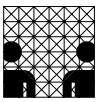
CADENCE Grundlagen

Werkzeuge : CADENCE IC
Design-Kits : AMS, ES2
designSetup : `ic` | `ic2` `ams` | `es2`

A. Mäder



Inhaltsverzeichnis

1 Einführung **2**
- 1.1 offenes System . . . 2
- 1.2 Bedienoberfläche . . . 3
- 1.3 Datenhaltung . . . 3
- 1.4 Properties . . . 4

2 Design Framework **6**
- 2.1 CADENCE starten . . . 6
- 2.2 CADENCE beenden . . . 7
- 2.3 Online-Dokumentation . . . 7
- 2.4 Library-Manager . . . 8
- 2.5 Designs bearbeiten und erzeugen . . . 9

3 Der Layout-Editor **10**
- 3.1 Starten, Sichern, Verlassen . . . 10
- 3.2 Auswahl der Layer . . . 10
- 3.3 Benutzung der Maus . . . 11
- 3.4 Fensterkontrolle . . . 12
- 3.5 Eingabehilfen . . . 12
- 3.6 Layout erzeugen . . . 13
- 3.7 Layout verändern . . . 14
- 3.8 Hierarchie . . . 15
- 3.9 Symbolisches Layout . . . 16
- 3.10 Properties . . . 18

4 Der Schematic-Editor **18**
- 4.1 Starten, Verlassen . . . 18
- 4.2 Rule Check und Sichern . . . 18
- 4.3 Benutzung der Maus . . . 19
- 4.4 Fensterkontrolle . . . 19
- 4.5 Eingabehilfen . . . 20
- 4.6 Schematic zeichnen . . . 21
- 4.7 Schematic verändern . . . 22
- 4.8 Hierarchie . . . 23
  - 4.8.1 Bottom-up Design . . . 24
  - 4.8.2 Top-down Design . . . 24
- 4.9 Properties . . . 25

5 Simulation and Test Language (STL) **26**

Legende

Für die Beschreibung der Aktionen bei der Benutzung unserer CAD-Programme wird die folgende Symbolik benutzt. Rechtsbündig steht jeweils der Titel des Fensters, in dem die Aktion durchzuführen ist, hier: [Xwin].

> `command` [Xwin]
> Eingabe von `command` an die Unix-Shell.

▷ `command` [Xwin]
Eingabe von `command` an das CADENCE-Kommandointerface.

≡ `item1` = `input1` [Xwin]
`item2` = `input2`
Eingabe zum Ausfüllen von „Fill-Forms“. Dabei wird für das Feld `item` der Wert `input` entweder über die Tastatur eingegeben oder aus einer Liste vorgegebener Möglichkeiten ausgewählt. Sind alle Werte entsprechend den Vorgaben eingestellt, so wird die Eingabe anschließend durch `OK`, bzw. `Apply` ausgeführt.

□ `menu - choice` [Xwin]
Auswahl von `choice` aus dem Menü `menu`.

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

⊙ `key` [Xwin]
alternative Eingabe von `key` als CADENCE-Bindkey über die Tastatur.

1 Einführung

Eine detailliertere Beschreibung der CADENCE Software würde den Rahmen dieser „Kurz“-Anleitung sprengen — dafür gibt es über 800 MB Online-Dokumentation. Hier werden nur einige Grundlagen angesprochen, die sich gezielt auf die in den Lehrveranstaltungen benutzten Teile des Systems beziehen. Sie sollen einen ersten Einstieg beim Umgang mit der Software ermöglichen.

1.1 offenes System

Das was hier immer als CADENCE IC bezeichnet wird ist eigentlich eine (sehr große) Menge von Programmen (Datenmanager, Editore, Simulatoren, Layoutprogramme...), die auf eine gemeinsame Designdatenbasis zugreifen. Die meisten Programmkomponenten sind über eine gemeinsame Oberfläche, das CADENCE DF II (Design Framework), erreichbar.

Dabei ist CADENCE ein offenes Design-System, in das eigene Komponenten eingebracht werden können. Eine eigene Programmierschnittstelle (SKILL) erlaubt es in das System einzugreifen und auf die Entwurfsdaten zuzugreifen.

1.2 Bedienoberfläche

Typischerweise wird man bei der Benutzung von CADENCE 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 Manager`, 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, Platzierung 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 (hier vorgestellten) Programme werden über pull-down Menüs gesteuert. Die Menüleiste wird dabei, beispielsweise bei den Editor-Fenstern, abhängig von den Teilaufgaben dynamisch verändert. Häufig benutzte Menüpunkte sind oft als „fixed-Menu's" am linken Fensterrand in Form von Symbolen aufgeführt.

1.3 Datenhaltung

Die Entwürfe sind hierarchisch, in drei Stufen, organisiert.

`Library` : 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 benutzt, um die Zellen nach Funktionen (`adders`, `boolean`, `latches`, `d-flipflops`...) zu gruppieren.

[1] Zellkategorien können im `Library Manager` über den Button `Show Categories` aktiviert werden.

`View` : Sichtweise oder Art der Zelle. Sie beschreibt (wie) welche Werkzeuge auf dieses Element der Datenbasis zugreifen dürfen. Die Namen der `View` 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

Achtung: Die Bibliotheken sollten nur über die CADENCE-Mechanismen – den `Library Manager` Abschnitt 2.4 – manipuliert werden. Das Löschen oder Kopieren mit Unix-Befehlen führt zu Inkonsistenzen.

Da eine `View` 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.

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 und Werte, 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.

