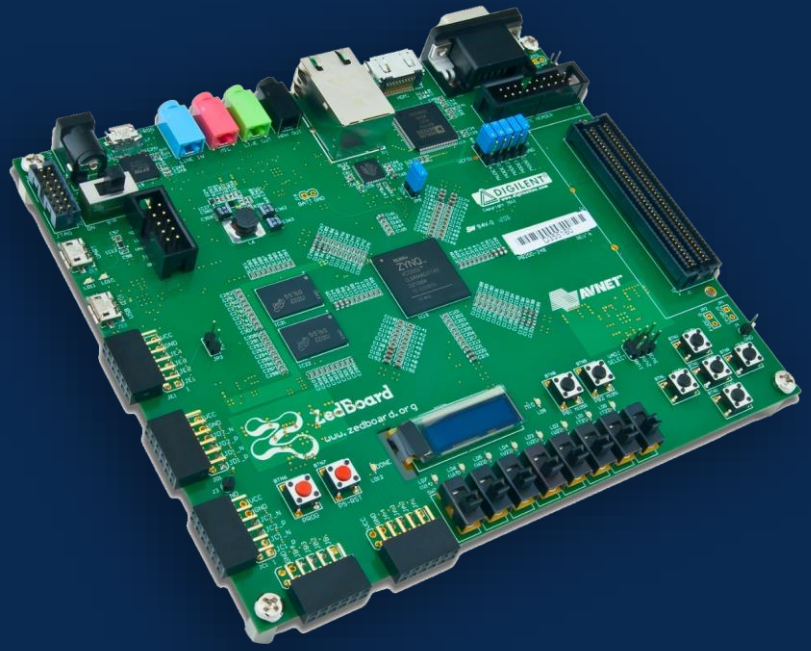
TECHNISCHE UNIVERSITÄT DRESDEN

**Informatik** » Institut für Technische Informatik » Professur VLSI-Entwurfssysteme, Diagnostik und Architektur

# Vereinbarkeit von partieller Rekonfiguration und Out-of-Context Synthese

## Forschungsprojekt

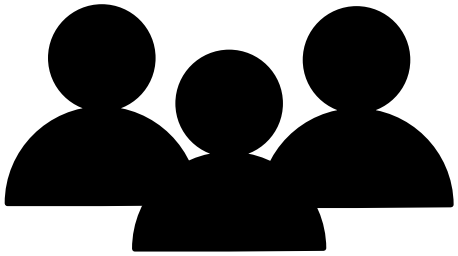



Björn Gottschall

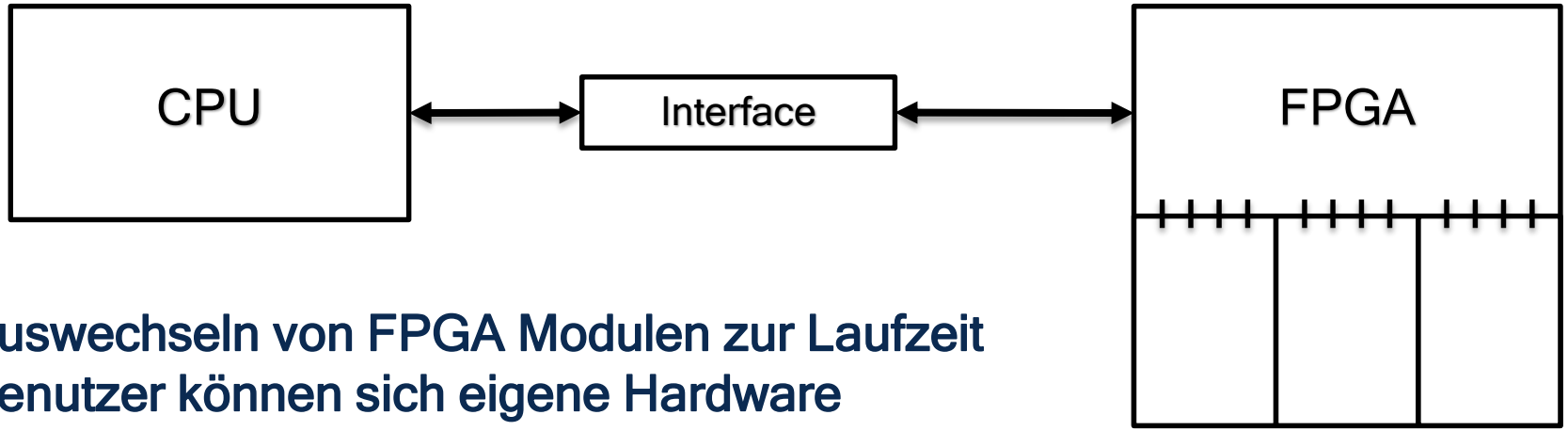
Dresden, 18.08.2016

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

# Inhaltsverzeichnis



- Unabhängige und isolierte Entwicklung von FPGA Modulen
- Trennung der Ressourcenkonkurrenz auf verschiedenen Ebenen




- Auswechseln von FPGA Modulen zur Laufzeit
- Benutzer können sich eigene Hardware Beschleuniger instanziieren
- Störungen von anderen sollen nicht möglich sein

