Kick-Off zu den praktischen Arbeiten ACE'10

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Übersicht

- Organisatorisches
  - Gruppeneinteilung
- Technischer Hintergrund
- Erste Aufgabe

Organisatorisches

Gruppeneinteilung

- 3er Gruppen
  - Werden nötigenfalls spontan gebildet
- Gruppenbogen ausfüllen
  - V2 oder IV5 wählen
- Bitte in Gruppe untereinander austauschen
  - Telefonnummer, E-Mail, Skype, ...

Programmierung

- In was?
  - Java
  - Abnahme auf Version 1.6.x
- Wo?
  - Auf dem heimischen Rechner
  - Informatik-Poolräume
- Art der Programme
  - Kommandozeilenorientiert
  - Dateiverarbeitend

Abgaben

- Am Abgabetag bis 23:59 Uhr MET
- E-Mail an ace@esa.informatik.tu-darmstadt.de
  - Betreff: „Gruppe *N* Aufgabe *M* ...“
  - Attachment: .jar-Datei
    - .java Quellen (mit gutem JavaDoc!)
      - In allen Dateien Gruppennummer!
    - .class vorkompilierte Klassen
  - README Textdatei
    - Beschreibt Kompilierung und Aufruf
    - Kurzer Überblick über
      - Programmaufbau
      - Algorithmen
      - Beiträge der einzelnen Gruppenmitglieder

Kolloquien & Vorträge

- Donnerstags, i.d.R. nachmittags
  - Je Gruppe ca. 30 Minuten, Zeitslots
- In der Regel am folgenden Freitag
  - Ausnahme: 1. Abgabe (1 Woche später)
  - Zur normalen Vorlesungszeit
  - Je Gruppe 10-15 Minuten Vortrag
    - Folien (PowerPoint/OpenOffice/PDF/Laptop)
      - Vorgehensweise, Kernalgorithmus und Datenstrukturen
      - Programmaufbau, Ergebnisse
      - Erfahrungen und Kommentare
- Beides benotet!

Programmierstil und Doku

- Vorschlag: „Writing Robust Java Code“
  - PDF auf Web-Seite
- Dokumentation
  - Aufgabe 1-3
    - Im wesentlichen JavaDoc und Kommentare
    - *Wichtig*: Historie im Dateikopfkommentar
      - Kann auch Log aus SVN oder CVS sein
  - Aufgabe 4
    - „Richtiges“ 20-30 seitiges Dokument (Prosa)
    - Zusammenfassend über alle bisherigen Arbeiten
    - Macht viel Arbeit, nicht unterschätzen!

Programmierung und Test

- Millionen von Rechenoperationen
- Auf Zehntausenden von Objekten
- Komplexität der Algorithmen wichtig!
  - Zeitbedarf: Hashing statt sequentieller Suche
  - Speicherbedarf: Objekte wiederverwenden
- Datenstrukturen aus Bibliothek

- Testdaten liegen auf Web-Seite
  - Minimalsatz ./. vollständiger Satz

Team-Organisation

- *Gruppen*arbeit entscheidend
- Probleme *rechtzeitig* ansprechen

- Aufteilung der Arbeiten
  - Vorschläge in Aufgabenstellung
  - Immer gut aufteilbare Bereiche
    - Test / Profiling
    - Dokumentation (nicht unterschätzen!)
      - Erfordert überblickende Fachkenntnisse

Vorgeschlagene Werkzeuge

- Versionsverwaltung: *Subversion*
  - Wichtigstes Werkzeug für Gruppenarbeit
- Java IDE: *Eclipse, NetBeans, IDEA*
  - Nützlich, insbesondere für *Refactoring*
- Automatisierte Regressionstests: *JUnit*
  - Müssen aber trotzdem gepflegt werden
- Profiler: *JIP*
  - Zur Analyse von Zeit-/Speicherbedarf
- Lexer/Parser: *ANTLR*
  - Aber eigentlich nicht nötig

Arbeitsphasen

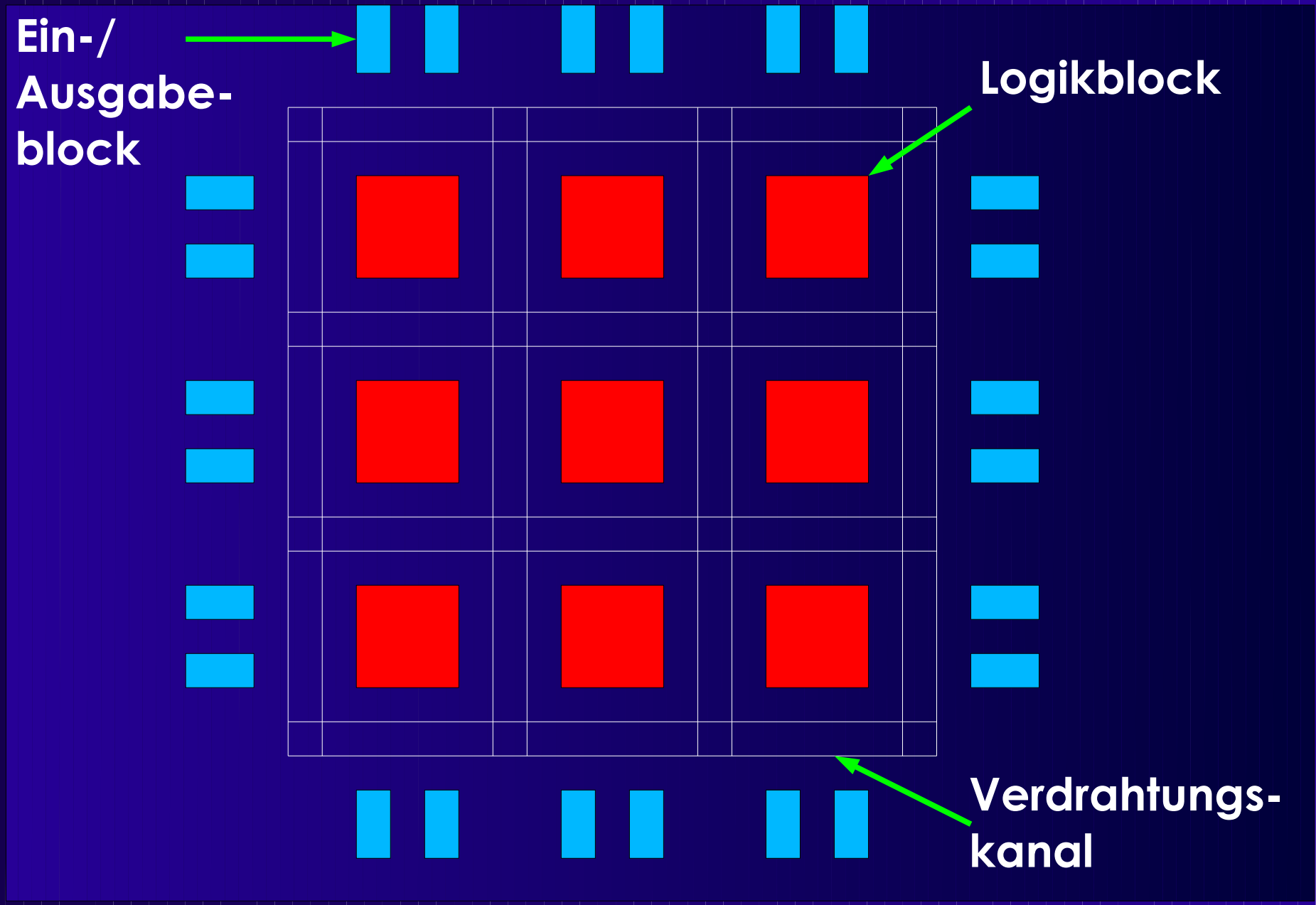
- Analyse von Schaltungen
  - Ca. 3 Wochen
  - Hilfreich für folgende Aufgaben
- Platzierung von Netzlisten
  - Ca. 4 Wochen
- Verdrahtung von platzierten Netzlisten
  - Ca. 3 Wochen (+ Weihnachten ...)
- Gezielte Verbesserungen
  - Auch letzte Fehlerkorrekturen
  - Ca. 3 Wochen