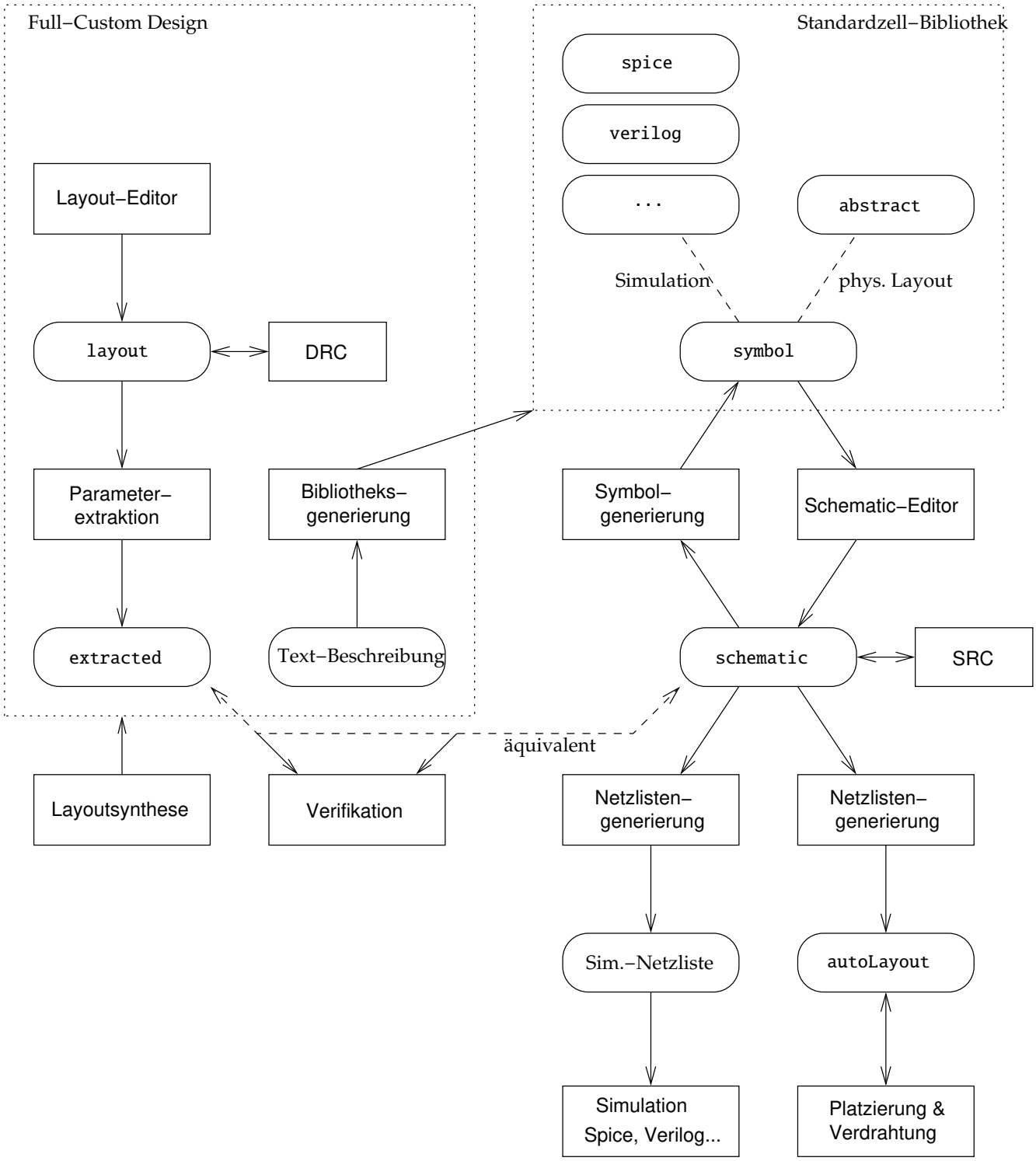


Abbildung 1: Abhängigkeiten zwischen Repräsentationen und den Werkzeugen

2 Design Framework

Neben der eigentlichen CADENCE Programmen stellen die Chip-Hersteller (AMS, Atmel (ES 2), Alcatel Mietec...) so genannte Design-Kits zur Verfügung, die die Programme an die Entwurfskonzepte des Herstellers anpassen. Darin sind zum Teil erhebliche Softwareanteile enthalten. Außerdem werden in den Design-Kits über die oben angesprochene Programmierschnittstelle Funktionen und Verhalten der Programme umdefiniert. Da dabei auch viel in der Bedienung der Programme verändert wird (Befehle, Menüs...), kann es vorkommen, dass einige der nachfolgend beschriebenen Befehle anders heißen.[2]

2.1 CADENCE starten

Die Anpassung an die Design-Kits geschieht über Skripte, die Initialisierungsdateien in das Login-Verzeichnis des Benutzers, beziehungsweise in das aktuelle Verzeichnis kopieren und anschließend das entsprechende CADENCE Programm starten.

Tipp: Erhält man beim Start des Systems die „falsche" Entwurfsumgebung oder werden Zellbibliotheken nicht mehr gefunden, so sollte man sich folgende Dateien ansehen — in den Design-Kits haben diese Dateien zum Teil andere (ähnliche) Namen:

`~/.cdsinit`, `~/.cdslocal`	Startup -Benutzer
`./.cdsinit`, `./.cdslocal`	Startup -lokal
`~/.simrc`, `~/.simlocal`	Sim.-Konfiguration -Benutzer
`./.simrc`, `./.simlocal`	Sim.-Konfiguration -lokal

AMS – Cadence Design-Kit für den Standardzellentwurf und Full-Custom Design

> `ams_cds -tech cub -tool dfii -mode fb` [xterm]

Nach der Initialisierung des Verzeichnisses wird gleich das CADENCE gestartet. Über die Parameter können unterschiedliche Betriebsmodi eingestellt werden. Die hier beschriebene Einstellung gilt für einen Standardzellentwurf mit den 0.6 μm Bibliotheken.

Die Parameter werden nur für den ersten Aufruf benötigt, später kann der Entwurf direkt mit `ams_cds` begonnen werden. Für bestimmte Werkzeuge ist allerdings der Parameter `-mode fb | ms | msfb...` erforderlich — siehe dazu die jeweilige Beschreibung!

ES 2 – Cadence Design-Kit für den Standardzellentwurf

> `ES2init` [xterm]

Diese Befehl ist nur in einem neuen Verzeichnis zur Initialisierung notwendig! Dabei wird nach dem aktuellen Prozess `-ecpd..-` und den Betriebsbedingungen `-ind-` gefragt.

