Power Aspects in Heigh-Performance Processor Design

Priv.-Doz. Dr. Silvia M. Müller
IBM Böblingen
smm@de.ibm.com

Uni Saarbrücken, Jan / 27 / 2004

Rechnerarchitektur: Themen

- Wie funktioniert ein Rechner und seine Komponenten
  - Addierer, Floatingpoint-Einheit, Graphic-Einheit, ...
  - Befehlsausführung, Speicheranbindung, ...
- Beschleunigen von Designs
  - Verschiedene Implementierungen von Komponenten / Funktionen
  - Parallelisieren der Verarbeitung
  - Schätzen / vorhersagen und mgl. korrigieren
- Bewerten und Optimieren von Designs
  - Satz verschiedener Gatter: Inverter, Nand, Nor, ...
  - Technologieunabh. Verzögerung und Fläche eines Gatters
    - Relativ zu Delay / Transitoranzahl eines Inverters
- Korrektheit des Designs

Rechnerarchitektur & Industrie

- Hardwareentwicklung in Deutschland
  - IBM, Infineon, Etas (Bosch), Motorola, ...
- Hardwareentwicklung - IBM Böblingen
  - Heigh-Performance Prozessoren für
    - Spielekonsolen, Desktop-Systeme und Highend Server
  - Entwickeln von vollständigen Systemen
    - Parallelrechner
    - Speicheranbindung und I/O
    - RAS: zuverlässig, geringe Ausfallzeit, leicht zu warten
    - Technologie
  - Hand in Hand mit Software- und Anwendungsentwicklung

➡ Herausfordernde Aufgaben für talentierte, gut ausgebildete Absolventen

Tripple Constraint

- Performance: Frequenz / Laufzeit von Anwendungen
  - Anwendung doppelt so schnell
  - Frequenz um x% steigern
- Price / Chip-Area
  - Senken der Systemkosten -> Limit für Chipgrösse
- Power
  - Kühlen des Systems
  - Lebensdauer der Batterie
  - Limit an den Stromverbrauch des Prozessors
- *Ziel: 2-fache Performance bei 1/2 Area und 1/2 Power*

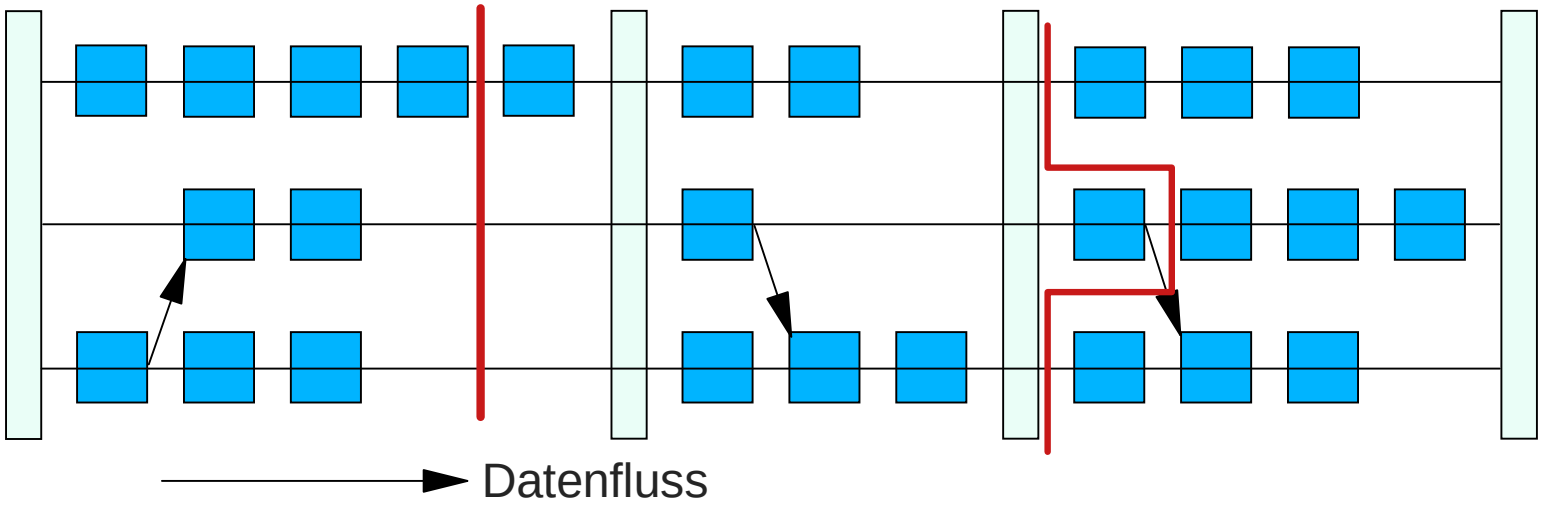
Überblick

- Performance Power Tradeoffs
  - Circuit-Optimierungen
  - Logik-Optimierungen
- Gating: Abschalten von Komponenten nach Bedarf
  - Granularität: Gatter ... Unit ... Prozessor ... System
  - Reaktionszeit: pro Takt .... Sekunden ... Minuten
- Reduzieren des Leckstroms - Forschung
- Zusammenfasung & Ausblick

