

Algorithmen für Chip-Entwurfswerkzeuge

Kick-Off zum Praktikum



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Praktikum
WS 2014/2015

Andreas Koch

Eingebette Systeme und Anwendungen
Technische Universität Darmstadt



- ▶ 2er Gruppe(n)
- ▶ Guppenbogen ausfüllen
- ▶ Kontaktinfos untereinander austauschen
 - ▶ Telefon, E-Mail, Skype, ICQ, ...



- ▶ In was?
 - ▶ Java
 - ▶ Abnahme auf Version 1.7
- ▶ Wo?
 - ▶ Auf dem heimischen Rechner
 - ▶ Informatik-Poolräume
- ▶ Art der Programme
 - ▶ Kommandozeilenorientiert
 - ▶ Dateiverarbeitend



- ▶ Am Abgabetag (Mittwochs) bis 23:59 Uhr MEZ!
- ▶ E-Mail an ace@esa.informatik.tu-darmstadt.de
 - ▶ Betreff: [Abgabe] Gruppe *N* Aufgabe *M*
 - ▶ Attachment: .jar-Datei
 - ▶ Kompilierbare .java-Quellen
 - ▶ In alled Dateien Gruppennummer!
 - ▶ Vorkompilierte .class Klassen
 - ▶ README Textdatei
 - ▶ Beschreibt Kompilierung und Aufruf
 - ▶ Kurzer Überblick über
 - Programmaufbau
 - Algorithmen und benutzte Datenstrukturen
 - Beiträge der einzelnen Gruppenmitglieder



- ▶ In der Regel am auf die Abgabe folgenden Freitag: **Vortrag**
- ▶ Zur normalen Vorlesungszeit
- ▶ In KW 49, 4 und 7
- ▶ Je Gruppe 10-15 Minuten Vortrag
- ▶ Inhalt:
 - ▶ Vorgehensweise, Kernalgorithmus und Datenstrukturen
 - ▶ Programmaufbau, Ergebnisse
 - ▶ Erfahrungen und Kommentare
- ▶ Beides geht in die Endnote ein!



- ▶ Vorschlag *Writing Robust Java Code*
 - ▶ PDF auf Web-Seite
- ▶ Dokumentation
 - ▶ Aufgabe 1-3
 - ▶ Im wesentlich JavaDoc und Kommentare
 - ▶ **Wichtig:** Historie im Dateikopfkommentar
Kann auch Log aus Versionskontrolle sein
 - ▶ Aufgabe 4
 - ▶ *Richtiges* 10-20 seitiges Dokument (Prosa)
 - ▶ Zusammenfassend über alle bisherigen Arbeiten
 - ▶ Inhalt:
 - Benutzte Algorithmen, Datenstrukturen und Bibliotheken
 - Erweiterungen
 - Ergebnisse (Profiling, Zeit, Lösung)
 - Erfahrungen
 - ▶ Macht viel Arbeit, nicht unterschätzen!



- ▶ 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



- ▶ Gruppenarbeit entscheidend
- ▶ Probleme rechtzeitig ansprechen
- ▶ Aufteilung der Arbeiten
 - ▶ Vorschläge in den Aufgabenstellungen
 - ▶ Immer gut aufteilbare Bereiche, empfohlene Arbeitspakete
 - ▶ Programmierung
 - ▶ Test
 - ▶ Profiling
 - ▶ Dokumentation (nicht unterschätzen!)