> `icfb` [xterm]

ES 2 – Full Custom Design dies ist eine lokal modifizierte Version des Design-Kits

> `ES2FCStart` [xterm]

[2] Anmerkung: ich habe deshalb versucht, die Beschreibung so allgemein wie möglich zu halten. Da aber sowohl die CADENCE Software, als auch die Design-Kits 1-2 mal pro Jahr aktualisiert werden sind kleine Fehler unvermeidlich! -AJM-

CADENCE standalone Start der Programme ohne Herstellerbibliotheken

> `icfb` [xterm]

Nach dem Programmstart (was etwas dauern kann) erscheint das Eingabefenster von CADENCE DF II ([icfb - Log:...]), über das dann alle anderen Programme (Editore, Simulatoren...) aufgerufen werden können.

2.2 CADENCE beenden

□ `File - Exit...` [icfb - Log:...]

≡ – bestätigen [Exit icfb?]
Damit das Programm 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...`), kann man diese Meldung ignorieren und die Sicherung der Daten ausschalten.

≡ $\uparrow_l$ `Cancel` [Save Display Information]
Informationen zur Darstellung der Layer sollten nicht gesichert werden!

2.3 Online-Dokumentation

Start 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 kontextsensitive Hilfe zu den gerade aktuellen Befehlen oder Menüs gibt, lässt sich die komplette CADENCE Online-Dokumentation von einer Unix-Shell aus aufrufen.

> `ichelp` [xterm]

Benutzung Die Dokumentation ist als Hypertext organisiert; hier die wichtigsten Befehle für die Benutzung von FRAMEVIEWER:

$\uparrow_l$ ⟨*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 (`Alphabetical ...`, `HDL Tools`, `IC Tools` ...) und erhallt anschließend eine Übersicht von Unterthemen. Nach Auswahl eines Unterthemas werden in rechten Teil des Fensters die passenden Manuals aufgezeigt.

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

□ `Help - Contents...` [⟨Frame window⟩]
Öffnet ein weiteres Fenster mit der Online-Hilfe zu FRAMEVIEWER.

□ `Search - All...` [⟨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 - 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. Hat man vorher eine der beiden oben beschriebenen Suchfunktionen benutzt, so sollte man den nächsten Befehl nehmen!

□ `File - Exit OpenBook` [⟨Frame window⟩]
Beenden des Programms, schließt alle offenen Fenster (insbesondere Suchfenster).

2.4 Library-Manager

Dieses Werkzeug 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 einen Library Browser – ähnlich dem Library Manager – startet und bei Auswahl von Elementen im Browser-Fenster den Eintrag in das entsprechende Feld des Formulars übernimmt.

□ `Tools - Library Manager...` [icfb - Log:...]
Erzeugt das Browser-Fenster, dabei werden alle Bibliotheken angezeigt, die sich im Suchpfad befinden. Um den Suchpfad zu verändern (was normalerweise nicht notwendig sein sollte) gibt es den Menüpunkt □ `Edit - Library Path...` [Library Manager].

Benutzung In dem Fenster des Library Managers können mit der Maus die einzelnen Befehle aufgerufen werden.

$\uparrow_l$ ⟨*item*⟩ [Library Manager]
Zeigt die nächstniedrigere Ebene der Bibliothekshierarchie zu ⟨*item*⟩ an.

$\uparrow_m$ ⟨*item*⟩ [Library Manager]
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 (opt. hierarchisch)
$\uparrow_m$⟨*Cell*⟩ `- Delete...`	löscht Zelle (und alle Views)
$\uparrow_m$⟨*View*⟩ `- Copy...`	kopiert einzelne View
$\uparrow_m$⟨*View*⟩ `- Delete...`	löscht einzelne View
$\uparrow_m$⟨*View*⟩ `- Open...`	öffnet Cellview, startet das „passenden“ Editor
$\uparrow_m$⟨*View*⟩ `- Open (Read-Only)`	öffnet Cellview – read-only Zugriff

2.5 Designs bearbeiten und erzeugen

existierende Entwürfe Die einfachste Möglichkeit ein schon vorhandenes Design zu bearbeiten – „Design“ hier als `Library Cell View` –, ist die oben beschriebene Auswahl mit dem Library Manager.

Alternativ dazu können auch die Menüs, entweder des Library Managers oder des CADENCE DF II-Fensters benutzt werden um Entwürfe zu öffnen.

□ `File - Open...` [icfb - Log:...]

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

Bibliotheken erzeugen Eigene Bibliotheken, die man braucht um seine Entwürfe durchzuführen, werden mit folgenden Befehlen eingerichtet. Sie werden dabei automatisch in den Suchpfad eingefügt.

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

≡ `Name` = ⟨*libId*⟩ [New Library]
`Technology File` = `Attach...` | `Don't need...`
`Design Manager` = `No DM`
Achtung: bei der Angabe der Technologiedatei ist typischerweise eine vom Hersteller vorgegebene Bibliothek einzutragen: `Attach to an existing techfile`!

≡ `Attach To...` = ⟨*libId*⟩ [Attach Design...]
Abhängig von dem verwendeten Prozess, sind hier folgende Einträge vorzunehmen:

`cdsDefTechLib`	CADENCE-standalone	
`techLib`	ES 2 Full Custom	CMOS 0.7 μm
`ecpd07`	ES 2 Standardzell Entwurf	CMOS 0.7 μm
`TECH_CUQ`	AMS Design	CMOS 0.6 μm

Entwürfe erzeugen In der eigenen Zellbibliothek können anschließend neue Entwürfe erzeugt werden.

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

≡ `Library Name` = ⟨*libId*⟩ [Create New File]
`Cell Name` = ⟨*cellId*⟩
`Tool` = ⟨*toolSel*⟩
Das Feld `View Name` wird bei der Auswahl der CADENCE Werkzeugs passend ausgefüllt. Hauptsächlich werden folgende Tools benötigt:

`Virtuoso`	Layout-Editor
`Composer - Schematic`	Schematic-Editor
`Composer - Symbol`	Symbol-Editor

3 Der Layout-Editor

3.1 Starten, Sichern, Verlassen

Starten Beim öffnen einer Zellview `layout` wird der Layout-Editor Virtuoso gestartet. Dazu es die drei, schon in Abschnitt 2.5 vorgestellten, Möglichkeiten:

1. über den `Library Manager`
2. als □ `File - Open...` [icfb - Log:...]
3. als □ `File - New - Cellview...` [icfb - Log:...]

Entwurf Sichern Sichert man seinen Entwurf unter einem anderen Namen, so ist zu beachten, dass man die Meldung bein Beenden von CADENCE ignorieren kann.

