



A. Koch

Einführung in Computer Microsystems

4. Modellierungskonzepte in Verilog

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

Sommersemester 2007



A. Koch

Organisatorisches



- **Klausurtermine**
 - 31.05.2007, 18:00-18:45 Uhr
 - 26.07.2007, 17:00-18:30 Uhr
- **Tutorensprechstunde** im C-Pool
 - Thomas Feller, Di. 13:30-15:00 Uhr in C005



A. Koch

Parallelität



- In konventionellen Programmiersprachen wie z.B. Pascal, C
 - Anweisungen werden **der Reihe nach** bearbeitet
 - Programmzähler zeigt auf **aktuelle** Anweisung
 - Es gibt nur **einen** Kontrollfluß
- In HDLs und realen Schaltungen
 - Alle Komponenten arbeiten **parallel**
 - Z.B. kann **eine** Taktflanke eine Vielzahl von **gleichzeitigen** Aktionen auslösen
 - Modelliert durch parallele **always**-Blöcke

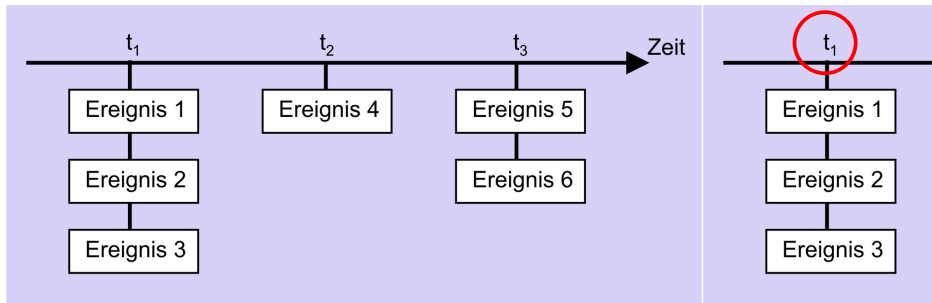
Modellierung von Parallelität in Verilog



A. Koch

- Instanzen von Modulen
- **always**- und **initial**-Blöcke
- ständige Zuweisungen (*continuous assignments*)
- nichtblockende Zuweisungen
- Mischformen

Ereignisgesteuerte Simulation der Parallelität



- globale Simulations-Zeitpunkte t_1, t_2, \dots
- ein oder mehrere Ereignisse sollen jeweils **parallel** ausgeführt werden
- Ereignis-**Scheduler** wählt **eines zufällig** aus
- wenn bei t_1 nichts mehr zu tun, gehe zu t_2 weiter



- Kein Verlass auf **bestimmte** Reihenfolge
 - Kann zwischen Simulatoren variieren
 - Kann auch durch Simulationsoptionen beeinflusst werden
- parallel = nicht-deterministisch
 - **ein** richtiges Ergebnis garantiert nicht allgemeine **Korrektheit**
 - exponentiell viele Ergebnisse möglich
- Unwägbarkeiten können durch Entwurstile reduziert werden
 - Synchroner Register-Transfer-Logik
 - Designer legt Zeitablauf explizit im Modell fest
 - Unterschiedliche Ereignisse finden in unterschiedlichen Takten statt



A. Koch

Register-Transfer-Logik

Register-Transfer-Logik (RTL)

Einführung



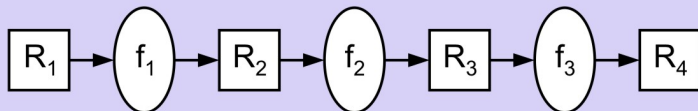
A. Koch

- Grundlegendes und universelles Konzept
- Beliebige **Automatennetze** übersichtlich realisierbar
- insbesondere effiziente Pipelines
- Ähnlichkeit zum Programmieren $y = f_3(f_2(f_1(x)))$
 - Aber **räumlich** parallel verteilt
- Synchron durch **gemeinsamen** Takt
- Gut testbar
- Sehr kompakt mit **nichtblockender** Zuweisung realisierbar

