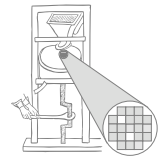


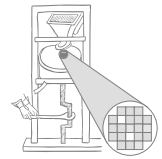
Einführung in Computer Microsystems

6. Systematischer Schaltungsentwurf

Andreas Koch

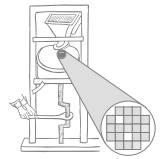
FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

Sommersemester 2007



Einleitung

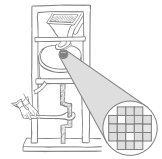
Zwischenstand



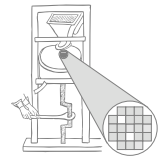
- Überblick über Verilog
  - Simulation
  - Logiksynthese
- Weitergehende Optimierungen
  - Auch jenseits der Logiksynthese
- Kleine und größere Beispiele

Aber wie ein Problem systematisch lösen?

Systematischer Entwurf

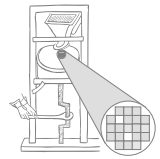


- Verschiedenste Techniken
- Abstraktion
  - Modell
  - Darstellungen
- Schrittweise Verfeinerung



Steuerwerk / Datenpfad-Modell

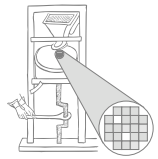
Steuerwerk / Datenpfad



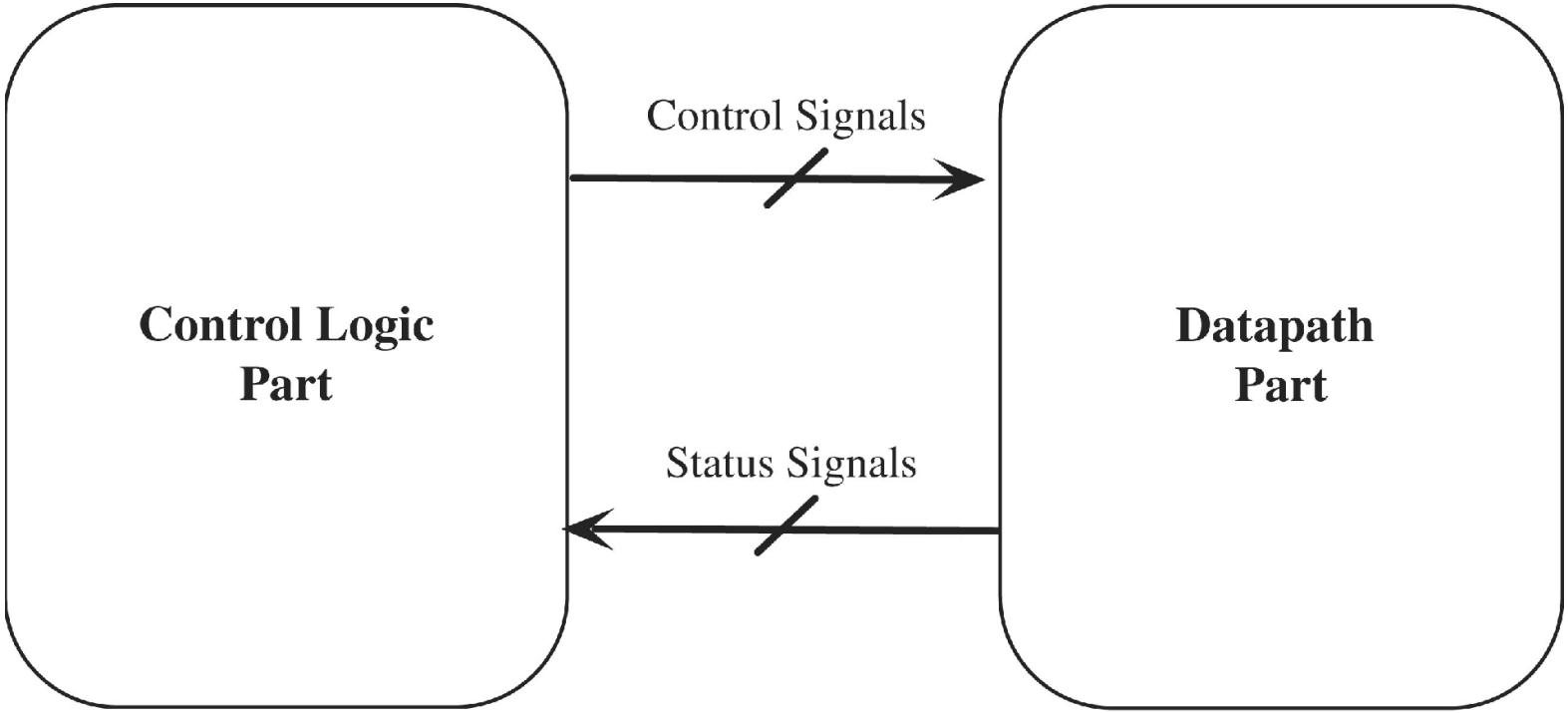
- Kernidee: Trennung von
  - Steuerung von Abläufen (*controller*)
  - Datenverarbeitung (*datapath*)
- In TGDI2 eingeführt als Steueroperationssystem
  - Steuerwerk
  - Operationswerk
- Hier: Verfeinerung und Umsetzung in Verilog

