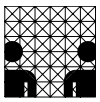
Full-Custom Design

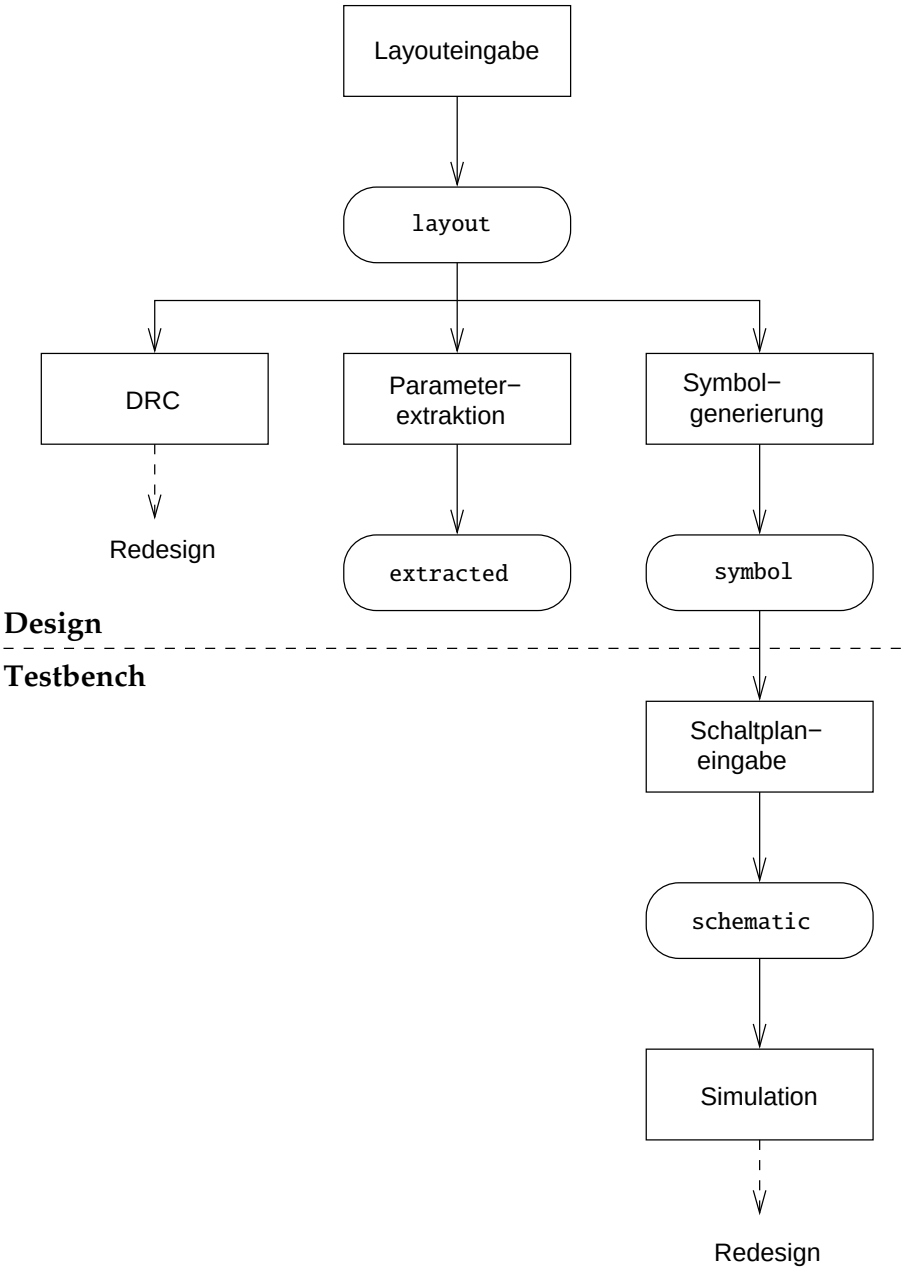
Werkzeuge : CADENCE IC, HSPICE
Design-Kits : ES2 Full-Custom
designSetup : `ic2 hsp es2`

A. Mäder



Diese Beschreibung bezieht sich in vielen Punkten auf die „CADENCE Grundlagen“, bei der Angabe der Benutzereingaben werden die dort beschriebenen Symbole verwendet.

Design-Flow



Dem Full-Custom Layout liegt folgender Entwurfsablauf zugrunde, er vorne skizziert:

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

Arbeitsschritte

Layouteingabe

1. Start des Systems

   > `ES2FCStart` [xterm]

2. Bibliothek erzeugen
   optional: Nur beim ersten Start ist eine eigene Bibliothek einzurichten.

   □ `File - New - Library...` [icfb - Log:...]