# Begriff der Synthese & Implementierung

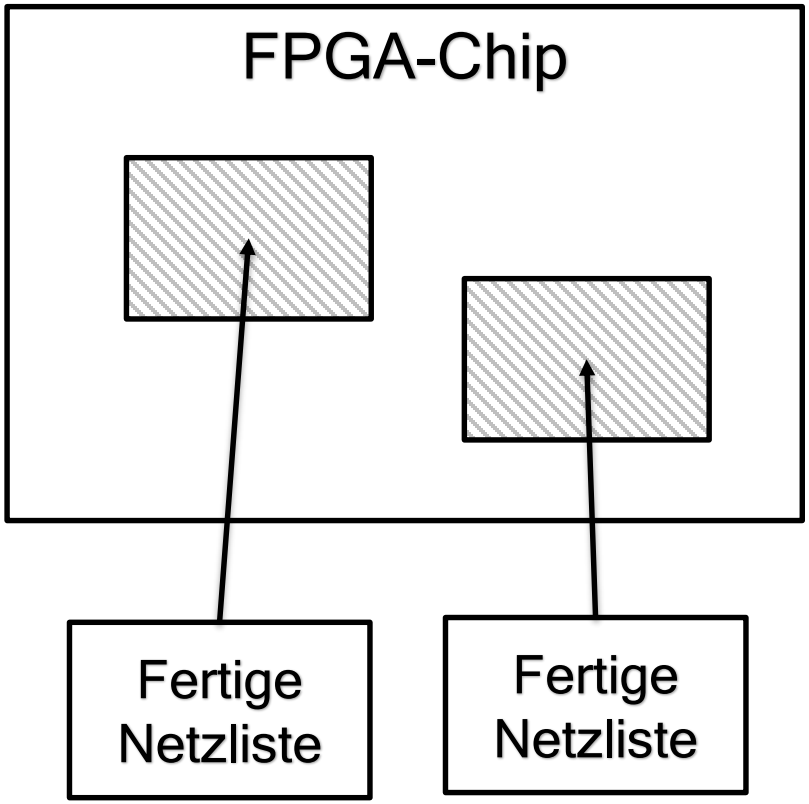
```
read_vhdl

synth_design

opt_design
place_design
phys_opt_design
route_design

write_bitstream
write_checkpoint
```

Synthese

Implementierung

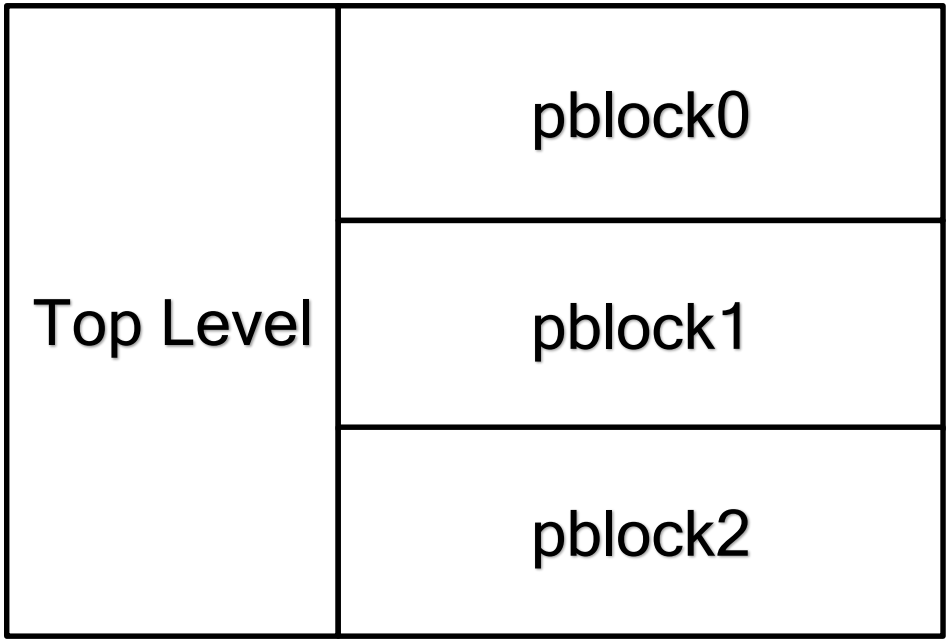
# Idee der Out-of-Context Synthese

- Unabhängige und isolierte Entwicklung von einzelnen Modulen in festgelegten Bereichen des FPGA Chips
- Jedes Modul erzeugt fertige Netzlisten inklusive des Routings
- Implementierung des Top-Level Designs um die fertigen Module herum



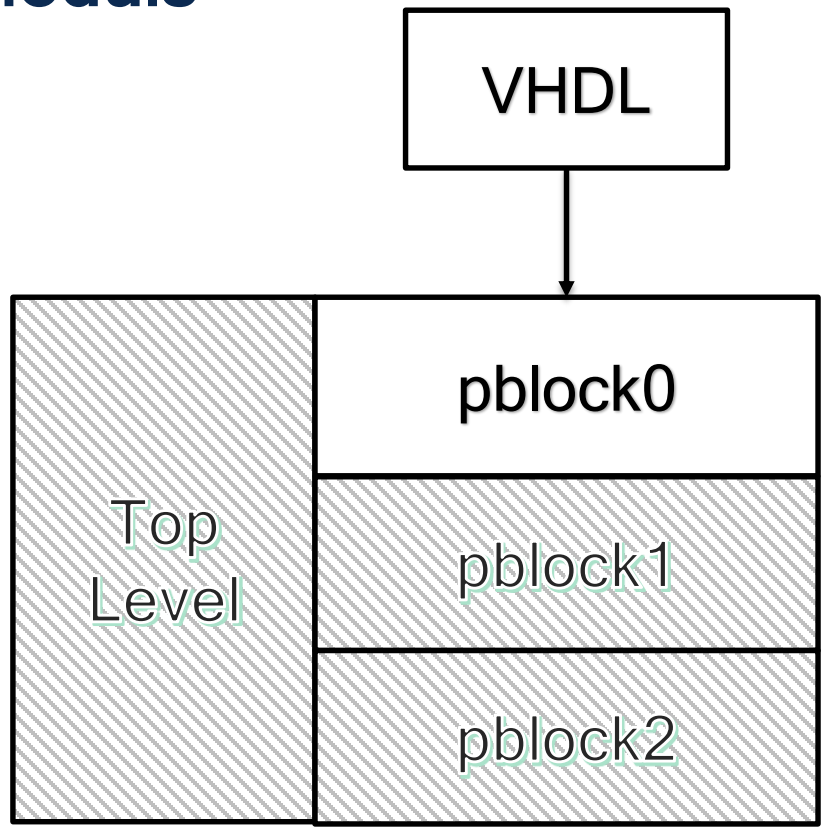
# 1. Aufteilung des FPGAs in geometrische Bereiche

