

Automatenorientierte Entwurfsmethoden für Asynchrones Design in Programmierbaren Logikschaltungen

Vortrag im Rahmen der Promotion

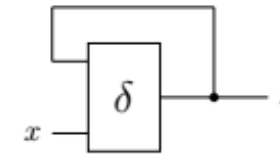
Florian Deeg

-
- 01 Einleitung und Motivation
 - 02 Taktloser Automatenentwurf
 - 03 Selbstsperrender Automatenentwurf
 - 04 Anwendungsbeispiel RISC-V Multicycle-Prozessor
 - 05 Zusammenfassung und Ausblick

Einleitung und Motivation

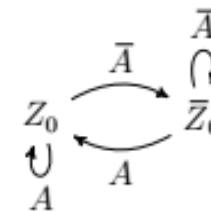
Warum automatenorientiert?

Gegensatz zu kombinatorischen Schaltungen:



Rückkoppelleitung

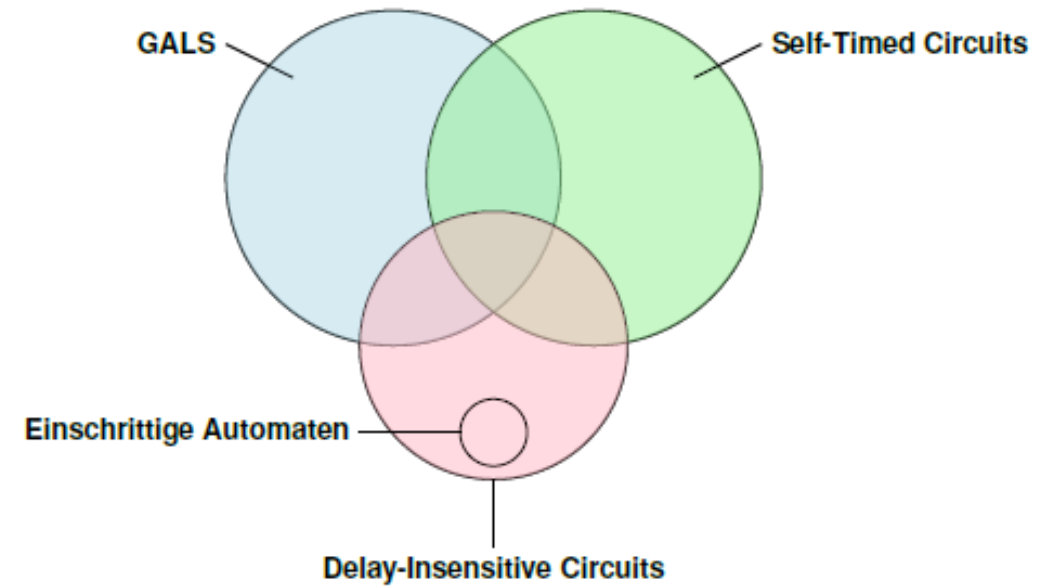
- Transitionen sind vom alten Zustand abhängig
- Funktionsstabilität muss gewährleistet werden



→ Rückkoppelleitung stellt große Herausforderung für **asynchrone Schaltungen** dar!

Alles außer synchrone Schaltungen:

- Delay-Insensitive Circuits
- Globally Asynchronous Locally Synchronous (GALS)
- Self-Timed Circuits





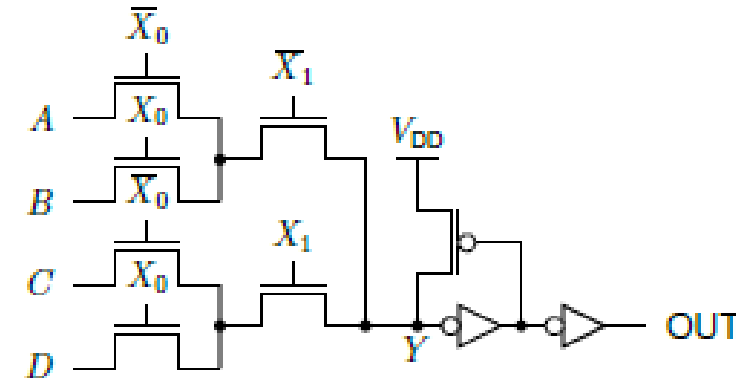
- **Geringerer Leistungsverbrauch**
 - Energie wird nur verbraucht, wenn tatsächlich gerechnet wird
 - Keine unnötige Taktverteilung
- **Höhere Performanz**
 - Keine Auslegung auf den Worst-Case-Pfad erforderlich
 - Schnelle Pfade können sofort weiterarbeiten
- **Skalierbarkeit und Flexibilität**
 - Leicht erweiterbar ohne globale Taktanpassung
 - Lokale Optimierung von Latenz und Durchsatz möglich
- **Robustheit**
 - unempfindlicher gegenüber:
 - Fertigungsschwankungen
 - Temperaturänderungen
 - Spannungsschwankungen
- **Zuverlässigkeit**
 - Kein zentraler Takt → kein Single-Point-of-Failure



Asynchrone Schaltungen ermöglichen **energieeffiziente**, **performante** und **robuste** Systeme – insbesondere dort, wo globale Taktung zum limitierenden Faktor wird.

- FPGAs sind für den synchronen Entwurf ausgelegt
- Bestehen aus:
 - Look-Up Table (LUT) v.a. für Logik
 - BRAM
 - DSP-Blöcken
 - PLLs
 - ...
- Kein high-Z innerhalb eines FPGA möglich
- Placement & Routing wird herausfordernd für asynchron
 - manuelles Platzieren
 - constraints
 - low-level

→ AMD Vivado für den Full-Custom-Entwurf

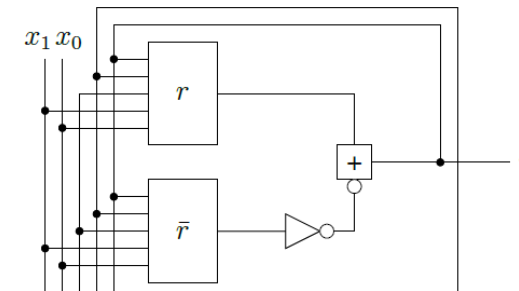
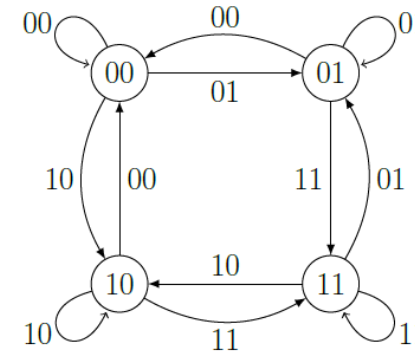