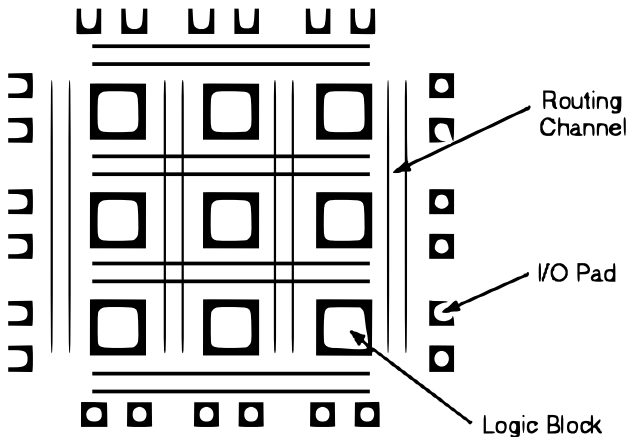
- ▶ Versionverwaltung: [Subversion](#), [git](#)
 - ▶ Wichtigstes Werkzeug für Gruppenarbeit
- ▶ Java IDE: [Eclipse](#), [NetBeans](#), [IDEA](#)
 - ▶ Nützlich, insbesondere für Refactoring
- ▶ Automatisierte Regressionstests: [JUnit](#)
 - ▶ Müssen aber trotzdem gepflegt werden
- ▶ Profiler: [JIP](#)
 - ▶ Zur Analyse von Zeit-/Speicherbedarf
- ▶ Lexer/Parser: [ANTLR](#)
 - ▶ Eigentlich nicht notwendig

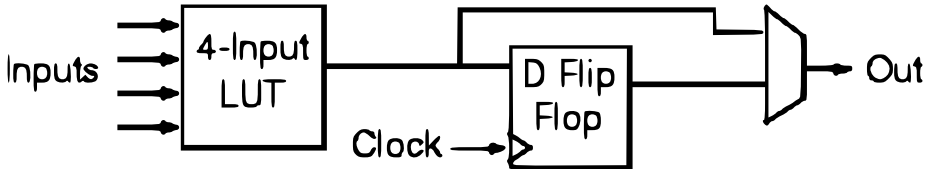


- ▶ Analyse von Schaltungen
 - ▶ Ca. 3 Wochen
 - ▶ Nützlich für die folgenden Aufgaben
- ▶ Platzierung von Netzlisten
 - ▶ Ca. 4 Wochen (+3 Wochen Weihnachtspause)
- ▶ Verdrahtung von platzierten Netzlisten
 - ▶ Ca. 3 Wochen
- ▶ Gezielte Verbesserungen
 - ▶ Auch letzte Fehlerkorrekturen
 - ▶ Ca. 4 Wochen

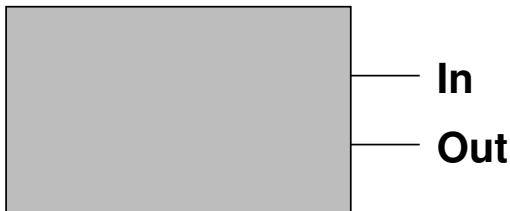


- ▶ Werkzeuge für FPGAs
- ▶ Grundprinzip
 - ▶ Sollte aus TGDI o.ä. bekannt sein
- ▶ Algorithmen
 - ▶ Vergleichbar mit denen für *echte* Chips
 - ▶ Einfacher
 - ▶ Diskrete statt kontinuierlicher Strukturen
 - ▶ Komplizierter
 - ▶ Feste Strukturen begrenzen Spielraum



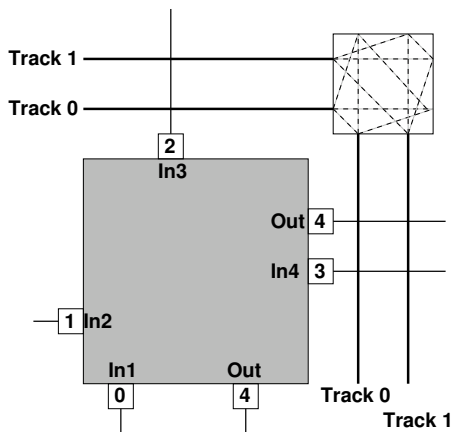


- ▶ Eingangs-Pins äquivalent
 - ▶ Untereinander austauschbar
 - ▶ Inhalt der Wertetabelle anpassen
 - ▶ Für unsere Tools nicht nötig ⇒ lassen wir weg