- Top Level Design muss vorhanden sein
- Jedes OOC Modul braucht einen pblock
- Legt Ressourcenverteilung fest
- Definition von interconnects zwischen Top Level und jedem pblock
- Definition des Taktports zu jedem pblock inklusive Frequenz



# 2. Entwicklung des OOC Moduls

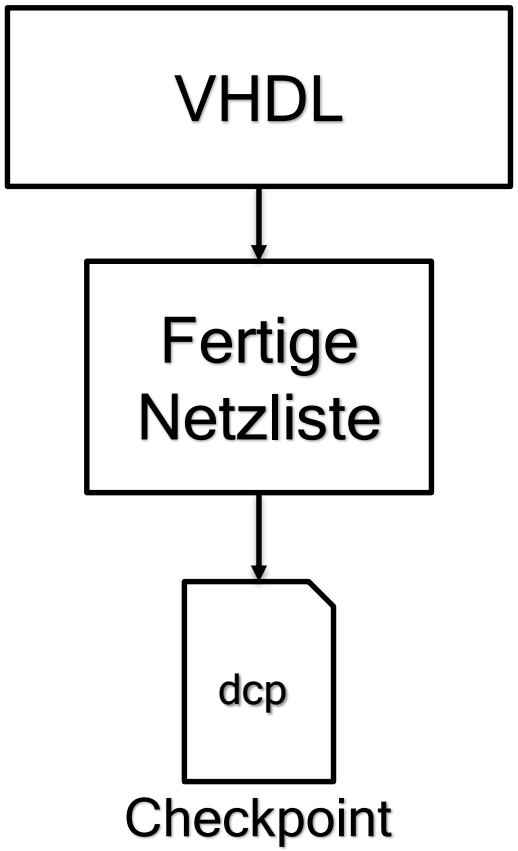
- Entwickler stehen nur die Ressourcen innerhalb seines pblocks zur Verfügung
- Restliche Entwicklung geschieht wie bei einem gewöhnlichen Top-Level Design



# 3. Synthese und Implementierung

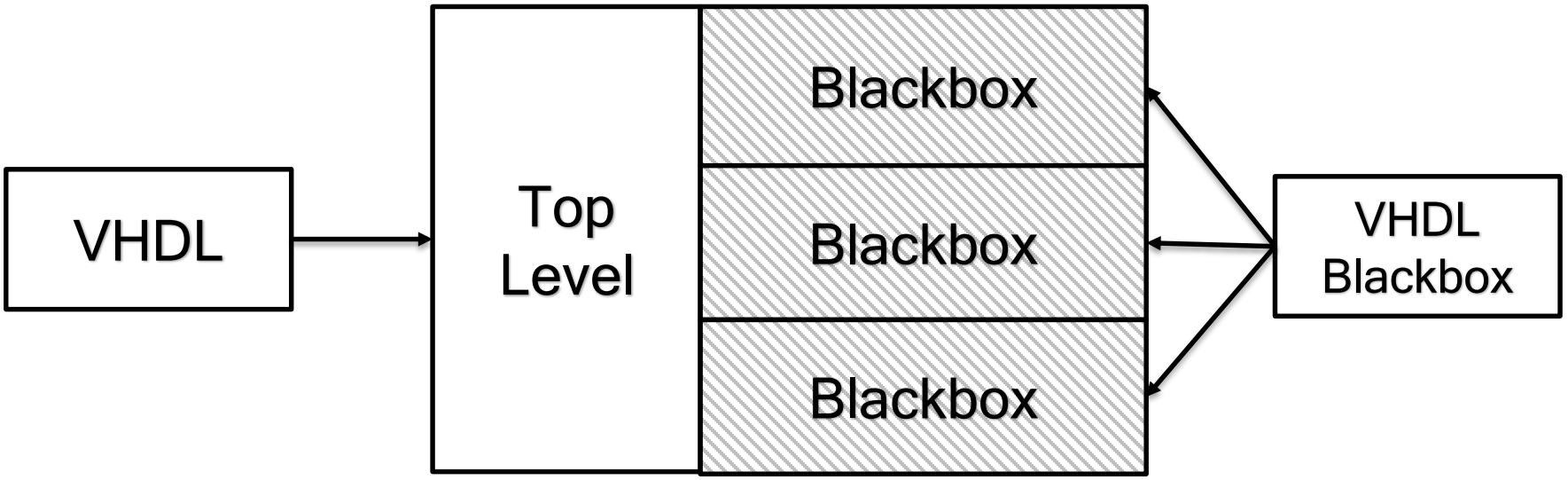
- Synthese mit einem spezielle OOC Flag
- Implementierung mit allen Constraints: pblock, clock, interconnects, routing und placement
- Speichern der erzeugten Netzliste in einem Checkpoint