Power

- Strom zum Schalten der Komponenten
  - Wenn Gatter den Zustand wechselt
  - Abhängig von der Anwendung
- Stromverbrauch durch Kurzschlüsse
  - Kurzfristig beim Schalten einiger Gatterarten
  - Durch fehlerhafte Logik / Circuits
- Leckstrom
  - Gatter schalten nicht vollständig ab

Balanziertes Design

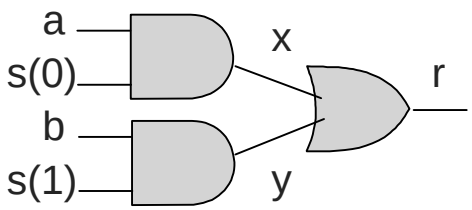
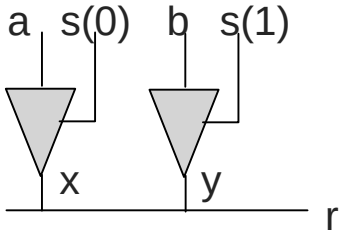
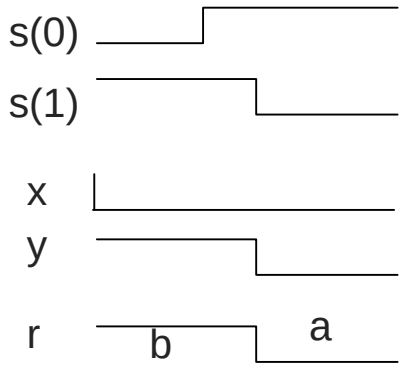
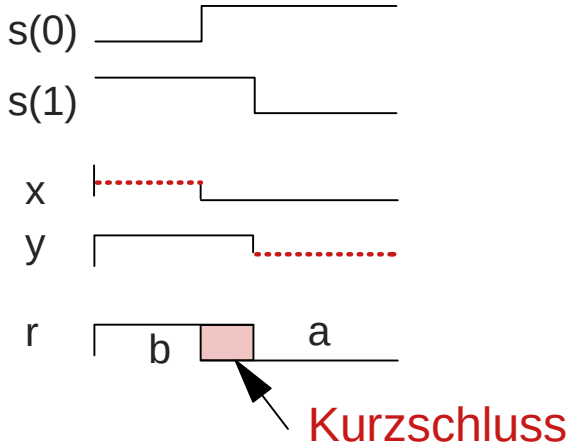


- Arbeit gleichmässig verteilen über die Takte / Stufen
  - Langsamste Teilrechung bestimmt die Cycle-Time
  - Verschieben der Registergrenzen
- Beschleunigen des kritschen Pfades
  - Anpassen der Transitorgrösse -> Power steigt
  - Schnellere Logik- / Circuit-Umsetzung
- Downsizing der unkritischen Pfade -> Power sinkt

Ciruit Optimierungen

- Verändern der Transitor-Grösse
  - Aspekt-Ratio: Länge, Breite
    - z.B.: geringerer Widerstand für kurzen, breiten Draht
  - Schaltverhalten: Last am Datenausgang
    - z.B.: Inverter kann etwa 4 Inverter seiner Grösse treiben
    - Grösse der Transitoren der Last anpassen
    - Grössere Devices benötigen mehr Power
- Verschiedene Implementierungen
  - Statischen vs. Dynamisches RAM
  - And-Or-Mux vs. Transmission-Gate-Mux
  - ....
- Verschiedene Circuit-Familien

Mutiplexor Designs (I)

2-Port Mux	And-Or-Mux	TG-Mux (transmission gate)
r = a if (s = 01) b if (s = 10)		
Latenz abhängig von # Ports:	steigt logarithmisch	steigt gering / konstant
Schaltverhalten: a = 0, b = 1		

Multiplexor Designs (II)

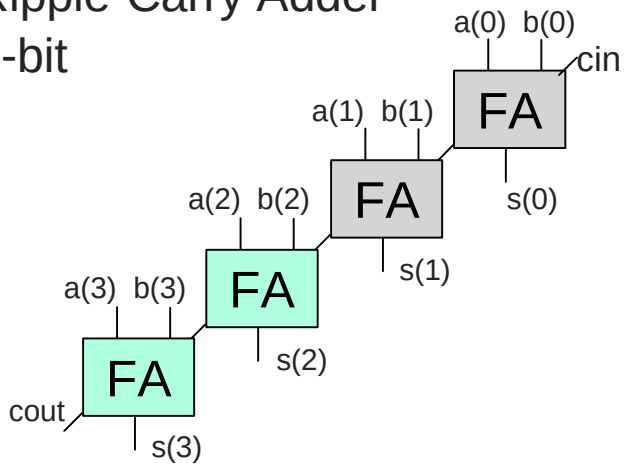
- And-Or Mux
  - Geringere Power als TG-Mux
  - Beliebige Select-Signale erlaubt -> sicherer
- TG-Mux
  - Schneller als And-Or-Mux
    - 4-port Mux: etwa Faktor 30% schneller
    - Vorteil steigt mit # Ports
  - Kurzzeitiger Kurzschluss beim Umschalten der Selects
    - Ursache für höhere Power
  - Select-Signale müssen orthogonal sein
    - Kurzschluss möglich wenn mehrere Ports aktiviert werden
    - Gefahr für die Schaltung und den Chip
    - Verifikation
- Tradeoff zwischen Performance & Power