Technischer Hintergrund

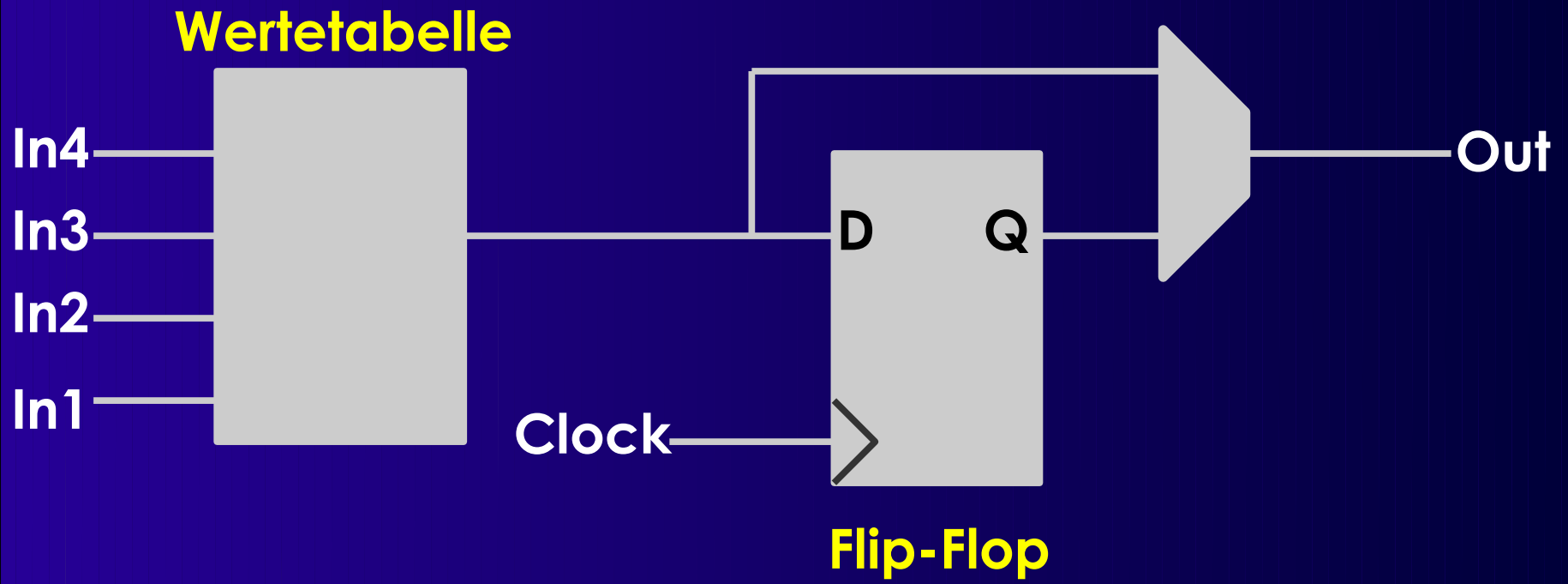
Problemgebiet

- Werkzeuge für FPGAs
- Grundprinzip
  - Sollte aus TgDI o.ä. bekannt sein
- Algorithmen
  - Vergleichbar denen für „echte“ Chips
  - Einfacher
    - Diskrete statt kontinuierlicher Strukturen
  - Komplizierter
    - Feste Strukturen begrenzen Spielraum