Taktloser Automatenentwurf



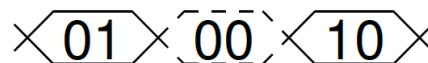
- Vehementer Ansatz: Eine Schaltung ohne Taktung sicher realisieren
- Herausforderungen:
 - **Hazards**
 - Strukturhazards: Hazards aufgrund der Struktur (verschiedene Pfadverzögerungen)
 - Funktionshazards: Hazards aufgrund Signalwechsel von mindestens zwei Eingangssignalen
 - **Races**
 - Wettlauf konkurrierender Rückkoppelpfade, die zu unterschiedlichen Folgezuständen führen
 - **Stabilität**
 - Zustand durch Funktionsstabilität (Eigenschleife)
 - Stabilisierung im Eingang (erst aus dem Zustand, wenn sich das Eingangssignal ändert)

- Realisierung:
 - Einschrittigkeit in
 - den Eingangsvariablen
→ vermeidet Funktions hazards
 - den Zustandsvariablen
→ vermeidet Races
 - Dual-Rail-Ansatz und überlappende Blöcke
→ vermeidet Struktur hazards einschrittiger Automaten



- Dual-Rail-Logik kodiert das binäre Signal x durch zwei Rails

- Als 1-Rail (X)

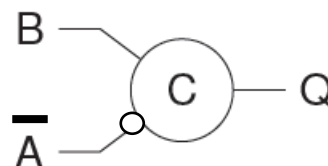


- Und 0-Rail (\bar{X})

- Handshaking durch Muller-C

- Änderung der Kodierung

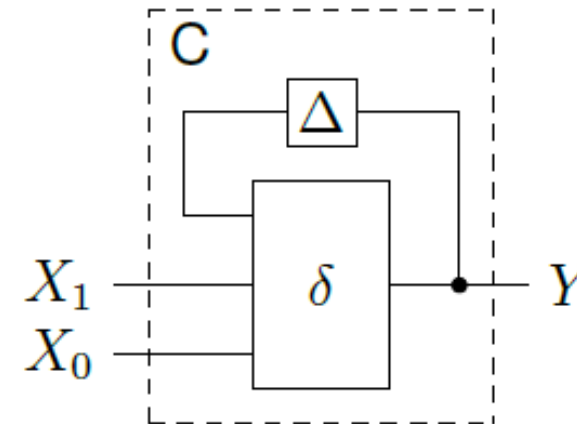
→ Eingangspins komplementär



\bar{A}	B	Q	
0	0	Q	Hold
0	1	1	Set
1	0	0	Reset
1	1	Q	Hold

- Gefordert: Sichere Realisierung der Struktur im FPGA durch Funktionsstabilität

- Realisierung als rückgekoppelte LUT
- Funktionsstabil, wenn gilt: $\tau_\delta > \tau_\Delta$
- Ziel: Konfigurationen der LUT finden, die dieses Kriterium erfüllen