- Wiederholung für jedes OOC Modul des Gesamtdesigns



# 4. Synthese des Top-Level Designs

- Jedes OOC Modul muss als Blackbox Definition für die Synthese bereit stehen

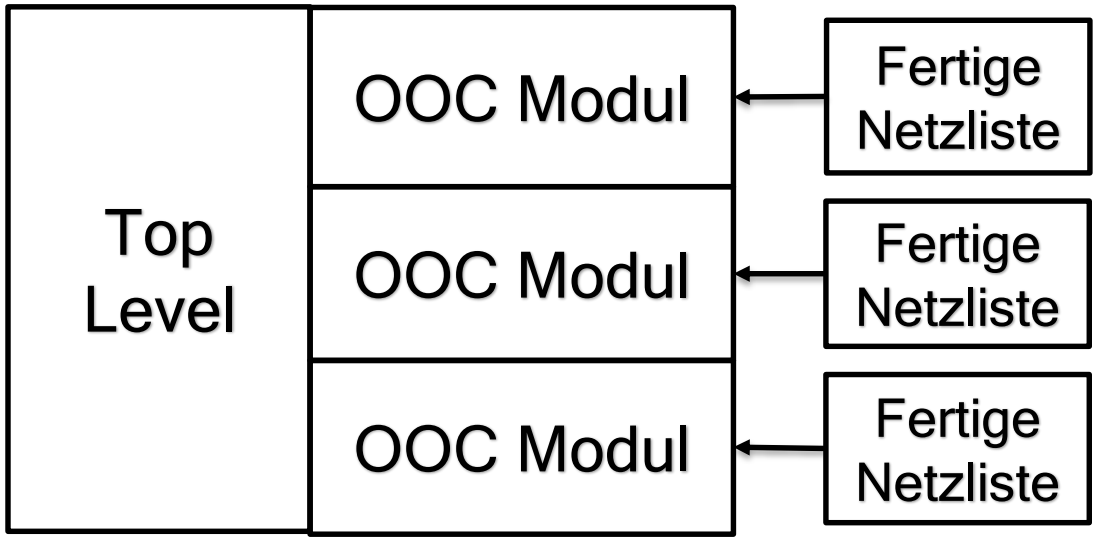


# 5. Implementierung des Top-Level Designs

- Die Blackboxen werden mit den jeweils fertigen Netzlisten der OOC Module ausgefüllt

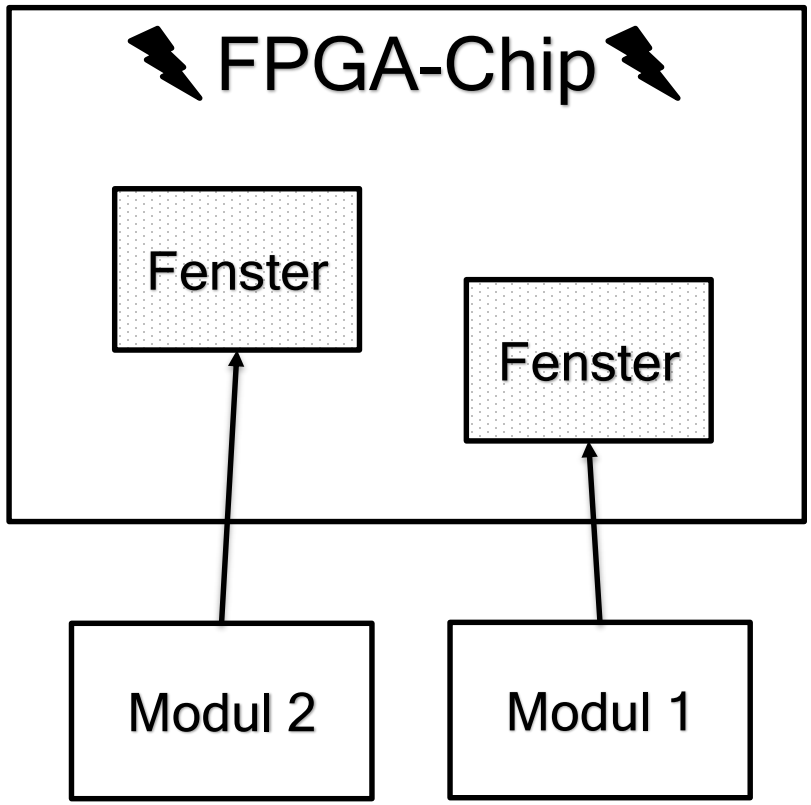
- Das Sperren der OOC Module verhindert das die Implementierung diese verändern darf

- Die Implementierung erzeugt nun das fertige Bitfile für den FPGA



# Idee der partiellen Rekonfiguration

- Austausch von Modulen innerhalb eines FPGAs zur Laufzeit
- Keine Unterbrechung des FPGAs oder anderer Module
- Aufteilung des FPGAs in statisches Design und rekonfigurierbaren Fenster
- Fixierung des statischen Designs für den Bottom-Up-Entwurf der partiellen Rekonfigurationen