Logik-Optimierungen

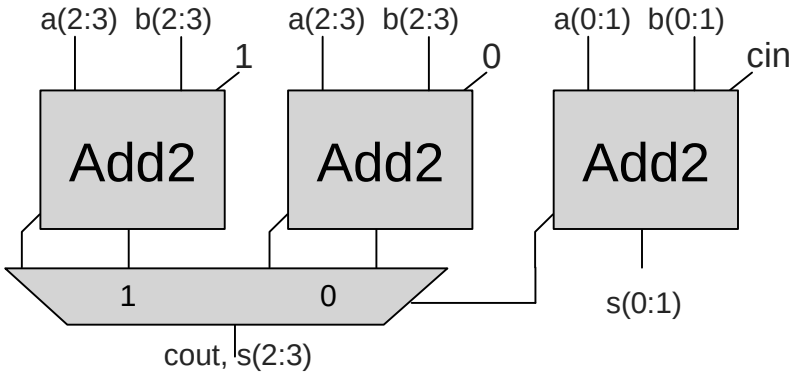
- Reduzieren der Latenz zu Lasten der Power
  - Vorberechnen von 2 Alternativen und später selektieren
    - z.B.: Conditional-Carry-Adder
  - Vorhersage und korrigieren falls notwendig
    - z.B.: Branch-Prediction, Leading-Zero-Anticipator
  - .....
- Reduzieren der Latenz und Power
  - Compound-Adder: berechnen von a+b und a+b+1
  - End-Around-Carry für Abs(x)
  - Sum-addressed Cache / Shifter
    - Kombinieren von Addition und Dekodiere
  - ...

Conditional Carry Prinzip

Ripple-Carry Adder
4-bit

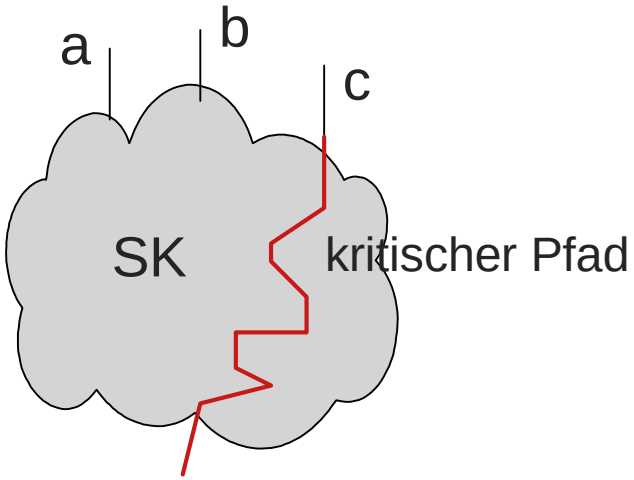


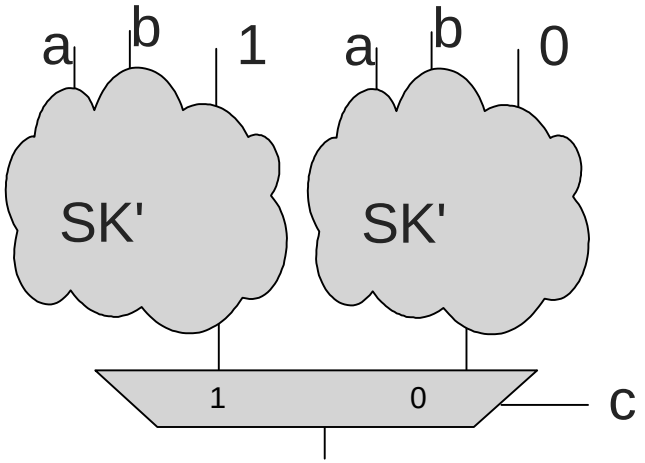
Conditional Carry Adder: 4-bit



- Carry-Berechnung ist zeitkritisch
  - Carry-Pfad in der Mitte aufbrechen:
  - Berechnen der Summe für Carry 0 und 1
  - Latenz etwa halbiert, Area / Power etwa 30% höher
- Verfahren kann rekursiv angewendet werden
  - Logarithmische Tiefe für einen n-bit Addierer

Conditional Select Prinzip



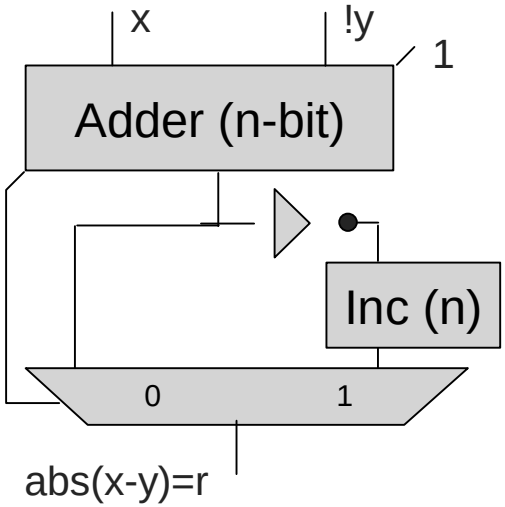


- Verallgemeinerung des Conditional-Carry Prinzips
  - Vorberechnen von 2 Versionen des Ergebnisses
    - Zeitkritischer Input wird auf 0 bzw 1 gesetzt
    - Optimieren der Schaltung SK für c=0 bzw c=1
  - Auswählen des Ergebnisses mit Hilfe von Signal c
  - Laufzeit wird reduziert, Kosten steigen an

