Inhaltsverzeichnis

Eine detailliertere Beschreibung der Software würde den Rahmen dieser „Kurz"-Anleitung sprengen, aber zum Umgang mit CADENCE sollen hier einige Grundlagen angesprochen werden.
Bei der Beschreibung gelten folgende Symbole:

$>$ `command` [Xwin]
Eingabe von `command` an Unix-Shell, in Fenster Xwin.

$\triangleright$ `command` [Xwin]
Eingabe von `command` an CADENCE-Kommandointerface, in Fenster Xwin.

$\equiv$ `item` = `input` [Xwin]
Eingabe zum Ausfüllen von „Fill-Forms", in Fenster Xwin. Dabei wird für das Feld `item` entweder `input` über die Tastatur eingegeben oder aus einer Liste vorgegebener Möglichkeiten ausgewählt. Anschließend wird die Form durch `OK`, bzw. `Apply` ausgeführt.

$\square$ `menu - choice` [Xwin]
Auswahl von `choice` aus Menü `menu`, in Fenster Xwin.

$\uparrow_x$ `object` [Xwin]
Auswahl von `object` mit der Maus (Maustaste: x), in Fenster Xwin.

$\odot$ `key` [Xwin]
alternative Eingabe von `key` als CADENCE-Bindkey.

1 Konzepte

1.1 offenes System

Das was hier immer als CADENCE DF II bezeichnet wird ist eigentlich eine (sehr große) Menge von Programmen (Datenmanager, Editore, Simulatoren, Layoutprogramme...), die auf eine gemeinsame Designdatenbasis zugreifen. Prinzipiell ist CADENCE ein offenes Design-System, in das eigene Komponenten eingebracht werden können. Eine eigene (Lisp-ähnliche) Sprache SKILL erlaubt es in das System einzugreifen und auf die Entwurfsdaten zuzugreifen.

1.2 Bedienoberfläche

Bei der Benutzung von CADENCE wird man normalerweise mehrere Fenster offen haben.

CADENCE DF II-Fenster
- listet die Log-Datei mit den Ausgaben der CADENCE-Programme
- zeigt die Funktion der Maustasten an
- ist Eingabefenster für den SKILL-Kommandointerpreter
- erlaubt Einstellungen an dem Gesamtsystem
- startet über Menüs die einzelnen Programme

Editor-Fenster
- hier wird ein Design als `schematic`, `layout`... bearbeitet
- weitere Entwurfsschritte, wie Simulationen dieses Designs, werden gestartet

Browser-Fenster stellen Hierarchien graphisch dar, wie den Aufbau von Designs oder die Struktur der CADENCE-Bibliotheken. Das letzte Beispiel, der `Library Browser`, wird später noch genauer beschrieben.

Ausgabe-Fenster listen (Text-) Ausgabedateien auf oder stellen Simulationsergebnisse als Waveforms dar und erlauben so deren Auswertung.

Auswahl-Fenster stellen als eigenes Fenster zusätzliche Menüs zu den Programmen bereit oder beeinflussen die Funktion von Befehlen (z.B. Layerselektion für Layout, Plazierung und Verdrahtung).

Formular-Fenster erlauben die Einstellung von Optionen und Parametern. Bei vielen Programmteilen von CADENCE wird die Arbeitsweise der einzelnen Werkzeuge über solche *Fill-Forms* gesteuert.

Die Programme werden über pull-down Menüs gesteuert; dabei wird, beispielsweise bei den Editor-Fenstern, die Menüleiste abhängig von den Teilaufgaben dynamisch verändert. Häufig benutzte Menüpunkte sind oft als „fixed-Menu's" am linken Fensterand in Form von Symbolen aufgeführt.

1.3 Datenhaltung

Die Entwürfe sind hierarchisch, in vier Stufen, organisiert.

`lib` : Bibliothek in der sich Entwürfe befinden. Dies sind einerseits vom Benutzer selbst erstellte Bibliotheken, in denen eigene Entwürfe gesammelt werden (z.B. `DESIGNS`); andererseits werden die Zellbibliotheken für Standardzell- oder Gate-Array Entwürfe über Bibliotheken in das System eingebunden.

`cell` : Name einer Zelle, dabei steht „Zelle" nur für den Namen eines Elements einer beliebigen Abstraktionsebene beim Entwurf — vom komplett (hierarchisch) entworfenen Chip bis hin zum einzelnen Transistor.

Innerhalb der Bibliotheken können die Zellen noch zu *logischen Gruppen* zusammengefaßt werden: den Zellkategorien.[1] Dieser Mechanismus wird beispielsweise bei den Standardzellbibliotheken verwandt, um die Zellen nach Funktionen (`adders`, `boolean`, `d-flipflops`...) zu gruppieren.

`cellview` : Sichtweise/Art der Zelle; beschreibt (wie) welche Werkzeuge auf sie zugreifen dürfen und damit auch wo eine Zelle in der Hierarchie einzuordnen ist. Die Namen der `cellview` sind dabei von System fest vorgegeben:

`layout`	geometrische Darstellung
`extracted`	schematische Netzliste
`schematic`	schematische Netzliste
`symbol`	schematisches Symbol
`autoLayout`	flache Netzliste für physikalisches Layout
`verilog`, `spice`...	Simulationsmodell
`abstract`	physikalisches (Flächen-) Modell

`version` : Version der Zelle (Sichtweise).

Achtung: Die Bibliotheken dürfen nur über die CADENCE-Mechanismen – den `Library Browser` – manipuliert werden. Das Löschen oder Kopieren mit Unix-Befehlen erzeugt inkonsistente Bibliotheken.

[1]Zellkategorien werden im `Library Browser` blau angezeigt.

Da eine `cellview` nur mit Hilfe bestimmter Werkzeuge erzeugt und bearbeitet werden kann, wird beim Öffnen des Designs automatisch das entsprechende Programm gestartet (z.B. der Schematic-Editor). Die Zusammenhänge zwischen den verschiedenen Sichtweisen einer Zelle und den einzelnen Programmteilen in CADENCE sind in Abb. 1 dargestellt.

