Diese Beschreibung bezieht sich in vielen Punkten auf die „Dokumentation zu CADENCE“, bei der Angabe der Benutzereingaben wird außerdem die gleiche Symbolik verwendet.

1 Design-Flow

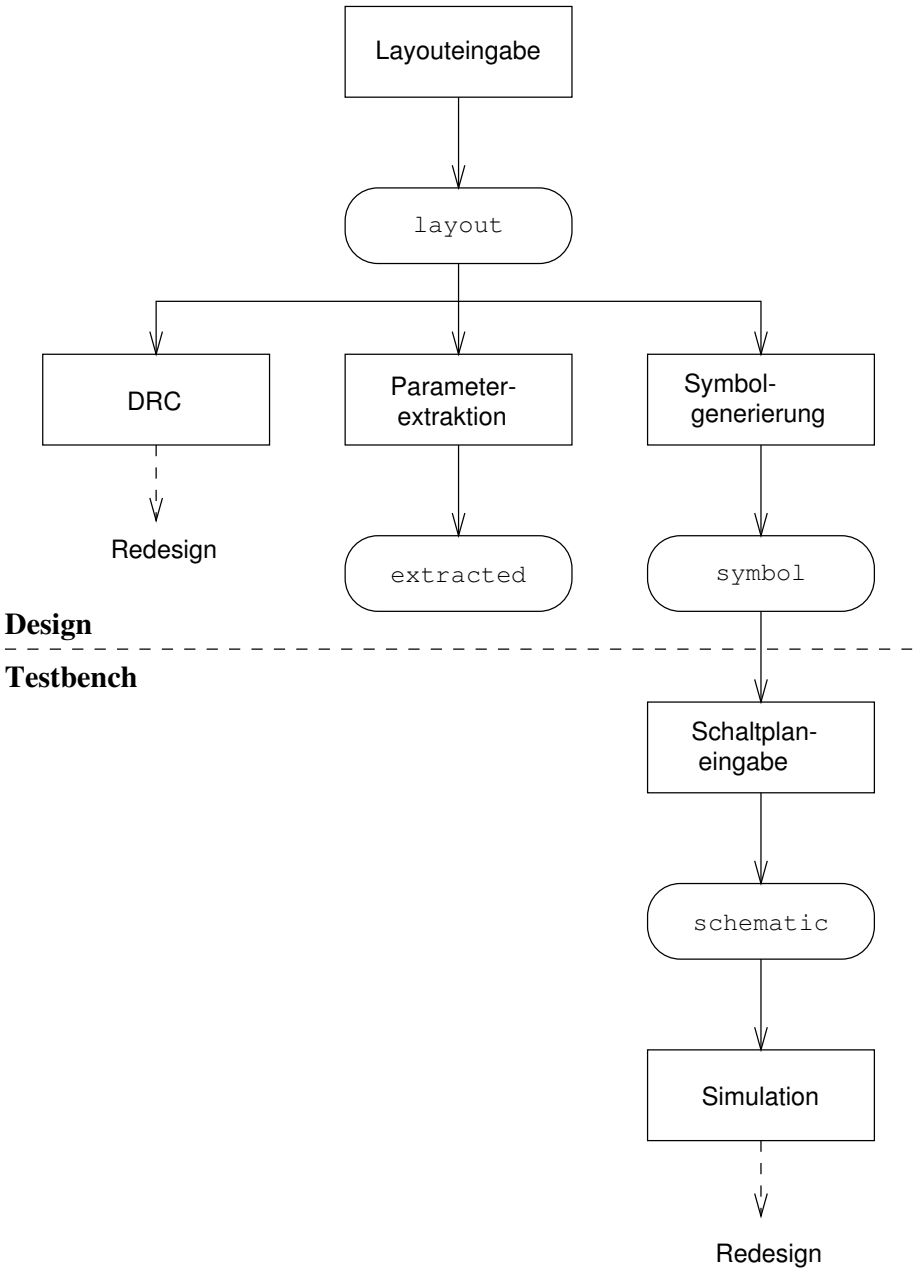

Abbildung 1: Design-Flow

Dem Full-Custom Layout liegt folgender Entwurfsablauf zugrunde, er ist in Abb. 1 skizziert:

1. Eingabe des Layouts mit dem graphischen Editor
2. Kontrolle durch einen Design-Rule-Check
3. Extraktion der elektrischen Netzliste
4. Generierung eines Symbols für den Schematic-Editor
5. Aufbau einer Testumgebung
6. Simulation der Schaltung

2 Full-Custom Layout

1. Start des Systems

   > `es2_cdk` [xterm]

   Achtung: Da dieser Befehl ein Skript startet, darf er nicht im Hintergrund ausgeführt werden. Beim ersten Aufruf (in einem Directory) wird nach dem Prozeß `-ecpd..-` und dem Programm `-icfb-` gefragt, dabei sind bei beiden Fragen die voreingestellten Werte zu bestätigen.

2. Design (Layout) öffnen

   □ `Open - Design...` [ES2 0.7um ...]

   ≡ `Library Name` = `<lib name>` [Open Design]
   `Cell Name` = `<cell name>`
   `View Name` = `layout`

3. Layout erstellen — siehe: „Dokumentation zu CADENCE“, 2.5 Der Layout-Editor.

4. Design Rule Check (DRC)

   □ `Verify - DRC...` [Editing:...]

   ≡ `Checking Method` = `flat` [DRC]
   `Checking Limit` = `full`
   `Select` = `slow`
   `Switch Names` = `slow`

   Während des DRC erscheinen in dem CADENCE Eingabefenster ([ES2 0.7um ...]) die Ausgaben des Programms. Die Ergebnisse werden am Ende des DRC zusammengefaßt unter: `*** Summary of rule violation for cell "..." ***`

   Traten DRC-Fehler auf, so werden sie mit einem speziellen Layer (blinkend) markiert. Die Größe dieser Polygone beschreibt oft schon die Regelverletzung, zum Beispiel bei Mindestabständen. Zu dem einzelnen Regelverletzungen kann man sich Texterklärungen ausgeben lassen.

   □ `Verify - Markers - Explain` [Editing:...]

   oder ↑$_l$ `Explain Error` [DRC].

$\uparrow_l$ `<drc figure>` [Editing:...]

Der Befehl öffnet ein eigenes Text-Fenster in dem, nach Auswahl einer DRC-Meldung, die entsprechende Regelverletzung ausgegeben wird.

Der Befehl kann durch Eingabe von `Esc` oder □ `File - Close Window` [marker text] beendet werden.

oder

□ `Verify - Markers - Find...` [Editing:...]

oder $\uparrow_l$ `Find Error` [DRC].

≡ – entsprechend ausfüllen [Find Marker]

Durch den Befehl wird ein Textfenster ([marker text]) geöffnet. Der jeweils aktuelle DRC-Fehler wird hell umrandet dargestellt.

`Previous / Next` : schaltet zwischen den Fehlern hin und her
`Delete` : löscht DRC-Fehler

Um die (teilweise störenden) DRC-Markierungen zu löschen, können folgende Befehle benutzt werden.

□ `Verify - Markers - Delete` [Editing:...]

$\uparrow_l$ `<drc figure>` [Editing:...]

oder

□ `Verify - Markers - Delete All...` [Editing:...]

oder $\uparrow_l$ `Delete All...` [DRC].

≡ –bestätigen [Delete All Markers]

Das Layout ist so lange nachzuarbeiten, bis keinerlei DRC-Fehler mehr vorhanden sind.

5. Markieren der Schaltungsanschlüsse für die Simulation — dieser Schritt ist in der Beschreibung des Layout-Editors dokumentiert (2.5.9 Kennzeichnung der Anschlüsse).

   Achtung: Es ist darauf zu achten, daß für die Spannungsversorgung die reservierten Bezeichner `vdd` und `gnd` benutzt werden.

6. Extraktion der elektrischen Netzliste für die Simulation

   □ `Verify - Extract...` [Editing:...]

   ≡ `Extract Method` = `flat` [Extractor]
   `Select` = `capall`
   `Switch Names` = `capall`

   Anmerkung: Da die Fertigungsprozesse der Hersteller (natürlich) gewissen Schwankungen unterliegen, kann man keine genaue Aussage machen, *wie* (Geschwindigkeit, Schaltverhalten...) sich gefertigte Elemente (z.B. Transistoren) letztendlich verhalten.