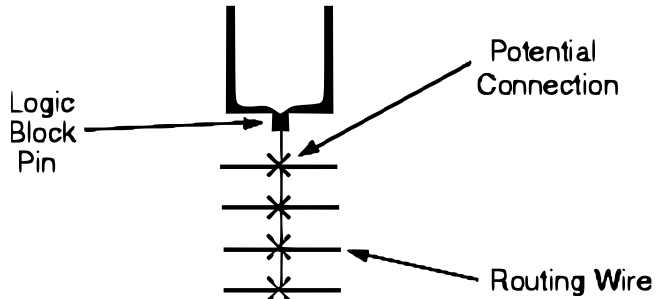


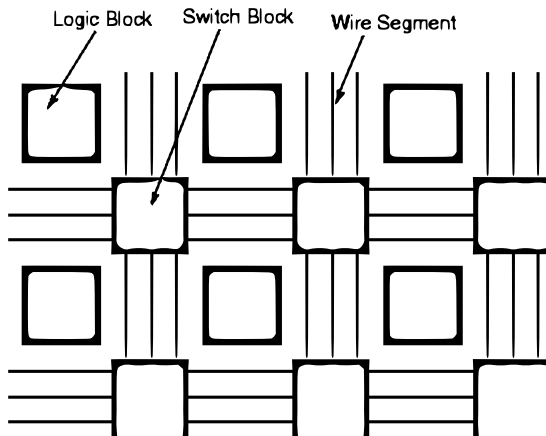
- ▶ *In* benutzt: Ausgangsblock
- ▶ *Out* benutzt: Eingangsblock
- ▶ **Keine** bidirektionalen Blöcke erlaubt
 - ▶ *In* und *Out* benutzt: Fehler

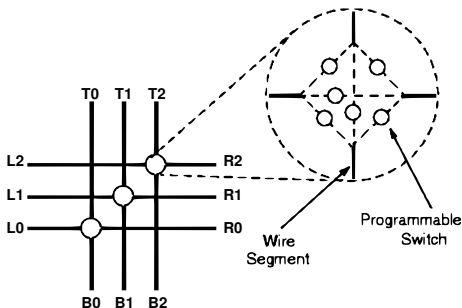
- ▶ Clock wird ignoriert
- ▶ Out ist doppelt vorhanden



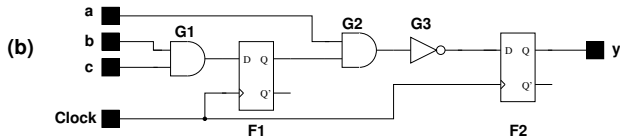
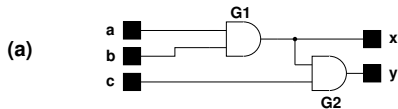
- ▶ Variable Anzahl von Leitungen
 - ▶ Parameter W : Breite (in V und H gleich!)
 - ▶ Architekturdatei
 - ▶ 1 Verbindung pro Eingang
 - ▶ Mehre bei Ausgang



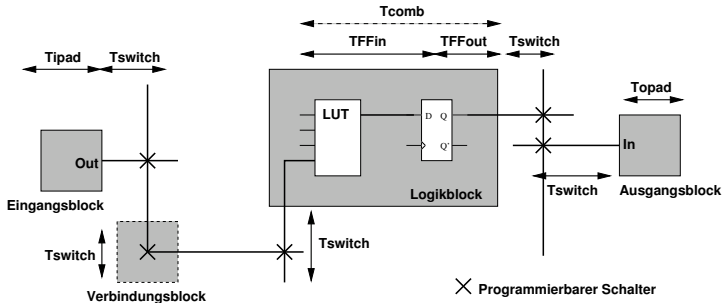




- ▶ Nur gleiche Spurnummern verbindbar
- ▶ Mehre Durchschaltungen OK (fanout)



- ▶ Kombinatorische Verzögerung: $a, b \rightarrow y$
- ▶ Sequentielle Verzögerung: $G2, G3$



- ▶ Elementweise Verzögerungsberechnung
 - ▶ Zeiten sind parametrisiert
 - ▶ Architekturdatei