□ `Design - Save/⊙ f2` [Virtuoso...]

□ `Design - Save As...` [Virtuoso...]

≡ `Library Name` = ⟨*libId*⟩ [Save As]
`Cell Name` = ⟨*cellId*⟩
`View Name` = `layout`

Editor beenden

□ `Window - Close/⊙ ^w` [Virtuoso...]

3.2 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/⊙ t` [Virtuoso...]
Das aktive Layer wird durch Auswahl eines Layers im Layout bestimmt.

Bedeutung der Layer Die derzeit bei uns benutzten CMOS Prozesse verwenden folgende Layer im Layout[3] — hier nur die Auswahl einiger „wichtiger“ Layer:

ES 2	AMS	Funktion	Benutzung
CNWI	NTUB	N-Wanne	für P-Kanal Transistoren
CTOX	DIFF	Dünnoxyd	für alle Diffusionsgebiete (P- und N-Diff.)
CPOL	POLY1	Polysilizium	für Gates der Transistoren
	POLY2	Polysilizium	für Kondensatoren
CNPI	NPLUS	N-Diffusion	für N-Kanal Transistoren und Wannenkontakte
CPPI	PPLUS	P-Diffusion	für P-Kanal Transistoren und Substratkontakte
CME1	MET1	Metall 1	für Metall 1 Leitungen, Pins
CME2	MET2	Metall 2	für Metall 2 Leitungen, Pins
CVIA	VIA	Vias	für Verbindungen zwischen Metall 1 und Metall 2
CCON	CONT	Kontakte	für Metall 1 Anschlüsse auf Poly oder Diffusion
CPAS	PAD	Passivation	Spezielles Layer für Padzellen
resistor	HRES		für Beschreibung von Widerständen
SIGTXT			für Signalnamen für CME1, CME2[4]
LABEL			für Label im Design
PNWI ... PME2			Spezielle Layer für Padzellen
XA ... PTINS			Spezielle ES 2 Layer

Außerdem haben die Layer auch noch eine Kennung, die die Art der Benutzung festlegt. Dabei sind für die Eingabe von Geometrien nur Layer mit der Kennung dg (drawing) zu benutzen. Kennungen sind:

dg	drawing
pn	pin
nt	net
by	boundary

3.3 Benutzung der Maus

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

$\uparrow_l$ ⟨*object*⟩ [Virtuoso...]
Auswahl (Selektion) von ⟨*object*⟩ für nachfolgende Befehle wie das Löschen, Kopieren, Verschieben... Die Selektion arbeitet dabei folgendermaßen:

- $\uparrow_l$: 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!
- $\uparrow_l$ festhalten: Selektionsfenster aufziehen
- <Shift> + $\uparrow_l$: Selektion ergänzen

Die Anzahl der selektierten Objekte wird in der Statuszeile des Layout-Editors (oben im Fenster) angezeigt.

[3] Die Funktionsweise und Herstellungstechnik von CMOS-Schaltungen werden hier als bekannt (aus den entsprechenden Vorlesungen) vorausgesetzt.

[4] 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!

Tipp: selektierte Objekte werden hell umrahmt dargestellt. Sollten sich Objekte nicht, oder nur schwierig selektieren lassen, so kann man über die Selektierbarkeit einzelner Layer (siehe 3.2) eine Vorauswahl treffen.

$\uparrow_m$ ⟨*object*⟩ [Virtuoso...]
Erzeugt ein (kontextsensitives) Menü, mit dem ⟨*object*⟩ direkt manipuliert werden kann.

$\uparrow_r$ ⟨*object*⟩ [Virtuoso...]
Wiederholt den letzten Befehl.

3.4 Fensterkontrolle

Scrolling

⊙ `Pfeiltasten:` ←, →, ↑, ↓ [Virtuoso...]

Vergrößern / Verkleinern

□ `Window - Zoom In/`⊙`z` [Virtuoso...]

□ `Window - Zoom In by 2/`⊙`^z` [Virtuoso...]

□ `Window - Zoom To Grid/`⊙`^g` [Virtuoso...]

□ `Window - Zoom To Sel Set/`⊙`^t` [Virtuoso...]

□ `Window - Zoom Out by 2/`⊙`Z` [Virtuoso...]

weiteres

□ `Window - Pan/`⊙`tab` [Virtuoso...]
Die Cursorposition wird der neue Fenstermittelpunkt.

□ `Window - Fit All/`⊙`f` [Virtuoso...]
Das Design wird verkleinert/vergrößert, so dass es vollständig im Fenster sichtbar ist.

3.5 Eingabehilfen

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

□ `Edit - Undo/`⊙`u` [Virtuoso...]

□ `Edit - Redo/`⊙`U` [Virtuoso...]

Maßstäbe Um Abstände in dem Design zu messen, beispielsweise um die Einhaltung von Design Rules zu prüfen, können Maßstäbe erzeugt werden. Sie sind nur temporär vorhanden und werden nicht abgespeichert.

□ `Misc - Ruler/`⊙`k` [Virtuoso...]

≡ `Keep Ruler` `=off|on` [Create Ruler]
`Multi-segment Ruler` `=off|on`
`Snap Mode` `=orthogonal|...`
Erzeugt einen oder mehrere Ruler.

□ `Misc - Clear Rulers/`⊙`K` [Virtuoso...]
Löscht alle Ruler im Design.

Suchfunktion Für die Suche nach Strukturen im Layout, hat man vielfältige Möglichkeiten Suchkriterien anzugeben, die gefundenen Elemente zu selektieren, zu verändern...

□ `Edit - Search.../⊙S` [Virtuoso...]

≡ – entsprechend ausfüllen [Search]

`Search`	`inst` | `contact` | `...`		wonach wird gesucht
	`area` | `current cellView` | `...`		wo wird gesucht
	↑$_l$ `Add Criteria`		Suchausdrücke werden gebildet
	↑$_l$ `Previous`	`Next`	wechselt zwischen den Elementen
	↑$_l$ `Add Select`	`Select All`	Selektion gefundener Elemente
`Replace`	`none` | `cell name` | `...`		zu verändernde Properties festlegen
	↑$_l$ `Replace`	`Replace All`	Properties ändern

3.6 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.