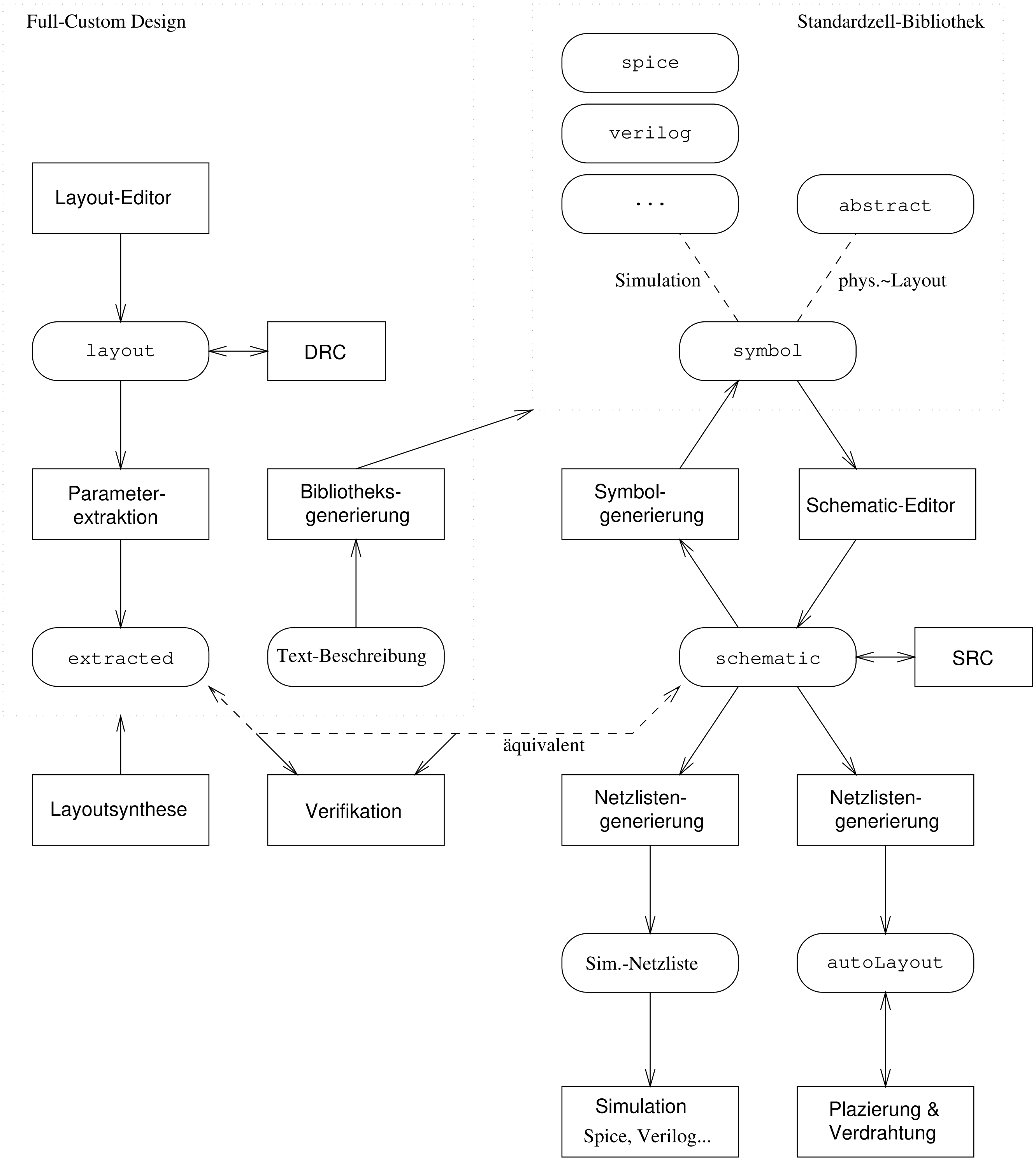

Abbildung 1: Abhängigkeiten zwischen Repräsentationen und den Werkzeugen

1.4 Properties

Ein grundlegendes Konzept von CADENCE sind *Properties*. Jedes Objekt (z.B. in einem Editor-Fenster) aber auch die einzelnen Zellen und Zellviews besitzen bestimmte Eigenschaften, über die der Entwurfsablauf und die Funktion der Programme gesteuert werden.

Im einfachsten Anwendungsfall werden Zellen damit parametrisiert: so erhalten Bauteile der elektrischen Ebene, wie Widerstände, Kondensatoren und Transistoren, ihre Werte als Property.

2 Bedienung von CADENCE DF II

2.1 Anfang und Ende

- 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.

  Nach dem Start erscheint das CADENCE DF II-Fenster ([ES2 0.7um ...]).

- Arbeit beenden :

  □ `Open - Quit...` [ES2 0.7um ...]

  ≡ – bestätigen [Quit]

  Damit das CADENCE DF II nicht verlassen werden kann, ohne daß vorher gemachte Änderungen gesichert worden sind, können beim Beenden noch folgende Meldungen erscheinen.

  ≡ – entsprechend ausfüllen [Save Cellviews]

  Achtung: für alle noch nicht gesicherten Designs wird gefragt, was mit ihnen geschehen soll und ggf. werden sie automatisch gesichert. Wurden die Daten schon vorher unter einem neuen Namen gesichert (z.B. mit `Design - Save As...`), so kann die automatische Sicherung ausgeschaltet werden.

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

  Änderungen in der Technologiedatei für die aktuelle Bibliothek (z.B. `DESIGNS`) brauchen nicht gesichert zu werden.

2.2 Online-Dokumentation

In allen Fenstern befindet sich ein `Help`-Button über den man sich die entsprechenden Abschnitte in den CADENCE-Manuals ansehen kann.

$\uparrow_l$ `Help` [<DF II window>]

Während obiger Mechanismus nur Hilfe zu den gerade aktuellen Befehlen oder Menüs gibt, läßt sich die komplette CADENCE DF II Online-Dokumentation von einer Unix-Shell aus aufrufen.

\> `cdshelp` [xterm]

Die Dokumentation ist als Hypertext organisiert; hier einige Befehle:

$\uparrow_l$ `<link>` [<Frame window>]

Durch Auswahl der Verweise kann man sich in dem Dokument bewegen. Dabei wählt man zuerst auf der linken Seite des Fensters eine Themengruppe (`Documents by ...`, `HDL Tools`, `IC Tools` ...) und erhalt anschließend eine Übersicht von Unterthemen. Nach Auswahl eines Unterthemas werden in rechten Teil des Fensters die passenden Manuals aufgezeigt. Bei Auswahl eines Manuals wird dieses dann als neues FRAMEVIEWER-Fenster angezeigt.

Abhängig vom jeweiligen Kontext erscheinen im unteren Fensterbereich Buttons für Querverweise, Inhaltsverzeichnisse, Seitenwechsel, Fenster schliessen ...

□ `Help - Main Menu...` [<Frame window>]

Öffnet ein weiteres Fenster mit dem Hypertext zur Benutzung von FRAMEVIEWER.

□ `Find - Search...` [<Frame window>]

Es wird ein Fenster geöffnet, in dem man „alle“ Dokumente nach Stichworten durchsuchen lassen kann. Anschließend können diese Dokumente an der entsprechenden Stelle direkt geöffnet werden.

□ `Search - In Current File...` [<Frame window>]

In dem aktuellen Dokument kann nach einem bestimmten Stichwort gesucht werden.

□ `File - Close` [<Frame window>]

Schließen eines Fensters, bzw. Beenden des Programms beim letzten Fenster.

2.3 Library-Browser

Der Browser stellt die Bibliothekshierarchie graphisch dar und erlaubt es über Menüs die Designs zu bearbeiten (öffnen), zu kopieren, zu löschen...

Er ist außerdem die einfachste Möglichkeit um Bibliotheken, Zellen und Zellviews für den Eintrag in *Fill-Forms* auszuwählen. Bei vielen Formularen ist deshalb ein Button `Browse` vorhanden, der den Library Browser startet und bei Auswahl von Elementen im Browser-Fenster den Eintrag in das entsprechende Feld des Formulars übernimmt.

□ `Design Manager - Library Browser...` [ES2 0.7um ...]

Erzeugt das Browser-Fenster, dabei werden alle Bibliotheken angezeigt, die sich im Suchpfad befinden. Die Anzeige `Level: lib...` in dem Fenster zeigt, wie die Bibliothekshierarchie zur Zeit dargestellt wird.

In dem Browser-Fenster können nun mit der Maus die einzelnen Befehle aufgerufen werden.