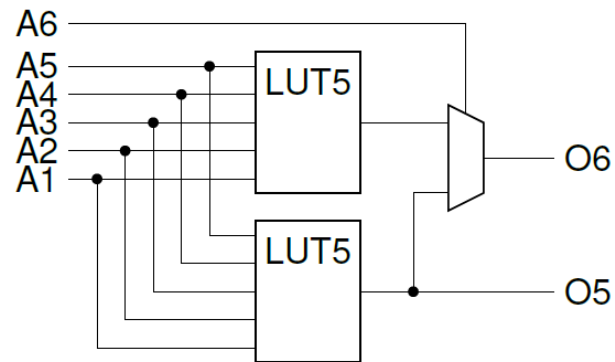


Taktloser Automatenentwurf

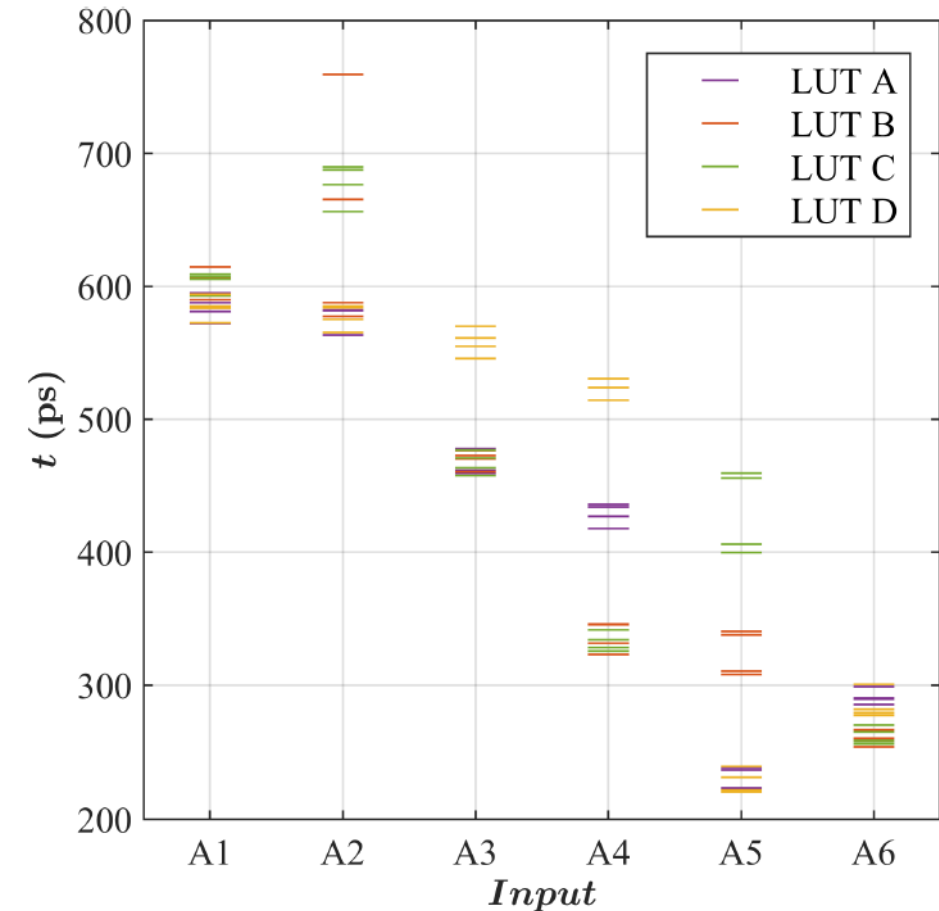
Funktionsstabiles Muller-C

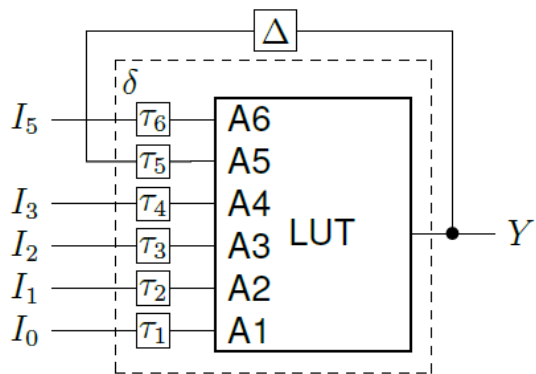


- Basis: LUT6_2
- Ziel: Minimales Feedbackdelay finden
 - LUT oszillieren lassen (T-Buffer)
 - Messung der Eingänge

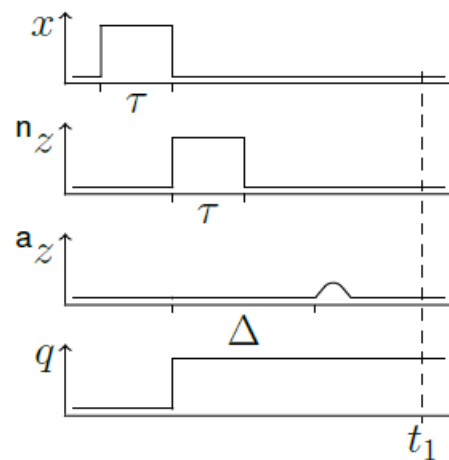
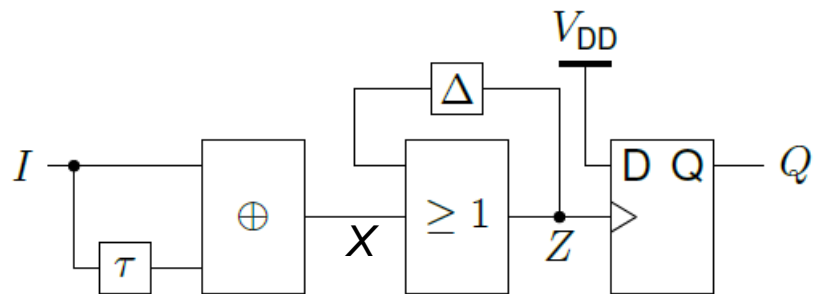


→ Kürzester Feedbackpfad für Eingang A5

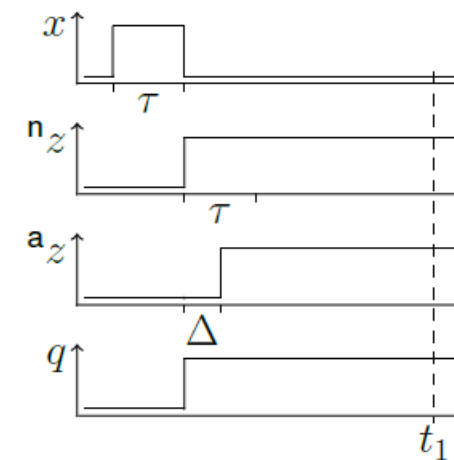




- Test zur Verifikation der gerouteten Schaltung



(a) Fehlerfall $\tau < \Delta$



(b) Fehlerfreier Fall $\tau > \Delta$

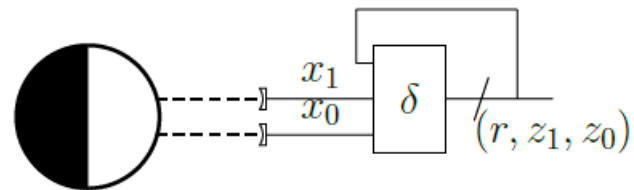
→ Verwendung der Dateneingänge A3 und A4 für maximal schnelles und funktionsstabiles Muller-C

Taktloser Automatenentwurf

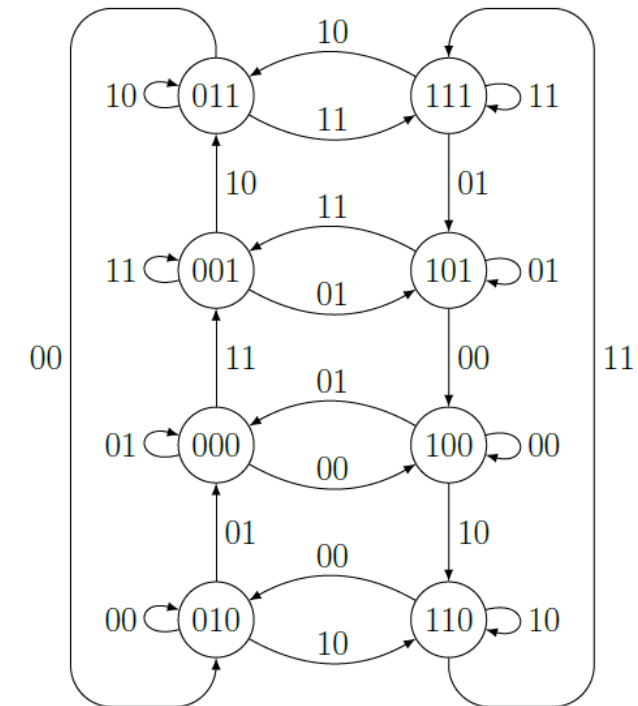
Beispielrealisierung: Rotationserkennungsautomat



- Drehscheibe mit Lasern und Reflektionssensoren
(Vergleich: Mechanische PC-Maus)



- Einschrittigkeit in den Eingängen ist vom physischen Aufbau gewährleistet
→ Funktionshazard-frei
- Zustandsübergänge einschrittig gewählt → Race-frei
- Überlappende Blöcke + Dual-Rail → Strukturhazard-frei





Taktloses Automatenentwurf ist

- in Spezialfällen möglich
- aufwendig in der Kodierung
 - Eingeschränkte Kodierungsmöglichkeiten
 - Eingangsvariablen- und Zustandsvariablenmenge steigt durch Einschrittigkeit rapide an

→ Simplerer allgemeingültiger Ansatz gesucht!