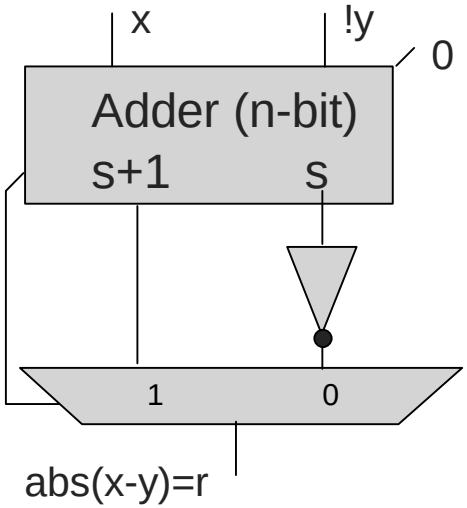
End-Around-Carry (EAC)

- Absoluten Differenz zweier n-bit Binär-Zahlen
  - abs(x-y) = x-y if x>y
    -x+y else
    x-y = x + !y +1 (modulo 2^n)
    -x+y = !(x + !y)

Conventionell:



EAC:

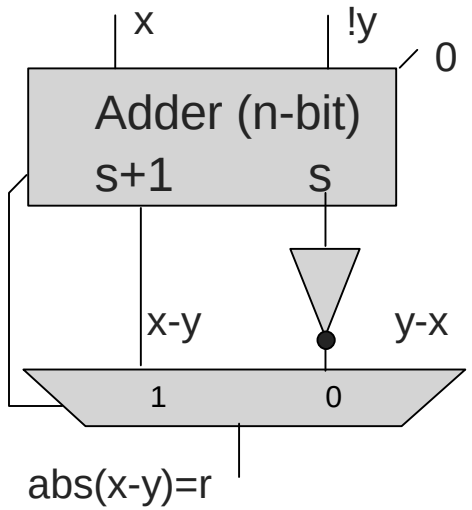


- Einsparen der Latenz des Incrementers (etwa 30%)
- Einsparen der Power des Incrementers

End-Around-Carry

- Rechnung in 2's Complement, modulo 2^n
  - -a = !a +1 => !a = -a -1
  - ! (x + !y) = -(x + !y) -1
    = -x - (!y + 1)
    = -x - (-y) = y -x
  - x > y <=> cout(x + !y) = 1
    - gleiche führende Stellen
    - erstes Bit mit x(i) != y(i)
    - da x, y positiv gilt x(i)=1, y(i)=0

```
     x =  x(n-1:i-1) 1 *.....*
     y =  x(n-1:i-1) 0 #....#
x+ !y >  1.........1 1 0.....0
                      +1
```



Überblick

- Performance Power Tradeoffs
  - Circuit- und Logic-Optimierungen meist zu Lasten der Power
  - Wenige Optimierungen sind Power neutral
  - Power einsparen: Downsizing auf unkritischen Pfaden
- Abschalten von Komponenten nach Bedarf
- Reduzieren des Leckstroms - Forschung
- Zusammenfasung & Ausblick

Abschalten von Hardware

- Rechner besteht aus vielen Komponenten
  - Prozessor (einer / mehrere)
  - Mehrstufige Speicheranbindung
    - Cache, Controller, Memory, Disk
  - I/O-Anbindung: Controller und Devices
    - z.B.: Tastatur, Bildschirm,
- Auslastung der Komponenten abh. von der Anwendung
- Idee:
  - Komponenten je nach Bedarf an- und abschalten
  - Besonderns wichtig bei portablen Geräten
- Abschalten: was und wie ?

Abschalten aber wie?

- Abschalten der Stromversorgung
  - Einschalten dauert lange
    - Power / Clock ramp-up, Initialisieren
  - Daten in Speicher und Register gehen verloren
    - Retten und Restaurieren von Daten
  - Nur gesamte Chips oder Bausteine
- Abschalten oder runtertakten der Clock
  - Lokale Daten bleiben erhalten (ausser Dynamische Speicher)
  - Einschalten: ramp-up der Clock im ms-Bereich
  - Separate Clock-Grids notwendig
  - Grosse Blöcke auf einem Chip
- Lokales Gaten der Clock
  - Clock wird am Register-Eingang aktiviert / deaktiviert
  - Feinste Granularität: pro Register
  - Umschalten pro Takt möglich ohne Clock ramp-up