FPGA-Zielarchitektur

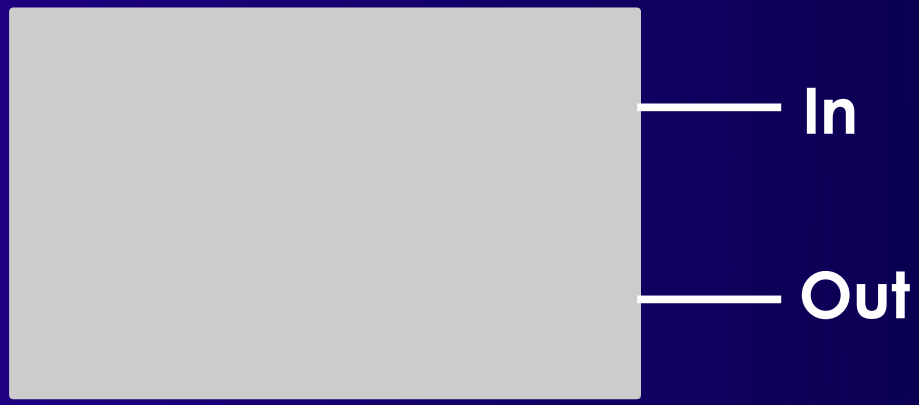


Logikblock



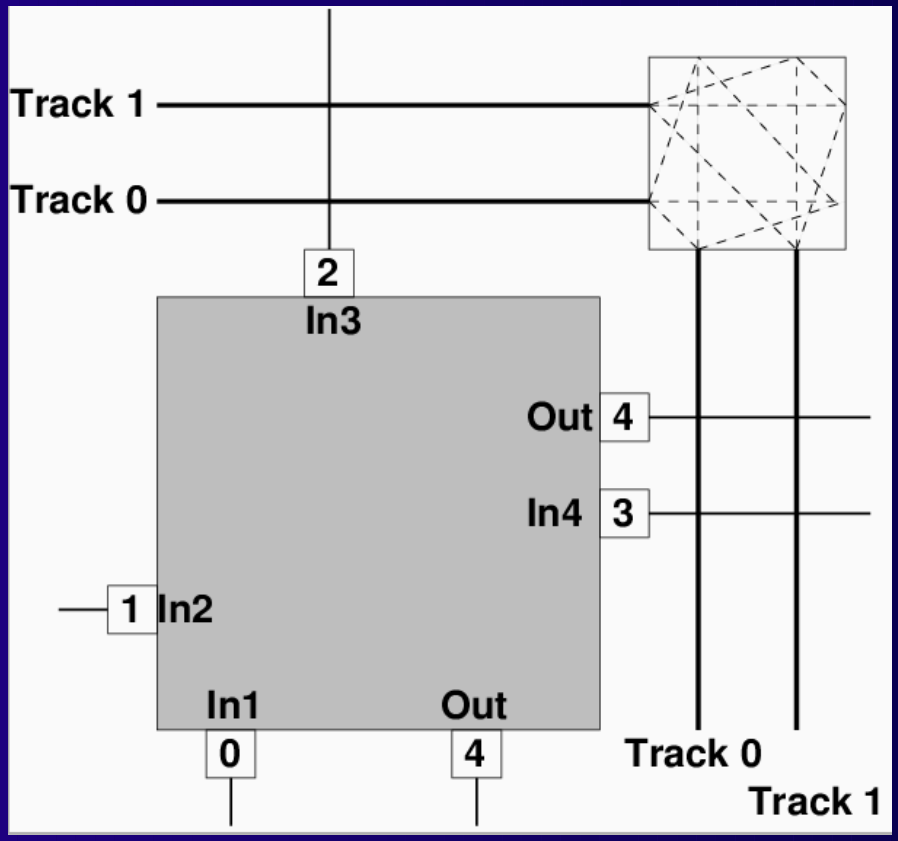
- Eingangs-Pins äquivalent
  - Untereinander austauschbar
  - Inhalt der Wertetabelle anpassen
    - Für unsere Tools nicht nötig, lassen wir weg

Ein-/Ausgabeblock



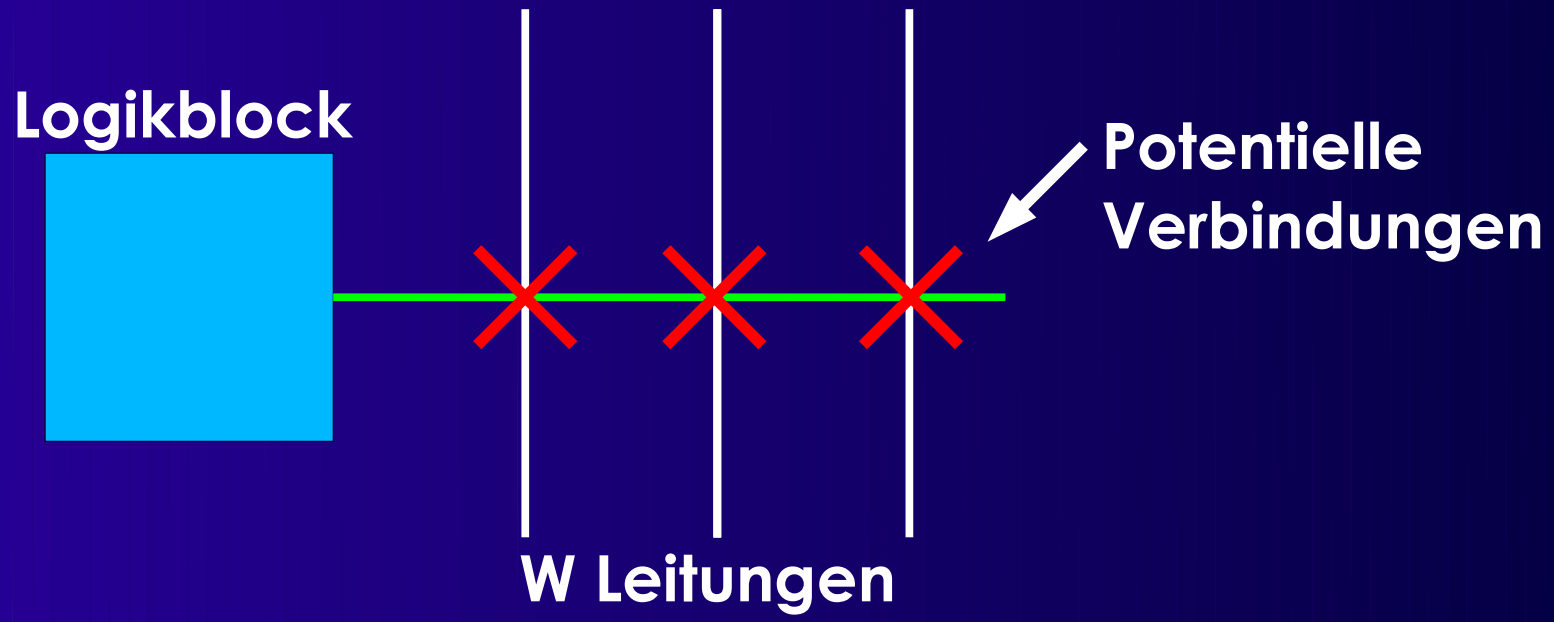
- „In“ benutzt: Ausgangsblock
- „Out“ benutzt: Eingangsblock
- *Keine* bidirektionalen Blöcke erlaubt
  - „In“ und „Out“ benutzt: Fehler