Steuerwerk / Datenpfad - Modell

Kurze Wiederholung



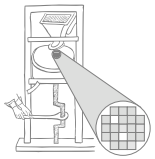
A. Koch



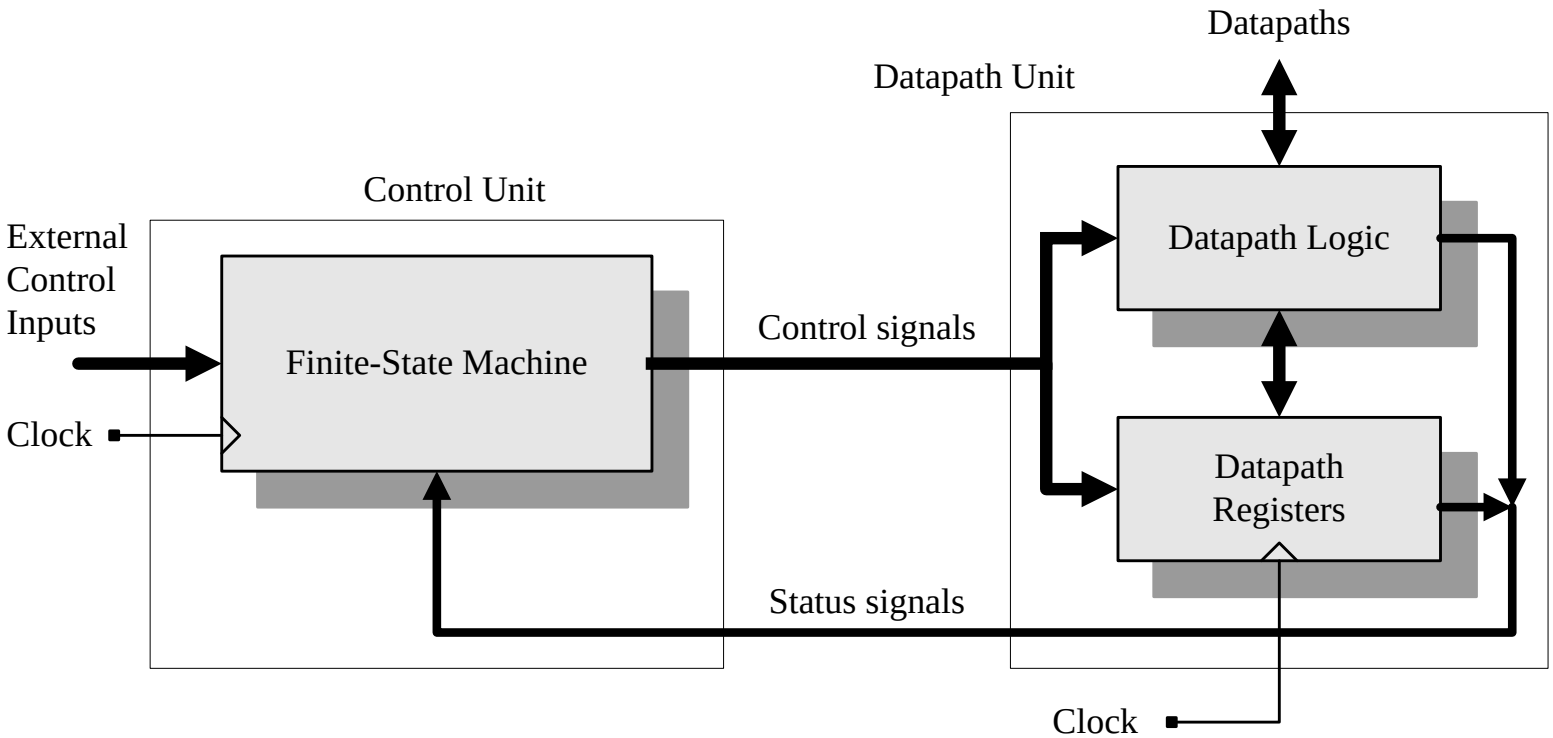
Quelle: Lee, Fig 5.1

Steuerwerk / Datenpfad - Modell

Genauerer Blick



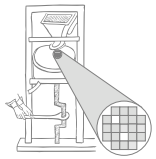
A. Koch



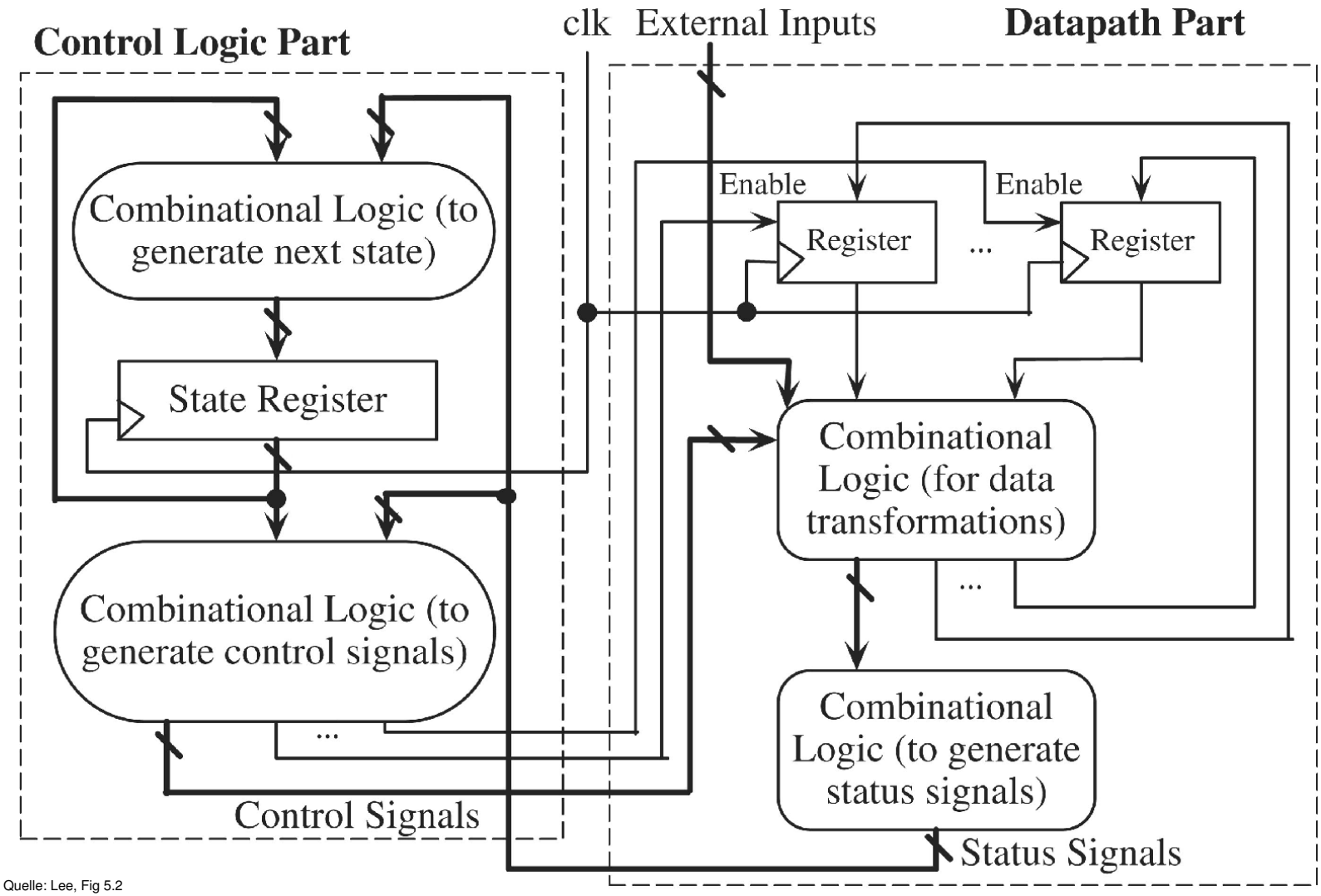
Quelle: Ciletti, Fig 7.1

Steuerwerk / Datenpfad-Modell

Noch genauerer Blick

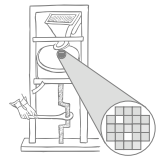


A. Koch



Quelle: Lee, Fig 5.2

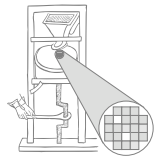
Diskussion



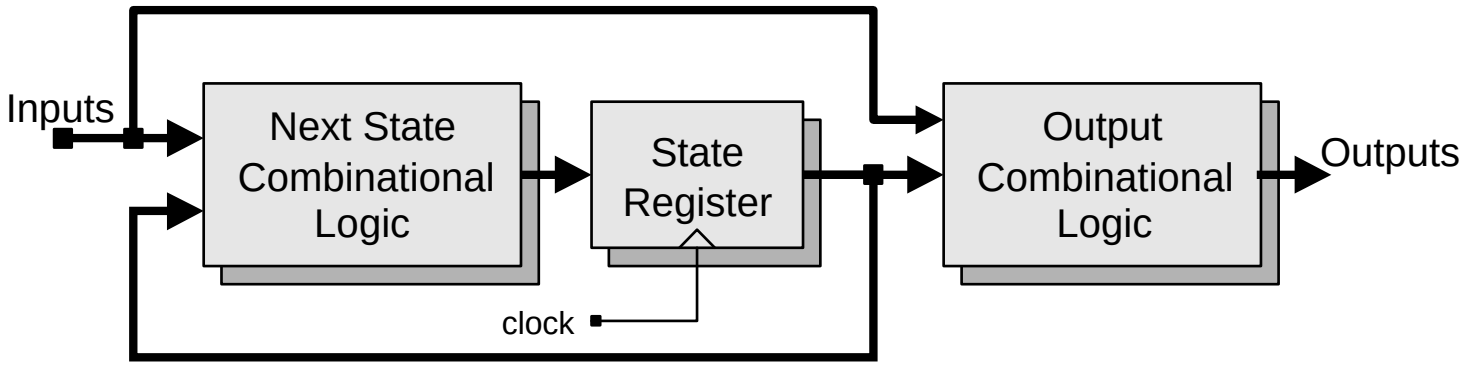
- Steuerwerk: Steuert Abläufe
  - Zustandsautomaten
  - Mealy / Moore / Mischformen
- Datenpfad: Manipuliert Daten
  - Speichern
    - Register
    - Echte Speicher (RAM, ROM)
  - Operationen
    - Arithmetisch
    - Logisch
  - Weiterleiten
    - Multiplexer
    - Tri-State-Busse

Exkurs: Zustandsautomaten in Hardware

Mealy-Automat



A. Koch



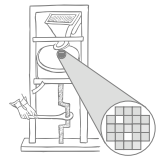
Ausgänge abhängig von

- aktuellem Zustand
- aktuellen Eingangssignalen

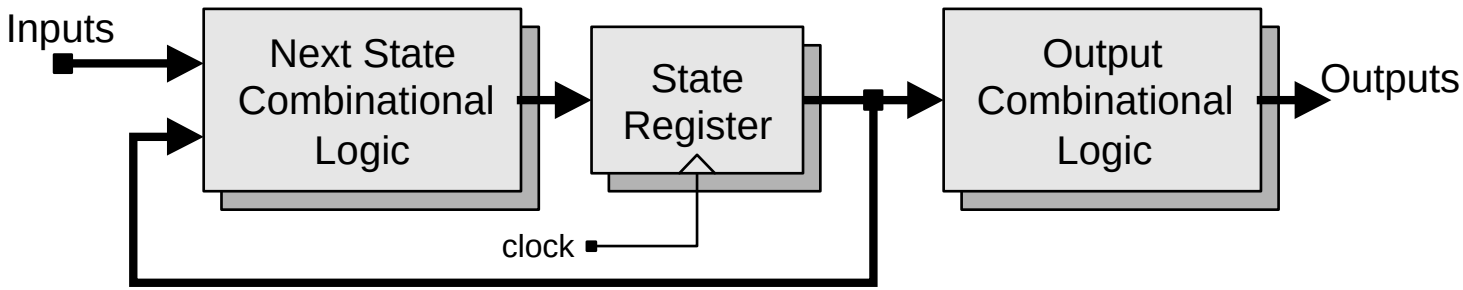
Quelle: Ciletti, Fig 6.44a

Exkurs: Zustandsautomat in Hardware

Moore-Automat



A. Koch

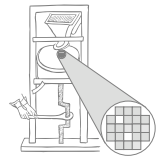


Ausgänge abhängig

- Nur von aktuellem Zustand

Quelle: Ciletti, Fig 6.44b

Beispiele für Kommunikation
Zwischen Steuerwerk und Datenpfad

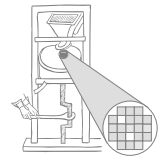


Steuersignale von Steuerwerk zum Datenpfad

- “Übernehme neuen Wert in Register”
- “Schreibe Daten in Speicher”
- “Leite Daten auf 4. Eingang weiter”
- “Führe Additions-Operation aus”

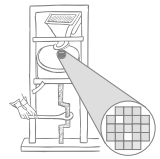
Statussignale von Datenpfad zum Steuerwerk

- “Ergebnis ist Null”
- “Ergebnis ist ungerade”
- “Beide Werte sind gleich”
- “1. Wert ist größer als 2. Wert”

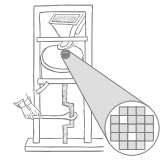


Systematische Konstruktion

Umsetzung von Algorithmen in Hardware

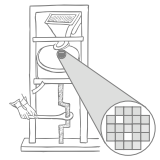


1. Beschreibe Algorithmus in Pseudo-Code
   - Wie beim Programmieren von Software
2. Schreibe Pseudo-Code in RTL-Beschreibung um
   - Keine **`for`**, **`while`**-Schleifen, Prozeduraufrufe
   - Aber Sprünge und **`if/then/else`** sind zugelassen!
   - Nur noch Konstrukte vergleichbar synthetisierbarem Verilog
   - Aber hier noch kein Verilog selbst erforderlich
3. Entwerfe Datenpfad-Struktur
   - Basierend auf Operationen in RTL-Beschreibung
4. Entwerfe Zustandsmaschine
   - auf Basis der RTL-Beschreibung
5. Realisiere Logik für Zustandsmaschine
   - Kann von Logiksyntheseübernommen werden
   - Schauen wir uns hier aber genauer an



Beispiel: Fakultätsberechnung

Pseudo-Code

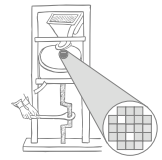


Annahmen

- Eingabe Zahl 0...7 (vorzeichenlos, 3b)
- Ausgabe ist 16b breit
- Signal **`start`**=1 startet Rechnung
- Signal **`done`**=1 zeigt Abschluss der Rechnung an

```
fact[15:0] := 1;
done := 0;

FOR count := 2 TO n[2:0] DO
  fact := fact * count;

done := 1;
```

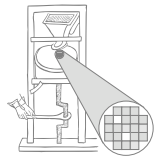
RTL-Beschreibung 1



- Schleifen auflösen
  - In Bedingung und Sprung
- Aufteilen der Rechnung in Einzelschritte
  - Zunächst vergleichbar Assembler-Anweisungen

```
1: fact[15:0] := 1;
2: done := 0;
3: count : =2;
4: IF (count <= n) THEN BEGIN
5:   fact := fact * count;
6:   count := count + 1;
7:   GOTO 4;
8: END
9: done := 1;
```

RTL-Beschreibung 2
Weitere Verfeinerung

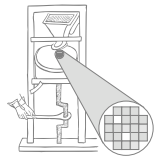


- Hardware rechnet parallel
- Alle parallel ausführbaren Operationen in einem Schritt

```
1: fact[15:0] := 1;
   done := 0;
   count : =2;
2: IF (count <= n) THEN BEGIN
     fact := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done := 1;
```

Jetzt parallele Operationen in einem Schritt

Diskussion: Parallele Operationen



A. Koch

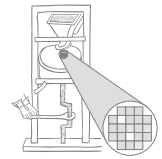
- Alle Operationen in einem Schritt rechnen mit gleichen Variablenwerten
- Zuweisungen werden erst im nächsten Schritt sichtbar

```
1: fact[15:0] := 1;
   done := 0;
   count : =2;
2: IF (count <= n) THEN BEGIN
     fact := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done := 1;
```

```
1: fact[15:0] := 1;
   done := 0;
   count : =2;
2: IF (count <= n) THEN BEGIN
     count := count + 1;
     fact := fact * count;
     GOTO 2;
   END
3: done := 1;
```

➥Gleiches Ergebnis!

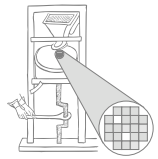
Datenpfad aus RTL-Beschreibung ableiten 1



A. Koch

```
1: fact[15:0] := 1;
   done := 0;
   count := 2;
2: IF (count <= n) THEN BEGIN
     fact := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done := 1;
```

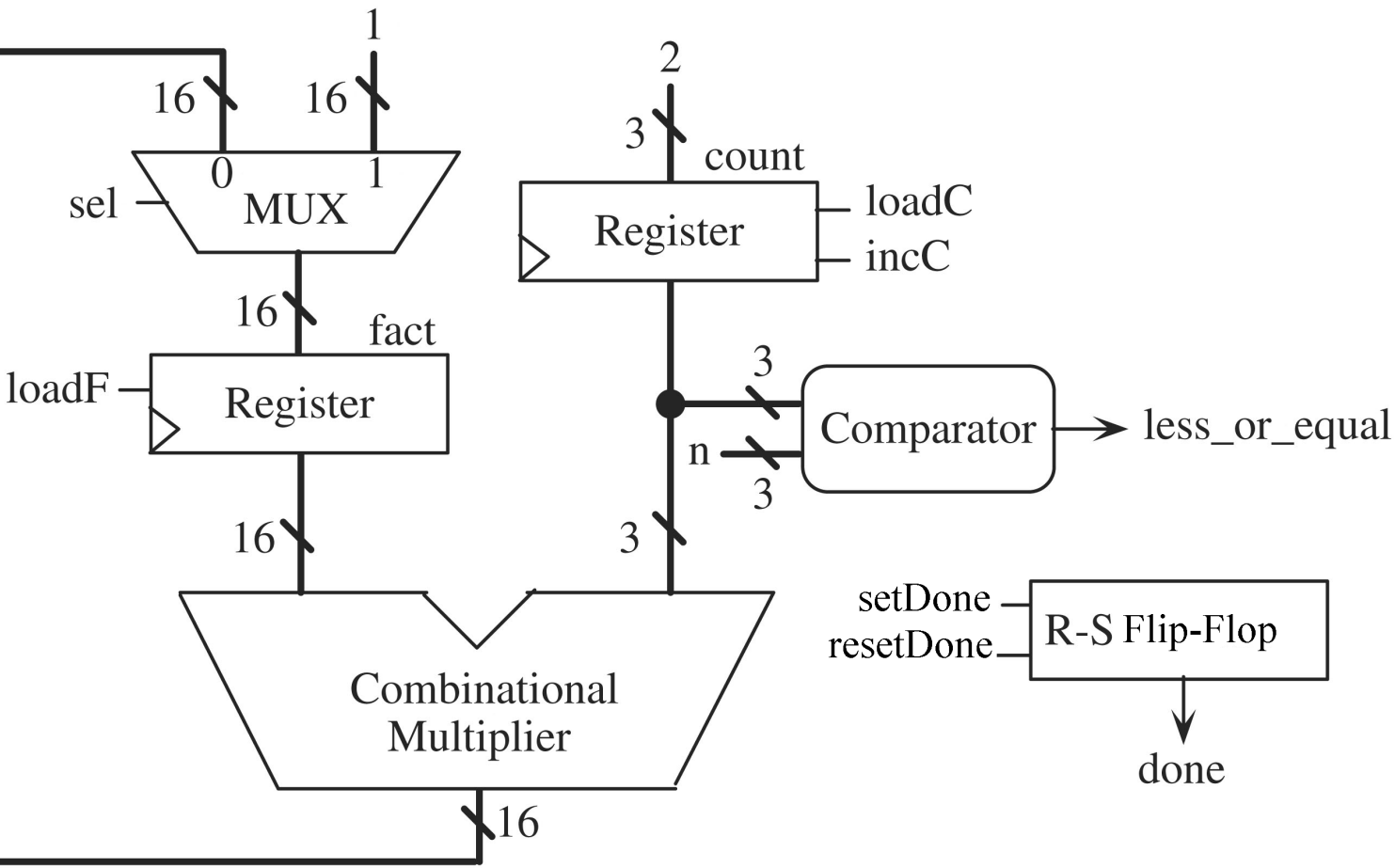
Datenpfad aus RTL-Beschreibung ableiten 2



1. Variablen werden Register
   - Ggf. Spezialregister ausnutzen
   - Zähler bei inkrementieren/dekrementieren
   - Schieberegister bei verdoppeln/halbieren
2. Variablen mit mehreren Quellen für Werte
   - Multiplexer oder Tri-State-Busse am Registereingang
   - Wählt aktuelle Quelle aus
3. Operatoren werden arithmetische/logische Blöcke
4. Steuersignale bestimmen, Beispiele:
   - Wann übernimmt Register neuen Wert?
   - Soll Zähler diesen Takt zählen?
   - Welcher Mux-Eingang soll auf den Ausgang gelegt werden?
   - Was war das Ergebnis eines Vergleichs?