Achtung: die nachfolgenden Zeichenbefehle beziehen sich immer auf das gerade ausgewählte Layer (siehe 3.2), dementsprechend ist *vorher* eine geeignete Wahl zu treffen.

Rechtecke Wird mit der Maus aufgezogen.

□ `Create - Rectangle/⊙r` [Virtuoso...]

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

□ `Create - Polygon/⊙P` [Virtuoso...]

≡ `Snap Mode` = `orthogonal` | `L90.First` [Create Polygon]

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

□ `Create - Path/⊙p` [Virtuoso...]

≡ `Width` = ⟨*layer1Val*⟩ [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` = ⟨*layer2Val*⟩ [Create Path]
`Snap Mode` = `orthogonal` | `L90.First`
`Change To` = ⟨*layer2*⟩
`Contact Justification` = ⟨*alignment*⟩

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

Texte Als Merkhilfe für den Designer.

□ `Create - Label.../⊙l` [Virtuoso...]

≡[5] `Label` = ⟨*string*⟩ [Create Label]
`Drafting` = `on`
`Attach` = `off`

3.7 Layout verändern

Nach der Eingabe der Befehle ist immer auszuwählen, welche Objekte bearbeitet werden sollen. Dies kann durch eine „normale“ Selektion geschehen, es ist aber auch möglich eine schon vorher selektierte Gruppe zu benutzen (Selektion 3.3).

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

Verschieben

□ `Edit - Move/⊙m` [Virtuoso...]

≡ `Change Layer` = `off` | `on` [Move]
`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.

Kopieren

□ `Edit - Copy/⊙c` [Virtuoso...]

≡ `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.

Verlängern / Verkürzen Für Rechtecke, Polygone und die Endpunkte von Linienzügen können Kanten oder Eckpunkte verschoben werden. 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` [Virtuoso...]

≡ `Lock Angles` = `on` [Stretch]
`Snap Mode` = `anyAngle` | `diagonal` | `orthogonal` | `...`

[5] Bei ES 2 Prozessen Eingabe mit Layer `LABEL`.

Form verändern 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` [Virtuoso...]

≡ `Reshape Type` = `rectangle` [Reshape]
`Snap Mode` = `anyAngle | diagonal | orthogonal | ...`

Löschen

□ `Edit - Delete/⊙del` [Virtuoso...]

3.8 Hierarchie

Erzeugen Durch Instantiierung anderer Designs (`layout`-View) wird eine Hierarchie aufgebaut.

□ `Create - Instance.../⊙i` [Virtuoso...]

≡ `Library` = ⟨*libId*⟩ [Create Instance]
`Cell` = ⟨*cellId*⟩
`View` = `layout`
`Names` = ⟨*instIdLis*⟩
`Rows` = ⟨*nr*⟩
`Delta Y` = ⟨*nr*⟩
`Columns` = ⟨*nr*⟩
`Delta X` = ⟨*nr*⟩
`Mag` = `1`
`Rotate | Sideways | Upsidedown`

Kennzeichnung der Anschlüsse Die Anschlusspunkte der Schaltung spielen, im Gegensatz zum Schematic, keine direkte Rolle in der (Layout-) Hierarchie. Das liegt allerdings daran, dass Layouthierarchien nur die geometrische Information umfassen.

Die Anschlusspunkte werden erst beim Wechsel der *Abstraktionsebene* – durch Extraktion wird ein Schematic aus elektrischen Bauelementen erzeugt – benötigt, um Netze in einer Simulation ansprechen zu können. Diese *Pins* werden als Netznamen in der Extraktion behandelt und stellen später die elektrischen Anschlüsse der Schaltung dar.

Von den unterschiedlichen Möglichkeiten Pins zu erzeugen, wird hier die Methode durch Zeichnen eines Rechtecks beschrieben. Wie bei den anderen Zeichenbefehlen muss vorher ein entsprechendes Layer eingestellt worden sein:

AMS `MET1 pn`, `MET2 pn`, `POLY1 pn`, `POLY2 pn`
ES 2 `CME1 dg`, `CME2 dg` oder (mit Einschränkung) `CPOL dg`

□ `Create - Pin.../⊙^p` [Virtuoso...]
Startet das Menu Create Symbolic Pin, so ist dort `shape pin` auszuwählen.

≡ `Terminal Names` = ⟨*pinIdLis*⟩ [Create Shape Pin]
`Mode` = `rectangle`
`Create Label` = `on | off`
`I/O Type` = `input | output | inputOutput`

Konventionen:

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

Traversieren Ausgehend von dem ursprünglichen Layout kann die Hierarchie durchlaufen werden, dabei ist es auch möglich Teile zu editieren.

- □ `Design - Hierarchy - Descend/⊙X` [Virtuoso...]
  Abstieg innerhalb der Hierarchie, die selektierte Zelle wird in den Layout-Editor geladen.
- □ `Design - Hierarchy - Edit In Place/⊙x` [Virtuoso...]
  Die selektierte Zelle wird editierbar gemacht, bleibt aber in der Umgebung des derzeitigen Designs sichtbar.
- □ `Design - Hierarchy - Return/⊙B` [Virtuoso...]
  Rückkehr innerhalb der Hierarchie zur nächsthöheren Ebene.

3.9 Symbolisches Layout

Während man beim normalen Layout alle Geometrien „von Hand“ eingeben muss, können beim symbolischen Layout[6] *technologieabhängig* vordefinierte Strukturen benutzt werden. Dies sind Kombinationen mehrerer Layer, die den Entwurfsregeln des Prozesses entsprechen. Beispielsweise sind für den ES 2 Prozess (`ecpd07`) folgende Elemente definiert:

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 contact : Kontakte zwischen verschiedenen Layern.

`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

symbolic device : vorgefertigte Transistoren.

`NTR`	n-Kanal Transistor, parametrisierbare Kanallänge und -weite (`l`, `w`)
`PTR`	p-Kanal Transistor, parametrisierbare Kanallänge und -weite (`l`, `w`)

symbolic wires : leitende Verbindungen aus mehreren Layern.

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

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

Anschlusspunkte Neben den schon oben vorgestellten Pins, gibt es auch symbolische Pins, die über Parameter angepasst werden können.