Logikblock-Konnektivität



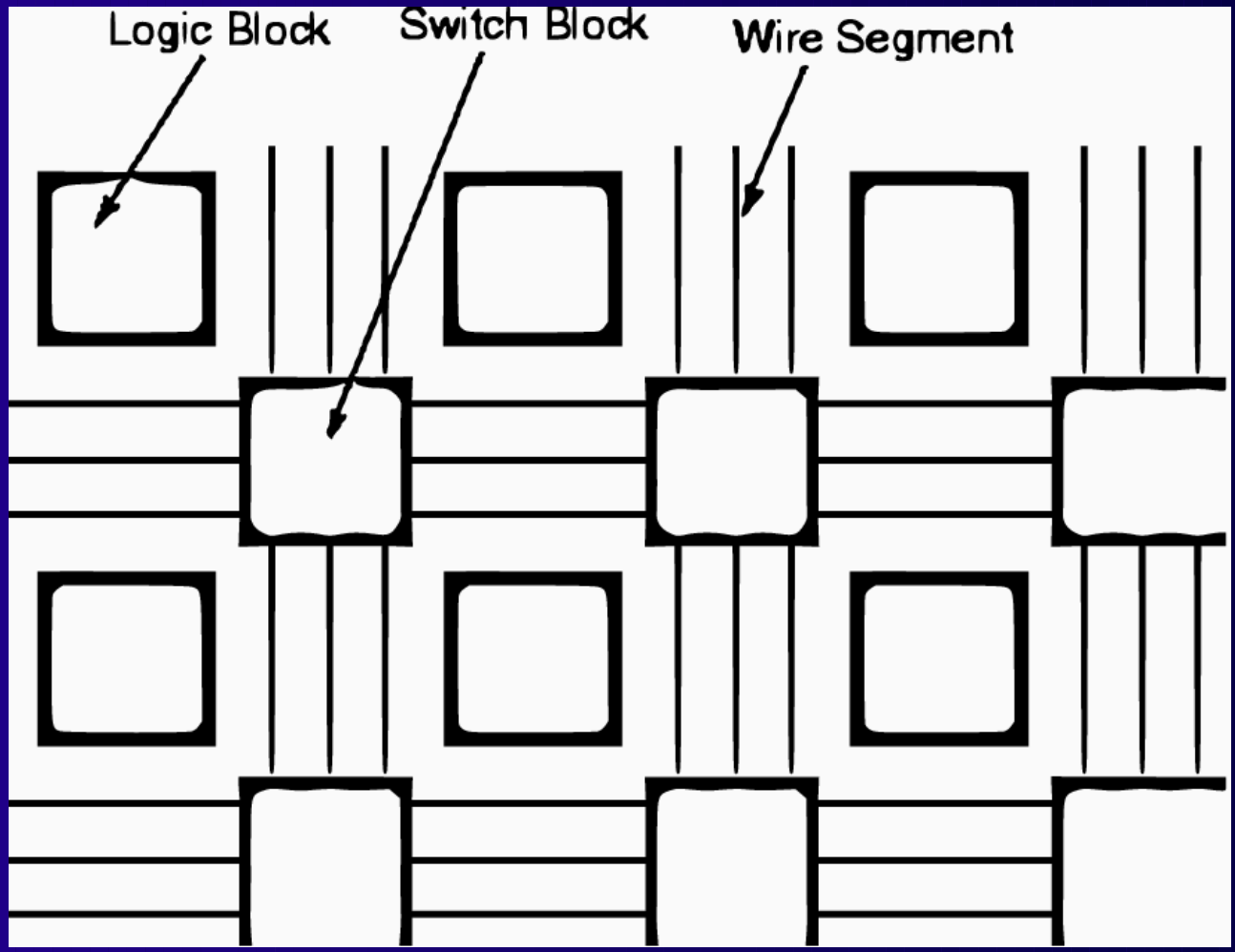
- Clock wird ignoriert
- Out ist doppelt vorhanden