Datenpfad

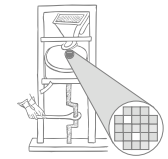
Eine Möglichkeit aus vielen!

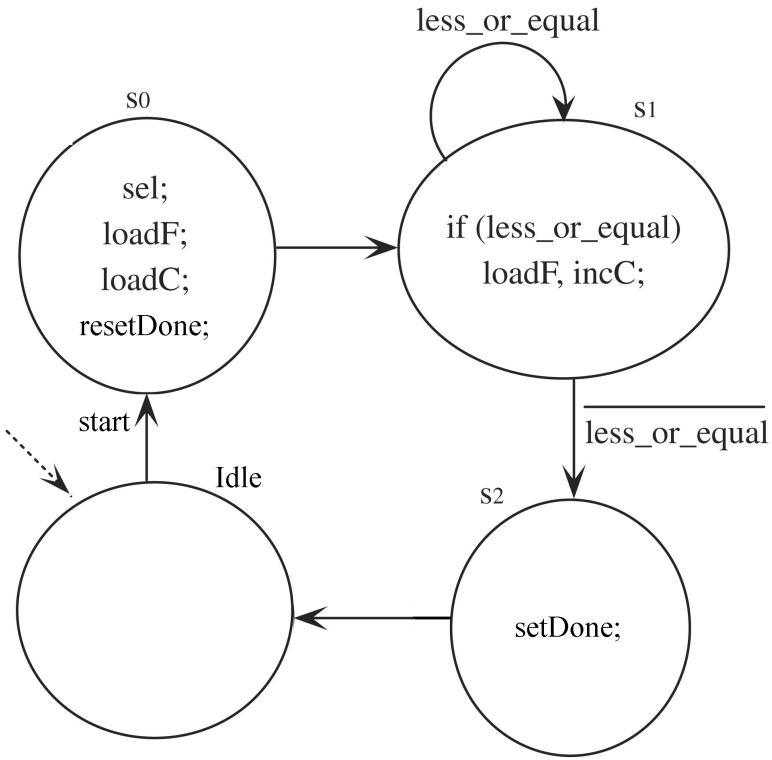


A. Koch

Zustandsautomat als Zustandsübergangsgraph

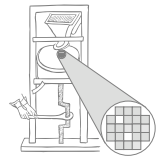
state transition graph (STG)





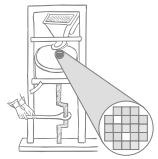
- Steuersignale in Zuständen
  - Nicht aufgeführte Signale → deaktiviert
- Statussignale
  - In Zuständen (Mealy-Ausgang)
  - An Übergängen: Boolesche Verknüpfung von Statussignalen
    - Und nichts anderes!

Verilog-Realisierung



- Nun Verilog-Modell formulierbar
- Saubere Trennung von Steuerwerk und Datenpfad
  - In eigene Module
- Im Steuerwerk saubere Trennung von
  - Kombinatorischen Rechnungen
  - Speicherelementen
  - Register-Transfer-Logik

Hauptmodul des Fakultätsrechners

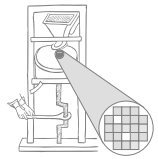


A. Koch

```
module fac (
  input           CLK, RESET,
  input           START,
  input    [2:0]  N,
  output [15:0]   FACT,
  output          DONE
);

  facdp   FACDP (CLK, RESET, SEL, LOADF, LOADC, INCC, RDONE, SDONE,
                 N, LEQ, DONE, FACT);
  facfsm FACFSM(CLK, RESET, START, LEQ,
                 SEL, LOADF, LOADC, INCC, RDONE, SDONE);

endmodule
```

Saubere Trennung von Steuerwerk und Datenpfad

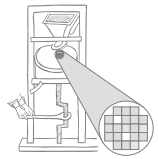
Datenpfad: Schnittstelle und Verwaltung



A. Koch

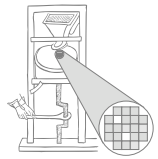
```
module facdp (
  input             CLK, RESET, SEL, LOADF, LOADC, INCC, RDONE, SDONE,
  input       [2:0] N,
  output            LEQ,
  output reg        DONE,
  output reg [15:0] FACT
);

  reg [2:0] COUNT;

  always @(posedge CLK, posedge RESET) begin
    if (RESET) begin
      FACT <= 0;
      COUNT <= 0;
      DONE <= 0;
    end else ...
```

Datenpfad: Aktiver Teil




```
... end else begin

    if (LOADF)                    // Behandlung von fact
      FACT <= (SEL) ? 1 : (FACT * COUNT);

    if (LOADC)                    // Behandlung von count
      COUNT <= 2;
    else if (INCC)
      COUNT <= COUNT + 1;

    case ({SDONE,RDONE})  // Behandlung von done
      2'b10: DONE <= 1;
      2'b01: DONE <= 0;
    endcase

  end
end

// Statussignal less_or_equal für Steuerwerk
assign LEQ = (COUNT <= N);

endmodule
```

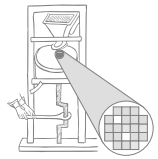
Steuerwerk: Schnittstelle und Verwaltung



A. Koch

```
module facfsm (
  input       CLK, RESET, START, LEQ,
  output reg SEL, LOADF, LOADC, INCC, RDONE, SDONE
);

  parameter IDLE = 0;
  parameter S0 = 1;
  parameter S1 = 2;
  parameter S2 = 3;
  reg [1:0]  STATE, NEXTSTATE;

   ...
```

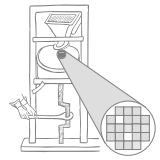
Steuerwerk: Aktiver Teil



A. Koch

```
always @(STATE,START,LEQ) begin

  SEL = 0; LOADF = 0; LOADC = 0; INCC = 0; RDONE = 0;SDONE = 0; // Latches vermeiden
  NEXTSTATE = IDLE;

  case (STATE)
    IDLE:    if (START)                    // Auf Startsignal warten
               NEXTSTATE = S0;
    S0:      begin                         // Datenpfad initialisieren
               SEL = 1; LOADF = 1; LOADC = 1; RDONE = 1;
               NEXTSTATE = S1;
             end
    S1:      if (LEQ) begin                // Schleife count <= n
               LOADF = 1; INCC = 1;
               NEXTSTATE = S1;
             end else
               NEXTSTATE = S2;
    S2:      begin                         // Ende der Berechnung anzeigen
               SDONE = 1;
               NEXTSTATE = IDLE;
             end
  endcase
end

always @(posedge CLK, posedge RESET) begin // Neuen Zustand übernehmen
  if (RESET)     STATE <= IDLE;
  else           STATE <= NEXTSTATE;
end
```

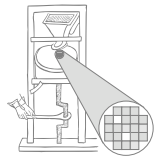
Testrahmen: Kopf und Verwaltung



A. Koch

```
module tb_fac;

reg          CLK;
reg          RESET;
reg          START;
reg    [2:0] N;
wire  [15:0] FACT;
wire         DONE;

  // Unit-under-Test instantiieren
  fac  FAC(CLK, RESET, START, N, FACT, DONE);

  // Takt erzeugen
  always begin
    CLK = 0;
    #5;
    CLK = 1;
    #5;
  end
```

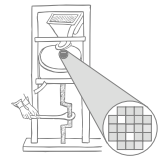
Testrahmen: Aktiver Teil



A. Koch

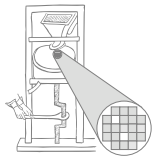
```
initial  begin
  $monitor("%t␣START=%d␣N=%d␣DONE=%d␣FACT=%d", $time, START, N, DONE, FACT);

  @(negedge CLK);          // Reset der Schaltung
  RESET = 1;
  @(negedge CLK);
  RESET = 0;

  @(negedge CLK);          // Berechne 3! = 6
  N = 3;
  START = 1;
  @(negedge CLK);
  START = 0;
  @(negedge CLK);
  while (!DONE)
    @(posedge CLK);

  @(negedge CLK);          // Berechne 5! = 120
  N = 5;
  START = 1;
  @(negedge CLK);
  START = 0;
  @(negedge CLK);
  while (!DONE)
    @(posedge CLK);

  $finish;
end
```

Simulationsergebnisse

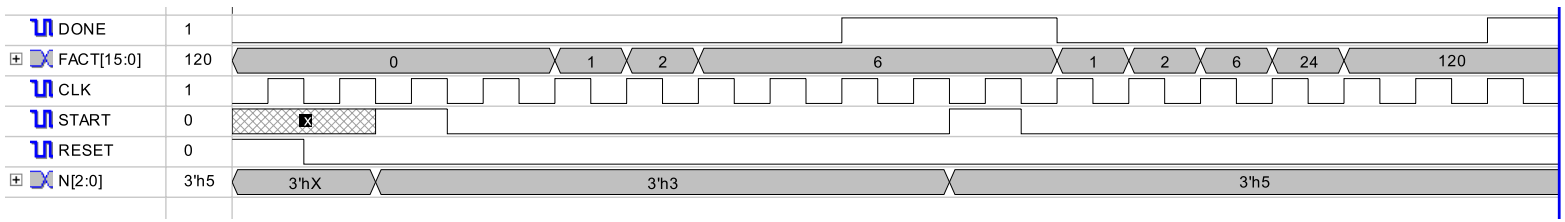



```
  0 START=x N=x DONE=x FACT=   x
 10 START=x N=x DONE=0 FACT=   0
 30 START=1 N=3 DONE=0 FACT=   0
 40 START=0 N=3 DONE=0 FACT=   0
 45 START=0 N=3 DONE=0 FACT=   1
 55 START=0 N=3 DONE=0 FACT=   2
 65 START=0 N=3 DONE=0 FACT=   6
 85 START=0 N=3 DONE=1 FACT=   6
100 START=1 N=5 DONE=1 FACT=   6
110 START=0 N=5 DONE=1 FACT=   6
115 START=0 N=5 DONE=0 FACT=   1
125 START=0 N=5 DONE=0 FACT=   2
135 START=0 N=5 DONE=0 FACT=   6
145 START=0 N=5 DONE=0 FACT=  24
155 START=0 N=5 DONE=0 FACT= 120
175 START=0 N=5 DONE=1 FACT= 120
```

Signalverlaufsdiagramm



A. Koch

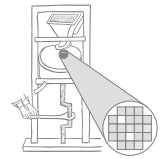


Beachte Verzögerung bei zweiter Berechnung 5!