.global clock

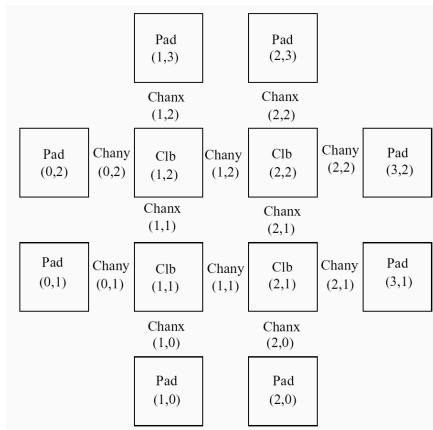
.output out:s27_out
pinlist: s27_out

.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



.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

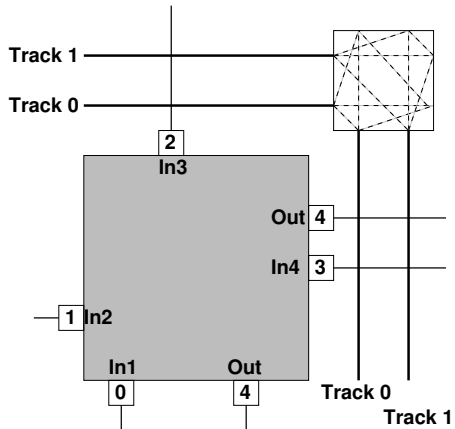
.r Verdrahtungsdatei

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
```



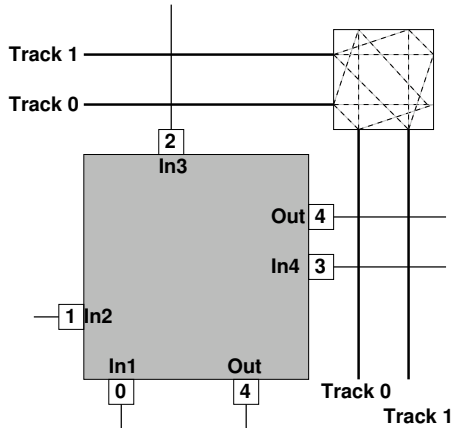
.r Verdrahtungsdatei

Array size: 3 x 3 logic blocks.

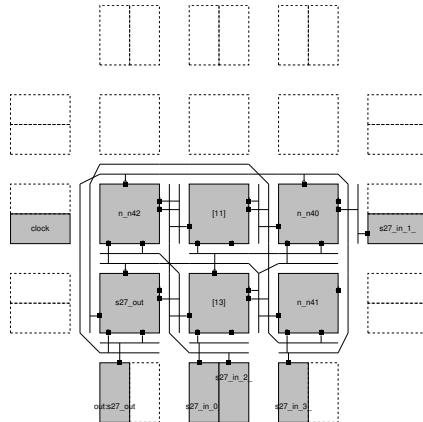
Routing:

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





- ▶ `.arch` Architekturparameterdatei
- ▶ X , Y Abmessungen in Logikblöcken
- ▶ W Anzahl horizontaler/vertikaler Leitungen im Kanal
- ▶ Verzögerungszeiten
 - ▶ T_{ipad} , T_{opad}
 - ▶ T_{switch}
 - ▶ T_{comb}
 - ▶ T_{FFin} , T_{FFout}
- ▶ Einer pro Zeile
- ▶ Überschreibbar in Kommandozeile

Aufgabe 1



- ▶ Abgabe bis Mittwoch 2014-12-03, 23:59 Uhr MEZ
- ▶ Einlesen von
 - ▶ Netzliste
 - ▶ Architekturbeschreibung
 - ▶ Platzierung
 - ▶ (falls angegeben) Verdrahtung
Muss aber auch ohne laufen!
- ▶ Analyse von Schaltungen



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



- ▶ 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



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



- ▶ 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 im nächsten VL Termin behandelt
 - ▶ Teilaufgaben aber schon jetzt lösbar



- ▶ Gruppenarbeit!!! 1 elf . . .
 - ▶ Aufgaben sonst kaum fristgerecht lösbar
- ▶ Tipp zur Vorgehensweise:
 - ▶ Als allererstes gemeinsam Datenstruktur festlegen
 - ▶ Dann damit parallel implementieren:
 - ▶ Einlesen / Aufbau der Datenstruktur
 - ▶ Prüfungen:
 - ▶ Getrennte .net/.p/.r-Prüfungen: Recht einfach
 - ▶ Übergreifende .net/.p/.r-Prüfungen: Schwieriger!
- ▶ Selber Testfälle entwickeln!