Selbstsperrender Automatenentwurf

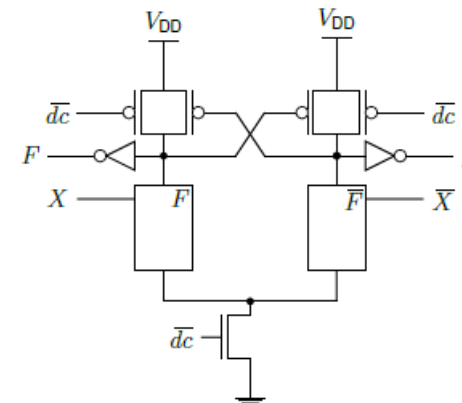
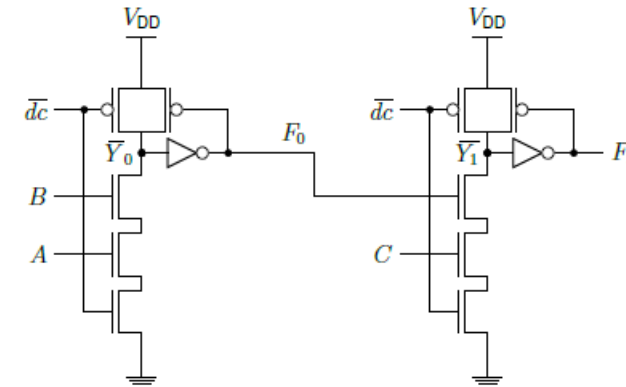
Idee: Selbstsperrende Dominologik

- Dual-Rail Dominologik
- Eingangspulsschaltung
 - Selbstsperrung am Eingang für Robustheit
 - Selbsttaktung

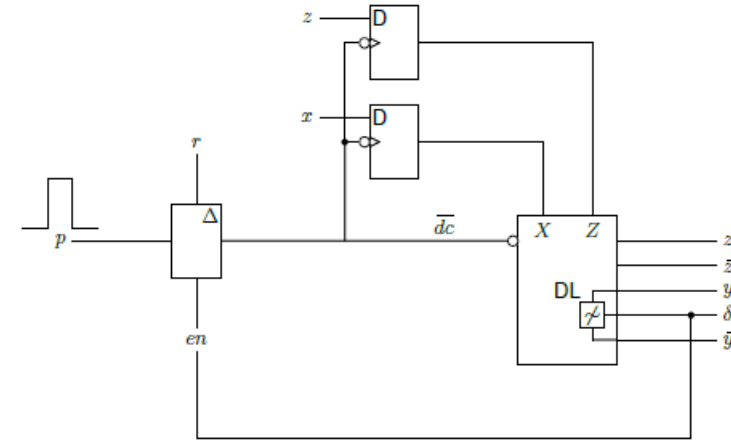


Dominologik

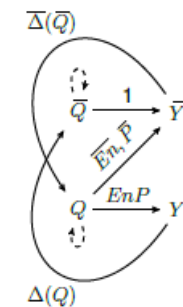
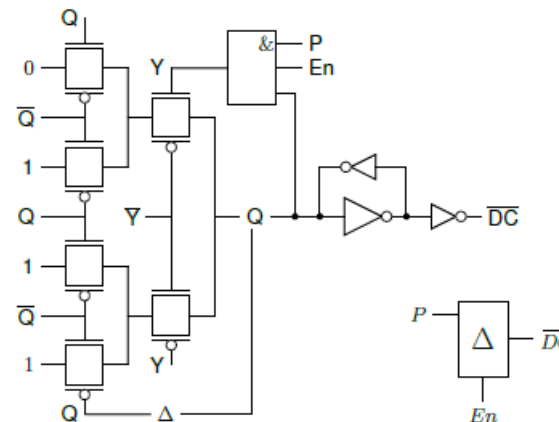
- Duty Cycle am Eingang steuert Precharge und Evaluation-Phase
 - $\overline{dc} = 0 \rightarrow$ Aufladen der inneren Knoten
 - $\overline{dc} = 1 \rightarrow$ Evaluierung der Eingänge
 - Kippen des ersten Dominosteins, dann kann der nächste kippen
 - Kombination von zwei disjunkten Dominogattern \rightarrow Dual-Rail Dominologik
 - Precharge eindeutig kodiert als [00] am Ausgang
 - Disjunkte Werte am Ausgang [01] oder [10] wenn Transition passiert ist
- \rightarrow Information über Vollständigkeit der Transition



- Struktur eines selbstsperrenden Automaten
 - Pulsschaltung am Eingang
 - Gebündelte Eingangs- und Zustandsvariablen
 - Dominologik-Kaskade
 - Komplettierungserkennung δy