Verdrahtungskanäle

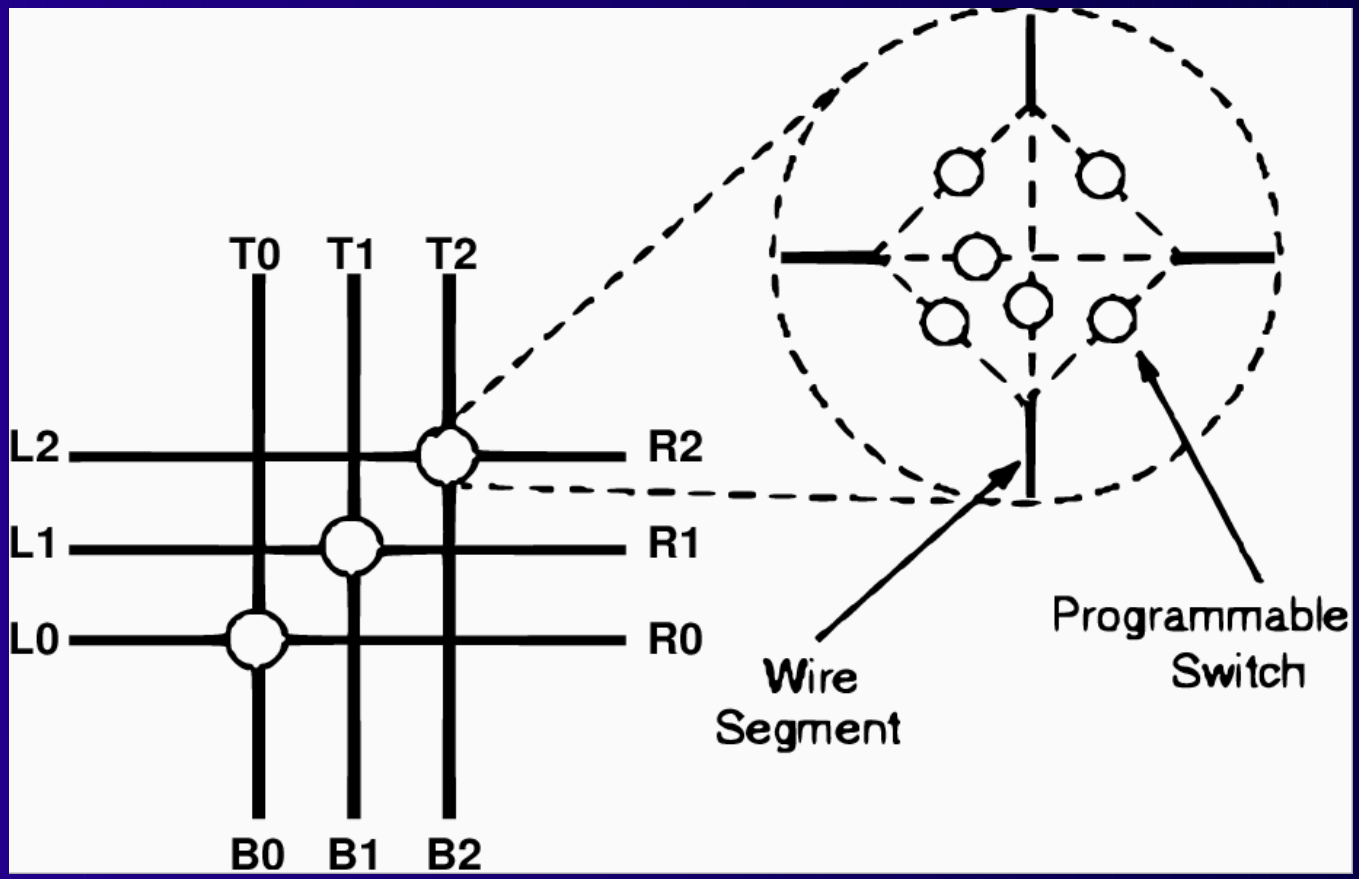


- Variable Anzahl von Leitungen
  - Parameter W: Breite (in V und H gleich!)
    - Architekturdatei
  - 1 Verbindung pro Eingang
  - Mehrere bei Ausgang