□ `Create - Pin.../⊙ ^p` [Virtuoso...]
Startet das Menu Create Shape Pin, so ist dort `sym pin` auszuwählen.

≡ `Mode` = `manual pin` [Create Symbolic Pin]
`Terminal Names` = ⟨*pinIdLis*⟩
`I/O Type` = `input` | `output` | `inputOutput`
`Pin Type` = ⟨*symPin*⟩
`Pin Width` = ⟨*symPinVal*⟩

Konventionen:

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

Kontakte

□ `Create - Contact.../⊙o` [Virtuoso]

≡ `Auto Contact` = `off` | `on` [Create Contact]
`Contact Type` = ⟨*contact*⟩
`Justification` = `centerCenter`
`Width` = `1`
`Length` = `1`
`Rows` = ⟨*nr*⟩
`Columns` = ⟨*nr*⟩
`Rotate` | `Sideways` | `Upsidedown`
Sollen größere Kontakte realisiert werden, so werden diese als „Mehrfachkontakte“, über die Angabe von `Rows` oder `Columns` in der Fill-Form, erzeugt. Ist `Auto Contact` aktiv, so können Kontakte automatisch an den Kreuzungspunkten von Pfaden generiert werden.

Transistoren Die symbolischen Transistoren können im Entwurf über Parameter dimensioniert werden.

□ `Create - Device...` [Virtuoso...]

≡ `Device Class` = ⟨*symDeviceCls*⟩ [Create Device]
`Device Type` = ⟨*symDeviceTyp*⟩
`Names` = ⟨*instIdLis*⟩
`Rows` = ⟨*nr*⟩
`Delta Y` = ⟨*nr*⟩
`Columns` = ⟨*nr*⟩
`Delta X` = ⟨*nr*⟩
`Mag` = `1`
`Rotate` | `Sideways` | `Upsidedown`
⟨*symDevicePar1*⟩ = ⟨*value1*⟩
`...` = `...`
Abhängig von der Art des `symbolic device` werden mehrere Parameter erscheinen, die dann entsprechend auszufüllen sind.

3.10 Properties

Die Arbeitsweise vieler CADENCE-Programme wird durch Eigenschaften der Designs, bzw. deren Elemente, beeinflusst. Diese Properties können angesehen und modifiziert werden.

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

□ `Edit - Properties.../⊙q` [Virtuoso...]

≡ –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.

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

□ `Design - Design Properties.../⊙Q` [Virtuoso...]

≡ –ansehen oder ändern [Edit Cellview Properties]

4 Der Schematic-Editor

4.1 Starten, Verlassen

Starten Beim öffnen einer Zellview `schematic` wird der Schematic-Editor Composer gestartet. Dazu es die drei, schon in Abschnitt 2.5 vorgestellten, Möglichkeiten:

1. über den `Library Manager`
2. als □ `File - Open...` [icfb - Log:...]
3. als □ `File - New - Cellview...` [icfb - Log:...]

Editor beenden

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

4.2 Rule Check und Sichern

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

□ `Check - Current Cellview/⊙x` [Composer-Schematic...]
SRC des aktuellen Schematic

Ausgehend von dem aktuellen Design kann auch die komplette Hierarchie geprüft werden, dazu sind die folgenden Schritte durchzuführen:

□ `Check - Hierarchy...` [Composer-Schematic...]

≡ – bestätigen [Check Hierarchy]
Der Rule Check wird bottom-up für die Designhierarchie durchgeführt, dabei werden die (Sub-) Designs nach dem Test gesichert.

Entwurf Sichern

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

□ `Design - Save/⊙S` [Composer-Schematic...]

□ `Design - Save As.../⊙^s` [Composer-Schematic...]

≡ `Library Name` = ⟨*libId*⟩ [Save As]
`Cell Name` = ⟨*cellId*⟩
`View Name` = `schematic`

4.3 Benutzung der Maus

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

$\uparrow_l$ ⟨*object*⟩ [Composer-Schematic...]
Auswahl (Selektion) von ⟨*object*⟩ für nachfolgende Befehle wie das Löschen, Kopieren, Verschieben... Die Selektion arbeitet dabei folgendermaßen:

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

Selektierte Objekte werden hell umrahmt dargestellt. Durch Angabe eines Filters kann die Selektion auf bestimmte Objekte eingeschränkt werden, siehe 4.5. Die Anzahl der selektierten Objekte wird in der Statuszeile des Schematic-Editors (oben im Fenster) angezeigt.

$\uparrow_m$ ⟨*object*⟩ [Composer-Schematic...]
Erzeugt ein (kontextsensitives) Menü, mit dem ⟨*object*⟩ direkt manipuliert werden kann.

$\uparrow_r$ ⟨*object*⟩ [Composer-Schematic...]
Wiederholt den letzten Befehl.

4.4 Fensterkontrolle

Scrolling

⊙ `Pfeiltasten:` ←, →, ↑, ↓ [Composer-Schematic...]

Vergrößern / Verkleinern

□ `Window - Zoom In/⊙z` [Composer-Schematic...]

□ `Window - Zoom In by 2/⊙]` [Composer-Schematic...]

□ `Window - Zoom Out by 2/⊙[` [Composer-Schematic...]

⊙ `^z` Zoom Out [Composer-Schematic...]

weiteres

□ `Window - Pan` [Composer-Schematic...]
Die Cursorposition wird der neue Fenstermittelpunkt.

⊙ `v` [Composer-Schematic...]
Für eine Punkt des Designs wird angegeben, wo er in dem Fenster platziert werden soll (relative Pan).

□ `Window - Fit/`⊙`f` [Composer-Schematic...]
Das Design wird verkleinert/vergrößert, so dass es vollständig im Fenster sichtbar ist.

4.5 Eingabehilfen

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

□ `Edit - Undo/`⊙`u` [Composer-Schematic...]

□ `Edit - Redo/`⊙`U` [Composer-Schematic...]

Selektion Die Selektierbarkeit über die Maus kann über einen Filter auf bestimmte Objektgruppen eingeschränkt werden — analog zur Selektierbarkeit einzelner Layer im Layout-Editor:

□ `Edit - Select - Filter.../`⊙`^f` [Composer-Schematic...]