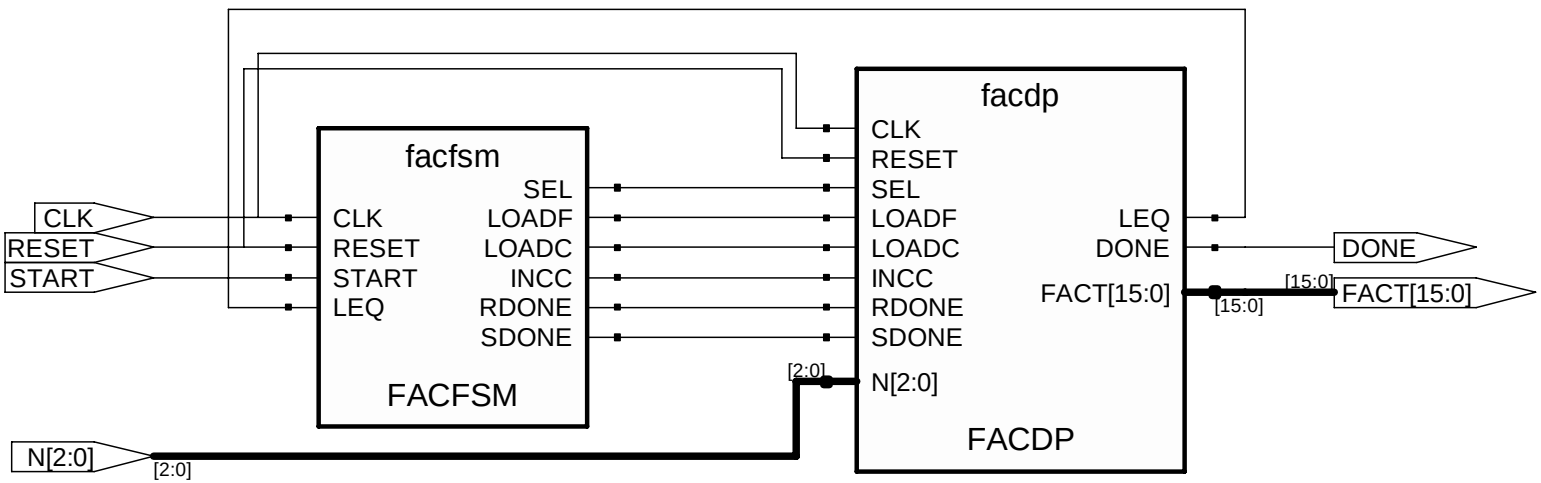
START=1 $\longrightarrow$ **RDONE=1** $\longrightarrow$ **DONE=0**