Verbindungen

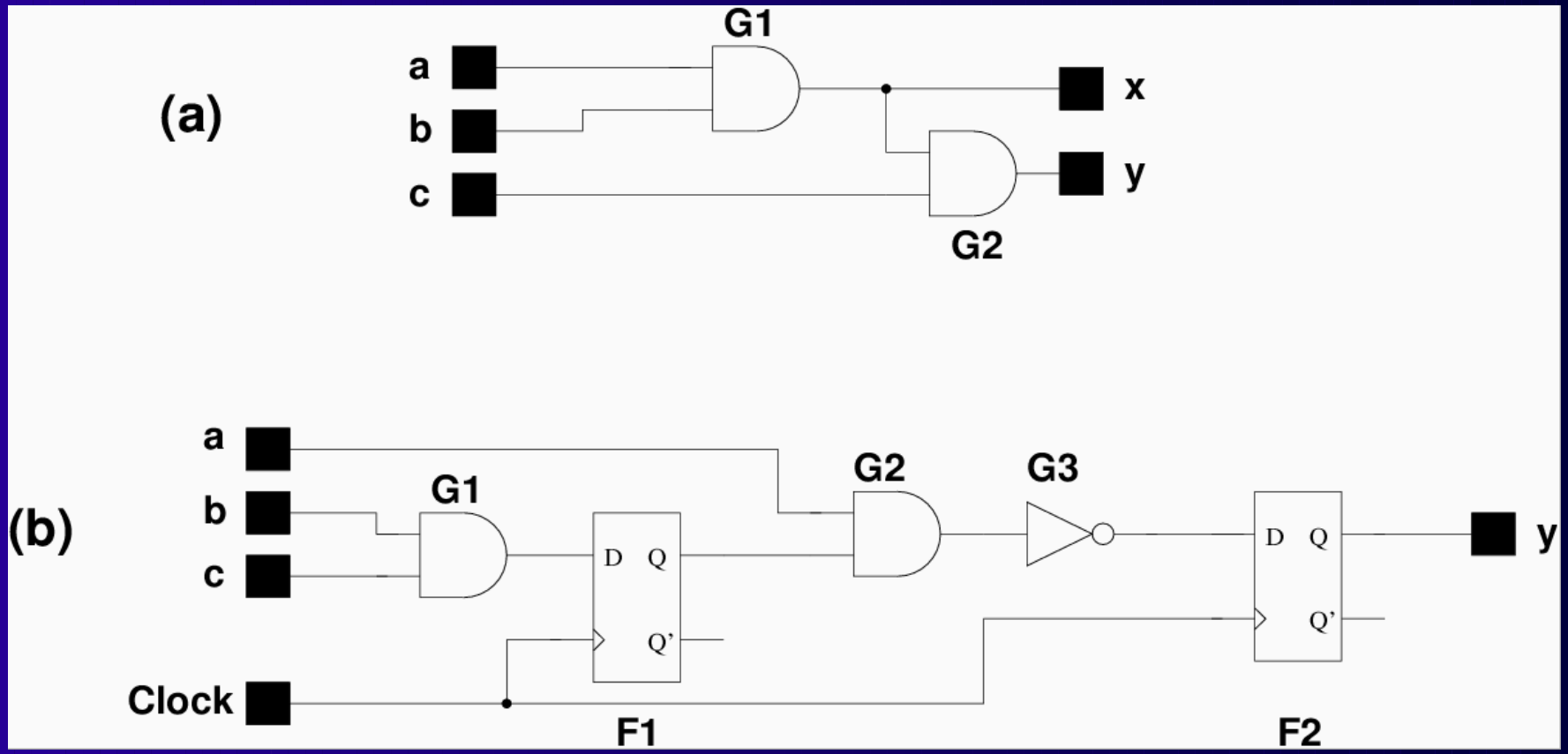


Verbindungsblöcke



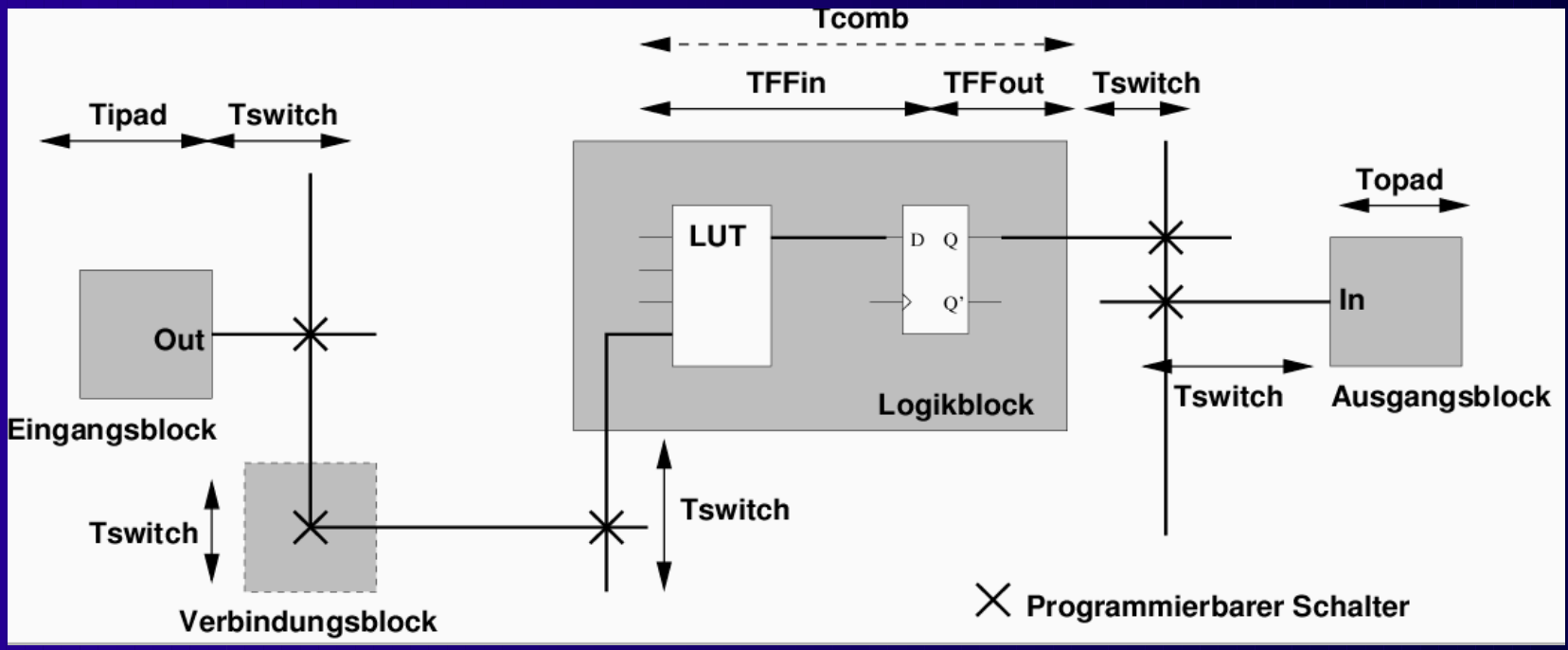
- Nur gleiche Spurnummern verbindbar
- Mehrere Durchschaltungen OK (fanout)