≡ `Area Partial Selection =off` [Schematic Selection Filter]
`Schematic Objects =wire|pin|instance|...`
`Instance Objects =name|pin|...`
Nachfolgende Selektionen mit der Maus (siehe 4.3) beziehen sich nur auf die angewählten Objekte des Schematic.

Ein globale Suchfunktion für das aktuelle Schematic:

□ `Edit - Select - All...` [Composer-Schematic...]

≡ `Schematic Objects =wire|pin|instance|...` [Schematic Select All]
`Instance Objects =name|pin|...`

Weiterhin ist die Selektion über Properties des Elemente möglich. Es können Ausdrücke gebildet werden, über die all diejenigen Objekte des Schematic die dem Suchausdruck entsprechen, werden selektiert, bzw. deselektiert werden:

□ `Edit - Select - By Property...` [Composer-Schematic...]

≡ `Search for =⟨propId⟩ ⟨op⟩ ⟨propVal⟩` [Schematic Select By ...]
`Form Action =select|deselect`
`Schematic Objects =wire|pin|instance|...`
`Instance Objects =name|pin|...`

Suchfunktionen Die beiden nachfolgend beschriebenen Suchfunktionen arbeiten jeweils in zwei Schritten. Zuerst werden Objekte (innerhalb der Hierarchie) gesucht, dabei hat man vielfältige Möglichkeiten Suchkriterien anzugeben. Anschließend können die gefundenen Objekte des Suchergebnisses einzeln selektiert (`Find`) oder verändert (`Replace`) werden.

Suche nach Objekten und deren Selektion:

□ `Edit - Search - Find...` [Composer-Schematic...]

≡ – entsprechend ausfüllen [Schematic Find]

⟨*propId*⟩ ⟨*op*⟩ ⟨*propVal*⟩	wonach wird gesucht
`Search Scope`	wo wird gesucht
`Object Filter`	schränkt Suche auf bestimmte Objekte ein
`Access Mode`	wie wird ein Design (in Hierarchie) geöffnet
↑$_l$ `Previous` | `Next`	wechselt zwischen gefundenen Elementen
↑$_l$ `Select`	selektiert ein gefundenes Objekt

Suche nach Objekten und Modifikation durch Veränderung von Properties:

□ `Edit - Search - Replace...` [Composer-Schematic...]

≡ – entsprechend ausfüllen [Schematic Replace]

⟨*propId*⟩ ⟨*op*⟩ ⟨*propVal*⟩	wonach wird gesucht
`Search Scope`	wo wird gesucht
`Object Filter`	schränkt Suche auf bestimmte Objekte ein
`Replace with:` ⟨*propId*⟩ ⟨*propVal*⟩	wodurch wird ersetzt
↑$_l$ `Replace`	ersetzt aktuelles Objekt
↑$_l$ `Skip`	keine Ersetzung
↑$_l$ `Replace All`	ersetzt alle gefundenen Objekte

4.6 Schematic zeichnen

Ein Schematic besteht aus Komponenten-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.

Symbole instantiieren

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

≡ [Add Component]

`Library Name`	= ⟨*libId*⟩
`Cell Name`	= ⟨*cellId*⟩
`View Name`	= `symbol`
`Instance Names`	= ⟨*instIdLis*⟩
`Columns`	= ⟨*nr*⟩
`Rows`	= ⟨*nr*⟩
`Rotate` | `Upsidedown` | `Sideways`	

Verbindungen erzeugen 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.

Netznamen Sollen explizite Namen für Netze vergeben werden, so wird erst das Label platziert und anschließend einer Leitung zugeordnet.

□ `Add - Wire Name.../⊙l` [Composer-Schematic...]

≡ `Names` = ⟨*netIdLis*⟩ [Add Wire Name]
`Bus Expansion` = `on | off`
`Placement` = `single | multiple`
`Purpose` = `label | alias`
`Rotate`

Texte Als Merkhilfe für den Designer.

□ `Add - Note - Note Text.../⊙L` [Composer-Schematic...]

≡ `Note Text` = ⟨*text*⟩ | ⟨*string*⟩ [Add Note Text]
`Rotate`

4.7 Schematic verändern

Nach der Eingabe der Befehle ist immer auszuwählen, welche Objekte bearbeitet werden sollen. Dies kann durch eine „normale“ Selektion geschehen, es ist aber auch möglich eine schon vorher selektierte Gruppe zu benutzen (Selektion 4.3).

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

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

□ `Edit - Stretch/⊙m` [Composer-Schematic...]

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

Der Move-Befehl verschiebt Elemente, lässt Leitungen aber liegen — deshalb ist Stretch vorzuziehen.

□ `Edit - Move/⊙M` [Composer-Schematic...]

≡ `Snap Mode` = `anyAngle | diagonal | orthogonal` [Move]
`Rotate | Upsidedown | Sideways`

Rotieren

□ `Edit - Rotate/⊙r` [Composer-Schematic...]

≡ `Rotate` = `on | off` [Rotate]
`UpsideDown` = `on | off`
`SideWays` = `on | off`

Kopieren

□ `Edit - Copy/⊙c` [Composer-Schematic...]

≡ `Snap Mode` = `anyAngle | diagonal | orthogonal` [Copy]
`Columns` = ⟨*nr*⟩
`Rows` = ⟨*nr*⟩
`Rotate | Upsidedown | Sideways`

Löschen

□ `Edit - Delete/⊙del` [Composer-Schematic...]

4.8 Hierarchie

Erzeugen Die eigentliche Instantiierung durch die Verwendung zuvor generierter Symbole wurde oben schon erläutert.

Kennzeichnung der Anschlüsse 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` [Composer-Schematic...]

≡ `Pin Names` = ⟨*pinIdLis*⟩ [Add Pin]
`Direction` = `input | output | inputOutput | switch`
`Usage` = `schematic`
`Bus Expansion` = `on | off`
`Placement` = `single | multiple`
`Rotate | Upsidedown | Sideways`

Symbolgenerierung Um Symbole für den Aufbau von Hierarchien zu erzeugen, gibt es mehrere Möglichkeiten, die im einzelnen in den Abschnitten 4.8.1 und 4.8.2 vorgestellt sind.

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

□ `Design - Hierarchy - Descend Edit.../⊙E` [Composer-Schematic...]