Alternative: Anderes Protokoll mit **DONE=1** nur für einen Takt

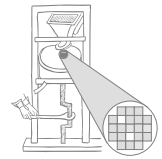
Syntheseergebnisse: Hauptmodul



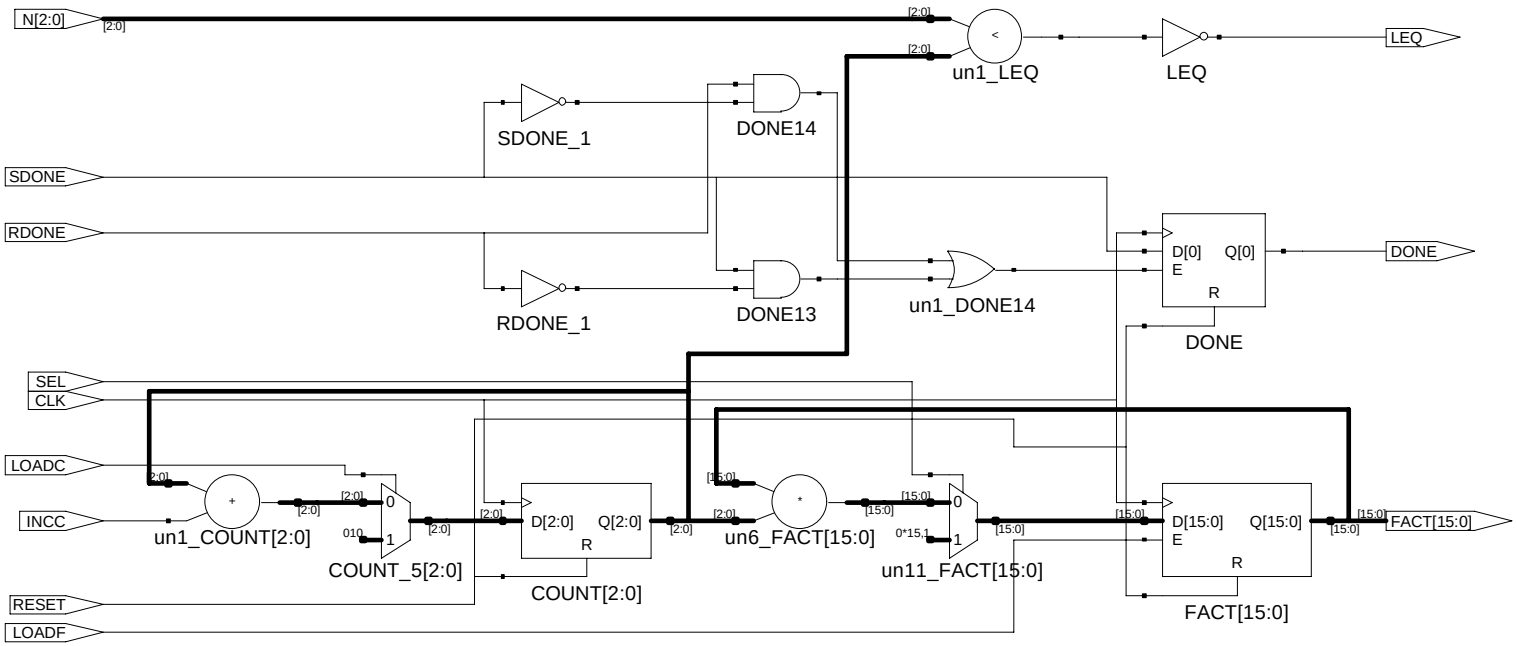
A. Koch



Syntheseergebnisse: Datenpfad

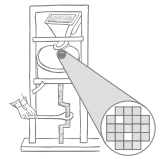


A. Koch

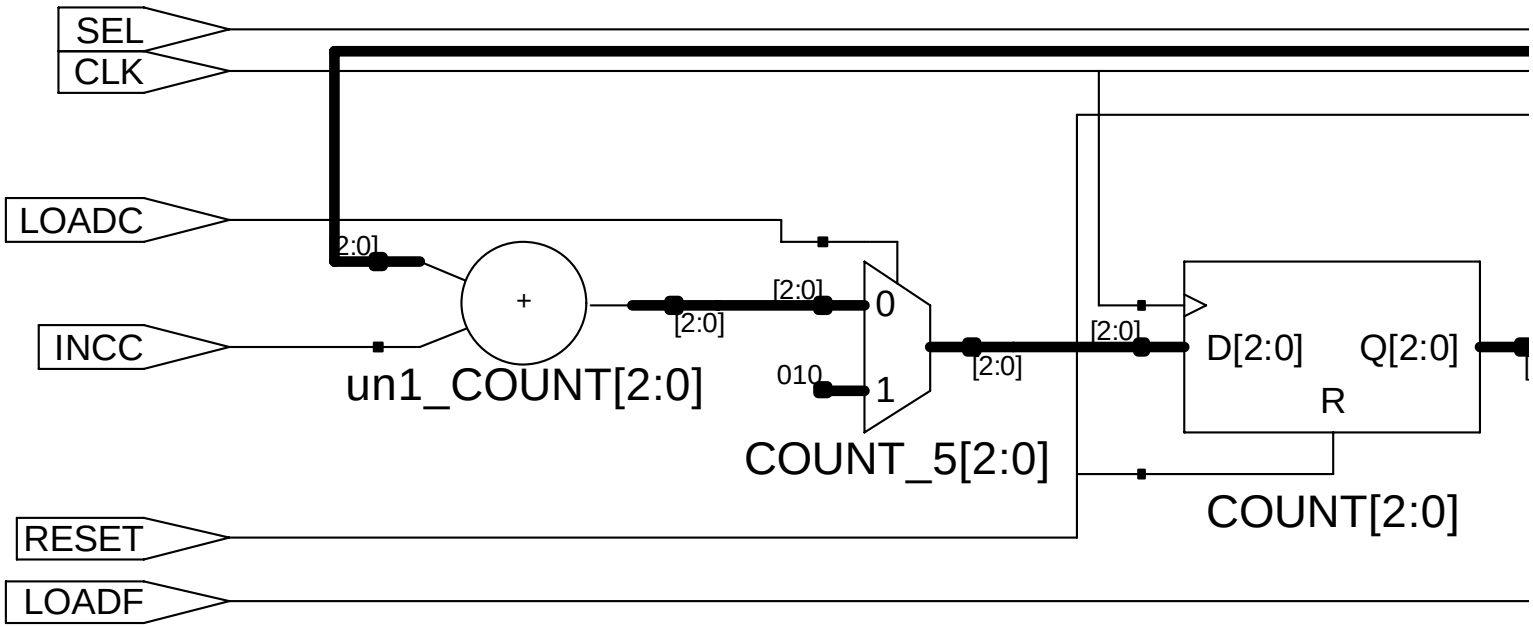


Syntheseergebnisse: Datenpfad

Schleifenzähler

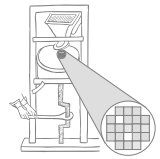


A. Koch

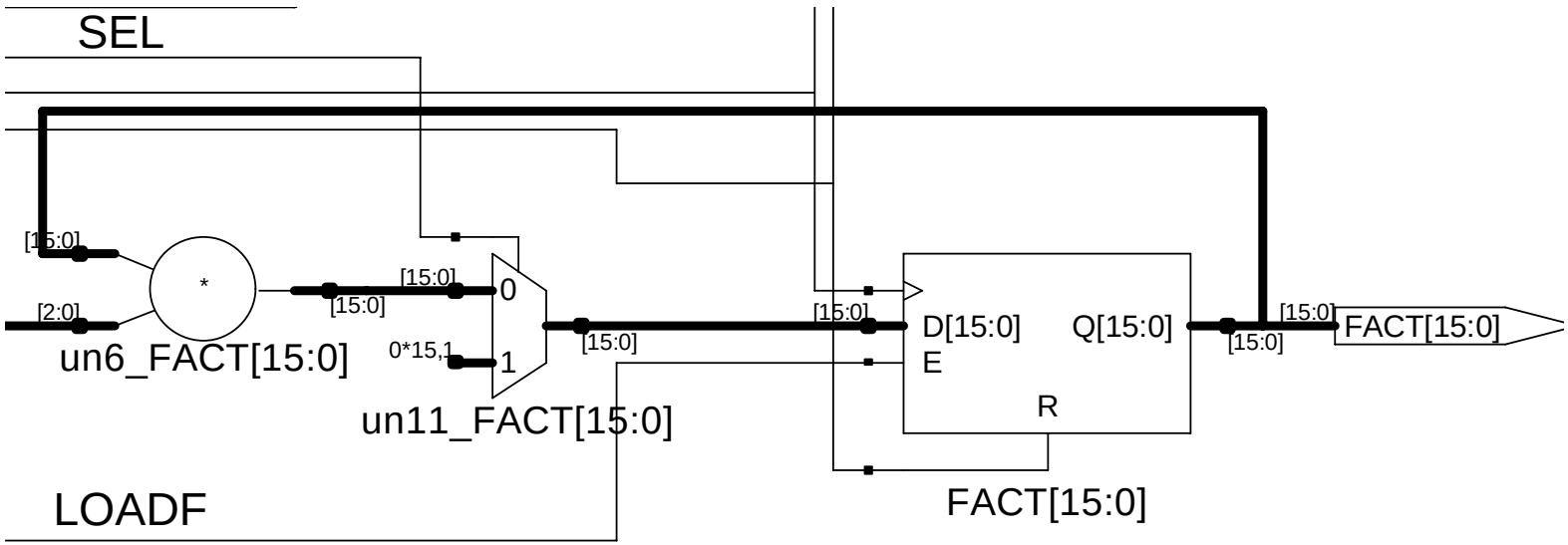


Syntheseergebnisse: Datenpfad

Produktberechnung

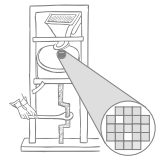


A. Koch

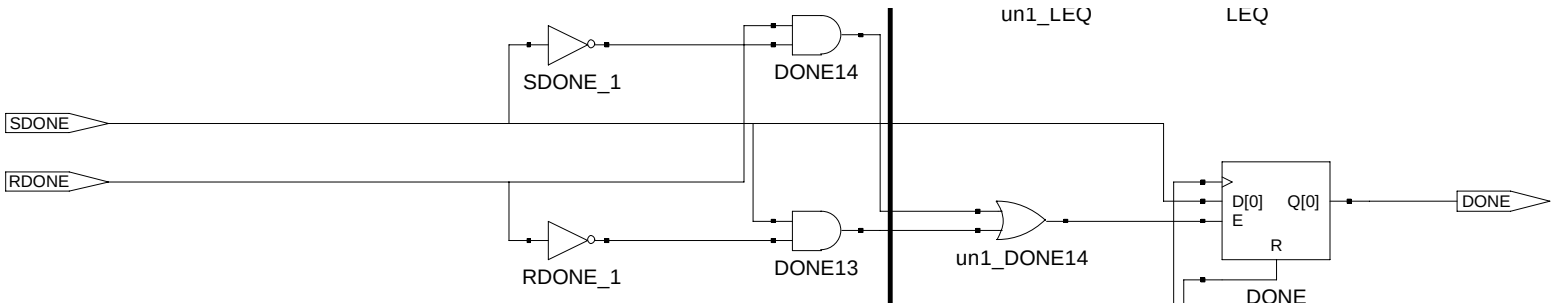


Syntheseergebnisse: Datenpfad

Berechnung des **DONE**-Signals

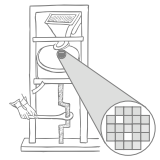


A. Koch

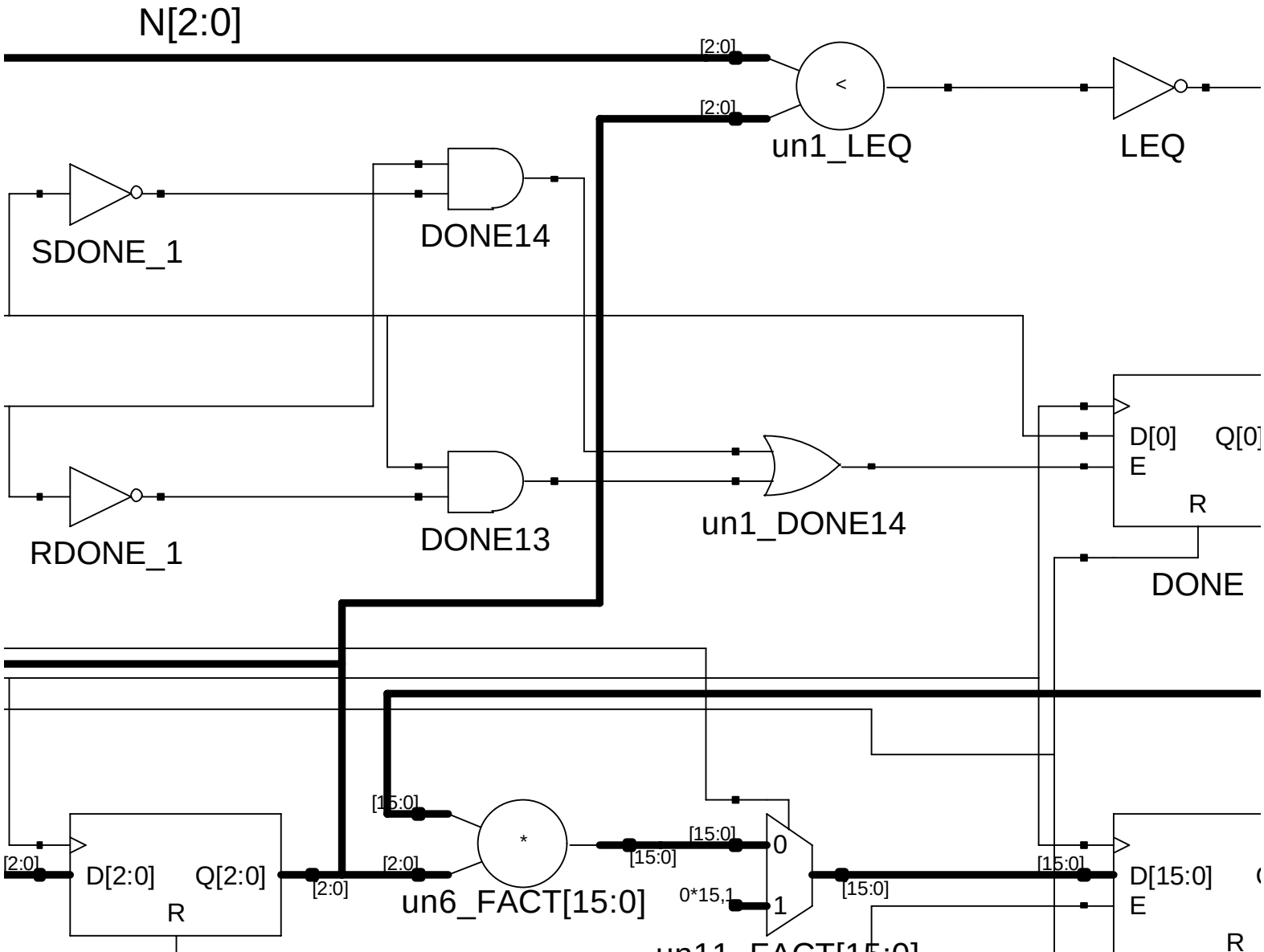


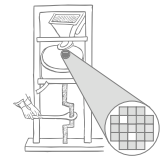
Syntheseergebnisse: Datenpfad

Berechnung des **LEQ**-Signals



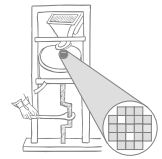
A. Koch





Graphische Beschreibungen

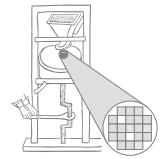
Alternative Darstellungsformen

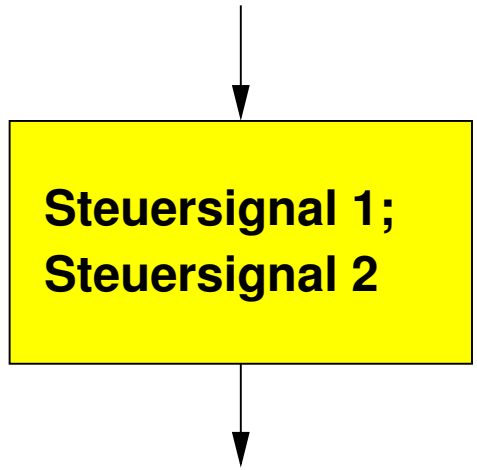


- Pseudo-Code
  - Textuell
- Nur eine Möglichkeit
- ASM(D)-Charts
- *Algorithmic State Machine* (*and Datapath*)
  - ASM-Chart stellt nur Steuerwerk dar
  - ASMD-Chart enthält zusätzlich noch Datenpfadoperationen

Zunächst nur ASM-Chart

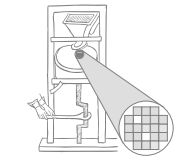
Zustand mit Ausgaben

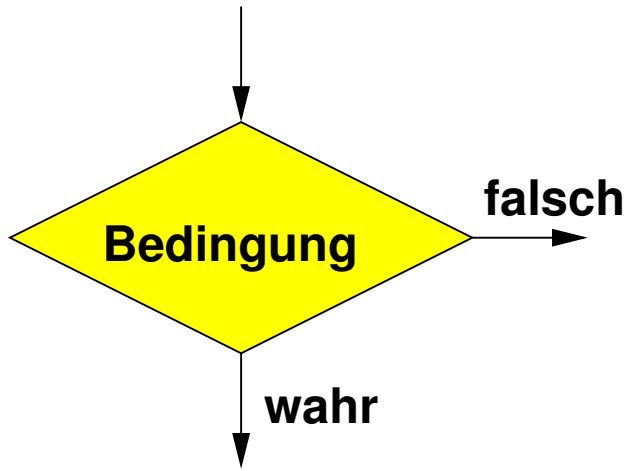




- Zählt aktive Steuersignale auf
- Nicht aufgeführte Signale sind inaktiv
- Die aufgeführten Signale sind parallel aktiv

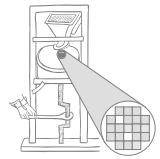
Bedingter Übergang

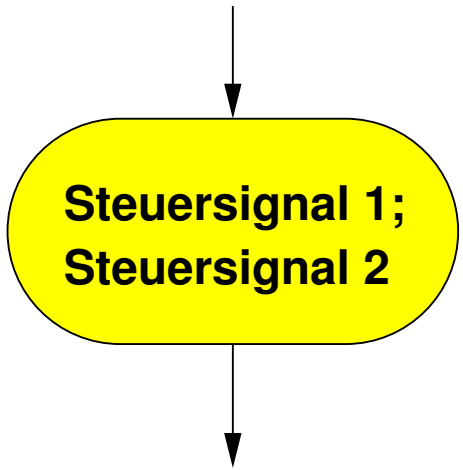




- Enthält logische Bedingung für Zustandsübergang
- Ausgangskanten müssen eindeutig beschriftet sein
  - True, 1, wahr
  - False, 0, falsch

Bedingte Ausgabe

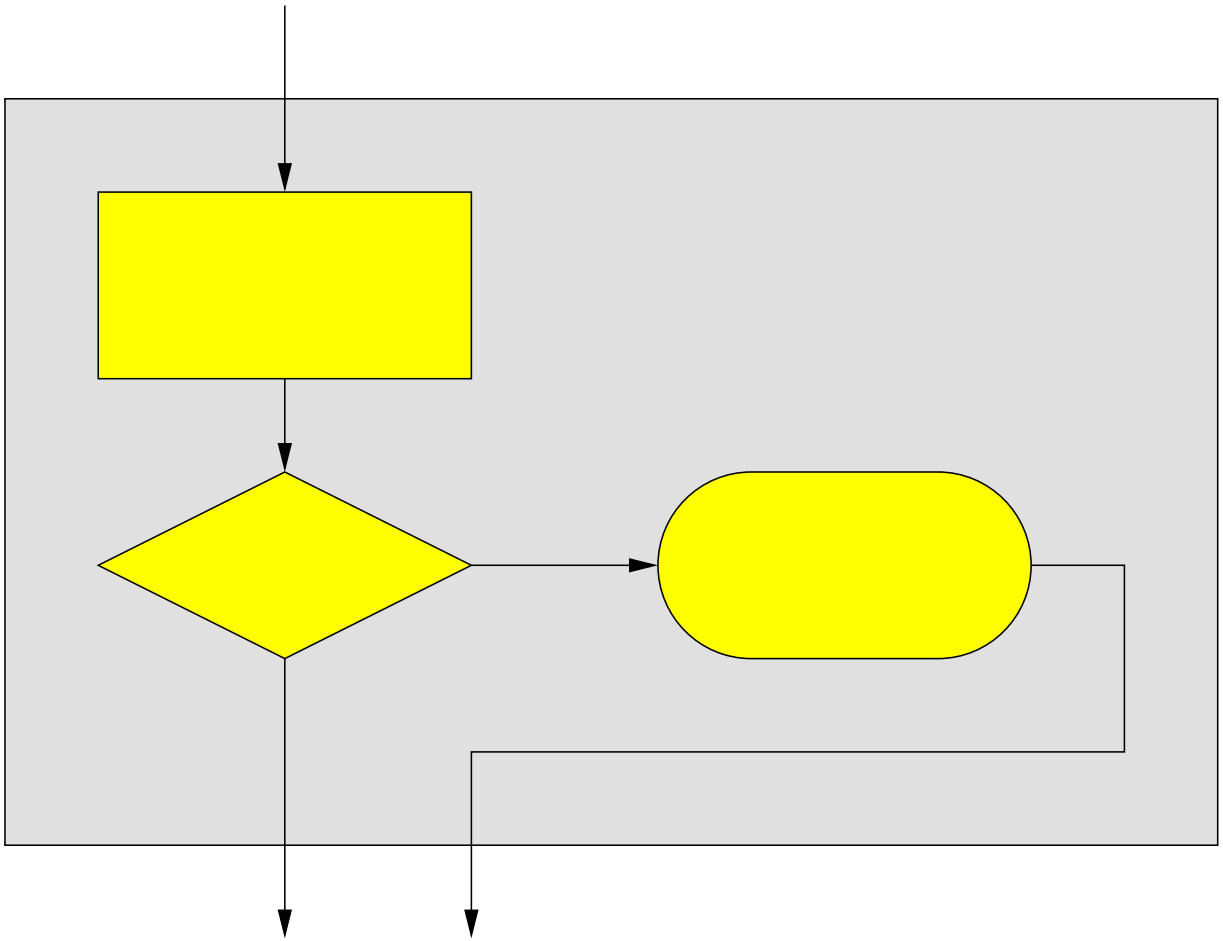




- Kann nur auf einen bedingten Übergang folgen
- Aktiviert Steuersignale abhängig von voriger Bedingung
- Beschreibung der Steuersignale wie bei Zustand