$\uparrow_l$ `<item>` [Library Browser]

Zeigt die nächstniedrigere Ebene der Bibliothekshierarchie zu `<item>` an. Bei Bibliotheken werden diese geöffnet (Darstellung in anderer Farbe).

$\uparrow_m$`<item>` [Library Browser]

Erzeugt kontextsensitive Menüs (abhängig von der Ebene in der Bibliothek), die es erlauben den Inhalt der Bibliotheken zu verändern und Designs zu öffnen. Hierzu einige Beispiele:

$\uparrow_m$`<cell> - Copy...`	kopiert Zelle und Views (wenn hierarchisch)
$\uparrow_m$`<cell> - Delete...`	löscht Zelle (und alle Views)
$\uparrow_m$`<cellview> - Copy...`	kopiert einzelne View
$\uparrow_m$`<cellview> - Delete...`	löscht einzelne View
$\uparrow_m$`<cellview> - Edit...`	öffnet Cellview, startet das entsprechende Werkzeug
$\uparrow_m$`<cellview> - Read...`	öffnet Cellview – read-only Zugriff

2.4 Designs bearbeiten und erzeugen

Die einfachste Möglichkeit ein schon existierendes Design zu bearbeiten – „Design“ hier als `lib cell cellview` –, ist über die Befehle `Edit...` und `Read...` des Library Browsers.

Die zweite Möglichkeit Designs zu öffnen, funktioniert über die Menüs. So können auch neue Designs erzeugt werden.

□ `Open - Design...` [ES2 0.7um ...]

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

Wurde unter `<lib name>` eine noch nicht vorhandene Bibliothek angegeben, so wird erst die Bibliothek erzeugt, bevor die Form weiter ausgefüllt werden kann:

≡ – bestätigen [Library not found]

≡ – bestätigen [Create Library]

Das Laden der Technologiedatei dauert etwas...

Bei einem neuen Design werden die entsprechenden Einträge für `<cell name>` und `<view name>` in der Bibliothek angelegt, auch hier ist eine Bestätigung notwendig:

≡ – bestätigen [Edit New Cell]

Das explizite Öffnen von Bibliotheken geschieht analog (als □ `Open - Library...` [ES2 0.7um ...]), ist im allgemeinen aber nicht notwendig, da Bibliotheken zusammen mit den Designs zusammen geöffnet werden.

2.5 Der Layout-Editor

Der Layout-Editor wird beim Öffnen einer Zellview `layout` gestartet

1. über den `Library Browser`
2. als □ `Open - Design...` [ES2 0.7um ...]

2.5.1 Auswahl der Layer

Neben dem eigentlichen Layout-Fenster des Editors wird noch ein zweites Fenster erzeugt, auf dem die zur Verfügung stehenden Layer dargestellt sind (LSW – Layer Selection Window).

Bevor Geometrien gezeichnet werden können, muß hier ein passendes Layer ausgewählt werden. Der nachfolgende Zeichenbefehl erzeugt die Geometrien auf diesem Layer.

$\uparrow_l$ `<layer>` [LSW]

Auswahl von `<layer>`, dabei wird das ausgewählte Layer oben im LSW angezeigt.

$\uparrow_m$ `<layer>` [LSW]

Für `<layer>` wird die Sichtbarkeit umgeschaltet.

$\uparrow_r$ `<layer>` [LSW]

Für `<layer>` wird die Selektierbarkeit umgeschaltet.

□ `Misc - Layer Tap` / $\odot$ `t` [Editing:...]

Das aktive Layer wird durch Auswahl eines Layers im Layout bestimmt.

Die ES2-Fertigungsprozesse ecpd07/ecpd10 (n-well, single poly, double metal CMOS) verwenden folgende Layer im Layout:

`CNWI`	N-Wanne	für P-Kanal Transistoren
`CTOX`	Dünnoxyd	für alle Diffusionsgebiete (P- und N-Diff.)
`CPOL`	Polysilizium	für Gates der Transistoren
`CNPI`	N-Diffusion	für N-Kanal Transistoren (Source/Drain) und Wannenkontakte
`CPPI`	P-Diffusion	für P-Kanal Transistoren (Source/Drain) und Substratkontakte
`CME1`	Metall 1	für Metall 1 Leitungen, Pins
`CME2`	Metall 2	für Metall 2 Leitungen, Pins
`CVIA`	Vias	für Verbindungen zwischen Metall 1 und Metall 2
`CCON`	Kontakte	für Metall 1 Anschlüsse auf Poly oder Diffusion
`CPAS`	Passivation	Spezielles Layer für Padzellen
`resistor`		für Beschreibung von Widerständen
`SIGTXT`		für Signalnamen für CME1, CME2[2]
`LABEL`		für Label im Design
`PNWI ... PME2`		Spezielle Layer für Padzellen
`XA ... PTINS`		Spezielle ES2 Layer

[2]Sollte nicht verwendet werden, außer für die „globale" Verbindung von Netzen über Namen bei der Extraktion. Label des Designers werden mit dem Layer `LABEL` eingegeben!

2.5.2 Benutzung der Maus

Die Belegung der Maustasten wird unten im Editor-Fenster angezeigt. Im allgemeinen gilt:

$\uparrow_l$ `<object>` [Editing:...]

Auswahl (Selektion) von `<object>` für nachfolgende Befehle wie das Löschen, Kopieren, Verschieben... Die Selektion arbeitet dabei folgendermaßen:

linke Maustaste	ein einzelnes Element — wird der Cursor auf ein Design-Objekt bewegt, dann zeigt eine weisse Strichmarkierung an, was bei einer nachfolgenden Selektion ausgewählt wird. Dabei ist zu unterscheiden, ob vollständige Geometrien oder nur die Kanten von Objekten markiert sind!
linke Maustaste festhalten	Selektionsfenster aufziehen
<Shift> linke Maustaste	Selektion ergänzen

Selektierte Objekte werden hell umrahmt dargestellt. Sollten sich Objekte nicht, oder nur schwierig selektieren lassen, so kann man über die Selektierbarkeit einzelner Layer (siehe 2.5.1) eine Vorauswahl treffen. Die Anzahl der selektierten Objekte wird in der Statuszeile des Layout-Editors (oben im Fenster) angezeigt.

$\uparrow_m$ `<object>` [Editing:...]

Erzeugt ein (kontextsensitives) Menü, mit dem `<object>` direkt manipuliert werden kann.

$\uparrow_r$ `<object>` [Editing:...]

Wiederholt den letzten Befehl.

2.5.3 Fensterkontrolle

Scrolling:

$\odot \leftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$ [Editing:...]

Zoom:

□ `Window - Zoom In` / $\odot$ `z` [Editing:...]

□ `Window - Zoom In by 2` / $\odot$ `^z` [Editing:...]

□ `Window - Zoom To Grid` / $\odot$ `^g` [Editing:...]

□ `Window - Zoom Out by 2` / $\odot$ `Z` [Editing:...]

Misc.

□ `Window - Pan` / $\odot$ `tab` [Editing:...]

Die Cursorpositon wird der neue Fenstermittelpunkt.

□ `Window - Fit All` / $\odot$ `f` [Editing:...]

Das Design wird verkleinert/vergrößert, so daß es vollständig im Fenster sichtbar ist.

2.5.4 Layout erzeugen

Fast alle Befehle des Layout-Editors sind so lange aktiv, bis sie explizit abgebrochen werden. Dazu muß entweder `Esc` eingegeben werden oder der `Cancel`-Button der entsprechenden Fill-Form.