Die Chiphersteller garantieren allerdings, daß sich alle gefertigten Elemente innerhalb gewisser Toleranzen befinden, dementsprechend werden drei Werte definiert:

Verzögerungszeit	Bedeutung
MAX	langsamste Version, untere Schranke
TYP	typischer Wert
MIN	schnellste Version, obere Schranke

Für die parasitären Kapazitäten, Widerstände, Dioden usw. werden hier die MAX-delay Werte extrahiert, um in der Simulation wirklich den worst-case Fall zu simulieren.[1]

Das Transistormodell, als bestimmender Faktor für die Geschwindigkeit der Schaltung, wird erst später bei der Simulation ausgewählt.

7. Extrahiertes Netz ansehen

   □ `Open - Design...` [ES2 0.7um ...]

   ≡ `Library Name` = `<lib name>` [Open Design]
   `Cell Name` = `<cell name>`
   `View Name` = `extracted`

   In der `extracted`-View sieht man die extrahierten Komponenten mit ihren Parametern.

   □ `Window - Close` / ⊙ `^w` [Editing:...]

8. Symbol generieren als Schnittstelle zum Schematic-Editor

   • erste Möglichkeit: nach dem Start des Schematic-Editors (z.B. beim Editieren des Schematic für die Testumgebung — Punkt 9) wird dessen Symbolgenerator benutzt.

   □ `Design - Create Cellview - From Pin List...` [Editing:... schematic ...]

   ≡ `Input Pins` = `<name list>` [Cellview From Pin List]
   `Output Pins` = `<name list>`
   `IO Pins` = `<name list>`
   `Switch Pins` = `<name list>`
   `Library Name` = `<lib name>`
   `Cell Name` = `<layoutcell name>`
   `Display Cellview` = `<off/on>`
   `Edit Options` = `<off/on>`
   `View Name` = `symbol`

   • zweite Möglichkeit: Symbolgenerator des Library-Compilers

   > `cp $CDS_INST_DIR/designKits/examples/tsgen tsgen` [xterm]

   > `vi tsgen` [xterm]

   Dabei sind in der Datei alle Einträge mit `<string>` durch passende Werte zu ersetzen. Auf der nächsten Seite ist die am Beispiel eines Inverters `myinv` beschrieben.

[1] Relativ zum typischen Wert sind die Abweichungen der anderen Parametersätze circa ±10%.

```
; Example for Inverter -AJM-
;
defcell("myinv"
  input("invin" "vdd" "gnd")
  output("invout")
  defsymbol(
    pinLocSpec(
      topPins("vdd")
      bottomPins("gnd")
      leftPins("invin")
      rightPins("invout")
      )
    symbolProps(partName = "myinv")
    controlParam(queryMode = t)
  )
)
```

▷ `lmLoadData("tsgen" "<lib name>")` [ES2 0.7um ...]

Das Symbol wird, entsprechend den Angaben in `tsgen` erzeugt. Meldungen zu dem Lauf erscheinen im CADENCE Eingabefenster ([ES2 0.7um ...]).

Existiert das Symbol schon in der Bibliothek `<lib name>`, so muß dessen Ersetzung bestätigt werden.

≡ –bestätigen [Overwrite Symbol]

9. Design der Testumgebung — siehe: „Dokumentation zu CADENCE“, 2.6 Der Schematic-Editor.
Um das Layout unter „realistischen“ Bedingungen zu simulieren, sollte eine Testumgebung aufgebaut werden, die externe Lasten simuliert.

□ `Open - Design...` [ES2 0.7um ...]

≡ `Library Name` = `<lib name>` [Open Design]
`Cell Name` = `<testcell name>`
`View Name` = `schematic`

In dem Schematic wird jetzt das Layout referenziert; seine Versorgungsspannungsanschlüsse `vdd` und `gnd` werden an entsprechende Instanzen aus der Bibliothek `basic` (`Supplies`) angeschlossen.

Außerdem erhält es an den	Eingängen :	Eingangswiderstand	40 Ω
		Eingangskapazität	0.1 pF
	Ausgängen :	Lastkapazität	0.1 pF

Die Instanzen für Widerstände und Kapazitäten sind in der Bibliothek `sample` in der Zellkategorie `PARAS` zu finden.