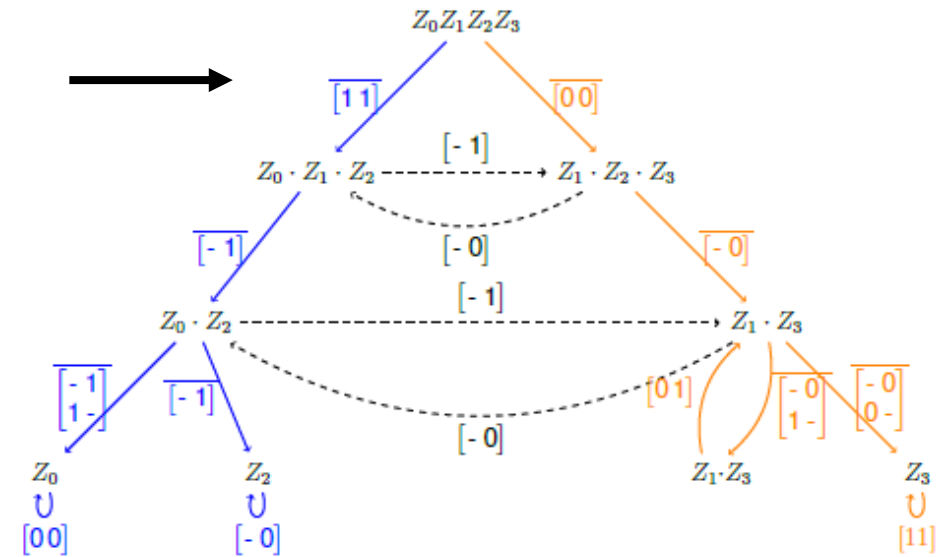
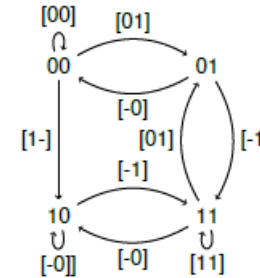
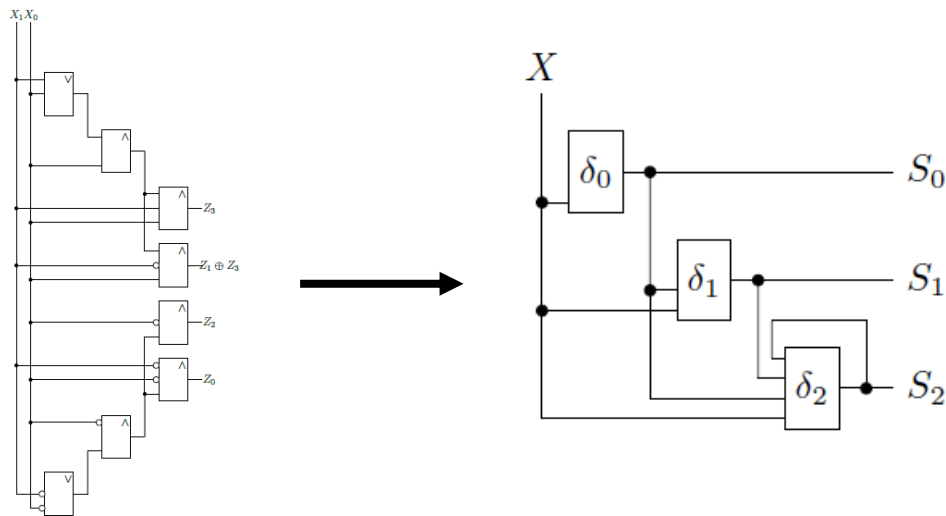


- Pulsschaltung (vereinfachte LUT-Struktur)
 - Self-resetting Schaltung
 - Selbstsperrung
 - Selbsttaktung (Erzeugung des Duty-Cycle)



Krohn-Rhodes-Theorie

- Besagt, dass jeder endliche Automat in eine Kaskade einfacher Bausteine zerlegt werden kann
- Erzeugung eines TSA
- Geeignete Kodierung führt zur Kaskadenstruktur





- Selbstgetaktet und -gesperrt durch Komplettierungsinformation
- Einfachere Entwurfsmethode
- Kaskadierung durch Krohn-Rhodes aufwändig aber möglich
→ One-hot Kodierung als guter Kompromiss zur einfachen Kodierung
- Im Allgemeinen höherer Flächenbedarf durch Dual-Rail und zusätzliche Selbstsperrung

Anwendungsbeispiel RISC-V Multicycle- Prozessor



- Basis: Synchroner Multicycle-Prozessor
- Idee: Steuereinheit asynchron implementieren, um Vorteile herauszuarbeiten
- Instruktionssatz:

Type	Bits [31:25]	Bits [24:20]	Bits [19:15]	Bits [14:12]	Bits [11:7]	Opcode[6:0]
R-Type	funct7	rs2	rs1	funct3	rd	0110011
I-Type	imm[11:0]		rs1	funct3	rd	0010011
L-Type	imm[11:0]		rs1	funct3	rd	0000011
S-Type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	0100011
B-Type	imm[12 10:5]	rs2	rs1	funct3	imm[4:1 11]	1100011
J-Type	imm[20 10:1 11 19:12]				rd	1101111