Zeitverhalten



- Kombinatorische Verzögerung: a,b -> y
- Sequentielle Verzögerung: G2,G3

Parameter

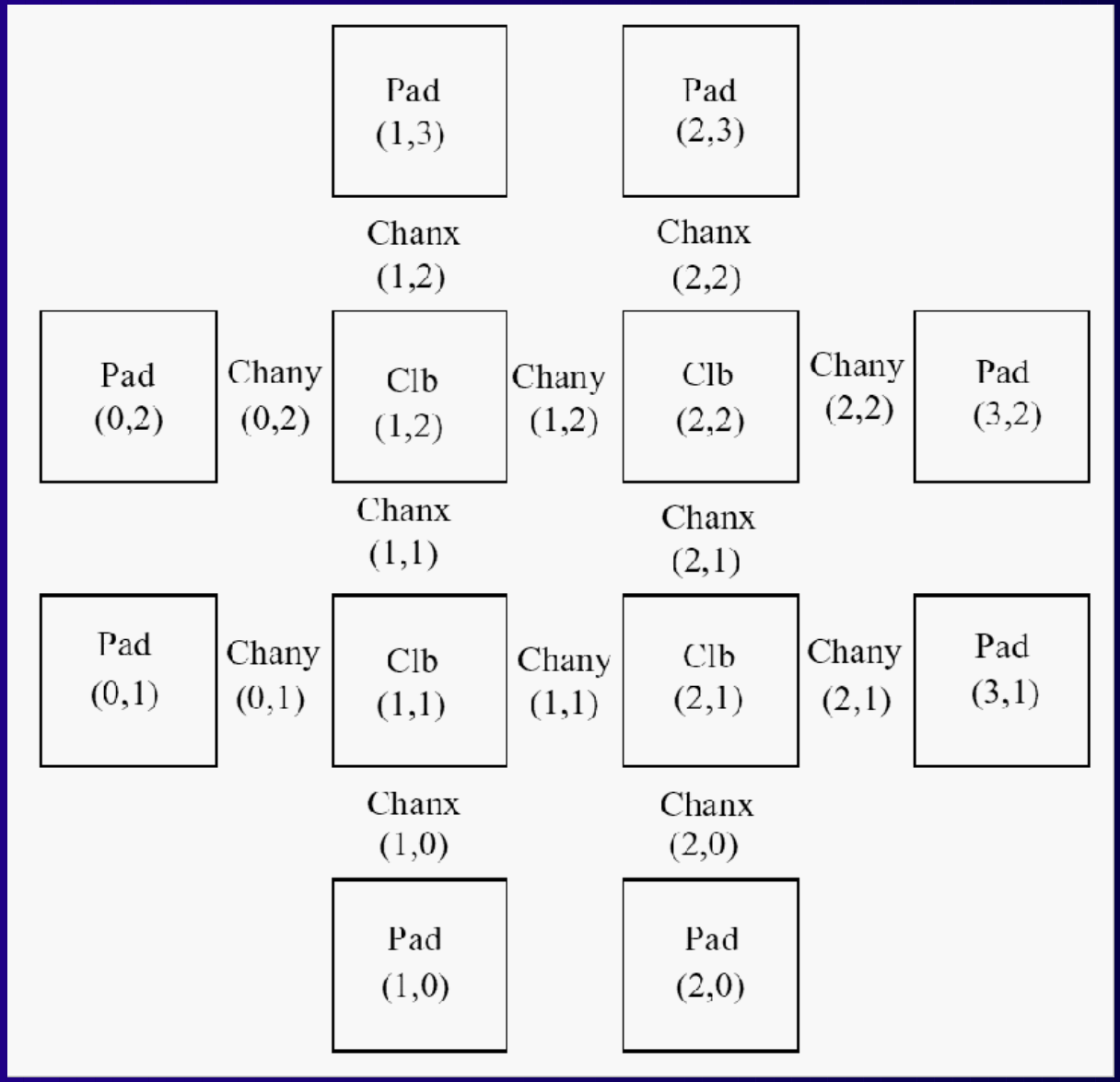


- Elementweise Verzögerungsberechnung
  - Zeiten sind parametrisiert
    - Architekturdatei

Netzlistendatei

```
.global clock

.output out:s27_out
pinlist: s27_out

.input in:s27_in_0_
pinlist: s27_in_0_

.clb [13]
pinlist: s27_in_2_ s27_in_0_ n_n40 n_n41 [13] open
subblock: [13] 0 1 2 3 4 open

.clb n_n40
pinlist: s27_in_1_ s27_in_3_ [13] [11] n_n40 clock
subblock: n_n40 0 1 2 3 4 5

.clb n_n41
pinlist: s27_in_3_ [13] open open n_n41 clock
subblock: n_n41 0 1 open open 4 5
```