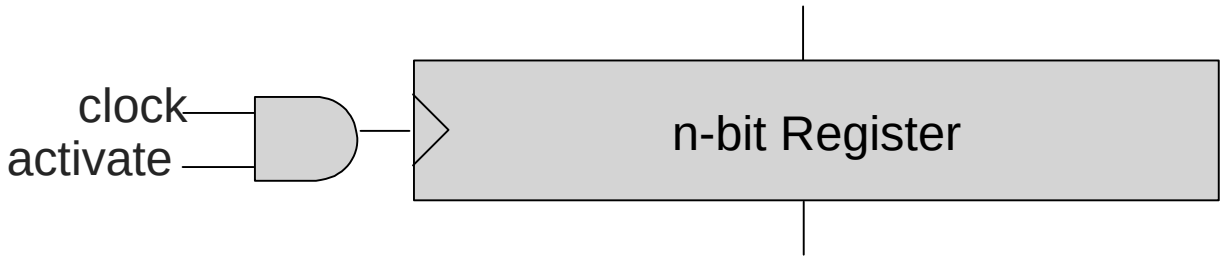
Power Save Modes

- Hibernate-Mode
  - Rechner lange idle, (Zeitschranke)
  - Abschalten der Stromversorgung => 0 Power
  - Retten aller Daten auf Festplatte: Speicher & Registerinhalte
  - Reaktionszeit: mehrere Minuten
- Sleep-Mode
  - Daten bleiben im Speicher und den Registern
  - Periferie abschalten: Treiber, Festplatte, Bildschirm, ...
  - Prozessor runterfahren soweit möglich
    - Check für Tastatureingabe bleibt aktiv
  - Kürzere Reaktionszeit: Sekunden

➡ Power einsparen wenn Rechner kurzfristig nicht genutzt

Clock Gating

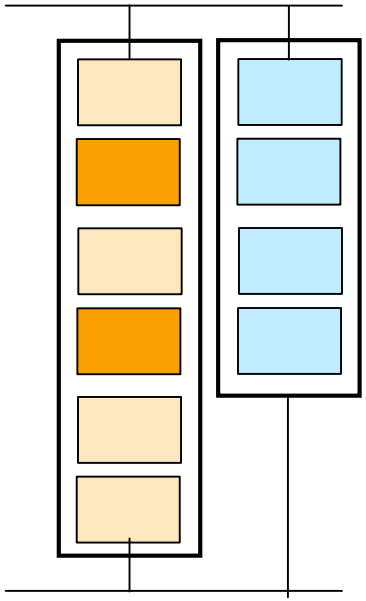
- Wie funktioniert das
  - Clock-Signal wird am Register mit Activate verknüpft
  - Lokale Clock des Register nur aktiv
    - wenn Clock schwingt und wenn Activate auf 1
- Warum spart das Power ?
  - Clock des Registers deaktiviert -> Daten-Output bleibt stabil
  - Inputs für nachfolgende Logic stabil
    - kein Umschalten der Gatter -> Einsparen von Power
- Gating auf Register-Ebene, Steuerung pro Takt



Gating: Granularität

1. Gaten einer gesamten Unit U
   - Test: Opcode der Instruktion
     - Unit aktiviert sobald eine Instruktion für U bearbeitet wird
     - Alle Register in U aktiviert mit Unit-activate
   - Spart Power wenn Unit nicht benutzt wird

(1)

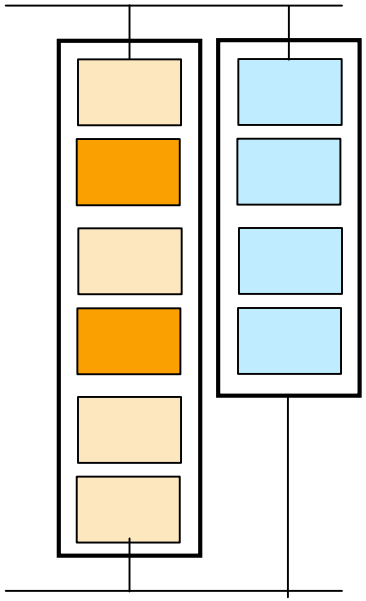


Gating: Granularität

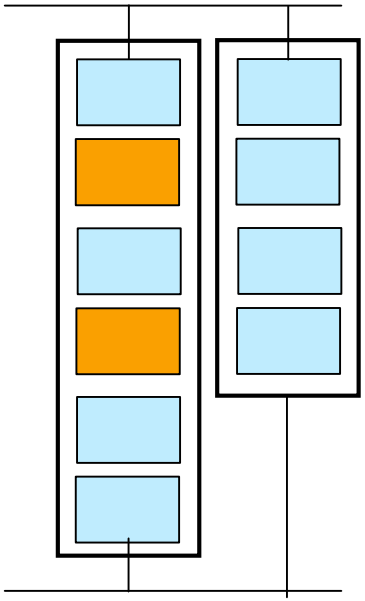
1. Gaten einer gesamten Unit U
   - Spart Power wenn Unit nicht benutzt wird