Die nachfolgenden Zeichenbefehle beziehen sich immer auf das gerade ausgewählte Layer (siehe 2.5.1).

Rectangle: wird mit der Maus aufgezogen.

□ `Create - Rectangle` / ⊙ `r` [Editing:...]

oder □ `SymEntry - Rectangle` [Editing:...]

Polygon: die Punkte werden der Reihe nach eingegeben. Die zweimalige Eingabe des gleichen Punktes beendet den Befehl.

□ `Create - Polygon` / ⊙ `P` [Editing:...]

oder □ `SymEntry - Polygon` [Editing:...]

≡ `Snap Mode` = `<orthogonal/L90.First>` [Create Polygon]

Path: die Punkte des Linienzuges werden der Reihe nach eingegeben. Die zweimalige Eingabe des gleichen Punktes beendet den Befehl.

□ `Create - Path` / ⊙ `p` [Editing:...]

≡ `Width` = `<value layer1>` [Create Path]
`Snap Mode` = `<orthogonal/L90.First>`
`Change To` = `<layer1>`

Der Linienzug wird mit dem vorher eingestellten Layer (in [LSW]) begonnen. Anschließend werden die Punkte des Linienzuges für `<layer1>` eingegeben.

Während der Linienzug erzeugt wird, kann die Fill-Form dazu benutzt werden, automatisch die Layer zu wechseln — die im Design notwendigen Kontakte werden dabei automatisch erzeugt.

≡ `Width` = `<value layer2>` [Create Path]
`Snap Mode` = `<orthogonal/L90.First>`
`Change To` = `<layer2>`
`Contact Orientation` = `<alignment>`

Es wird der Kontakt von `<layer1>` auf `<layer2>` erzeugt und `<layer2>` wird für die Eingabe aktiv. Über die Angabe der Orientierung (`<alignment>`) kann der Kontakt relativ zum letzten Linienpunkt ausgerichtet werden.

Nach der Auswahl eine „neuen“ Layers wird der Kontakt im Layout plaziert. Anschließend werden alle weiteren Punkte des Pfades mit `<layer2>` verbunden.

Eine zweite Möglichkeit Pfade einzugeben ist später in dem Abschnitt zum *symbolischen Layout* beschreiben.

Label: als Merkhilfe für den Designer.

□ `Create - Label...` / ⊙ `l` [Editing:...]

oder □ `SymEntry - Label...` [Editing:...]

≡ `Label` = `<string>` [3] [Create Label]
`Drafting` = `<on>`
`Attach` = `<off>`

[3] Eingabe mit Layer `LABEL`.

2.5.5 Hierarchie

Instance: durch Instanziierung anderer Designs (`layout`-View) wird eine Hierarchie aufgebaut.

☐ `Create - Instance... / ⊙ i` [Editing:...]

oder ☐ `Sym Entry - Instance...` [Editing:...]

≡ [Create Instance]

`Library`	= `<lib name>`
`Cell`	= `<cell name>`
`View`	= `layout`
`Names`	= `<inst names>`
`Rows`	= `<nr>`
`Delta Y`	= `<nr>`
`Columns`	= `<nr>`
`Delta X`	= `<nr>`
`Mag`	= `1`
`Rotate / Sideways / Upsidedown`	

- Ausgehend von dem ursprünglichen Layout kann die Hierarchie durchlaufen, und dort sogar Änderungen vorgenommen werden.

  ☐ `Design - Hierarchy - Descend / ⊙ X` [Editing:...]

  Abstieg innerhalb der Hierarchie, die selektierte Zelle wird in den Layout-Editor geladen.

  ☐ `Design - Hierarchy - Edit In Place / ⊙ x` [Editing:...]

  Die selektierte Zelle wird editierbar gemacht, bleibt aber in der Umgebung des derzeitigen Designs sichtbar.

  ☐ `Design - Hierarchy - Return / ⊙ B` [Editing:...]

  Rückkehr innerhalb der Hierarchie zur nächsthöheren Ebene.

2.5.6 Symbolisches Layout

[4] Während man beim normalen Layout alle Geometrien „von Hand“ eingeben muß, können beim symbolischen Layout *technologieabhängig* vordefinierte Strukturen benutzt werden. Dies sind Kombinationen mehrerer Layer, die den Design-Rules des Prozesses entsprechen. Dazu werden in der Technologiedatei (`ecpd07.tf` / `ecpd10.tf`) folgende Elemente definiert:

symbolic devices : Kontakte und Transistoren.

`M1_N`	Kontakt Metall 1 auf n-Diffusion
`M1_P`	Kontakt Metall 1 auf p-Diffusion
`M1_Pol`	Kontakt Metall 1 auf Polysilizium
`NTAP`	Substratkontakt
`PTAP`	Wannenkontakt
`via1`	Kontakt Metall2 auf Metall 1
`NTR`	n-Kanal Transistor, parametrisierbare Kanallänge und -weite (`l`, `w`)
`PTR`	p-Kanal Transistor, parametrisierbare Kanallänge und -weite (`l`, `w`)

[4] Auf die Möglichkeiten des *symbolischen Layouts* wird hier nur im Rahmen des Layouteditors eingegangen.

symbolic wires : leitende Verbindungen aus mehreren Layern.

ndiff_wire	Leitende n-Diffusion (CTOX + CNPI)
pdiff_wire	Leitende p-Diffusion (CTOX + CPPI)

symbolic pins : Anschlüsse innerhalb der Hierarchie.

CME1_T	Metall 1 Terminal
CME2_T	Metall 2 Terminal
CPOL_T	Polysilizium Terminal
CME12_T	Metall 1 + Metall 2 Terminal

Symbolic Path: die Punkte des Linienzuges werden der Reihe nach eingegeben. Die zweimalige Eingabe des gleichen Punktes beendet den Befehl.

Im Gegensatz zum einfachen Path-Befehl können stehen neben den „normalen“ Layern des Layouts auch die symbolischen Verbindungen zur Verfügung.

□ Sym Entry - Sym Path... [Editing:...]

≡ Wire name = <sym layer1> [Place Wire Form]
Wire snap = <orthogonal/L90.First>
Wire width = <value layer1>

Ein Linienzug wird mit <sym layer1> erzeugt. Die Mindestbreite für dieses Layer (aus der Technologiedatei) wird rechts neben dem eingestellten Layer angezeigt.

Bei der Eingabe kann die Fill-Form dazu benutzt werden Kontakte einzufügen oder die Layer zu wechseln.

≡ Switch wires ... = <on/off> [Place Wire Form]
Drop a contact = $\uparrow_l$ <contact>

Es wird ein Kontakt an dem letzten Linienpunkt erzeugt. War Switch wires ... eingestellt, so wird mit dem über den Kontakt angesprochenen/verbundenen Layer weitergearbeitet. Das Feld Wire name zeigt den „neuen“ Layer an; seine Eigenschaften (z.B. die Breite) können in der Fill-Form eingestellt werden.

Contact: symbolische Kontakte werden erzeugt.

□ Create - Contact... / ⊙ o [Editing]

oder □ SymEntry - Contact... [Editing:...]