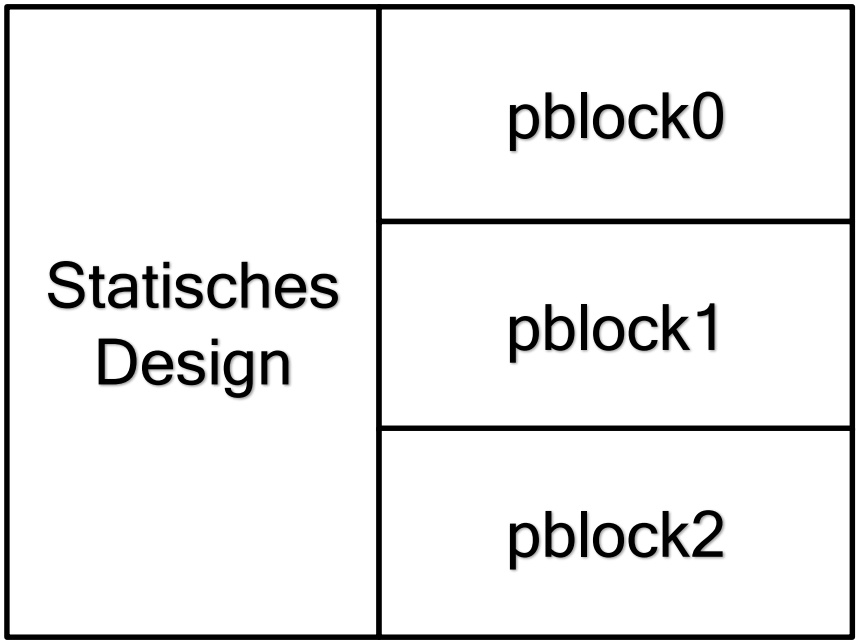


# 1. Aufteilung des FPGAs in geometrische Bereiche

- Aufteilung des FPGA Chips in statisches Design und rekonfigurierbare Fenster

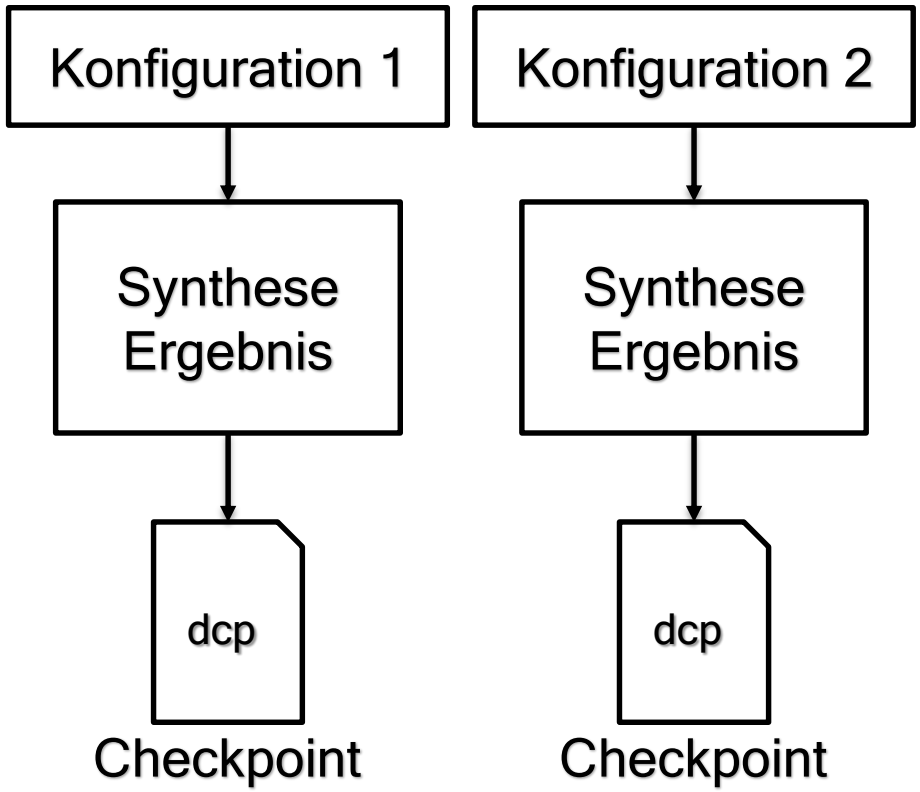
- Jedes rekonfigurierbare Fenster brauch einen pblock
- Jeder pblock darf keine Logik- oder Placement-Ressourcen außerhalb verwenden

- Definition von Interconnects zwischen Top Level und jedem pblock sowie des Takts



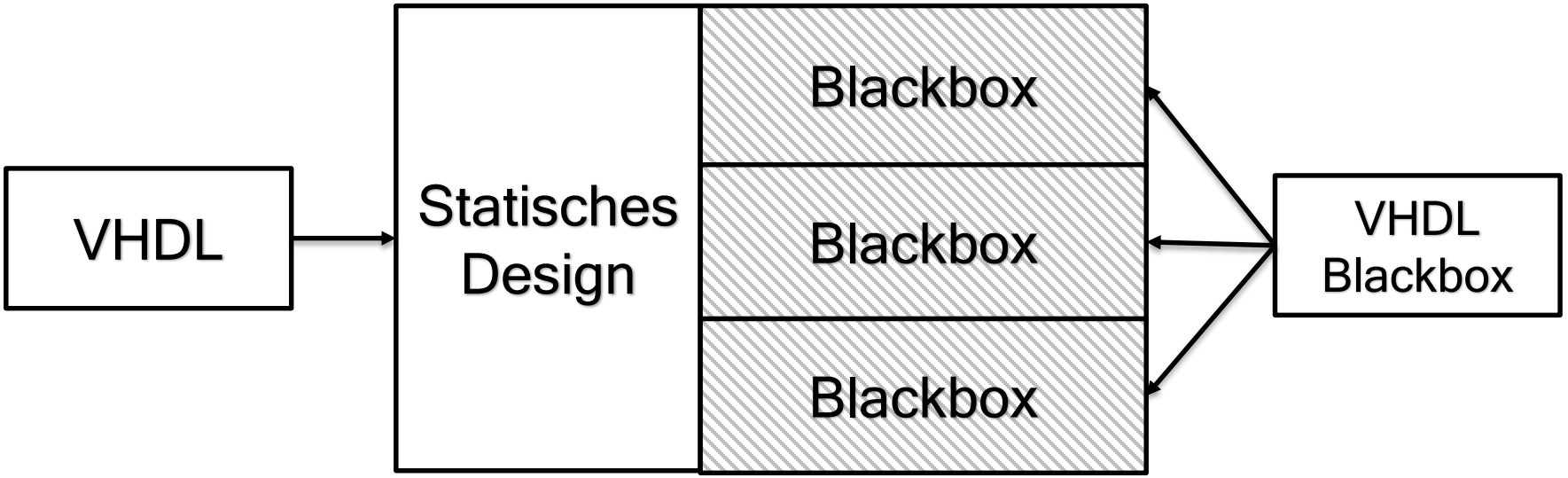
# 2. Synthese der partiellen Konfigurationen