   ≡ `Name` = ⟨*libId*⟩ [New Library]
   `Technology File` = `Attach to an existing techfile`
   `Design Manager` = `No DM`

   ≡ `Attach To...` = `techLib` [Attach Design...]

3. Design (Layout) öffnen — werden neue Entwürfe erstellt:

   □ `File - New - Cellview...` [icfb - Log:...]

   ≡ `Library Name` = ⟨*libId*⟩ [Create New File]
   `Cell Name` = ⟨*cellId*⟩
   `Tool` = `Virtuoso`

   Um bereits existierende Entwürfe zu bearbeiten:

   □ `File - Open...` [icfb - Log:...]

   ≡ `Library Name` = ⟨*libId*⟩ [Open File]
   `Cell Name` = ⟨*cellId*⟩
   `View Name` = `layout`
   Dabei stehen für Library und View entsprechende Auswahlfelder zur Verfügung.

4. Layout erstellen — siehe: „CADENCE Grundlagen“, 3 Der Layout-Editor.

Design Rule Check

5. Design Rule Check (DRC)

   □ `Verify - DRC...` [Virtuoso...]

   ≡ `Checking Method` = `flat` [DRC]
   `Checking Limit` = `full`
   `Switch Names` = `slow`
   Während des DRC erscheinen in dem CADENCE Eingabefenster ([icfb - Log:...]) 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` [Virtuoso...]

$\uparrow_l$ ⟨*drcFigure*⟩ [Virtuoso...]

Der Befehl öffnet ein eigenes Text-Fenster in dem, nach Auswahl einer DRC-Meldung, die entsprechende Regelverletzung ausgegeben wird.
Er wird durch Eingabe von `Esc` oder □ `File - Close Window` [marker text] beendet.

oder

□ `Verify - Markers - Find...` [Virtuoso...]

≡ – entsprechend ausfüllen [Find Marker]

Durch den Befehl wird ein Textfenster ([marker text]) geöffnet. Der jeweils aktuelle DRC-Fehler wird violett 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` [Virtuoso...]

$\uparrow_l$ ⟨*drcFigure*⟩ [Virtuoso...]

oder

□ `Verify - Markers - Delete All...` [Virtuoso...]

≡ –bestätigen [Delete All Markers]

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

Netzlistenextraktion

6. Markieren der Schaltungsanschlüsse für die Simulation — dieser Schritt ist in der Beschreibung des Layout-Editors dokumentiert (3.8 Hierarchie).

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

7. Extraktion der elektrischen Netzliste für die Simulation

□ `Verify - Extract...` [Virtuoso...]

≡ `Extract Method` = `flat` [Extractor]
`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.

8. Extrahiertes Netz ansehen

□ `File - Open...` [icfb - Log:...]

≡ `Library Name` = ⟨*libId*⟩ [Open File]
`Cell Name` = ⟨*cellId*⟩
`View Name` = `extracted`

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

□ `Window - Close/`⊙ `^w` [Virtuoso... extracted]

Aufbau einer Testumgebung

9. Start des Schematic-Editors für die Testumgebung
Um das Layout unter „realistischen" Bedingungen zu simulieren, sollte eine Testumgebung aufgebaut werden, die externe Lasten simuliert. Werden neue Entwürfe erstellt:

□ `File - New - Cellview...` [icfb - Log:...]

≡ `Library Name` = ⟨*libId*⟩ [Create New File]
`Cell Name` = ⟨*testCellId*⟩
`Tool` = `Composer-Schematic`

Achtung: die Testumgebung ⟨*testCellId*⟩ muss anders heißen als das zu testende Layout, schließlich soll eine neue Hierarchieebene erzeugt werden, die das Layout (die zu simulierende extrahierte Netzliste) referenziert!

Um bereits existierende Entwürfe zu bearbeiten:

□ `File - Open...` [icfb - Log:...]

≡ `Library Name` = ⟨*libId*⟩ [Open File]
`Cell Name` = ⟨*testCellId*⟩
`View Name` = `schematic`

Dabei stehen für Library und View entsprechende Auswahlfelder zur Verfügung.

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

10. Symbol generieren als Schnittstelle zum Schematic-Editor
Nach dem Start des Composer-Schematic wird dessen Symbolgenerator benutzt.

□ `Design - Create Cellview - From Pin List...` [Composer-Schematic...]