□ `Add - Component...` / ⊙ `i` [Editing:...]

≡ `Library` = `<lib>` / `basic` / `sample` [... Library Menu]
`Cell` = `<cell>` / `vdd, gnd` / `resistor, cap`
`View` = `symbol`

Anmerkung: da die Kapazität ohnehin mit `gnd` verbunden wird, sollte `cap` als Kondensator eingesetzt werden.[2]

Die Werte werden anschließend über Properties angegeben.

□ `Edit - Properties - Objects... / ⊙ q` [Editing:...]

↑$_l$ `Add` [Edit Object Properties]

≡ `Name` `=r / c` [Add Property]
`Type` `=float`
`Value` `=40 / 0.1p`

Die Ein- und Ausgänge des Schematic werden mit Pins versehen, über die die Schaltung in der Simulation angesprochen wird.

□ `Add - Pin... / ⊙ p` [Editing:...]

≡ `Pin Names` `=<string>` [Add Pin]
`Direction` `=<input/output>`
`Usage` `=schematic`

Anschließend kann gesichert werden — dabei sollte auch gleich ein Schematic-Rule-Check durchgeführt werden.

□ `Design - Check and Save / ⊙ X` [Editing:...]

10. Simulation vorbereiten

□ `Tools - Simulation - Other` [Editing:...]

□ `Simulation - Initialize...` [Editing:...]

≡ `Simulation Run Directory` `=<simdir name>` [Initialize Environment]

Wenn bei der Initialisierung ein schon vorher benutztes Verzeichnis angegeben wird, so erscheinen die nachfolgenden Fill-Forms nicht mehr, da die entsprechenden Werte aus (vorhandenen) Kontrolldateien übernommen werden.

≡ `Simulator Name` `=hspice` [Initialize Environment]
`Library Name` `=<lib name>`
`Cell Name` `=<testcell name>`
`View Name` `=schematic`

≡ `HSpice Transistor Model` `=<typ/slow/fast>` [Initialize HSpice Simulation]
`HSpice run directory` `=<simdir name>.<model>`

Achtung: Das vorher eingegebene Simulationsverzeichnis (`<simdir name>`) wird automatisch mit dem Namen des Transistormodells verlängert. Der „richtige“ Name wird in der Fill-Form als `HSpice run directory` angezeigt.

[2] **Achtung:** bei der Benutzung von `capacitor` ist ein Wert für die Property `IC` einzutragen (Spannung über dem Kondensator bei Simulationsbeginn).

11. Stimuli editieren
    Die Stimuli (Eingangswerte) der Simulation werden am einfachsten im STL-Format eingegeben — siehe: „Dokumentation zu CADENCE“, 2.7 Simulation and Test Language.

    □ `Simulation - Stimulus - Edit STL Stimulus...` [Editing:...]

    ≡ `Compile STL after Editing =<on>` [Edit STL Stimulus]

    In einem Fenster wird ein Editor (üblicherweise `vi`) mit einer STL-Templatedatei gestartet. In dieser sind die Ein- und Ausgänge der Schaltung schon eingetragen, Änderungen betreffen:
    - ggf. die Definition von Clocksignalen
    - `deftiming 10ps 1ns 10ns`
    - das Einfügen der Eingangsstimuli

    Hierzu ein Beispiel:

```
; Example for Inverter -AJM-
defpin tstin      in tr=1e-10 tf=1e-10
defpin tstout     out
deflevel  vil=0 vih=5 vol=0 voh=5
deftiming 10ps 1ns 10ns
defformat tstin
deftest
xv 0
xv 1
xv 0
endtest
```

    Nach Beendigung des Editors werden die Stimuli in HSPICE-Eingaben konvertiert; man beachte dabei die Ausgabe in [ES2 0.7um ...].

12. Simulation starten

    □ `Simulation - Netlist/Simulate...` [Editing:...]

    ≡ [Netlist and Simulate]

```
Library Name            =<lib name>
Cell Name               =<testcell name>
View Name               =schematic
Simulator Name          =hspice
Run Actions netlist     =<on>
Run Actions simulate    =<on>
Run in Background       =<off>
```

    Geht das Fenster der Fill-Form zu, so ist die Simulation beendet — was allerdings noch nicht heißt, daß der HSPICE überhaupt richtig gelaufen ist. Deshalb sollte man noch folgende Schritte ausführen.