- Synthese von jeder Konfiguration noch unabhängig von seinem Fenster
- Module werden im Out-of-Context Modus synthetisiert
- Speichern des Ergebnis in einem Checkpoint



# 3. Synthese des statischen Designs

- Jedes rekonfigurierbare Fenster muss als Blackbox Definition bereit stehen

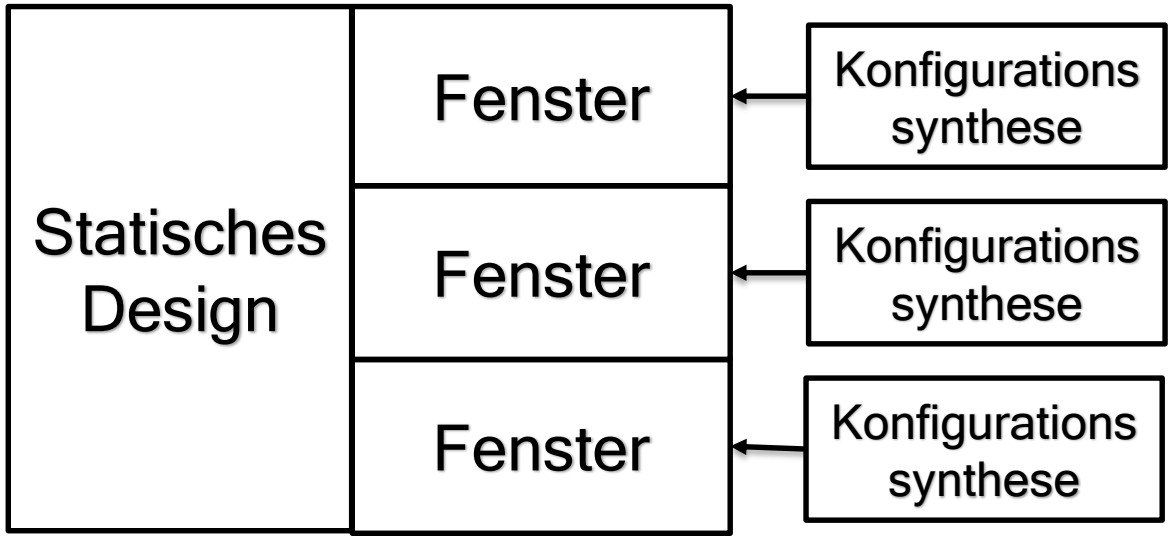


# 4. Implementierung des statischen Designs

- Jedes Fenster besitzt entweder gepufferte Ports oder ist mit einer Implementierung gefüllt
- Es entsteht immer ein fertiges Gesamtdesign

- Markierung von jedem Fenster als rekonfigurierbar
- Laden der Syntheseergebnisse
- Laden aller Constraints

- Implementierung des Gesamtdesigns

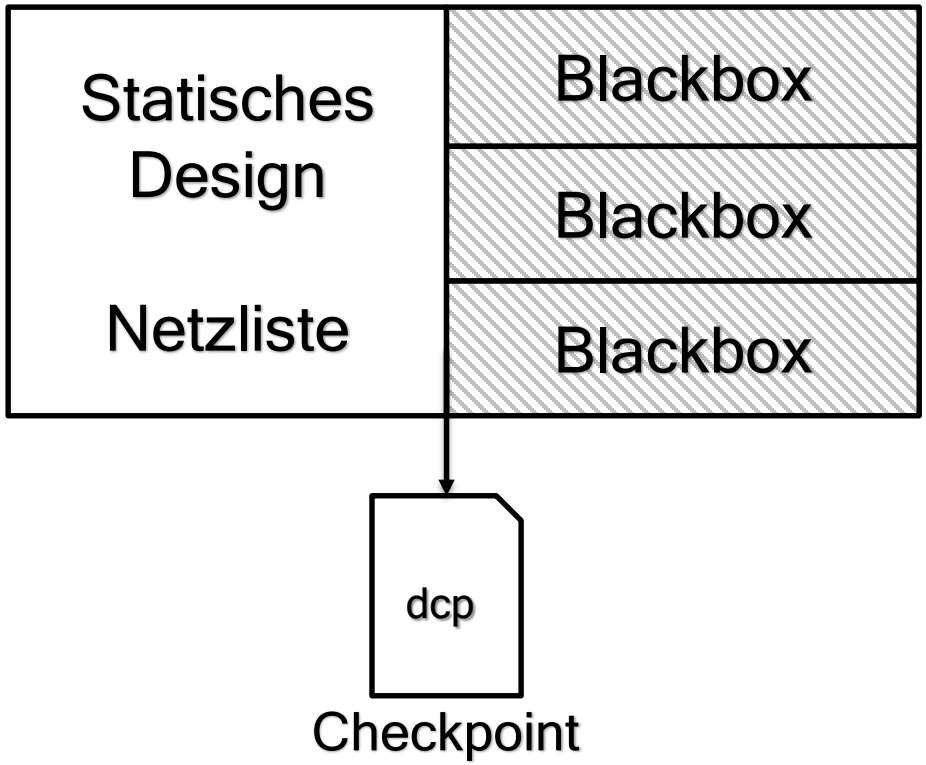


# 5. Vorbereitung des statischen Designs

- Rekonfigurierbaren Fenster werden wieder zu Blackboxen gewandelt

- Sperren des statischen Designs, damit jede weitere Implementierung dieses nicht verändern kann