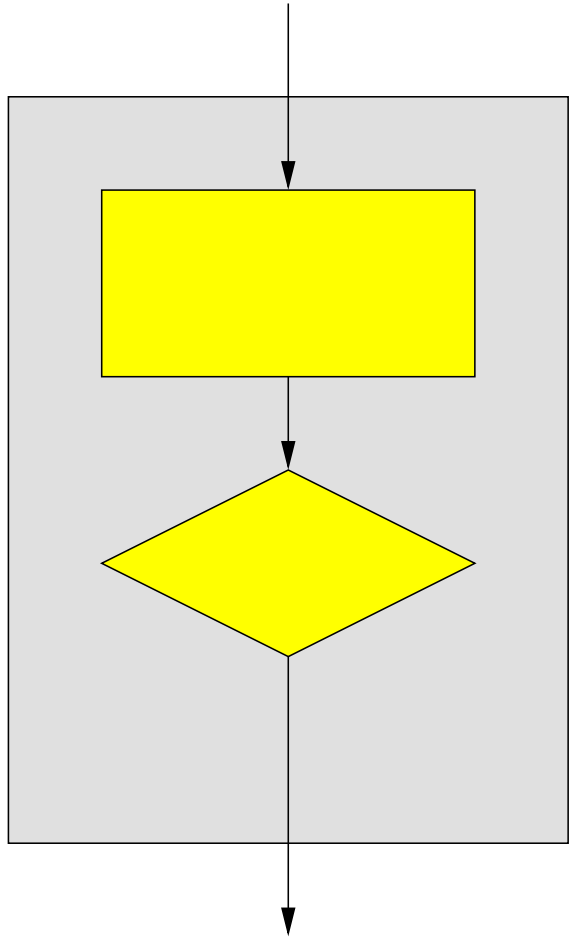
Zusammensetzen der Elemente

Mealy-Block



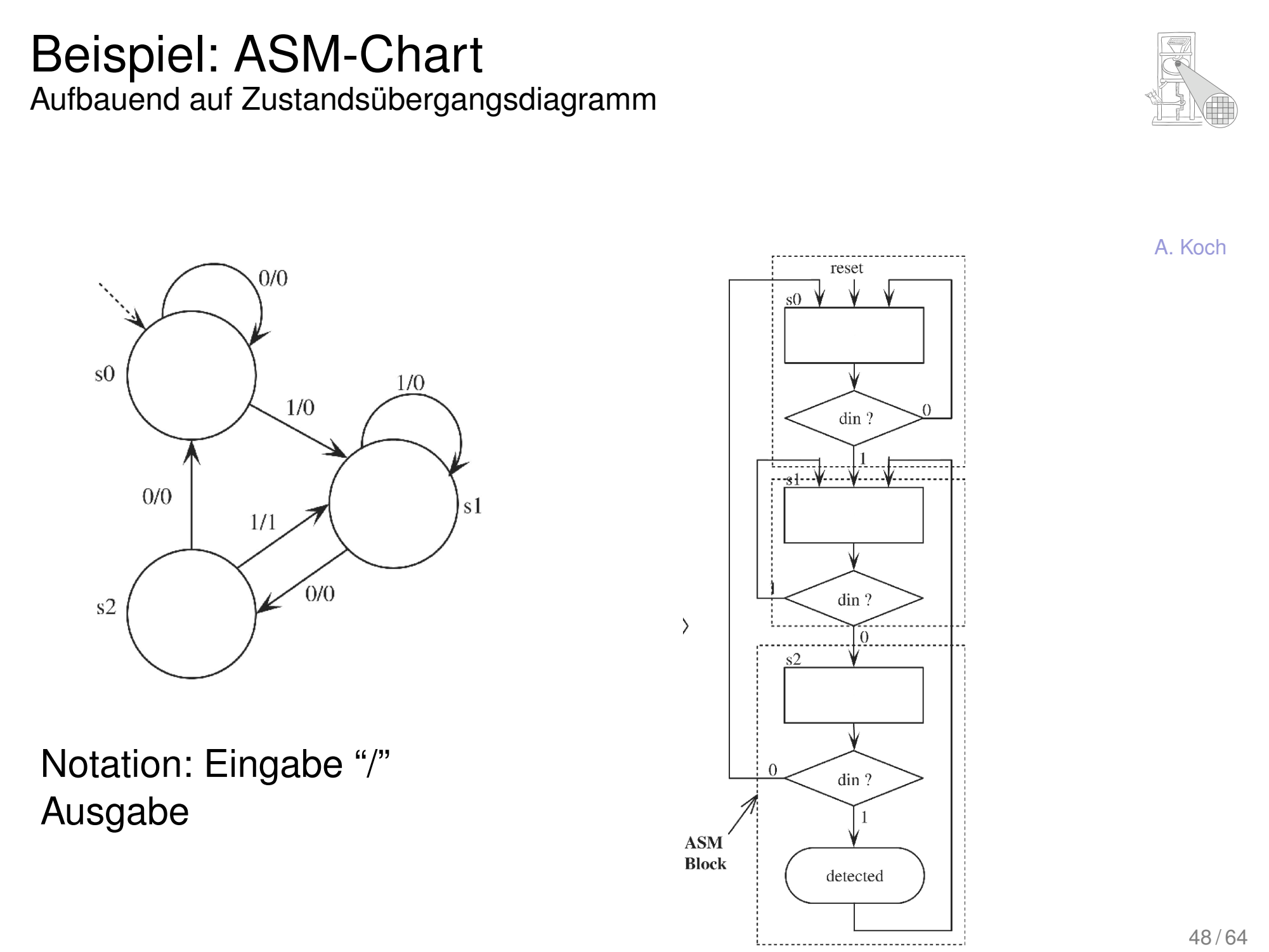
Zusammensetzen der Elemente

Moore-Block



Beispiel: ASM-Chart

Aufbauend auf Zustandsübergangsdiagramm

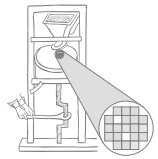


A. Koch

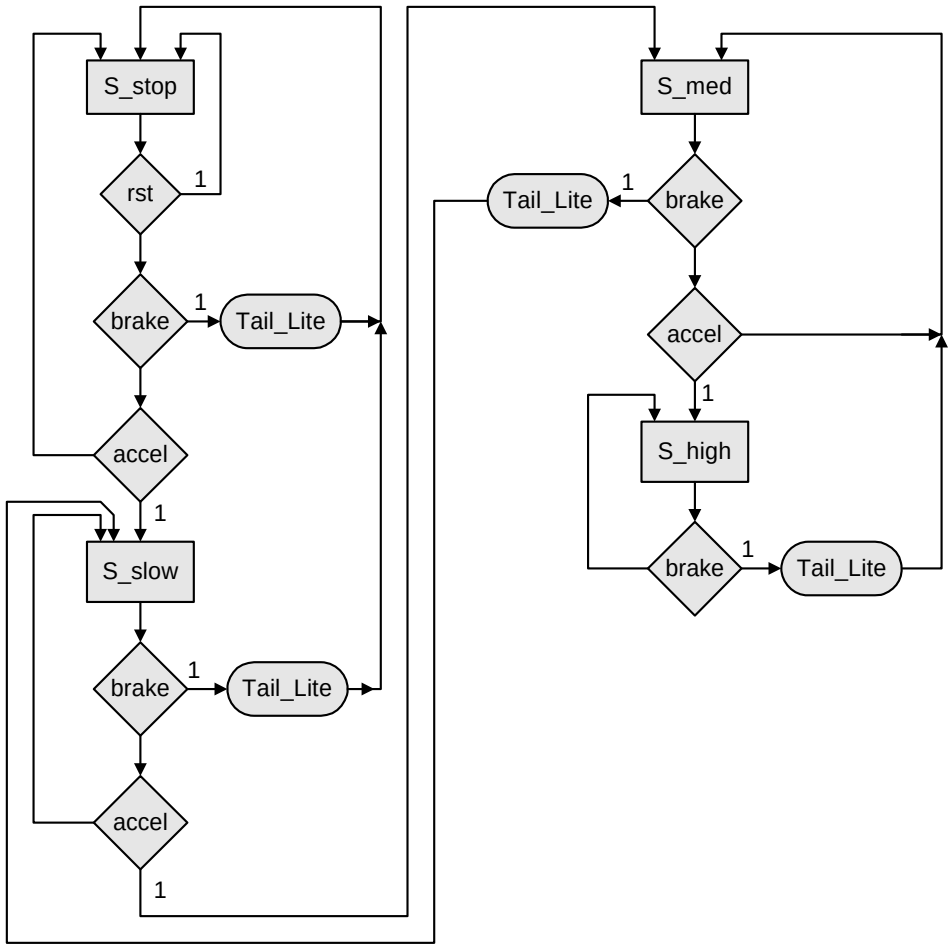
Notation: Eingabe “/” Ausgabe

Beispiel: ASM-Chart

Steuergerät für Auto



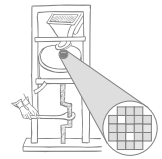
A. Koch



rst = Reset, brake=Bremspedal, accel=Gaspedal, Tail_Lite=Bremslicht

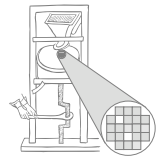
Quelle: Ciletti, Fig 5.23

Diskussion



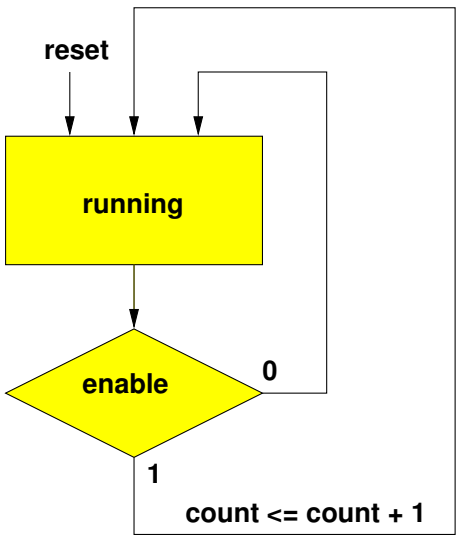
- Bedingte Übergänge sind priorisiert
  - In der Reihenfolge der Auswertung
    - Im Beispiel: Bremspedal vor Gaspedal abfragen
  - Konvention: Reset-Übergang nur einmal darstellen
    - Kennzeichnet Reset-Zustand
    - Implizit: Aus jedem Zustand bei Reset in diesen Zustand wechseln
  - Vorteile von ASM-Chart
    - Sehr ähnlich zu Flußdiagrammen
    - Wenn Algorithmus leicht als Flußdiagramm darstellbar
    - . . . dann auch leicht als ASM-Chart darstellbar