≡ [Cellview From Pin List]

`Input Pins`	= ⟨*pinIdLis*⟩
`Output Pins`	= ⟨*pinIdLis*⟩
`IO Pins`	= ⟨*pinIdLis*⟩
`Switch Pins`	= ⟨*pinIdLis*⟩
`Library Name`	= ⟨*libId*⟩
`Cell Name`	= ⟨*layoutCellId*⟩
`Display Cellview`	= `off`|`on`
`Edit Options`	= `off`|`on`
`View Name`	= `symbol`

11. Design der Testumgebung
— siehe: „CADENCE Grundlagen", 4 Der Schematic-Editor.
In dem Schematic wird jetzt das Layout über das neu erzeugte Symbol 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. Da die Kapazität ohnehin mit `gnd` verbunden wird, sollte `cap` als Kondensator eingesetzt werden.[2]

□ `Add - Component.../⊙ i` [Composer-Schematic...]

≡ [Add Component]

`Library`	= ⟨*libId*⟩	`basic`	`sample`
`Cell`	= ⟨*cellId*⟩	`vdd`|`gnd`	`cap`|`resistor`
`View`	= `symbol`		

Die Werte werden anschließend über Properties angegeben.

□ `Edit - Properties - Objects.../⊙ q` [Composer-Schematic...]

↑$_l$ `Add` [Edit Object Properties]

≡ [Add Property]

`Name`	= `r`	`c`
`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` [Composer-Schematic...]

≡ [Add Pin]

`Pin Names`	= ⟨*pinIdLis*⟩
`Direction`	= `input`|`output`
`Usage`	= `schematic`

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

□ `Design - Check and Save/⊙ X` [Composer-Schematic...]

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

Simulation der Schaltung

12. Simulation vorbereiten

- □ `Tools - Simulation - Other` [Composer-Schematic...]
- □ `Simulation - Initialize...` [Composer-Schematic...]
- ≡ `Simulation Run Directory` = ⟨*simDir*⟩ [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` = ⟨*libId*⟩
  `Cell Name` = ⟨*testCellId*⟩
  `View Name` = `schematic`
- ≡ `HSpice Transistor Model` = `typ` | `slow` | `fast` [Initialize HSpice Simulation]
  `HSpice run directory` = ⟨*simDir*⟩`.`⟨*model*⟩

  Achtung: Das vorher eingegebene Simulationsverzeichnis ⟨*simDir*⟩ wird automatisch mit dem Namen des Transistormodells verlängert. Der „richtige“ Name wird in der Fill-Form als `HSpice run directory` angezeigt.

13. Stimuli editieren

Die Stimuli (Eingangswerte) der Simulation werden am einfachsten im STL-Format eingegeben — siehe: „CADENCE Grundlagen“, 5 Simulation and Test Language (STL).

- □ `Simulation - Stimulus - Edit STL Stimulus...` [Composer-Schematic...]
- ≡ `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-
stlinit
defpin invi  in  tr=1e-10 tf=1e-10
defpin invo  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 [icfb - Log:...].

14. Simulation starten

□ `Simulation - Netlist/Simulate...` [Composer-Schematic...]

≡ `Library Name` = ⟨*libId*⟩ [Netlist and Simulate]
`Cell Name` = ⟨*testCellId*⟩
`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` [Composer-Schematic...]
In dem Text muss in der ersten Zeile der Datei als Ausgabe erscheinen:
`** >info: ***** hspice 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 [icfb - Log:...].

□ `File - Close Window` [...si.out]

15. 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` [Composer-Schematic...]
Wird nicht der Bindkey benutzt, erscheint folgende Fill-Form.

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

↑$_l$ ⟨*net*⟩ [Composer-Schematic...]
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.

↑$_l$ ⟨*subDesign*⟩ [Composer-Schematic...]

□ `Design - Hierarchy - Descend Read.../⊙e` [Composer-Schematic...]

≡ `View Name` = `extracted` [Descend]
Abstieg innerhalb der Hierarchie zu der extrahierten Netzliste.

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

≡ `Probing Method` = `single probe` [Probing]
`Probe Type` = `device or net`
↑$_l$ `Add Device or Net`

↑$_l$ ⟨*net*⟩ [Virtuoso 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` [Virtuoso Reading:... extracted]
In der Hierarchie zurückgehen...

16. Kontrolle der Ergebnisse

□ `Simulation - Show Waveforms...` [Composer-Schematic...]

≡ –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]

17. ...fertig: CADENCE beenden

□ `Window - Close` [Composer-Schematic...]

≡ – ausfüllen [Save Changes]
Wurde das Schematic noch nicht gesichert, geschieht dies jetzt automatisch.

□ `File - Exit...` [icfb - Log:...]

≡ – bestätigen [Exit icfb?]

≡ – entsprechend ausfüllen [Save Cellviews]
Unter Umständen sind noch weitere Daten zu sichern.