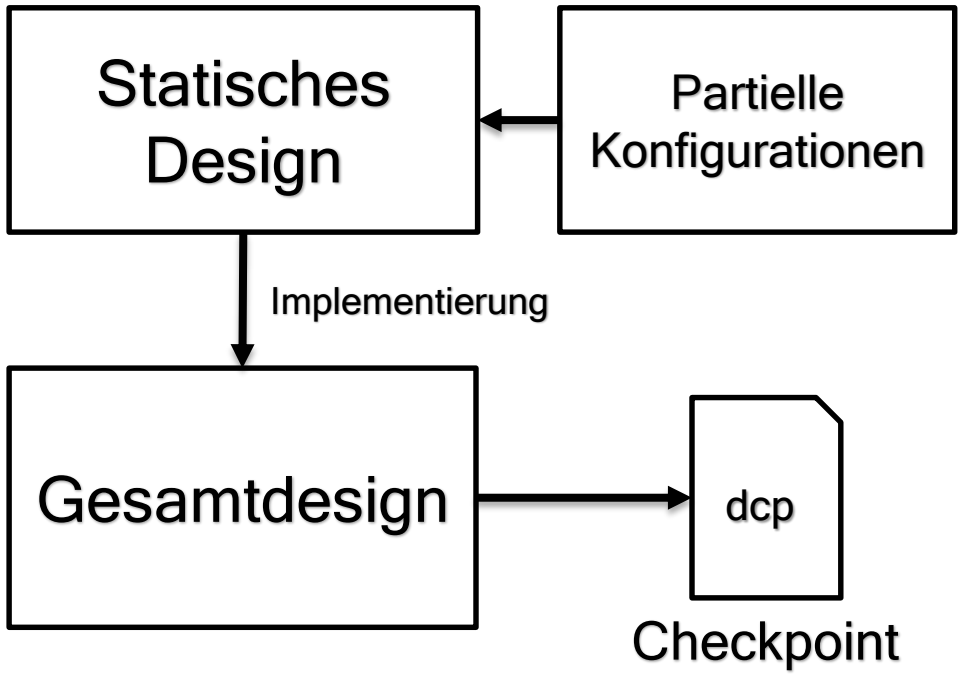
- Statisches Design steht nun als Template für jede weitere Implementierung bereit.



# 6. Implementierung der partiellen Rekonfigurationen

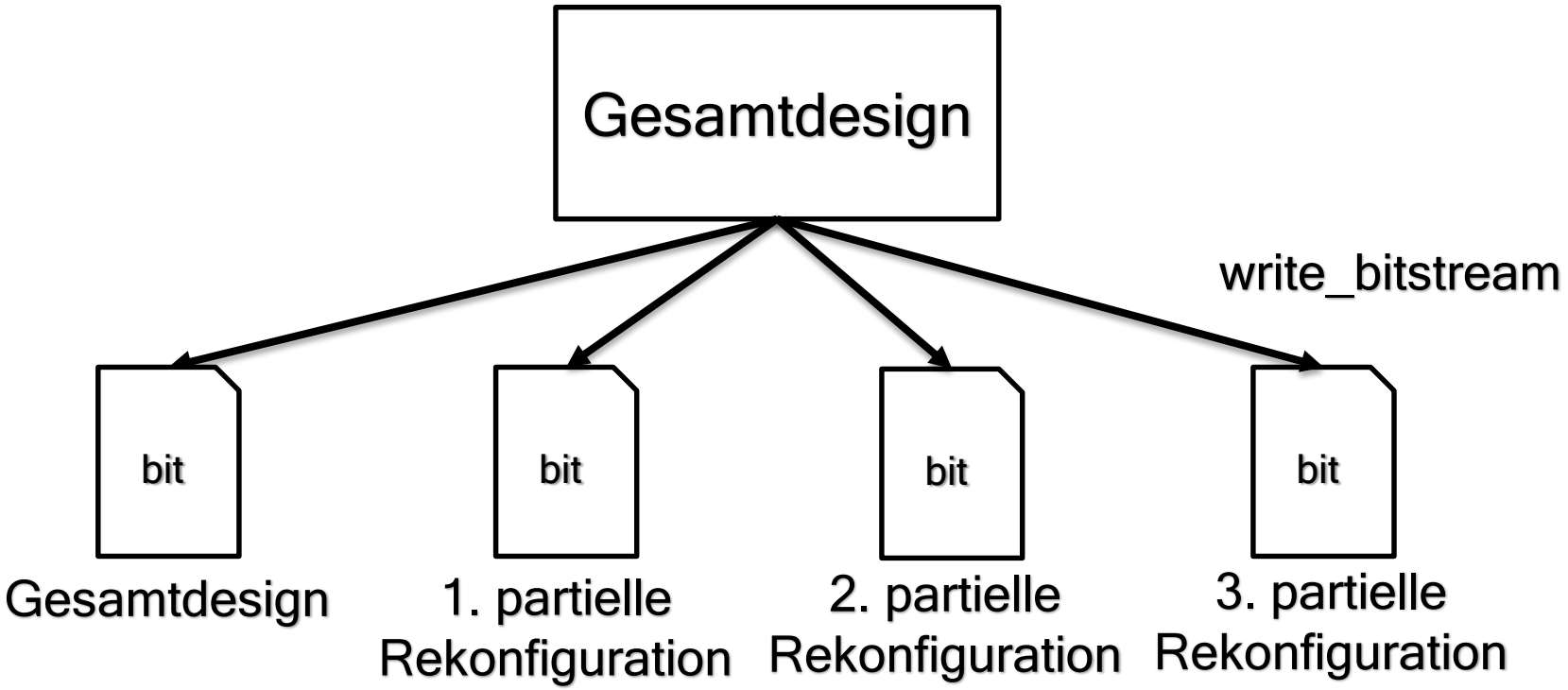
- festes statisches Design mit Blackboxen laden