Koordinatensystem

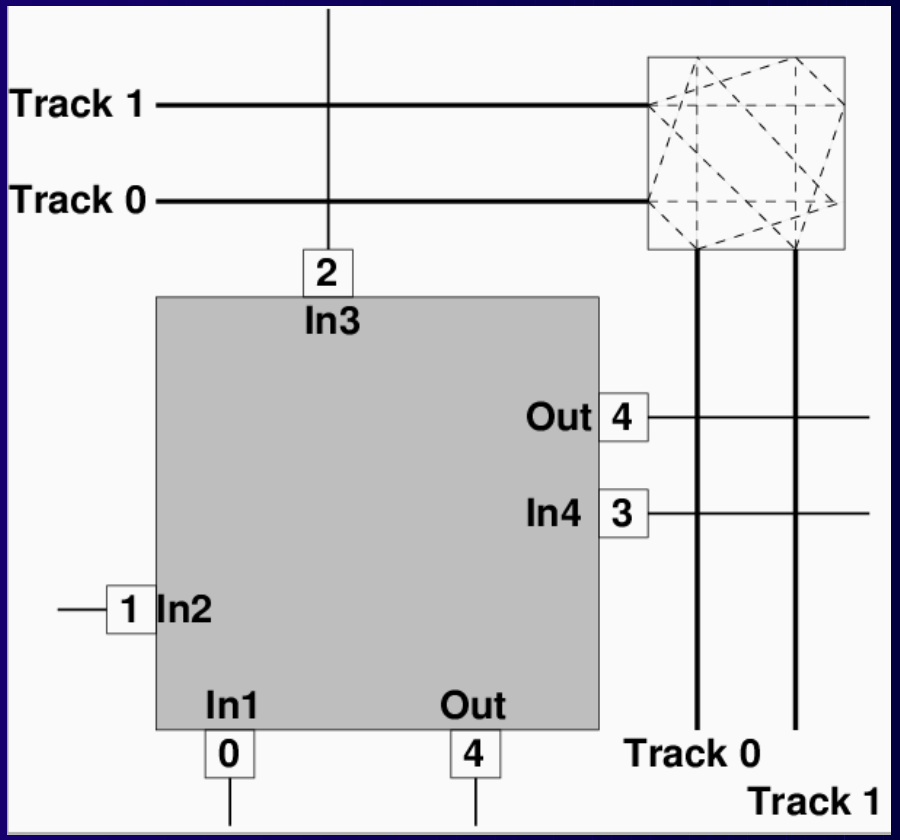


.p Platzierungsdatei

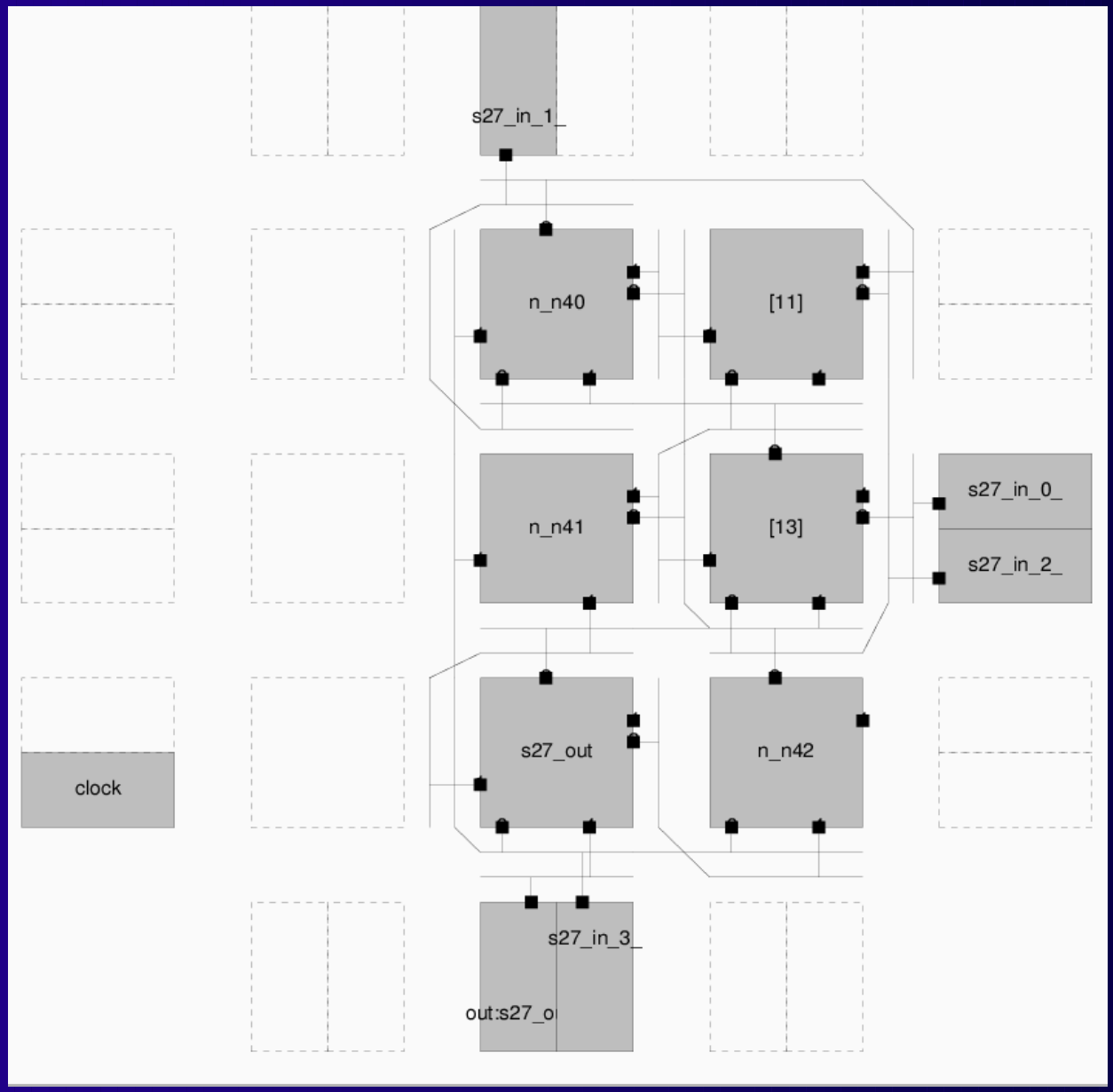
```
Netlist file: BLIF/s27.net   Architecture file: prak07.arch
Array size: 3 x 3 logic blocks

#block name     x    y    subblk     block number
#----------     --   --   ------     ------------
s27_in_2_       4    2    0               #0
s27_in_1_       2    4    0               #1
s27_in_3_       2    0    1               #2
s27_in_0_       4    2    1               #3
clock           0    1    0               #4
out:s27_out     2    0    0               #5
[13]            3    2    0               #6
s27_out         2    1    0               #7
n_n40           2    3    0               #8
n_n41           2    2    0               #9
n_n42           3    1    0               #10
[11]            3    3    0               #11
```

.r Verdrahtungsdatei

```
Array size: 3 x 3 logic blocks.

Routing:

Net 0 (s27_in_2_)

SOURCE (4,2)  Pad: 0
  OPIN (4,2)  Pad: 0
 CHANY (3,2)  Track: 0
 CHANX (3,1)  Track: 0
  IPIN (3,2)  Pin: 0
  SINK (3,2)  Class: 0
 CHANY (3,2)  Track: 0
 CHANY (3,3)  Track: 0
  IPIN (3,3)  Pin: 3
  SINK (3,3)  Class: 0


Net 10 ([11])

SOURCE (3,3)  Class: 1
  OPIN (3,3)  Pin: 4
 CHANY (3,3)  Track: 1
 CHANX (3,3)  Track: 1
 CHANX (2,3)  Track: 1
  IPIN (2,3)  Pin: 2
  SINK (2,3)  Class: 0
```



Beispiel


s27_in_1_
n_n40
[11]
n_n41
[13]
s27_in_0_
s27_in_2_
clock
s27_out
n_n42
s27_in_3_
out:s27_o

Architekturparameter

- X, Y Abmessungen in Logikblöcken
- W Anzahl horiz./vert. Leitungen in Kanal
- Verzögerungszeiten
  - Tipad, Topad
  - Tswitch
  - Tcomb
  - TFFin, TFFout
- Einer pro Zeile
- Überschreibbar in Kommandozeile

Aufgabe 1

- Abgabe bis Mo, 15.11.2010 23:59 MET
  - Für V2/3CP: 14.01.2011
- Analyse von Schaltungen
- Einlesen von
  - Netzliste
  - Architekturbeschreibung
  - Platzierung
  - (falls angegeben) Verdrahtung
    - Muss aber auch ohne laufen!

Platzierungsprüfung

- Alle Blöcke aus Netzliste platziert?
- Platzierung gültig?
  - Keine Überlappungen?
  - Koordinaten in Ordnung?
- Fehler ausgeben

Verdrahtungsprüfung

- Genau 1 Quelle pro Netz
- Mindestens 1 Senke pro Netz
- Alle Netzteile miteinander verbunden
  - Prüfe Koordinaten auf passende Anreihung
    - SOURCE, OPIN, CHANX, ..., IPIN, SINK
- Gültige Koordinaten
- Ressourcen nur einmal benutzt

Konsistenzprüfung

- Anspruchsvollste Prüfung!
- Netzliste ./. Verdrahtung
- Konnektivität muss gleich sein
  - Alle Verbindungen aus .net müssen existieren
  - Keine weiteren Verbindungen dürfen in .r existieren
- Vertauschbarkeit von Logikblockeingangs-Pins beachten!
  - Logikblockinhalte aber ignorieren

Timing-Analyse (nicht für V2!)

- Wie „schnell“ ist die Lösung?
  - Kritischer Pfad
- Bei unverdrahteten Schaltungen
  - Verdrahtungsverzögerung schätzen
  - Kürzesten Weg annehmen
- Bei verdrahteten Schaltungen
  - Verzögerung genau berechnen
- Mittels längster Pfade
  - Wird am Freitag behandelt
  - Teilaufgaben aber schon jetzt lösbar

Wichtig

- Gruppenarbeit!!!
  - Aufgaben sonst kaum fristgerecht lösbar
- Tipp zur Vorgehensweise
  - Als allererstes gemeinsam Datenstruktur festlegen
  - Damit dann parallel implementieren
    - Einlesen / Aufbauen der Datenstruktur
    - Prüfungen
      - Getrennte .net/.p/.r-Prüfungen: Recht einfach
      - Übergreifende .net/.p/.r-Prüfung: Schwieriger!
  - Selber Testfälle entwickeln!