≡ Auto Contact = <off> [Create Contact]
Contact Type = <contact>
Justification = <centerCenter>
Width = 1
Length = 1
Rows = <nr>
Columns = <nr>
Rotate / Sideways / Upsidedown

Große Kontakte werden als „Mehrfachkontakte“, über die Angabe von Rows oder Columns in der Fill-Form, realisiert.

Ist Auto Contact aktiv, so können Kontakte automatisch an den Kreuzungspunkten von Pfaden generiert werden.

Instance: die symbolischen Elemente können beim Entwurf instanziiert und über Parameter angepaßt werden.

☐ `Create - Instance... / ⊙ i` [Editing:...]

oder ☐ `Sym Entry - Instance...` [Editing:...]

≡ [Create Instance]

Library	= <design-lib name>
Cell	= <symbolic device name>
View	= symbolic
Names	= <inst names>
Rows	= <nr>
Delta Y	= <nr>
Columns	= <nr>
Delta X	= <nr>
Mag	= 1
Rotate / Sideways / Upsidedown	
<device par1>	= <value1>
...	= ...

Abhängig von der Art des `symbolic device` werden mehrere Parameter erscheinen, die dann entsprechend auszufüllen sind.

Symbolic Pin: symbolische Pins werden erzeugt, dabei können sie über Parameter angepaßt werden.

☐ `Sym Entry - Sym Pin...` [Editing:...]

≡ [Create Sym Pin]

Mode	= manual inst pin
Terminal Names	= <name>
I/O Type	= <input/output/inputOutput>
Pin Type	= <symbolic pin name>
Pin Width	= <value pin>

Achtung: folgende Konventionen sind zu beachten:

Pin	`I/O Type`	`Terminal Names`
Eingänge	`input`	beliebiger Name
Ausgänge	`output`	beliebiger Name
Spannungsversorgung	`inputOutput`	festgelegt: `vdd`, `gnd`

2.5.7 Layout verändern

Nach der Eingabe der Befehle ist immer auszuwählen, welche Objekte bearbeitet werden sollen. Dies kann durch eine „normale“ Selektion geschehen, es kann hier aber auch schon eine vorher selektierte Gruppe gewählt werden (Selektion 2.5.2).

Bei den Befehlen `move`, `copy`, `stretch` wird, vor dem endgültigen Absetzen, die Wirkung des Befehls durch eine helle Umrandung dargestellt.

Move:

☐ `Edit - Move / ⊙ m` [Editing:...]

≡ [Move]

Change Layer	= <off/on>
Snap Mode	= <anyAngle/diagonal/orthogonal/...>
Rotate / Sideways / Upsidedown	

Ist `Change Layer` aktiv, so kann man in dem Layerauswahlfeld angeben, mit welchem Layer die Figur dargestellt wird.

Copy:

□ `Edit - Copy` / ⊙ `c` [Editing:...]

≡ `Change Layer` = `<off/on>` [Copy]
`Snap Mode` = `<anyAngle/diagonal/orthogonal/...>`
`Rows` = `<nr>`
`Columns` = `<nr>`
`Rotate / Sideways / Upsidedown`

Ist `Change Layer` aktiv, so kann man in dem Layerauswahlfeld angeben, mit welchem Layer die kopierte Figur dargestellt wird.

Stretch: erlaubt es einzelne Kanten von Geometrien zu verlängern. Es können aber auch ganze Bereiche modifiziert werden; dabei werden Kanten verlängert, die die Selektion schneiden, während Objekte, die sich vollständig in der Selektionsbox befinden, verschoben werden.

□ `Edit - Stretch` / ⊙ `s` [Editing:...]

≡ `Lock Angles` = `<on>` [Stretch]
`Snap Mode` = `<anyAngle/diagonal/orthogonal/...>`

Reshape: zu bestehenden Rechtecken oder Polygone werden weitere Rechtecke hinzugefügt. Dazu wird an ein selektiertes Element ein schneidendes Rechteck angesetzt und mit dem entsprechenden Layer gefüllt.

□ `Edit - Reshape` / ⊙ `R` [Editing:...]

≡ `Reshape Type` = `rectangle` [Reshape]
`Snap Mode` = `<anyAngle/diagonal/orthogonal/...>`

Delete:

□ `Edit - Delete` / ⊙ `del` [Editing:...]

2.5.8 Diverses

Undo / Redo: die jeweils letzten 5 Befehle können wieder Rückgängig gemacht werden.

□ `Edit - Undo` / ⊙ `u` [Editing:...]

□ `Edit - Redo` / ⊙ `U` [Editing:...]

Properties: die Arbeitsweise vieler CADENCE DF II-Programme wird durch Eigenschaften der Designs, bzw. deren Elemente, beeinflußt. Diese Properties können angesehen und modifiziert werden.

So lassen sich beispielsweise die Layer gezeichneter Rechtecke oder Linienzüge über die Property Fill-Form nachträglich ändern.

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

≡ –ansehen oder ändern [Edit Properties]

Sind mehrere Elemente selektiert, so kann in der Fill-Form über `Next` und `Previous` direkt zwischen der selektierten Elementen umgeschaltet werden.

Die Properties des aktuellen Designs kann man sich mit folgendem Befehl ansehen:

☐ `Design - Design Properties... / ⊙ Q` [Editing:...]

≡ –ansehen oder ändern [Edit Cellview Properties]

Ruler: ermöglichen es Abstände in dem Design zu messen, um beispielsweise die Einhaltung von Design-Rules zu prüfen.

☐ `Misc - Ruler / ⊙ k` [Editing:...]

≡ `Keep Ruler` = `<off/on>` [Create
`Multi-segment Ruler` = `<off/on>`
`Snap Mode` = `<anyAngle/diagonal/orthogonal/...>`
Ruler]

Erzeugt einen oder mehrere Ruler.

☐ `Misc - Clear Rulers / ⊙ K` [Editing:...]

Löscht alle Ruler im Design.

Search: sucht nach Strukturen in Layouthierarchien, dabei hat man vielfältige Möglichkeiten Suchkriterien anzugeben, die gefundenen Elemente zu selektieren, zu verändern...

☐ `Edit - Search... / ⊙ S` [Editing:...]

≡ – entsprechend ausfüllen [Search]

`<inst/contact/...>`	: wonach wird gesucht
`<area/current cellView/...>`	: wo wird gesucht
↑ₗ `Add Criteria`	: es können Suchausdrücke gebildet werden
↑ₗ `Previous / Next`	: schaltet zwischen den Elementen hin und her
↑ₗ `Add Select / Select All`	: gefundene Elemente werden selektiert
`Replace <none/cell name/...>`	: zu verändende Properties werden festgelegt
↑ₗ `Replace / Replace All`	: Properties werden verändert

2.5.9 Kennzeichnung der Anschlüsse

Um elektrische Netze nach der Extraktion in einer Simulation ansprechen zu können, müssen die Anschlüsse der Schaltung gekennzeichnet werden. Diese *Pins* werden als Netznamen in der Extraktion behandelt und stellen später die elektrischen Anschlüsse der Schaltung dar.

Pin: wie bei den Zeichenbefehlen muß vorher ein entsprechendes Layer eingestellt worden sein: `CME1`, `CME2` oder (mit Einschränkung) `CPOL`.

☐ `Create - Pin... / ⊙ ^p` [Editing:...]

oder ☐ `Sym Entry - Pin...` [Editing:...]