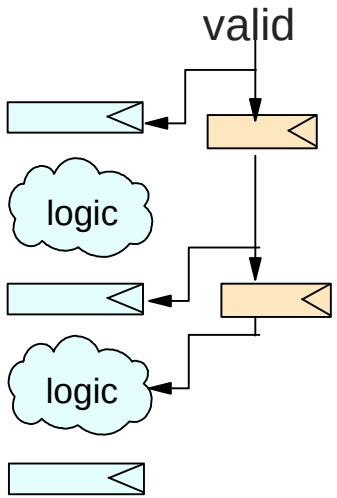
2. Gaten von Pipelinestufen
   - Test: aktivieren anhand von einem Valid Bit
     - Nur Stufen mit gültiger Instruktion werden aktiviert
   - Spart Power wenn Unit nicht voll ausgelastet

(1)



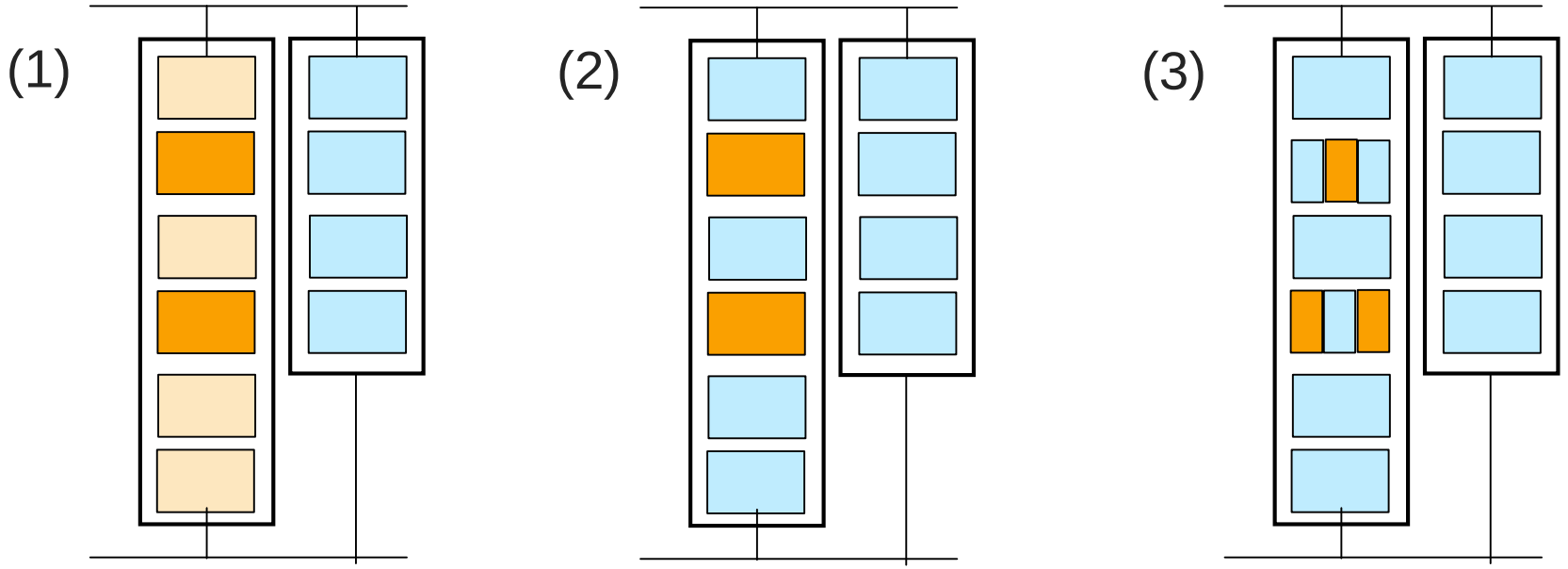
(2)