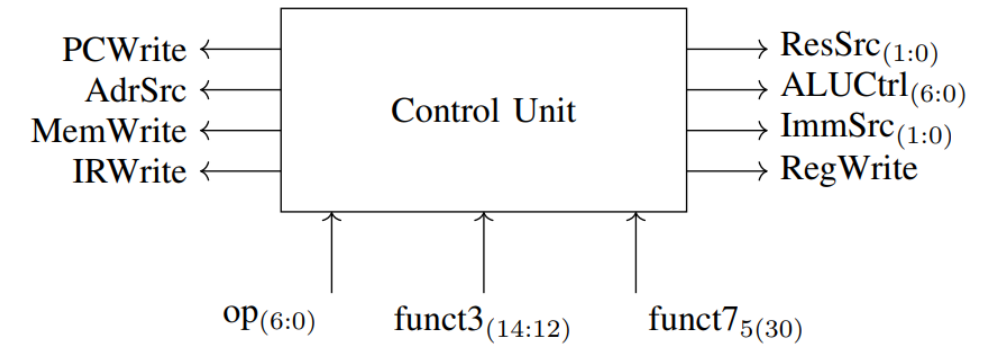
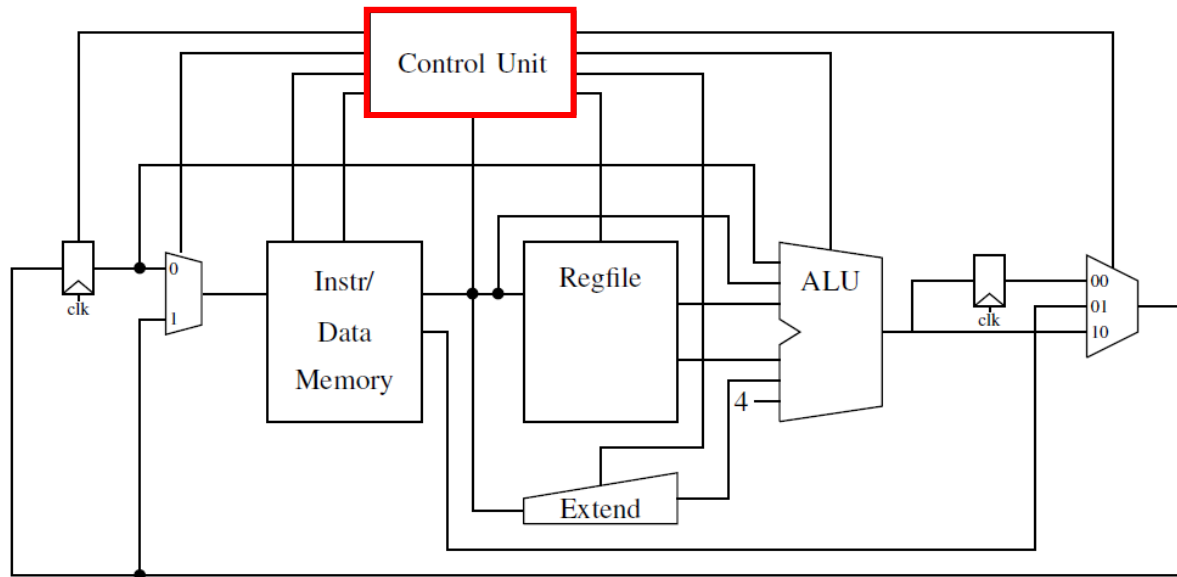
Anwendungsbeispiel: RISC-V

Architektur



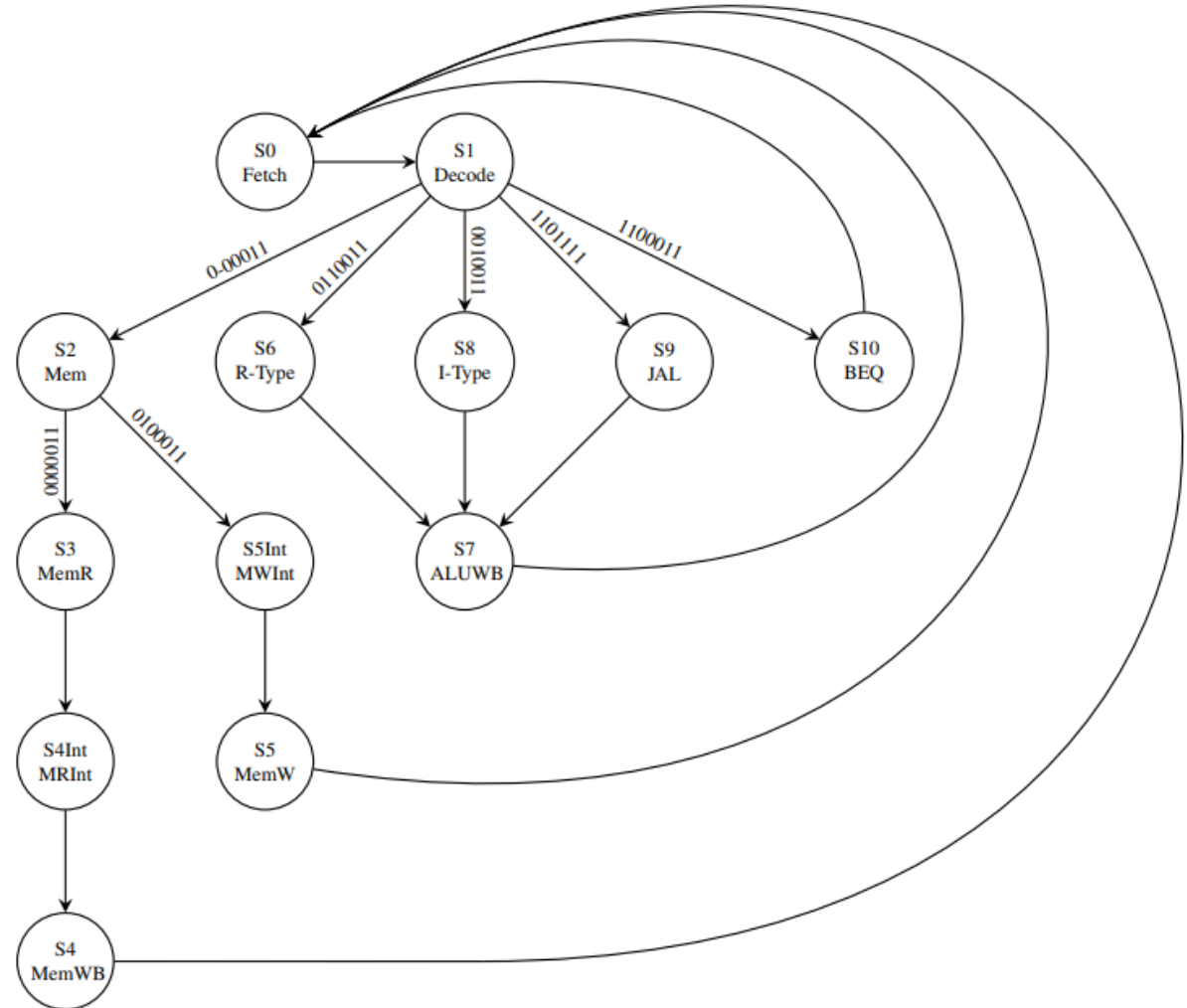
Bestehender synchroner RISC-V Multicycle-Prozessor mit

Harvard Architektur (Getrennter Programm- und Instruktionspeicher)