≡ `Terminal Names` = `<name>` [Create Pin]
`Pin Shape` = `rectangle`
`Display Name` = `<on/off>`
`I/O Type` = `<input/output/inputOutput>`

Achtung: folgende Konventionen sind zu beachten:

Pin	`I/O Type`	`Terminal Names`
Eingänge	`input`	beliebiger Name
Ausgänge	`output`	beliebiger Name
Spannungsversorgung	`inputOutput`	festgelegt: `vdd`, `gnd`

2.5.10 Sichern

☐ `Design - Save` / ⊙ `f2` [Editing:...]

☐ `Design - Save As...` [Editing:...]

≡ `Library Name`	= `<lib name>`	[Save As]
`Cell Name`	= `<cell name>`	
`View Name`	= `layout`	

2.6 Der Schematic-Editor

Der Schematic-Editor wird beim Öffnen einer Zellview `schematic` gestartet

1. über den `Library Browser`
2. als ☐ `Open - Design...` [ES2 0.7um ...]

2.6.1 Benutzung der Maus

Die Belegung der Maustasten wird unten im Editor-Fenster angezeigt. Im allgemeinen gilt:

$\uparrow_l$ `<object>` [Editing:...]

Auswahl (Selektion) von `<object>` für nachfolgende Befehle wie das Löschen, Kopieren, Verschieben... Die Selektion arbeitet dabei folgendermaßen:

linke Maustaste	ein einzelnes Element — wird der Cursor auf ein Design-Objekt bewegt, dann zeigt eine weisse Strichmarkierung an, was bei einer nachfolgenden Selektion ausgewählt wird.
linke Maustaste festhalten	Selektionsfenster aufziehen
<Shift> linke Maustaste	Selektion ergänzen

Selektierte Objekte werden hell umrahmt dargestellt. Durch Angabe eines Filters kann die Selektion auf bestimmte Objekte eingeschränkt werden, siehe 2.6.6. Die Anzahl der selektierten Objekte wird in der Statuszeile des Schematic-Editors (oben im Fenster) angezeigt.

$\uparrow_m$ `<object>` [Editing:...]

Erzeugt ein (kontextsensitives) Menü, mit dem `<object>` direkt manipuliert werden kann.

$\uparrow_r$ `<object>` [Editing:...]

Wiederholt den letzten Befehl.

2.6.2 Fensterkontrolle

Scrolling:

⊙ ←, →, ↑, ↓ [Editing:...]

Zoom:

☐ `Window - Zoom In` / ⊙ `z` [Editing:...]

☐ `Window - Zoom In by 2` [Editing:...]

☐ `Window - Zoom Out by 2` [Editing:...]

⊙ `^z` Zoom Out [Editing:...]

Misc.

- ☐ `Window - Pan` [Editing:...]

  Die Cursorpositon wird der neue Fenstermittelpunkt.
- ⊙ `v` [Editing:...]

  Für eine Punkt des Designs wird angegeben, wo er in dem Fenster plaziert werden soll (relative Pan).
- ☐ `Window - Zoom To Fit` / ⊙ `f` [Editing:...]

  Das Design wird verkleinert/vergrößert, so daß es vollständig im Fenster sichtbar ist.

2.6.3 Schematic zeichnen

Ein Schematic besteht aus Symbolen (Zellview `symbol`) und deren Verbindung untereinander. Diese Symbole können

1. aus vorgegebenen Zellbibliotheken kommen (Gatterbibliotheken mit Standardzellen, Bibliotheken mit elektrischen Bauteilen...).
2. aus selbst entworfenen Schematics generiert worden sein. Die Verwendung solcher *eigener* Symbole entspricht dem Aufbau einer Hierarchie im Design.

Fast alle Befehle des Schematic-Editors sind so lange aktiv, bis sie explizit abgebrochen werden. Dazu muß entweder `Esc` eingegeben werden oder der `Cancel`-Button der entsprechenden Fill-Form.

Instance:

- ☐ `Add - Component...` / ⊙ `i` [Editing:...]
- ≡ `Library Name` = `<lib name>` [ES2 ... Library Menu]

  `Cell Name` = `<cell name>`

  `View Name` = `symbol`

  `Instance Names` = `<string>`

  `Columns` = `<nr>`

  `Rows` = `<nr>`

  `Rotate / Upsidedown / Sideways`

Wire: über Leitungen werden die Anschlüsse der Instanzen miteinander verbunden. Für die Darstellung von Bussen verwendet man dabei üblicherweise breitere Leitungen. Die einzelnen Punkte der Leitung werden der Reihe nach eingegeben. Geht eine Leitung an einen Anschluß eines Symbols, so wird sie abgesetzt, ansonsten muß der gleiche Punkt zweimal eingegeben werden um die Leitung zu beenden.

- ☐ `Add - Wire (narrow)` / ⊙ `w` [Editing]
- ☐ `Add - Wire (wide)` / ⊙ `W` [Editing]
- ≡ `Draw Mode` = `<route/...>` [Add Wire]

  `Route Method` = `<full/direct/flight>`

  `Width` = `<0/0.0625> (narrow/wide)`
- ⊙ `s` [Editing]

  Wenn sich eine Leitung in der Nähe von Anschlußpunkten oder anderen Leitungen befindet, dann wird durch ein Rautensymbol ein *möglicher* Anfangs- oder Endpunkt gekennzeichnet. Durch Eingabe des Bindkeys wird die Leitung dort angeschlossen.

Label:

- als explizite Namen für Netze. Dabei wird erst das Label plaziert und anschließend einer Leitung zugeordnet.

  ☐ `Add - Wire Name... / ⊙ l` [Editing:...]

  ≡ `Names` = `<string>` [Add Wire Name]
  `Bus Expansion` = `<on/off>`
  `Placement` = `<single/multiple>`
  `Purpose` = `<label/alias>`
  `Rotate`

- als Merkhilfe für den Designer

  ☐ `Add - Note - Text... / ⊙ L` [Editing:...]

  ≡ `Note Text` = `<text/string>` [Add Note Text]
  `Rotate`

2.6.4 Hierarchie

Die eigentliche Instanziierung durch die Verwendung von zuvor generierten Symbolen wurde oben schon erläutert.

Pin: die Anschlüsse der Schaltung, die in einer Hierarchie verwendet werden, müssen in dem Schematic als Pins gekennzeichnet werden. Bei einer Simulation der Schaltung können nur diese Pins angesprochen werden.

☐ `Add - Pin... / ⊙ p` [Editing:...]

≡ `Pin Names` = `<string>` [Add Pin]
`Direction` = `<input/output/inputOutput/switch>`
`Usage` = `schematic`
`Bus Expansion` = `<on/off>`
`Placement` = `<single/multiple>`
`Rotate / Upsidedown / Sideways`

Bottom-up Design: aus einer vorhandenen Zellview `schematic` kann, über die Information der Pins, automatisch ein Symbol generiert werden, das dann in der nächsthöheren Hierarchieebene benutzt werden kann.

☐ `Design - Create Cellview - From Cellview...` [Editing:...]

≡ `Library Name` = `<lib name>` [Cellview From Cellview]
`Cell Name` = `<cell name>`
`Display Cellview` = `<off/on>`
`Edit Options` = `<off/on>`
`From View Name` = `schematic`
`To View Name` = `symbol`

Ist `Edit Options` aktiviert, so erscheint nachfolgend eine Fill-Form ([Symbol Generation Form]) in der man beispielsweise die Anordnung der Pins ändern kann.