≡ `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` [Composer-Schematic...]

≡ `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` [Composer-Schematic...]

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

4.8.1 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...` [Composer-Schematic...]

≡ [Cellview From Cellview]

`Library Name`	`=⟨libId⟩`
`Cell Name`	`=⟨cellId⟩`
`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 Options]) in der man beispielsweise die Anordnung der Pins ändern kann.

Durch `Display Cellview` wird der Symboleditor Composer-Symbol für das neu erzeugte Symbol gestartet.[7]

4.8.2 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.

[7] 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.

□ `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`	= ⟨*cellId*⟩
`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.

`Display Cellview` startet den Symboleditor.[7]

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 instantiiert und untereinander verbunden werden.

□ `Add - Block.../⊙b` [Composer-Schematic...]

≡ [Add Block]

`Library Name`	= ⟨*libId*⟩
`Cell Name`	= ⟨*cellId*⟩
`View Name`	= `symbol`
`Instance Names`	= ⟨*instIdLis*⟩
`Pin Name Seed`	= ⟨*pinId*⟩
`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 ⟨*pinId*⟩ 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.

4.9 Properties

Die Arbeitsweise vieler CADENCE-Programme wird durch Eigenschaften der Designs, bzw. deren Elemente, beeinflusst. Diese Properties können angesehen und modifiziert werden.

Schematic-Objekte

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

≡ –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.

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

□ `Edit - Properties - Cellview.../⊙Q` [Composer-Schematic...]
≡ –ansehen oder ändern [Edit Cellview Properties]

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

↑$_l$ `Add` [Edit Object/Cellview Properties]
≡ `Name` = ⟨*propId*⟩ [Add Property]
`Type` = `int|float|string|...`
`Value` = ⟨*propVal*⟩
`Choices` = ⟨*val1 val2 val3...*⟩

↑$_l$ `Modify` [Edit Object/Cellview Properties]
≡ ⟨*propSel*⟩ [Modify Property]
`Type` = `int|float|string|...`
`Name` = ⟨*propId*⟩
`Value` = ⟨*propVal*⟩
`Choices` = ⟨*val1 val2 val3...*⟩

↑$_l$ `Delete` [Edit Object/Cellview Properties]
≡ ⟨*propSel*⟩ [Delete Property]
`Delete All` = `off|on`

5 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.

Beispiel

```
stlinit
defpin dffD  in  tr=1e-10 tf=1e-10
defpin dffC  clk tr=1e-10 tf=1e-10
defpin dff0  out
deflevel  vil=0  vih=5 vol=0 voh=5
deftiming 10ps 1ns 10ns
defclock                ".11111...." dffC
defstrobe    out window  "....%%%%%%" dff0
defformat dffD dff0
deftest
  xv 0 0
  xv 1 1
```

```
  xv 0 0
endtest
```

Aufbau Die wichtigsten Bestandteile einer STL-Datei sind hier kurz erläutert.

1. Initialisierung
   `stlinit`
2. Definition der Pins
   `defpin` ⟨*pinId*⟩ `in` [`tr=`⟨*timeExpr*⟩ `tf=`⟨*timeExpr*⟩] Eingänge
   `defpin` ⟨*pinId*⟩ `clk` Clock-Eingänge
   `defpin` ⟨*pinId*⟩ `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*⟩ elektrische Sim. (V)
4. Zeitschema
   `deftiming` ⟨*simTimeExpr*⟩ ⟨*clkTimeExpr*⟩ ⟨*vecTimeExpr*⟩ Timing (s)
   mit ⟨*simTimeExpr*⟩ Zeitauflösung des Simulators
   ⟨*clkTimeExpr*⟩ Zeitraster der Clock (n × ⟨*simTimeExpr*⟩)
   ⟨*vecTimeExpr*⟩ Zeitraster der Stimuli (m × ⟨*clkTimeExpr*⟩)
5. Definition der Clocks — nur bei `clk`-Eingängen
   `defclock "`⟨*clkPattern*⟩`"` ⟨*pinId*⟩ String aus `'.'` und `'1'`

Clockeingänge	Eingänge
⟨*clkTimeExpr*⟩	⟨*vecTimeExpr*⟩ = m × ⟨*clkTimeExpr*⟩
m-Clockmuster für	einen Eingangsvektor

   Sind beispielsweise im Zeitschema ⟨*clkTimeExpr*⟩ und ⟨*vecTimeExpr*⟩ im Verhältnis 1:10 definiert, dann sieht ein Taktschemas mit zwei nichtüberlappenden Takten so aus:
   ```
   defclock ".1111....." clk1
   defclock "......1111" clk2
   ```
6. Definition der Abtastzeitpunkte — nur bei Erwartungswerten
   Bei der Simulation könne auch zu erwartende Ergebnisse überprüft werden. Dazu werden hier Abtastzeitpunkte (analog zur Definition von Clocksignalen) festgelegt.
   ```
   defstrobe out [window|edge] ".....%%%%%" out1
   defstrobe out [window|edge] "........%%" out2
   ```
   Bei der nachfolgenden Definition des Eingabeformats sind dann auch Ausgänge der Schaltung mit aufgeführt.

   Neben den hier vorgestellten Strobes für Erwartungswerte, gibt es auch Definitionen für Eingangsstimuli. Diese sind jedoch nur für Testvektoren von Bedeutung und werden nicht weiter erläutert.

7. Definition des Eingabeformats
   In der hier festgelegten Reihenfolge der Leitungen werden nachher die einzelnen Vektoren ausgewertet.
   `defformat` ⟨*pinIdLis*⟩
   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.
8. Definition der Stimuli / Erwartungswerte

[⟨*procedureDef*⟩]	Prozeduren (Rumpf)
[⟨*sequenceDef*⟩]	Vektor-Sequenzen

```
deftest
  ⟨vecInput⟩
endtest
```

Stimulidefinition Für die Beschreibung der Stimuli (bzw. Erwartungswerte) 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 ⟨sequenceId⟩                      Deklaration
  v ⟨vector⟩                              entspricht xv...
  v ⟨vector⟩
  ...
endseq
callseq ⟨sequenceId⟩ [ ⟨nr⟩ ]            Aufruf (mit Wiederholung)
```