Automatengraph des synchronen Moore-Automaten

- Input: opcode(6:2)
- S0: Fetch
- S1: Decode
- S2-S10 Instructions



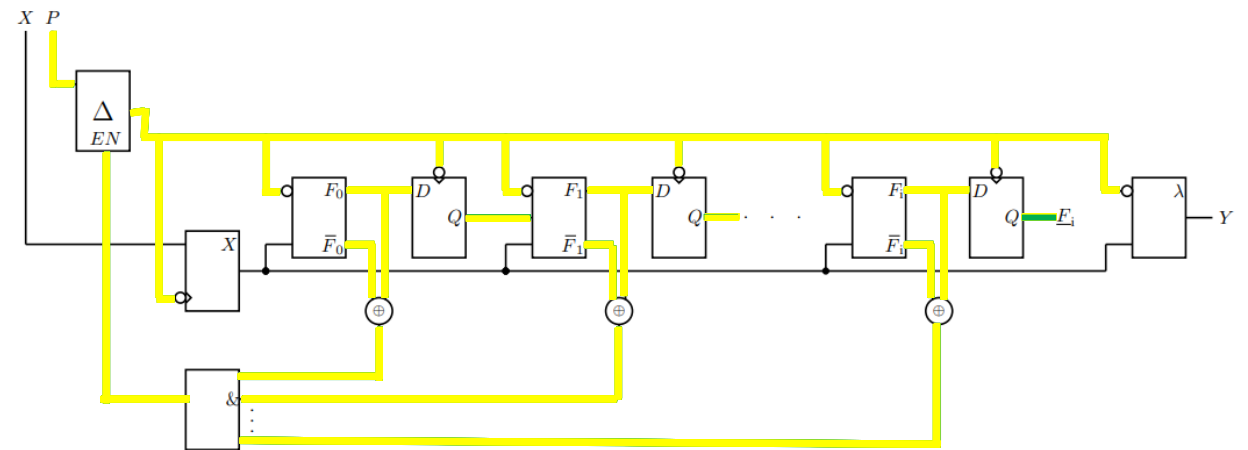
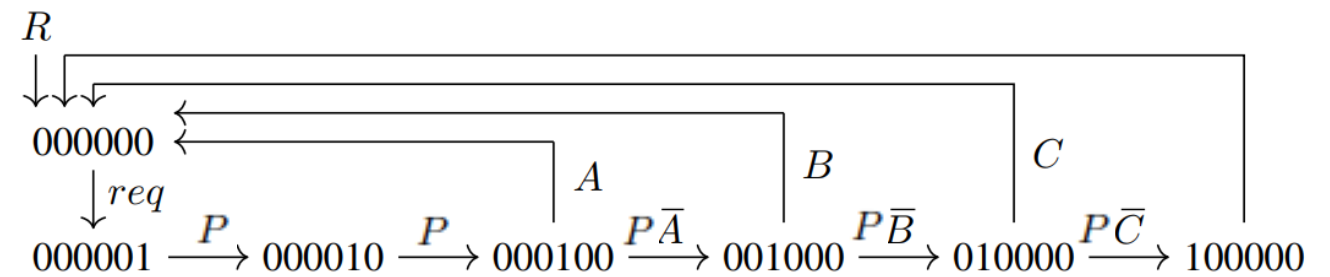
Anwendungsbeispiel: RISC-V

Implementierung der asynchronen Steuereinheit



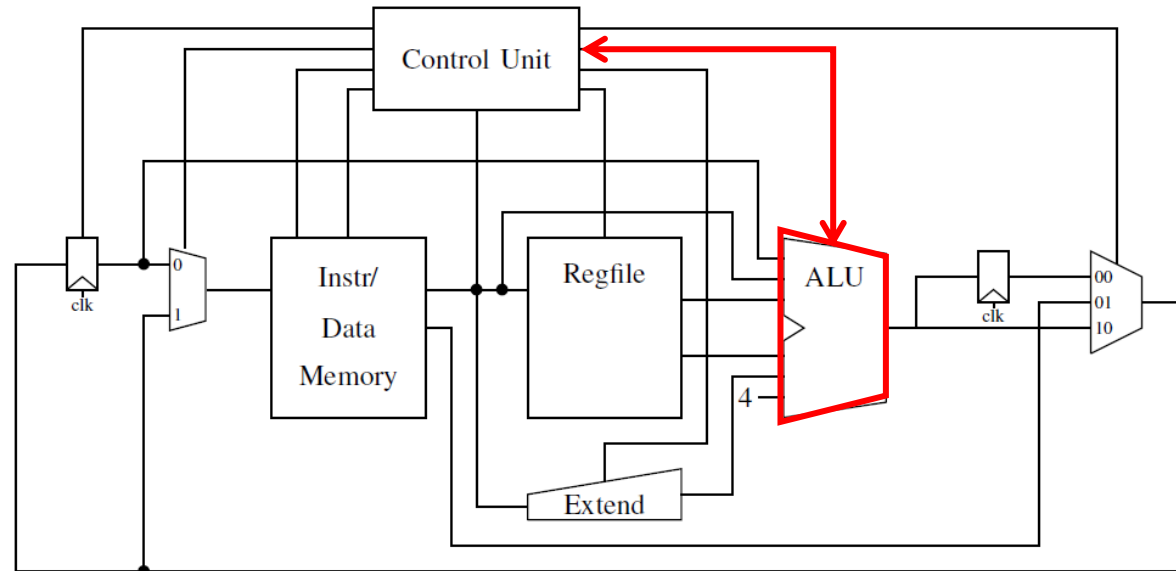
Asynchroner Mealy-Automat in Dominologik

- Input: opcode(6:2)
- One-hot kodiert (außer IDLE[000000])
- S0: IDLE
- S1: FETCH
- S2: DECODE
- S4-S32: INSTRUCTIONS