Durch `Display Cellview` wird der Symboleditor für das neu erzeugte Symbol gestartet.[5]

[5] Der Symboleditor ist im Rahmen dieser Kurzeinführung nicht weiter beschrieben, da da die automatisch generierten Symbole „im Regelfall“ ohne Probleme benutzt werden können — ansonsten muß auf die CADENCE Online-Dokumentation verwiesen werden.

Top-down Design: wenn kein Schematic vorhanden ist, wie beim top-down Entwurf oder wenn ein Symbol zu einer `layout`-View generiert werden soll, kann ein Symbol aus einer Textliste der Pins generiert werden.

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

≡ `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`	= `<cell name>`	
`Display Cellview`	= `<off/on>`	
`Edit Options`	= `<off/on>`	
`View Name`	= `symbol`	

Ist `Edit Options` aktiviert, so erscheint nachfolgend eine Fill-Form ([Symbol Generation Form]) in der man beispielsweise die Anordnung der Pins ändern kann.

Durch `Display Cellview` wird der Symboleditor für das neu erzeugte Symbol gestartet.[5]

Eine zweite Möglichkeit einen top-down Entwurf durchzuführen hat man mit dem Block-Befehl. Mit seiner Hilfe können im Schematic automatisch `symbol`-Views für referenzierte Komponenten erzeugt und in dem gerade aktuellen Schematic instanziiert und untereinander verbunden werden.

□ `Add - Block...` / ⊙ `b` [Editing:...]

≡ `Library Name`	= `<lib name>`	[Add Block]
`Cell Name`	= `<cell name>`	
`View Name`	= `symbol`	
`Instance Names`	= `<string>`	
`Pin Name Seed`	= `<pin name>`	
`Block Sample`	= `<freeform/...>`	

Nach seiner Generierung hat der Block noch keinerlei Ein- und Ausgänge; wird der Block in dem Schematic an Leitungen angeschlossen, so werden die Anschlüsse (Pins) automatisch generiert. Dabei werden die Namen der Pins aus `<pin name>` und einer Nummer gebildet. Um „sinnvolle“ Namen zu vergeben ist eine Nachbearbeitung des Symbols mit dem Symboleditor notwendig.

Eine `schematic`-View dieses Blocks muß man dann später noch erzeugen/bearbeiten.

- Ausgehend von dem aktuellen Schematic kann die Hierarchie durchlaufen, und dort sogar Änderungen vorgenommen werden.

□ `Design - Hierarchy - Descend Edit...` / ⊙ `E` [Editing:...]

≡ `View Name` = `<schematic/symbol/layout/...>` [Descend]

Lädt das selektierte Design in den entsprechenden Editor — dies wird normalerweise ein Schematic sein.

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

≡ `View Name` = `<schematic/symbol/layout/...>` [Descend]

Bei der Rückkehr innerhalb der Hierarchie ist darauf zu achten, daß sich der Editor (entsprechend der Zellview) und damit auch der Kontext der Bindkeys geändert haben kann.

□ `Design - Hierarchy - Return` / ⊙ `^e` [Editing:...]

Dies ist der Normalfall (Schematic) — falls man sich aber im Layout-Editor befindet, hat der entsprechende Befehl den Bindkey `B` !

2.6.5 Schematic verändern

Nach der Eingabe der Befehle ist immer auszuwählen, welche Objekte bearbeitet werden sollen. Dies kann durch eine „normale“ Selektion geschehen, es kann hier aber auch schon eine vorher selektierte Gruppe gewählt werden (Selektion 2.6.1).

Bei den Befehlen `move`, `copy`, `stretch` wird, vor dem endgültigen Absetzen, die Wirkung des Befehls durch eine helle Umrandung dargestellt.

Stretch: verschiebt Elemente (Symbole), wobei Leitungen die an die Symbole angeschlossen sind, mitgeführt (verlängert, bzw. neu gelegt) werden.

□ `Edit - Stretch` / ⊙ `m` [Editing:...]

≡ `Back Trace Wire From` = `comp / wire / pin` [Stretch]
`Snap Mode` = <anyAngle/diagonal/orthogonal>
`Route Method` = <full/direct/flight>
`Rotate / Upsidedown / Sideways`

Move: der Move-Befehl verschiebt Elemente, läßt Leitungen aber liegen — deshalb ist Stretch vorzuziehen.

□ `Edit - Move` / ⊙ `M` [Editing:...]

≡ `Snap Mode` = <anyAngle/diagonal/orthogonal> [Move]
`Rotate / Upsidedown / Sideways`

Rotate:

□ `Edit - Rotate` / ⊙ `r` [Editing:...]

≡ `Rotate` = <on/off> [Rotate]
`UpsideDown` = <on/off>
`SideWays` = <on/off>

Copy:

□ `Edit - Copy` / ⊙ `c` [Editing:...]

≡ `Snap Mode` = <anyAngle/diagonal/orthogonal> [Copy]
`Columns` = <nr>
`Rows` = <nr>
`Rotate / Upsidedown / Sideways`

Delete:

□ `Edit - Delete` / ⊙ `del` [Editing:...]

2.6.6 Diverses

Undo / Redo: die jeweils letzten 5 Befehle können wieder Rückgängig gemacht werden.

☐ `Edit - Undo` / ⊙ `u` [Editing:...]

☐ `Edit - Redo` / ⊙ `U` [Editing:...]

Properties: die Arbeitsweise vieler CADENCE DF II-Programme wird durch Eigenschaften der Designs, bzw. deren Elemente, beeinflußt. Diese Properties können angesehen und modifiziert werden.

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

≡ –ansehen oder ändern [Edit Object Properties]

Für Elemente des Schematic (Instanzen, Leitungen, Pins, Label...). Sind mehrere Elemente selektiert, so kann in der Fill-Form über `Next` und `Previous` direkt zwischen der selektierten Elementen umgeschaltet werden.

Die Properties des aktuellen Designs kann man sich mit folgendem Befehl ansehen:

☐ `Edit - Properties - Cellview...` / ⊙ `Q` [Editing:...]

≡ –ansehen oder ändern [Edit CellView Properties]

In den entsprechenden Fill-Forms lassen sich neben den vordefinierten auch eigene Properties eintragen und verändern.

↑ₗ `Add` [Edit Object/CellView Properties]

≡ `Name` = `<string>` [Add Property]
`Type` = `<int/float/string/...>`
`Value` = `<value>`
`Choices` = `<val1 val2 val3...>`

↑ₗ `Modify` [Edit Object/CellView Properties]

≡ `<property selection>` [Modify Property]
`Type` = `<int/float/string/...>`
`Name` = `<string>`
`Value` = `<value>`
`Choices` = `<val1 val2 val3...>`

↑ₗ `Delete` [Edit Object/CellView Properties]

≡ `<property selection>` [Delete Property]
`Delete All` = `<off/on>`

Select:

☐ `Edit - Select - All...` [Editing:...]

≡ `Objects` = `wire/instance/pin/...` [Schematic Select All]

Alle angewählten Objekte des Schematic werden selektiert.

☐ `Edit - Select - By Property...` [Editing:...]

≡ `Search for` = `<property> <op> <value>` [Schematic Select By ...]
`Object Filter` = `wire/instance/pin/...`
`Form Action` = `select/deselect`

Alle angewählten Objekte des Schematic die dem Suchausdruck entsprechen, werden selektiert, bzw. deselektiert.