ASMD-Chart
Erweiterung um Datenoperationen

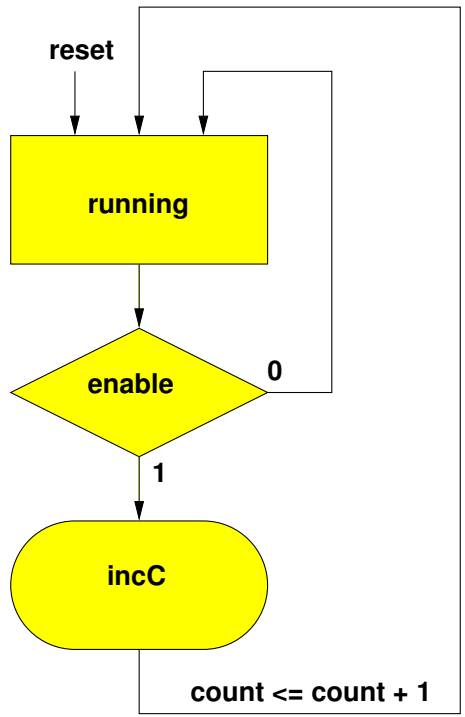


- Annotieren von Datenpfadoperationen
- Nun ASMD-Chart

A. Koch



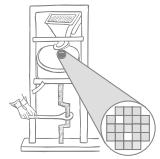
Noch ohne Steuersignale



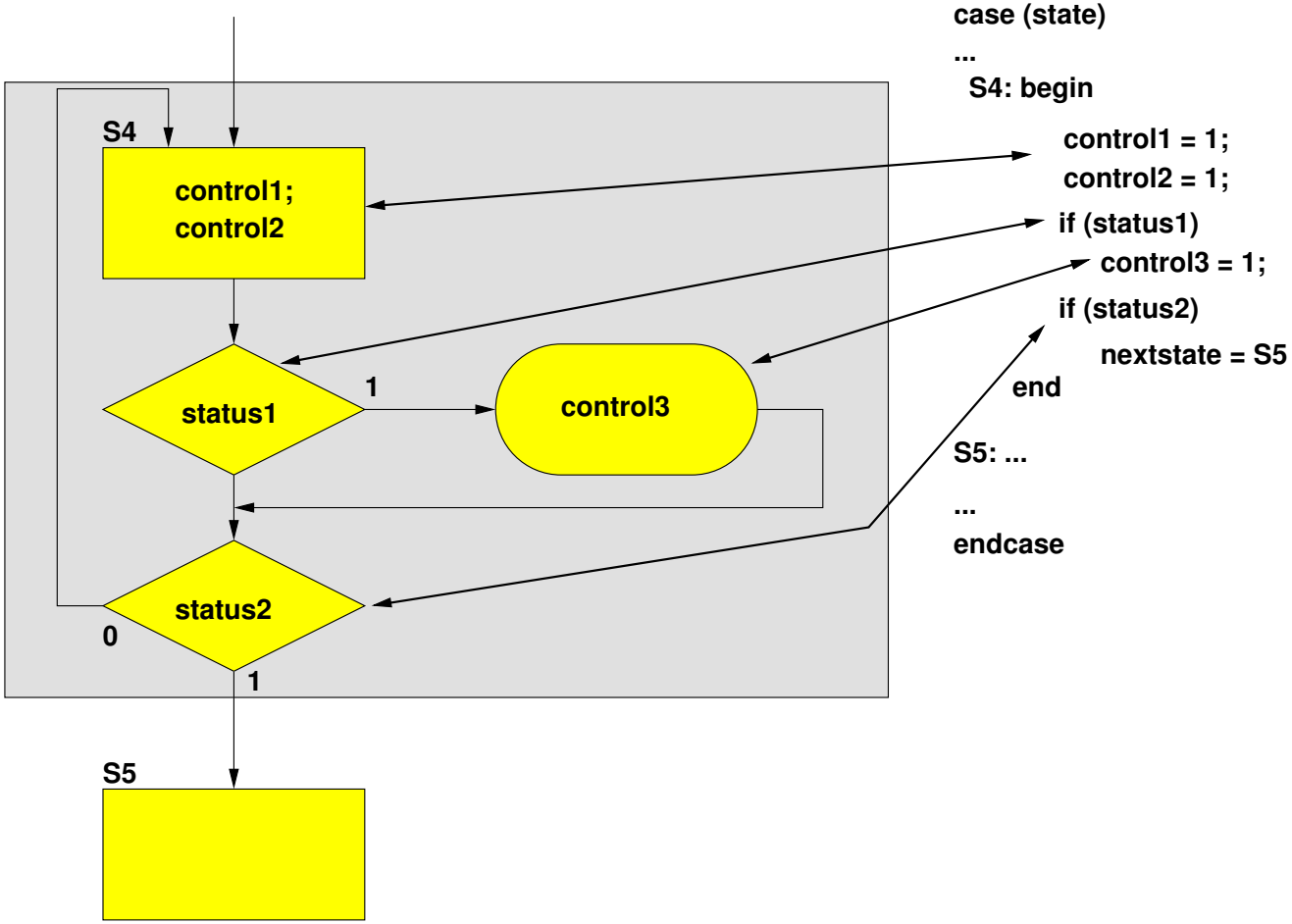
Mit Steuersignalen

Umsetzung von ASM-Charts in Verilog

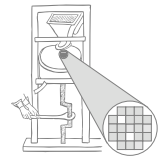
Hier nur Steuerwerk, Datenpfad wie vorher



A. Koch



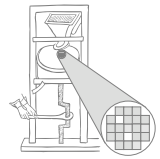

```
Alle Signale deaktivieren
control1 = 0;
control2 = 0;
control3 = 0;
Im gleichen Zustand bleiben
nextstate = state;
case (state)
...
 S4: begin
      control1 = 1;
      control2 = 1;
      if (status1)
         control3 = 1;
      if (status2)
         nextstate = S5
      end
S5: ...
...
endcase
```



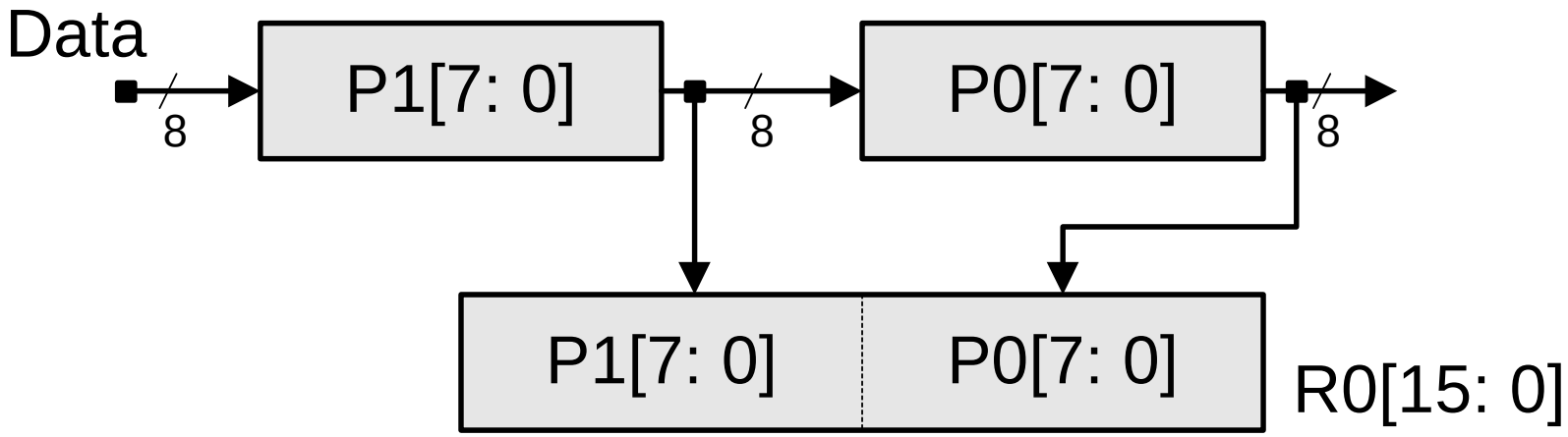
Beispiel: Seriell-Parallel-Wandlung

Beispiel: Seriell-Parallel Wandlung

Wandle 8b Datenstrom in 16b Datenstrom um



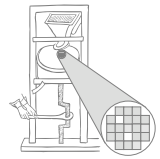
A. Koch



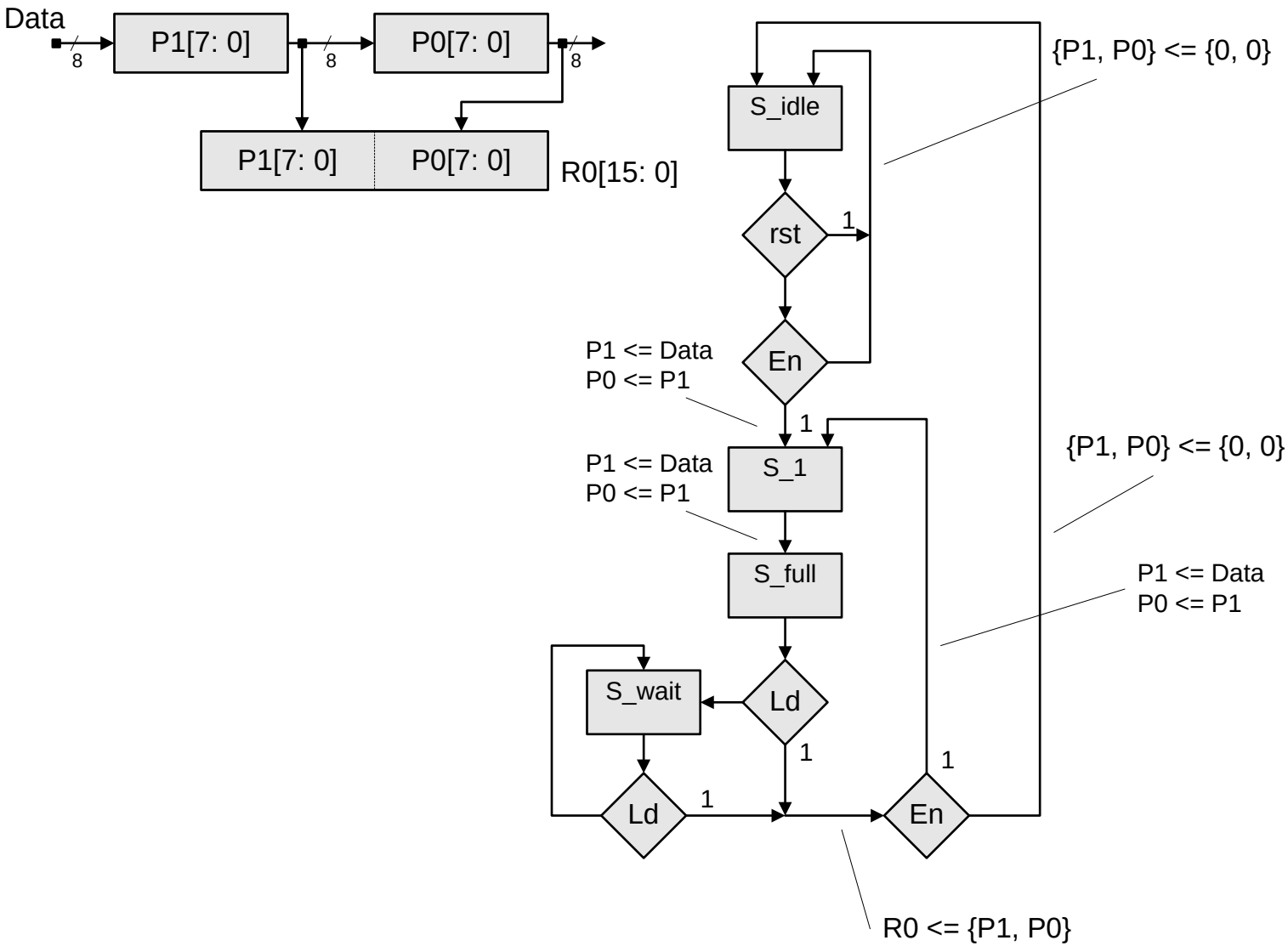
- Zwei 8b Daten kommen über **`Data`** an
- . . . wenn Steuereingang **`En`** aktiviert ist
- Je zwei Bytes zu einem 16b Wort zusammensetzen
- Ergebnis wurde von außen übernommen, wenn Steuereingang **`Ld`** aktiviert ist
- Dann mit neuen Bytes wiederholen