- Laden der gewünschten partiellen Konfigurationen
- Implementierung eines Gesamtdesigns
- Speichern als Checkpoint

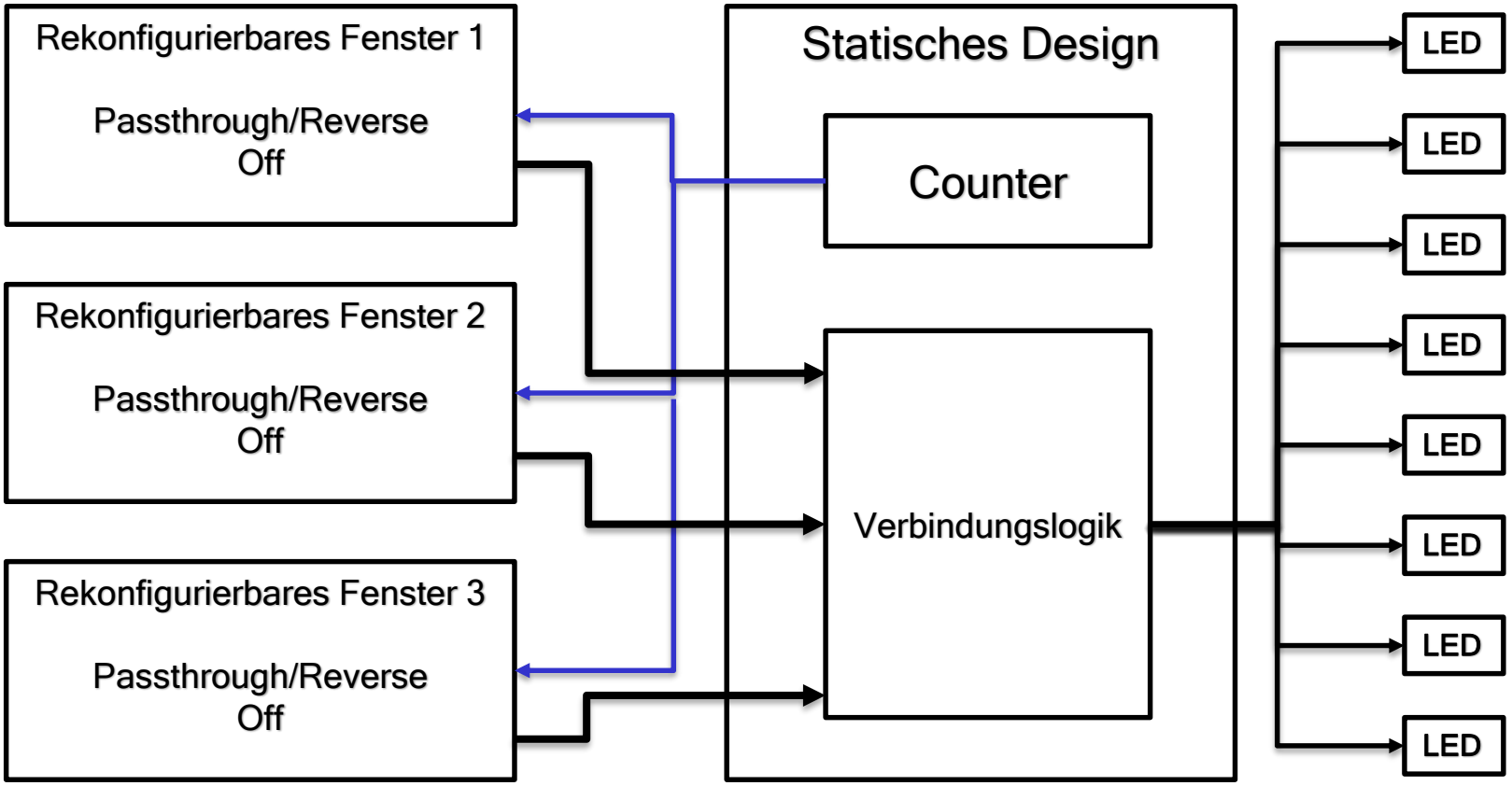


# 6. Bitstreams der partiellen Rekonfigurationen

- Gesamtdesign mit den gewünschten partiellen Konfigurationen laden und Bitstreams schreiben



# Praktisches Beispiel



# Zusammenfassung

- Die OOC Synthese bietet eine isolierte und unabhängige Entwicklung von einzelnen Modulen
- Nur die partielle Rekonfiguration bietet das Austauschen von einzelnen Modulen in einem statischen Design

- Die Ressourcentrennung ist bei beiden Methoden gleich streng umsetzbar und die Synthese ist sogar identisch
- OOC bietet einen höheren Freiheitsgrad bei der Implementierung

- OOC und partielle Rekonfiguration lassen sich noch nicht miteinander kombinieren
- Für das Routing des Top-Level Design konnte nicht garantiert werden, dass diese nicht durch andere Module hindurchgeht.

# Vielen Dank!

- [1] Xilinx UG905: Hierarchical Design, Reuse Design
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_3/ug905-vivado-hierarchical-design.pdf
- [2] Xilinx UG702: Partial Reconfiguration (ISE)
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf
- [3] Xilinx UG909: Partial Reconfiguration (Vivado)
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug909-vivado-partial-reconfiguration.pdf
- [4] Xilinx UG835: Vivado TCL Command
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug835-vivado-tcl-commands.pdf