelldateien
   Dieser Schritt ist für alle VHDL-Dateien des Entwurfs (bottom-up) durchzuführen[2].

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

   ≡ `File Names(s)` `=<vhdl file name>` [Read File]
   `File Format` `=VHDL`

   Beim Einlesen beachte man die log-Datei ([VHDL]); traten kein Fehler auf, so wurden die entsprechenden Objekte in der SYNOPSYS Datenbasis erzeugt. Für Entities die über generische Werte parametrisierbar sind, werden nur „templates“ erzeugt.

   Achtung: In der log-Datei ([VHDL]) 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 '<process name>'
    in routine <entity name> line <line nr> in file
    '<file name>'
============================================================...
|      Register Name     |      Type         |  Width   | ...
============================================================...
| <sign/var name>_reg    | Flip-flop / Latch | <nr bit> | ...
============================================================...
```

   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 Datetyp werden 32-bit breite Register erzeugt.
   - Obwohl nur das Verhalten eines Schaltnetzes bechrieben 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 — und nach unseren Erfahrungen z.T. auch *fehlerhafte* Hardware (im Falle der Latches).

5. Selektion des top-level Elements

   $\uparrow_l$ `<top-level element>` [Synopsys Design Analyzer]

   Das Element der obersten Hierarchieebene wird schraffiert umrandet dargestellt, alle weiteren Befehle beziehen sich auf das so selektierte Entity.

[2] Anmerkung: wurden die Elemente der Hierarchie vorher mit der Option `-spc_elab` analysiert (Aufruf von `vhdlan`), so kann auf das Einlesen der gesamten Hierarchie verzichtet werden. In diesem Fall genügt es, nur das top-level Element der Hierarchie einzulesen.

6. Auflösen der Hierarchie[3]

   □ `Edit - Uniquify - Hierarchy` [Synopsys Design Analyzer]

   Elemente, die in der Hierarchie mehrfach vorkommen, werden eindeutig (entsprechend oft) erzeugt und aus den Templates generischer Entities werden db-Objekte generiert.

   Diese Identifikation einzelner Instanzen hat den Vorteil, daß später bei der Sythese die Instanziierungen individuell behandelt werden können.

   Achtung: Wenn sehr große Designs zu bearbeitet werden oder wenn viele wirklich „identische“ Subdesigns referenziert werden, so sollte die Synthese (aus Effizienzgründen) hierarchisch durchgeführt werden:

   1. Synthese der Subdesigns.
   2. Vergabe der Attributes `don't touch` für bereits synthetisierte Subdesigns.
   3. Synthese des top-level Designs.

   Für das genaue Vorgehen hier nur der Verweis auf die Online-Dokumentation.

7. Automatisches Einfügen von Padzellen[4]
   Ist die Hierarchie schon so weit als VHDL-Beschreibung vorhanden, daß man das Gesamt-IC bearbeitet, können an dieser Stelle automatisch Pad-Zellen eingefügt werden — alternativ dazu könnten diese auch „von Hand“ in Cadence DF II hinzugefügt werden.

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

   ≡ `Port is Pad` `=<on>` [Design Attributes]

   Die VHDL-Ports werden als Pads der Schaltung gekennzeichnet.

   ▷ `set_pad_type -exact LIBIPS8B all_inputs()` [Command Window]

   Gegebenenfalls müssen die Pad Typen noch genauer spezifiziert werden. Dies ist sowohl für einzelne Pads als auch für ganze Gruppen möglich (siehe Synopsys Design-Analyzer Dokumantation).

   Hier werden beispielsweise für alle Eingänge der Schaltung CMOS-kompatible Pads festgelegt — ohne weitere Angaben von z.B. Spannungspegeln würden sonst TTL-Eingänge LIBIPS8G generiert werden.

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

   ≡ – bestätigen [Insert Pads]

8. Festlegung der Operationsbedingungen
   Durch diese Auswahl wird das Timingmodell der Gatter für die folgenden Schritte (Synthese und Timinganalyse) festgelegt.

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

   ≡ `Design Name` `=<design name>` [Operating Conditions]
   `Operating Conditions` `=MIN / TYP / MAX-IND`

   Eingabe durch ↑$_l$ `<condition>`.

[3] optional: dieser Schritt ist natürlich nur notwendig, wenn auch eine Hierarchie vorhanden ist — was bei größeren Designs der Fall sein sollte!

[4] optional: nur bei Synthese des top-level Elements, wobei dieses dann dem kompletten IC entspricht und mit Cadence weiterbearbeitet werden soll (Plazierung und Verdrahtung)

9. Hardwaresynthese und Abbildung auf die Zellbibliothek (`ecpd07` / `ecpd10`)
   Ausgehend von dem selektierten (obersten) Element, läuft die Synthese für die gesamte Hierarchie ab. Wegen der Möglichkeiten den Syntheseprozeß zu beeinflussen, sei hier nochmals auf die SYNOPSYS Dokumentation verwiesen. Hier nur die „einfachen“ Alternativen:[5]

   1. minimale Fläche (Voreinstellung): [Command Window]
      ▷ `set_max_area 0.0`
   2. maximale Geschwindigkeit: [Command Window]
      ▷ `set_max_delay 0.0 {all_outputs() all_registers(-data_pins)}`