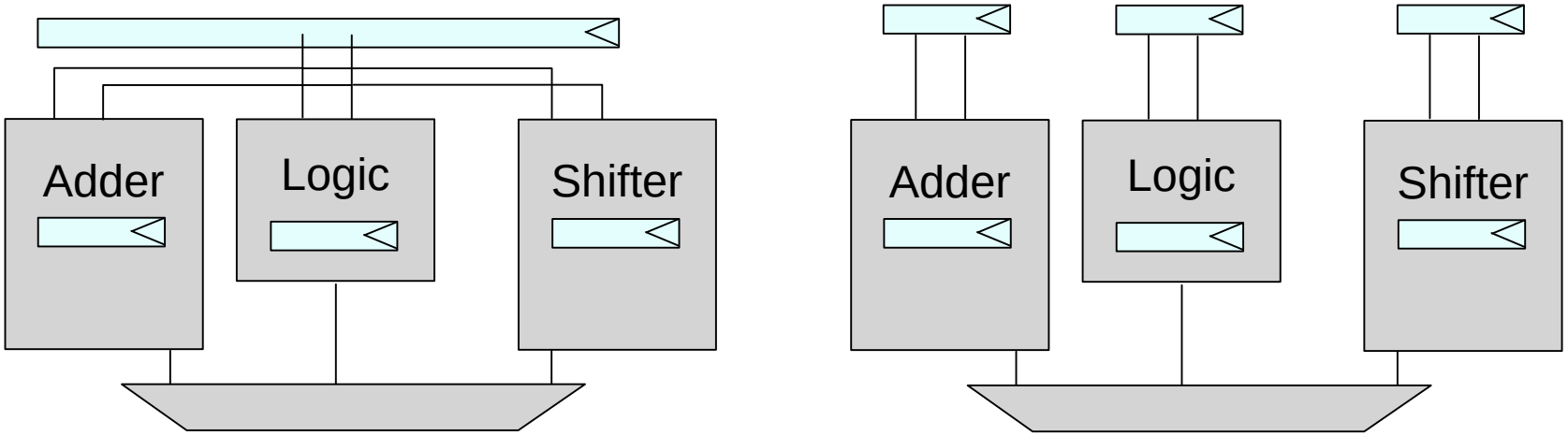


Gating: Granularität

1. Gaten einer gesamten Unit U
2. Gaten von Pipelinestufen innerhalb der Unit
3. Gaten von Komponenten innerhalb einer Pipelinestufe
   - Test: anhängig von Instruktion und Daten
   - Verfeinern des valid-Bits
     - Mehrere Aktivate-Bits pro Pipeline-Stufe
   - Spart Power selbts wenn Unit voll ausgelastet ist



Gating: ALU



- Gemeinsame Operand-Register
  - Erste Stufe der Blöcke gemeinsam aktiviert
- Getrennte Operand-Register
  - Adder, Logic-Unit und Shifter einzeln zu aktivieren
  - Lohnt die extra Hardware?
  - Power / Area Tradeoff

Gating - Fragestellungen

- Abhängig von Instruktionen und Daten
  - Welche Teile des Prozessor / der Einheit werden benötigt
  - Wie kann der Rest abgeschaltet werden
  - Ist zum Abschalten zusätzliche Hardware notwendig
- Rentiert sich die extra Hardware fürs Gating
  - z.B.: extra Register um funktionale Blöcke abzukoppeln
  - Evaluierung anhand von Anwendungen
- Verifikation des Gating
  - Funktional: berechnet die Hardware noch das richtige
  - Wird alles abgeschaltet wie geplant

Überblick

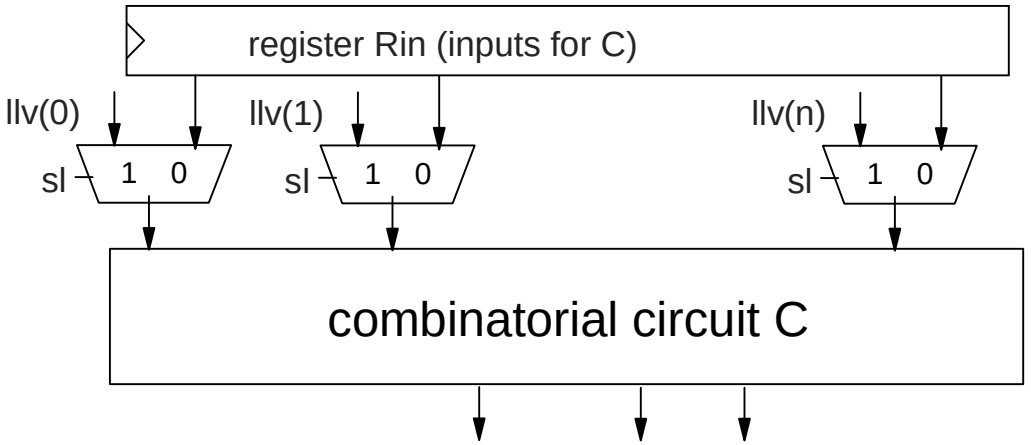
- Performance Power Tradeoffs
- Abschalten von Komponenten nach Bedarf
  - Anfangs Abschalten von ganzen Chips
    - Einsparen von Power wenn System nicht ausgelastet
  - Gating bis auf Register-Ebene / innerhalb von Pipe-Stufen
    - Einsparen von Power wenn System ausgelastet ist
- Reduzieren des Leckstroms - Forschung
- Zusammenfasung & Ausblick

Reduzieren des Leckstroms

- A. Abdollahi, F Fallah, M Pedran, ISQED-2003
  - Leakage Current Reduction ... by Modifying the Scan Chain
- Beobachtung
  - Leckstrom eines Gatters abh. von den Inputs
  - Geeigneten Inputs kann Leckstrom von Schaltungen um 35% - 50% reduzieren
  - Gute Input-Belegungen können berechnet werden
    - low leakage vector LLV
- Wie kann man Schaltung in LLV versetzen?