Ziel: Vorteil durch Handshaking aufzeigen

→ ALU asynchron realisieren

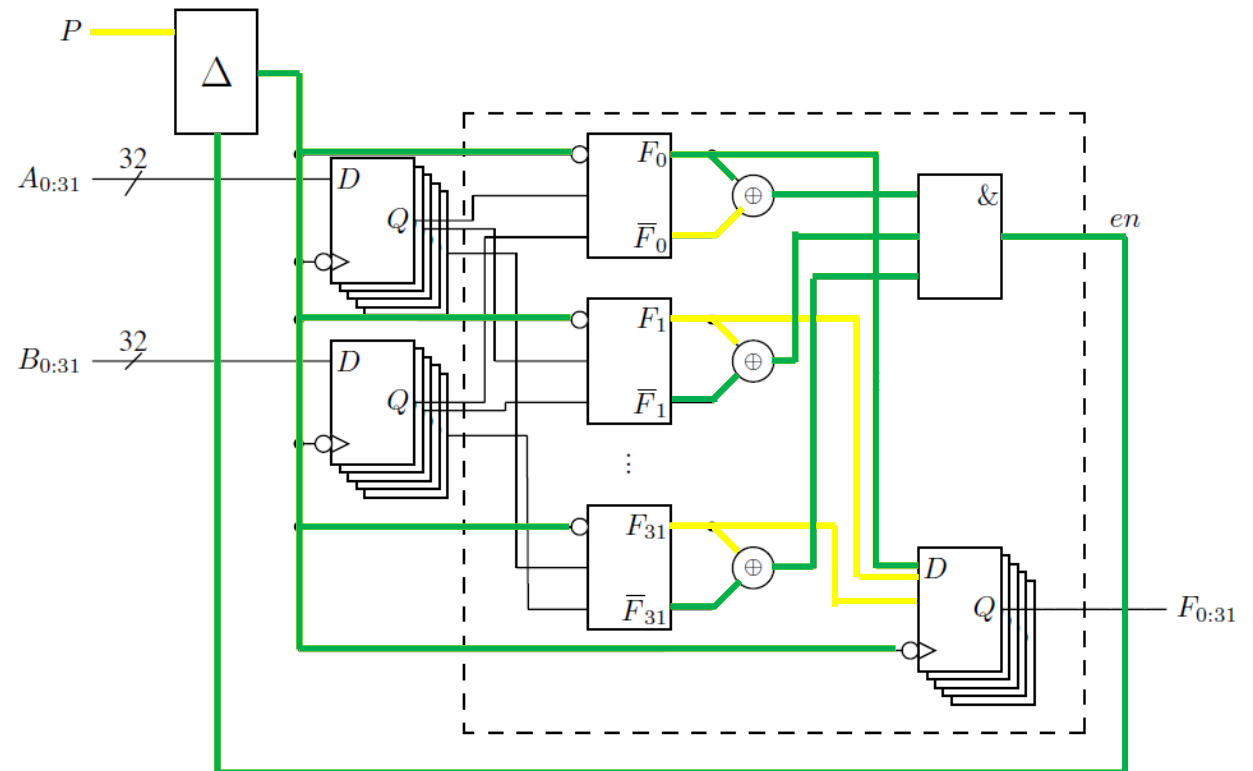


Anwendungsbeispiel: RISC-V

Implementierung der selbstsperrenden ALU



- Handshaking mit der Steuereinheit
- Parallelisierung von Dual-Rail Dominogattern

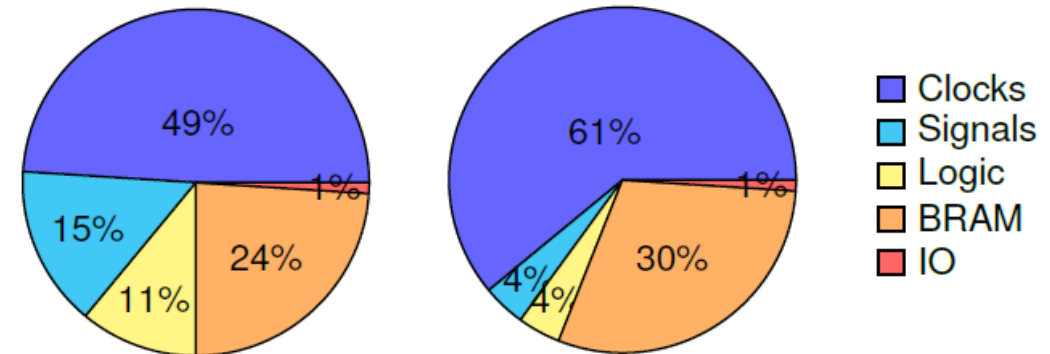




- Asynchron benötigt mehr Fläche

Resource Type	Utilization in (%) Async	Utilization in (%) Sync
LUTs	6.67%	6.32%
Slice Registers	4.85%	4.9%
Slices	9.27%	8.9%

- Geringerer Leistungsverbrauch von Clocks



- Höhere Performanz

Parameter	Asynchronous CPU	Synchronous CPU
Throughput (MIPS)	25.64	22.73
Average Latency/Instruction	39.5 ns	44 ns

Zusammenfassung und Ausblick

- Fokus der Arbeit: Entwurf und Umsetzung asynchroner Automaten in FPGAs
- Asynchrone Schaltungen mit Vorteilen gegenüber synchronen
- Analyse und Vergleich zweier Implementierungen
 - Taktloser Automat
 - Für Spezialanwendungen möglich
 - Selbstsperrende Dominologik
 - Selbstsperrende Dominologikschaltungen mit allgemeingültigem und einfacherem Entwurfsprozess
- Anwendungsbeispiel: RISC-V Multicycle-Prozessor



→ Asynchron bedeutet komplexerer Entwurfsprozess und größerer Flächenverbrauch, bietet aber deutliche Vorteile gegenüber synchronen Schaltungen.

- Entwicklung spezieller FPGA-Architekturen und Software
 - optimierte LUTs
 - Routing-Algorithmen
- Zukünftige Schwerpunkte und Optimierungen bezüglich des Prozessor-Designs:
 - Realisierung als Pipeline mittels asynchroner Handshaking-Protokolle
 - Speicher: Einsatz LUT-basierter asynchroner Speicher zur Minimierung von Zugriffszeiten
 - Hybrid-Ansatz: Kombination von synchronen und asynchronen Methoden (GALS-Architekturen)
- Potenzielle Anwendungsfelder für asynchrone FPGA-Entwürfe:
 - Energieeffiziente Embedded-Systeme
 - Neuromorphe Chips
 - Sicherheitskritische Anwendungen

Vielen Dank für die Aufmerksamkeit!

The background of the slide is a light blue color with a series of concentric, curved lines that create a sense of depth and movement, resembling a stylized landscape or a series of ripples.