Quelle: Ciletti, Fig 5.24

ASMD mit Datenpfadoperationen



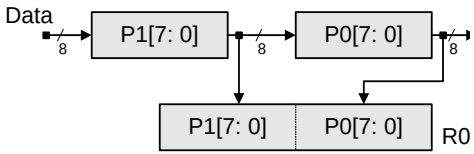
A. Koch

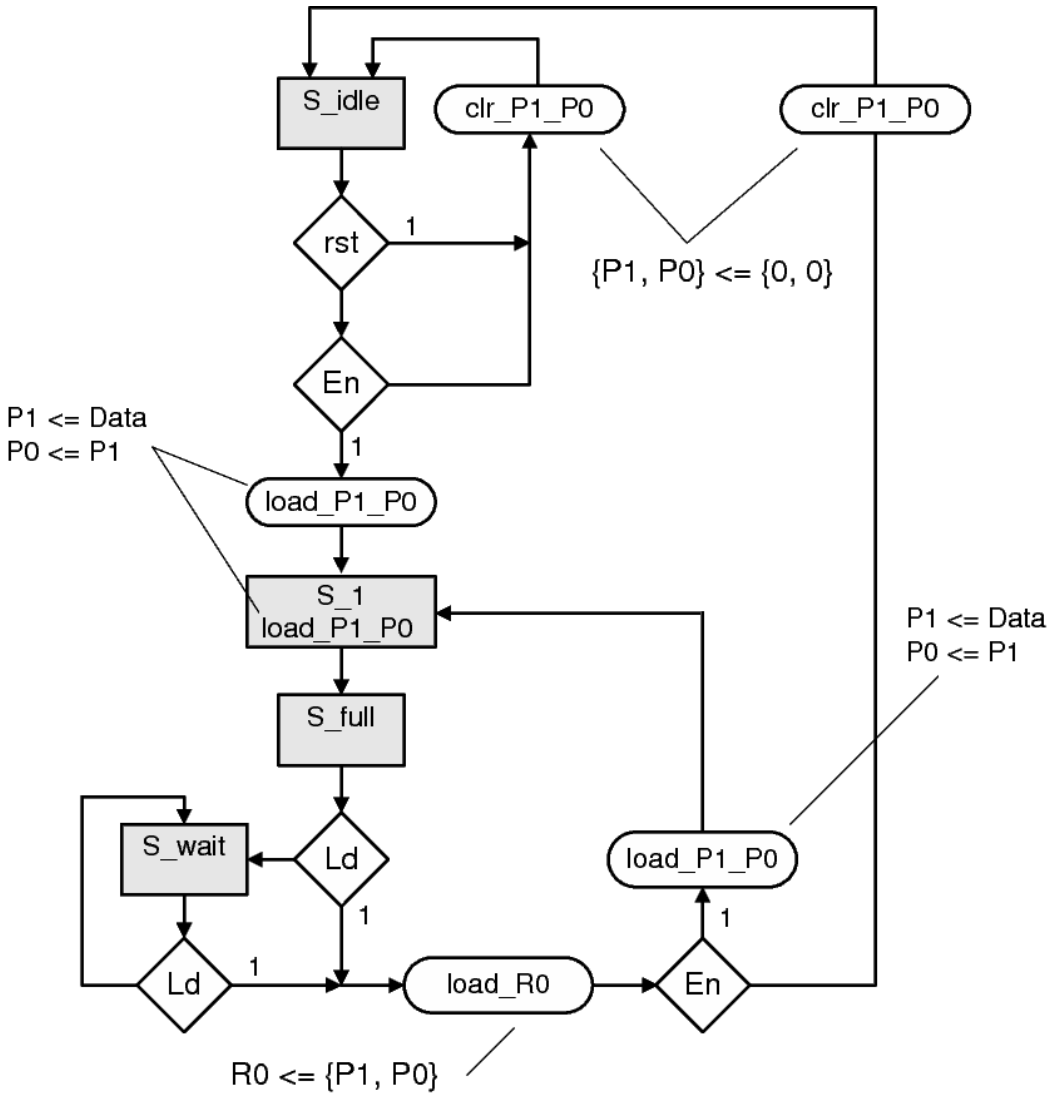


Quelle: Ciletti, Fig 5.24

ASMD mit Steuersignalen

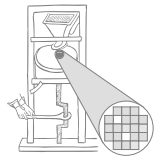
Für späteren Anschluss an Datenpfad





A. Koch

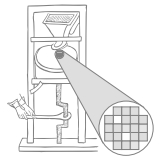
Top-Level-Modul



A. Koch

```
module byteword (
  input              CLK,
  input              RST,
  input        [7:0] DATA,    // 8b serieller Dateneingang
  input              EN,
  input              LD,
  output      [15:0] R0        // 16b paralleler Datenausgang
);

  // Steuerwerk
  ctlbyteword CTL (CLK,RST,EN,LD,LOAD_P1_P0,CLR_P1_P0,LOAD_R0);

  // Datenpfad
  dpbyteword DP  (CLK,RST,DATA,LOAD_P1_P0,CLR_P1_P0,LOAD_R0,R0);

endmodule
```

Steuerwerk 1/2
Kopf und Verwaltung

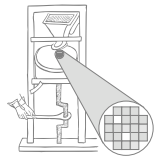


A. Koch

```
module ctlbyteword (
  input  CLK, RST, EN, LD,
  output reg LOAD_P1_P0, CLR_P1_P0, LOAD_R0
);

  parameter S_idle = 0;
  parameter S_1    = 1;
  parameter S_full = 2;
  parameter S_wait = 3;

  reg [1:0]  STATE, NEXTSTATE;

...

  always @(posedge CLK, posedge RST)
    if (RST)
      STATE <= S_idle;
    else
      STATE <= NEXTSTATE;

endmodule
```

Steuerwerk 2/2

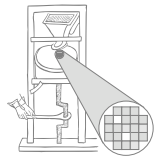
Zustandsübergangs- und Ausgabefunktion



```
always @(STATE, EN, LD, RST) begin
  LOAD_P1_P0 = 0; CLR_P1_P0 = 0; LOAD_R0 = 0;    // alle Ausgänge inaktiv
  NEXTSTATE = STATE;                             // verharre in diesem Zustand
  case (STATE)
    S_idle: if (RST | ~EN)
              CLR_P1_P0 = 1;
            else if (EN) begin
              LOAD_P1_P0 = 1;   NEXTSTATE = S_1;
            end
    S_1:    begin
              LOAD_P1_P0 = 1;   NEXTSTATE = S_full;
            end
    S_full: if (LD) begin
              LOAD_R0 = 1;
              if (EN) begin
                LOAD_P1_P0 = 1; NEXTSTATE = S_1;
              end else begin
                CLR_P1_P0 = 1;  NEXTSTATE = S_idle;
              end
            end else
              NEXTSTATE = S_wait;
    S_wait: if (LD) begin
              LOAD_R0 = 1;
              if (EN) begin
                LOAD_P1_P0 = 1; NEXTSTATE = S_1;
              end else begin
                CLR_P1_P0 = 1; NEXTSTATE = S_idle;
              end
            end
  endcase
end
```

A. Koch

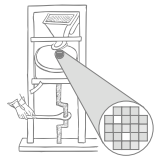
Datenpfad



```
module dpbyteword (
  input              CLK, RST,
  input        [7:0] DATA,
  input              LOAD_P1_P0, CLR_P1_P0,LOAD_R0,
  output reg [15:0] R0
);

  reg [7:0]  P1, P0;

  always @(posedge CLK, posedge RST) begin
    if  (RST) begin
      R0 <= 0;
      {P1, P0} <= 0;
    end else begin
      if  (CLR_P1_P0)
        {P1,P0} <= 0;
      else if  (LOAD_P1_P0)
        {P1,P0} <= {DATA,P1};
      if  (LOAD_R0)
        R0 <= {P1,P0};
    end
  end
endmodule
```

A. Koch

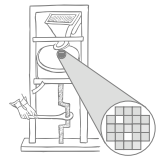
Testrahmen



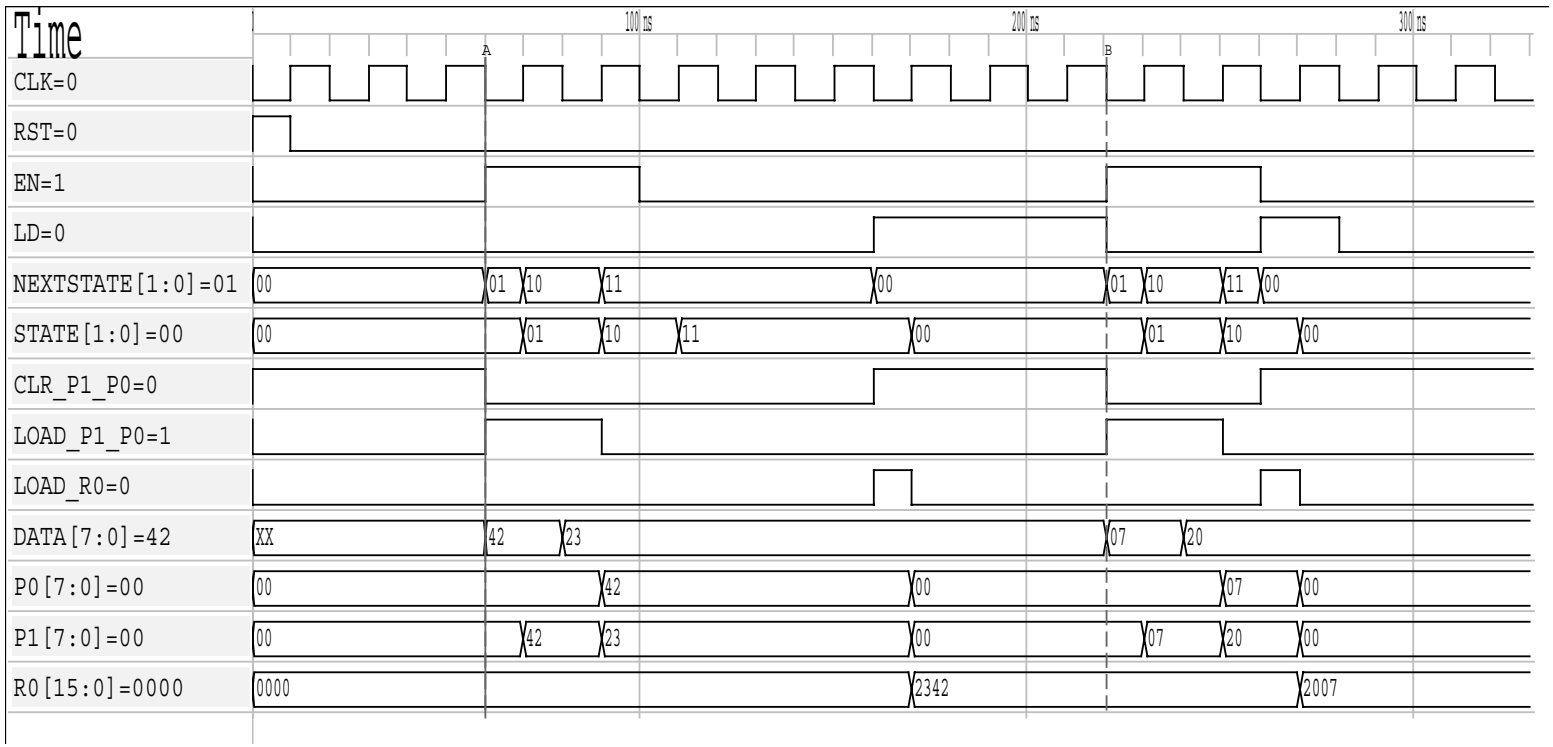
```
module testbench;

  reg CLK, RST, EN, LD;
  reg [7:0] DATA;
  wire [15:0] R0;

  byteword UUT (CLK, RST, DATA, EN, LD, R0);

  always begin        // Takterzeugung
    CLK = 0;
    #10;
    CLK = 1;
    #10;
  end

   initial  begin

    EN = 0;           // System initialisieren
    LD = 0;
    RST = 1;
    @(posedge CLK);
    RST = 0;
    #50;

    @(negedge CLK); // Ersten Datenblock anlegen
    EN = 1;
    DATA = 8'h42;
    #0;
    @(negedge CLK);
    DATA = 8'h23;
    #0;
    @(negedge CLK);
    EN = 0;
    #50;

    @(negedge CLK);     // Gepackte Daten abrufen
    LD = 1;

    #50;

    @(negedge CLK);     // Zweiten Datenblock anlegen
    LD = 0;
    EN = 1;
    DATA = 8'h07;
    #0;
    @(negedge CLK);
    DATA = 8'h20;

    #0;                 // Gepackte Daten abrufen
    @(negedge CLK);
    EN = 0;
    LD = 1;
    #0

    @(negedge CLK);     // System herunterfahren
    LD = 0;

    #50;
    $finish;
  end

endmodule
```

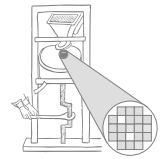
A. Koch

Signalverlaufsdiagramme



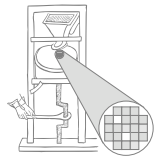
A. Koch





Ausblick

Auswahl weiterer Lehrveranstaltungen im WS



- Algorithmen im Chip-Entwurf (Koch)
- Prozessorarchitekturen für rechenstarke eingebettete Systeme (Koch)
- Systementwurf mit Mikroprozessoren (Hoffmann)
- Eingebettete Systeme 1 (Huss)