    □ `Simulation - Show Outputs - Show Output` [Editing:...]

    In dem Text muß gegen Ende der Datei als Ausgabe erscheinen:

```
*****job concluded
```

    Ist dies nicht der Fall, so gab es noch Probleme, wobei die entsprechenden Ausgaben entweder in dieser Datei stehen oder (Datei wurde nicht erzeugt!) in [ES2 0.7um ...].

    □ `File - Close Window` [...si.out]

13. Signale für Waveforms markieren
Vor dem Öffnen des Waveform-Fensters müssen in dem Schematic (im Layout) die Netze ausgewählt werden, die man sehen möchte.

□ `Design - Probe - Add Net` / ⊙ `9` [Editing:...]

Wird nicht der Bindkey benutzt, erscheint folgende Fill-Form.

≡ `View Types all` = `<on>` [Applications to Cross Probe]

↑$_l$ `<net>` [Editing:...]

Die ausgewählten Netze werden farbig gekennzeichnet,

Da bei der Simulation die Signalverläufe aller Knoten der Schaltung aufgezeichnet wurden, können außer den Netzen der Testumgebung (Ein- und Ausgänge der „eigentlichen“ Schaltung) auch die Simulationsergebnisse aller internen Netze angezeigt werden.

↑$_l$ `<sub design>` [Editing:...]

□ `Design - Hierarchy - Descend Read...` / ⊙ `e` [Editing:...]

≡ `View Name` = `extracted` [Descend]

Abstieg innerhalb der Hierarchie zu der extrahierten Netzliste.

□ `Verify - Probe...` [Reading:... extracted]

≡ `Probing Method` = `single w/o parasitics` [Probing]
↑$_l$ `Add Net`

↑$_l$ `<net>` [Reading:... extracted]

Die Figur eines leitenden Layers (`CME2`, `CME1` und `CPOL`) wird zur Anzeige ausgewählt.

Nach Festlegung aller Probes ist die Fill-Form [Probing] mit `Cancel` zu schließen.

□ `Design - Hierarchy- Return` / ⊙ `B` [Reading:... extracted]

In der Hierarchie zurückgehen...

14. Kontrolle der Ergebnisse

□ `Simulation - Show Waveforms...` [Editing:...]

≡ –bestätigen [Show Waveforms]

Die wichtigsten Befehle für die Arbeit im Waveform-Fenster sind unten aufgeführt.

Fensterkontrolle

□ `Display - Full` - `X / Y / X&Y` [Waveform]

□ `Display - Zoom In` - `X / Y / X&Y` [Waveform]

□ `Display - Zoom In By 2` - `X / Y / X&Y` [Waveform]

□ `Display - Zoom Out By 2` - `X / Y / X&Y` [Waveform]

□ `Display - Pan` - `X / Y / X&Y` [Waveform]

Messen von Zeiten und Spannungen

Ist der Cursor nah genug an einer Kurve, so folgt er, als Meßpunkt, deren Verlauf. Die Cursorkoordinaten werden in dem Fenster [Waveform] angezeigt: `X:... Y:...`

☐ `Measure - Set Anchor` [Waveform]

Setzt einen Punkt der Kurve als Referenzpunkt. Anschließend wird in der Anzeige neben den absoluten Cursorkoordinaten auch der relative Wert angezeigt: `X:... Y:... DX:... DY:...`

Nach der Kontrolle der Simulationsergebnisse kann das Waveform-Fenster wieder geschlossen werden.

☐ `Display - Close Window...` [Waveform]

≡ – bestätigen [Close window]

15. fertig !

☐ `Window - Close` [Editing:...]

≡ – ausfüllen [Save Changes]

Wurde das Schematic noch nicht gesichert, geschieht dies jetzt automatisch.

☐ `Open - Quit...` [ES2 0.7um ...]

≡ – bestätigen [Quit]

≡ – entsprechend ausfüllen [Save Cellviews]

Unter Umständen sind noch weitere Sicherungen durchzuführen.

≡ `Save Techfile Changes?  =discard changes` [Save Technology File Changes?]