      oder, bei Verwendung von Clocks,
      ▷ `create_clock -period <time> <clock-input>`

   □ `Tools - Design Optimization...` [Synopsys Design Analyzer]
   ≡ – bestätigen [Design Optimization]
   Während des Laufs beachte man die log-Datei ([Compile Log]).

10. Bewertung der Syntheseergebnisse
    Hier werden nur einige Möglichkeiten von SYNOPSYS DESIGN-ANALYZER vorgestellt. Die bei der Timinganalyse ausgegebene Information bezieht sich immer auf das unter Punkt 8. festgelegten Zeitmodell. Durch Auswahl anderer Operationsbedingungen können „worst-case“ und „best-case“ Timing der synthetisierten Struktur ermittelt werden.

    - Kontrolle des Schematic / Ansehen der Hierarchie
      ↑$_l$ `<db-Objekt> / <Symbol im Schematic>` [Synopsys Design Analyzer]
      ↑$_l$ `<Darstellung>` - Button: linke Seite im Fenster [Synopsys Design Analyzer]
    - Timing einzelner Netze
      ↑$_l$ `<Netz im Schematic>` [Synopsys Design Analyzer]
      □ `Analysis - Show Timing...` [Synopsys Design Analyzer]
      ↑$_l$ `<Netz im Schematic>` Auswahl weiterer Netze... [Synopsys Design Analyzer]
    - Lasten einzelner Netze
      ↑$_l$ `<Netz im Schematic>` [Synopsys Design Analyzer]
      □ `Analysis - Show Net Load...` [Synopsys Design Analyzer]
      ↑$_l$ `<Netz im Schematic>` weitere Netze... [Synopsys Design Analyzer]
    - Timing von Pfaden
      Kritischer (längster) Pfad
      □ `Analysis - Highlight - Critical Path` [Synopsys Design Analyzer]
      Längste / Kürzeste Pfade zu Elementen
      ↑$_l$ `<Symbol im Schematic>` [Synopsys Design Analyzer]
      □ `Analysis - Highlight - Max / Min Path - ... Path / Rise / Fall` [Synopsys ...]

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

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

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

  $\uparrow_l$ `<Report Optionen>` [Report]

11. Schematics für die gesamte Hierarchie generieren[6]
    Da hier über EDIF Schematics ausgetauscht werden, müssen diese für alle Hierarchieelemente vorhanden sein.

    ▷ `create_schematic -hierarchy -no_bus` [Command Window]

12. EDIF-Datei schreiben[6]

    □ `File - Save As...` [Synopsys Design Analyzer]

    ≡ `File Name: =<edif file name>` [Save File]
    `File Format: =EDIF`
    `Save All Designs in H.. =<on>`

    Dieser Schritt erzeugt die EDIF-Datei; beim Lauf beachte man Fehlermeldungen und Warnungen, die in dem Kommandofenster ([Command Window]) ausgegeben werden. Gegebenenfalls muß der Entwurf überarbeitet werden...

13. VHDL-Simulation der synthetisierten Netzliste[7]
    Die synthetisierte Netzliste kann jetzt als VHDL-Datei geschrieben und mit SYNOPSYS simuliert werden. Diese abschließende Simulation empfieht sich hier, da eine (vorhandene) VHDL-Testumgebung wieder benutzt werden kann und, für eine erste Überprüfung des Syntheseergebnisses, viele der zeitraubende Schritte entfallen, wie: Datenkonvertierung für CADENCE DF II, Initialisierung einer Verilog-Simulation, erstellen der Stimuli....

    Die synthetisierte VHDL-Netzliste wird folgendermaßen geschrieben:

    □ `File - Save As...` [Synopsys Design Analyzer]

    ≡ `File Name: =<vhdl file name>` [Save File]
    `File Format: =VHDL`
    `Save All Designs in H.. =<on>`

    Dieser Schritt erzeugt die VHDL-Datei — eine strukturelle Beschreibung der synthetisierten Schaltung.

    Vor der Simulation sind noch einige Änderungen vorzunehmen, um die neue VHDL-Beschreibung benutzen zu können.

    > `vi <vhdl file name>` [xterm]

    Bei mit dem SYNOPSYS DESIGN-ANALYZER erzeugten Dateien wird vor jeder Entity ein `library IEEE;` eingefügt. Die Referenz auf die Zellbibliothek fehlt noch, kann hier aber bequem mit einem Search/Replace erzeugt werden, beispielsweise mit

    ```
    1,$s/library IEEE;/library ecpd07,IEEE;/
    ```

[6] optional: nur notwendig, wenn ein Datentransfer zu CADENCE DF II stattfinden soll, um dort Logiksimulationen und das physikalische Layout durchzuführen.

[7] optional: für eine VHDL-Gattersimulation

> `vi <vhdl-testbench file name>` [xterm]

Wurde für die VHDL-Simulation der Schaltung eine Testumgebung benutzt, so muß die „neue“ synthetisierte Architecture durch eine Konfigurationsanweisung referenziert werden. Die Namen werden dabei folgendermaßen vergeben: der `entity`-Name bleibt gleich, während der `architecture`-Name um `syn_` ergänzt wird:

```
configuration <cfg name> of <test entity name> is
  for <test architecture name>
    for <instance label>: <design entity name> use entity
        WORK.<design entity name>(syn_<design architecture name>);
    end for;
  end for;
end;
```

Anschließend kann wie gewohnt simuliert werden: Simulationsmodelle mit `vhdlan` erzeugen und mit `vhdlsim`/`vhdldbx` simulieren.

14. Beenden von Synopsys Design-Analyzer

□ `File - Quit` [Synopsys Design Analyzer]

≡ – bestätigen [Quit Design Analyzer]