☐ `Edit - Select - Filter...` / ⊙ `^f` [Editing:...]

≡ `Area Partial Selection` = `<off>` [Schematic Selection Filter]
`Objects` = `wire/instance/pin/...`

Nachfolgende Selektionen mit der Maus (siehe 2.6.1) beziehen sich nur auf die angewählten Objekte des Schematic.

Search: sucht nach Strukturen in Designhierarchien, dabei hat man vielfältige Möglichkeiten Suchkriterien anzugeben, die gefundenen Elemente zu selektieren (`Find`) oder zu verändern (`Replace`).

☐ `Edit - Search - Find...` [Editing:...]

Objekte suchen und selektieren.

☐ `Edit - Search - Replace...` [Editing:...]

Objekte suchen und durch Veränderung von Properties ersetzen.

≡ – entsprechend ausfüllen [Schematic Find/Replace]

`<property> <op> <value>`	: wonach wird gesucht
`Object Filter`	: schränkt Suche auf bestimmte Objekte ein
`Search Scope`	: wo wird gesucht

in [Schematic Find]:

`Access Mode`	: wie wird ein Design (in Hierarchie) geöffnet
$\uparrow_l$ `Previous / Next`	: wechselt zwischen gefundenen Elementen
$\uparrow_l$ `Select`	: gefundenes Objekt wird selektiert

in [Schematic Replace]:

`Replace with: <property> <value>`	: wodurch wird ersetzt
$\uparrow_l$ `Replace`	: ersetzt aktuelles Objekt
$\uparrow_l$ `Skip`	: keine Ersetzung
$\uparrow_l$ `Replace All`	: ersetzt alle gefundenen Objekte

2.6.7 Rule Check und Sichern

Check: zur Überprüfung eines Schematic sollte ein *Schematic Rule Check* durchgeführt werden, um beispielsweise offene Eingänge, Netze ohne Treiber und ähnliche Fehlerquellen zu finden.

☐ `Check - Current Cellview` / ⊙ `x` [Editing:...]

Ausgehend von dem aktuellen Design kann auch ein kompletter SRC der gesamten Hierarchie durchgeführt werden.

☐ `Check - Hierarchy...` [Editing:...]

≡ – bestätigen [Check Hierarchy]

Der Rule-Check wird bottom-up für die Designhierarchie durchgeführt, dabei werden die (Sub-) Designs nach dem Test gesichert.

Save:

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

☐ Design - Save / ⊙ S [Editing:...]

☐ Design - Save As... / ⊙ ^s [Editing:...]

≡ Library Name = <lib name> [Save As]
Cell Name = <cell name>
View Name = schematic

2.7 Simulation and Test Language (STL)

Die CADENCE-eigene Sprache STL erlaubt die Beschreibung von Simulationsstimuli unabhängig von einem speziellen Simulator und dessen Eingabesprache. Anschließend werden die STL-Beschreibungen in die Eingaben des Simulators (Verilog, Silos, Spice, HSpice...) übersetzt.

Die wichtigsten Bestandteile einer STL-Datei sind hier kurz erläutert:

1. Initialisierung
   `stlinit`
2. Definition der Pins

defpin <name> in [tr=<time> tf=<time>]	Eingänge
defpin <name> clk	Clock-Eingänge
defpin <name> out	Ausgänge

   Die Timing-Information der Eingänge für *Rise* und *Fall* ist nur bei elektrischen Simulatoren (Spice, HSpice...) sinnvoll, und beschreibt den Flankenanstieg (bzw. -Abfall) zwischen den logischen Pegeln 0 und 1.

   Clock-Eingänge werden speziell gekennzeichnet, weil sie einem anderen Zeitschema unterliegen, als „normale" Eingänge (siehe dazu: Zeitschema, Definition der Clocks).
3. Definition der Spannungspegel
   Die Angaben zu den Spannungspegeln werden nur bei elektrischen Simulatoren ausgewertet und dienen dazu die Stimuli, als logische Werte, elektrischen Spannungen zuzuordnen.
   deflevel vil=<u0> vih=<u1> vol=<u0> voh=<u1> [V] nur elektrische Sim.
4. Zeitschema
   deftiming <sim-res> <clk-tim> <in-tim> [s]

   mit <sim-res> Zeitauflösung des Simulators
   <clk-tim> Zeitraster der Clock (n × <sim-res>)
   <in-tim> Zeitraster der Stimuli (m × <clk-tim>)
5. Definition der Clocks — nur bei clk-Eingängen
   defclock "<clk-pattern>" <clk-name> pattern aus '.' und '1'

Clockeingänge	Eingänge
<clk-tim>	<in-tim> = m × <clk-tim>
m-Clockmuster für	einen Eingangsvektor

   Sind beispielsweise im Zeitschema <clk-tim> und <in-tim> im Verhältnis 1:10 definiert, so sieht die Beschreibung eines Taktschemas mit zwei nichtüberlappenden Takten so aus:

   ```
   defclock ".1111....." clk1
   defclock "......1111" clk2
   ```

6. Definition des Eingabeformats
   In der hier festgelegten Reihenfolge der Leitungen werden nachher die einzelnen Vektoren ausgewertet.
   ```
   defformat <wire-list>
   ```
   Für die Vektoren können, außer den Stimuli der Eingänge, auch Erwartungswerte der Ausgänge vorgegeben werden. In diesem Fall müssen auch diese Leitungen in `defformat` aufgeführt werden.
7. Definition der Stimuli / Erwartungswerte

[`<procedure definition>`]	Prozeduren (Rumpf)
[`<sequence  definition>`]	Vektor-Sequenzen

   ```
   deftest
     <vector input>
   endtest
   ```
   Für die Stimuli Beschreibung stehen mehrere Möglichkeiten zur Verfügung:

   - `xv <vector>`
     Dies ist die einfachste Beschreibungsform: ein Vektor wird an die Schaltung angelegt. Der Ausdruck `<vector>` muß zu der vorherigen Definition in `defformat` passen.

     Mögliche Eingaben sind:

`0`, `1`, `Z`	einzelne Leitungen
`X`	don't care
`?`	unbekannte Erwartungswerte
`<int>`, `0b<bin>`, `0<oct>`, `0x<hex>`	Busse

   - `rptv <nr> <vector>`
     `<vector>` wird mehrmals hintereinander angelegt.
   - SKILL-Programmierung erlaubt die Benutzung von Funktionen und Variablen. Ausdrücke zur Berechnung von Werten sind möglich.

     Als Beispiel hier eine `for`-Schleife — alle Wertekombinationen werden an drei Eingänge angelegt: `for( i 0 7 a=i<2> b=i<1> c=i<0> xv(a b c))`
   - Prozeduren können definiert werden und dann bei der Eingabe der Vektoren aufgerufen werden. Hier ein Beispiel:
     ```
     procedure( example(i)              : Deklaration
                a=i<2> b=i<1> c=i<0>
                xv(a b c)
              )
     example(5)                         : Aufruf
     ```
   - Sequenzen von Vektoren können unter bestimmten Namen definiert und dann bei der Eingabe der Vektoren aufgerufen werden.
     ```
     defseq <seq-name>                  : Deklaration
       v <vector>                         entspricht xv...
       v <vector>
       ...
     endseq
     callseq <seq-name> [<nr>]          : Aufruf (mit Wiederholung)
     ```