Reduzieren des Leckstroms

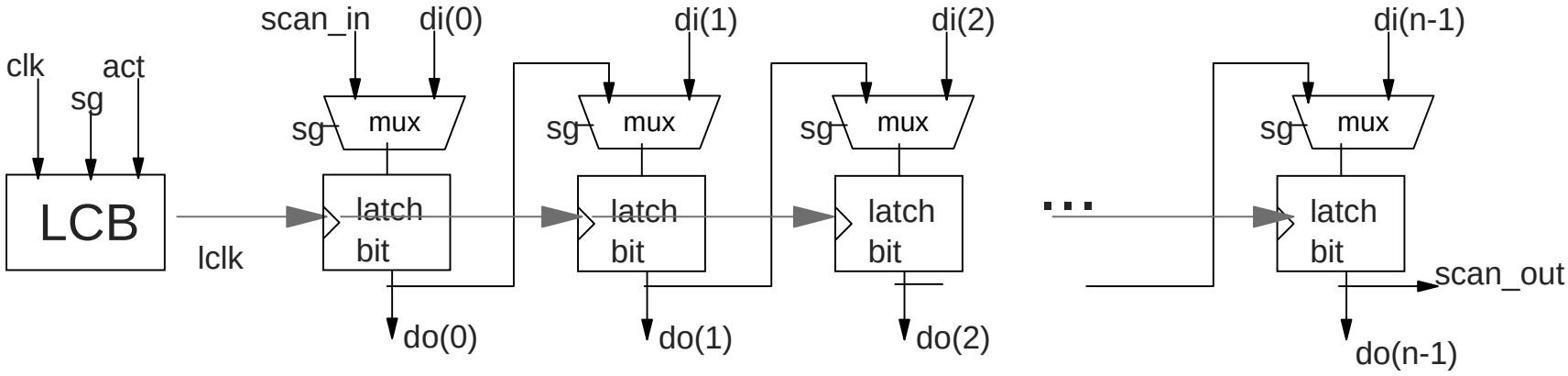
- Input-Multiplexor Methode
  - Mux hinter jedem Register um LLV anzulegen
  - Vorteil: einfach über select-Signal zu steuern
  - Nachteil: Latenz der Schaltung steigt an



Reduzieren des Leckstroms

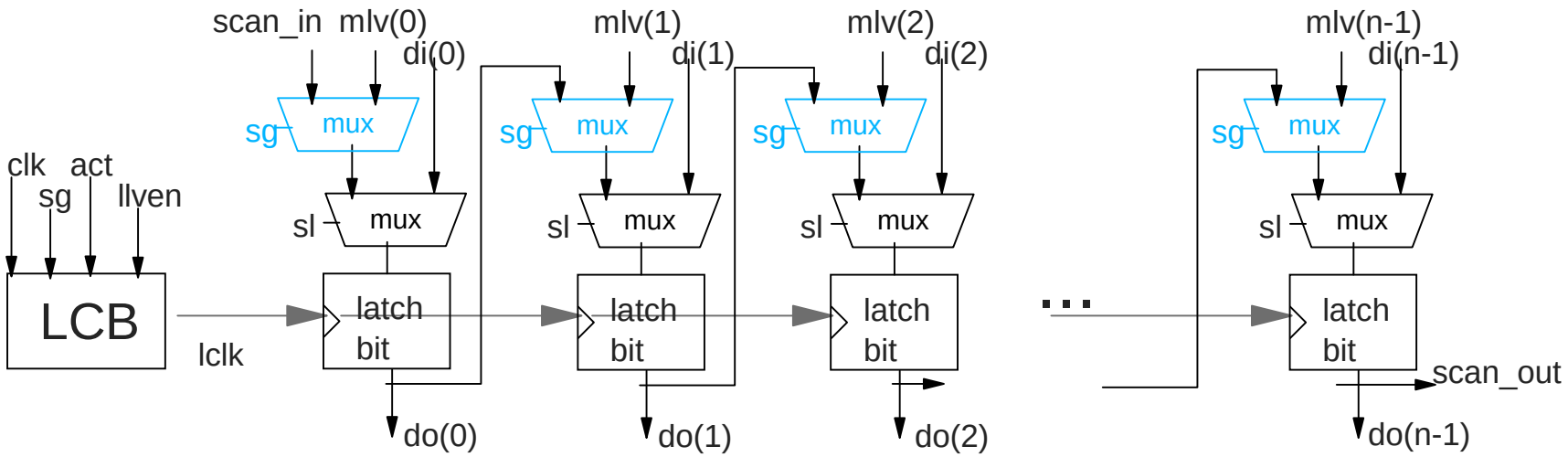
- Scan-Methode
  - Alle Register sind für Test als grosse Schieberegister organisiert
    - Scan-Chain ist um die 1000 Bits lang
  - Einscannen der LLV Wertes in die Register
  - Vorteil:
    - Latenz der Schaltung bleibt gleich,
    - keine extra Hardware
  - **Nachteil:** viele extra Takte Switching bis LLV am richtigen Ort
    - Power-Verlust beim Scannen grösser als Einsparung durch LLV ?

n-bit scan register



Reduzieren des Leckstroms

- Modifizierte Scan-Methode
  - Spezielle Scan-Register mit 2tem parallelem Daten-Port
  - LLV wird über neuen Port eingelesen als specielle Scan-Funktion
  - Vorteil:
    - Latenz der Schaltung bleibt gleich
    - Einlesen in einem Takt unabh. von der Länge der Scan-Chain
  - Nachteil: Umschalten in Scan-Mode



Überblick & Ausblick

- Performance Power Tradeoffs
- Abschalten von Komponenten nach Bedarf

- Reduzieren des Leckstroms - Forschung
  - Kann man den Scan-Mode vermeiden ?
  - Wann wird LLV in Register transferiert ?
    - Kann Applikation Hilfestellung geben?

- **Power-Saving für Hochleistungs-Rechner**
  - Aufwand für Power-Management steigt
  - Forschung auf vielen Gebieten nötig
    - Technologie
    - Bessere Circuit- & Logik-Verfahren
    - Kann OS / Anwendung Hilfestellungen geben