Pipeline in Register-Transfer-Logik



A. Koch



- Kombinatorische Logik **zwischen** den Registern
 - f_1 : verdoppeln
 - f_2 : plus 5
 - f_3 : quadrieren
- Pipeline berechnet $R_4 = (2R_1 + 5)^2$
 - bearbeitet 3 Datensätze **gleichzeitig**
 - gibt **pro Takt** ein Ergebnis aus
 - Damit 3x schneller als sequentielle Berechnung der drei Funktionen



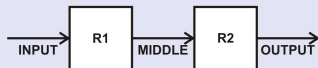
A. Koch

Konstruktion von Pipelines in RTL

1. Schritt: Flip-Flop-Kette



A. Koch

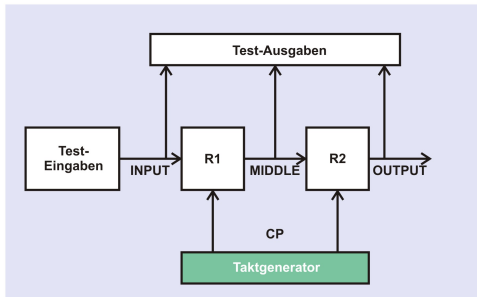


- Mini-Pipeline aus zwei Flip-Flops
- Flip-Flops sind **flankengesteuert**
 - Unterschied zu Latches (pegelgesteuert)
 - Aufbau z.B. aus Master-Slave-Latches (TGDI1)
- Annahme hier: **vorderflankengesteuert**
 - **always** @(posedge CLOCK)

2. Schritt: Takterzeugung



A. Koch

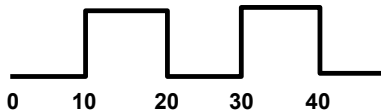


always begin

```
CP = 0; #10;
```

```
CP = 1; #10;
```

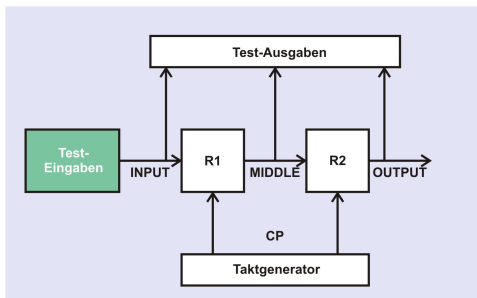
end



3. Schritt: Testeingaben (Stimuli)



A. Koch



```
initial begin
```

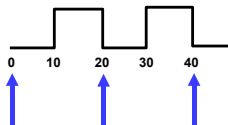
```
  INPUT = 0;    #20;
```

```
  INPUT = 255; #20;
```

```
  INPUT = 8'haa; #20;
```

```
  $finish;
```

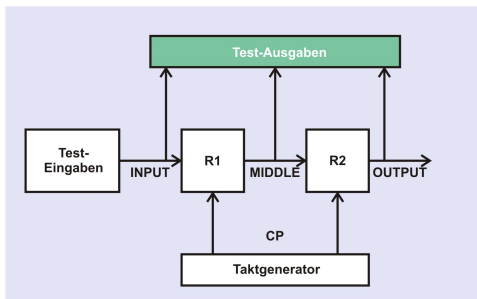
```
end
```



4. Schritt: Testausgaben



A. Koch

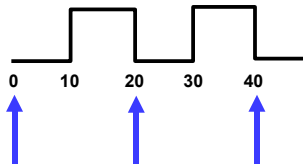
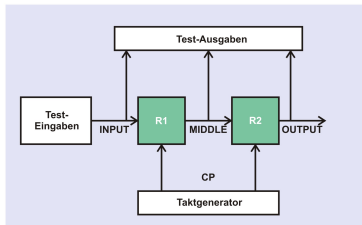


always @(INPUT, MIDDLE, OUTPUT)

\$display

```
("Zeit: %2.0f, INPUT=%h, MIDDLE=%h, OUTPUT=%h",  
$time, INPUT, MIDDLE, OUTPUT);
```


5. Schritt: Modellierung der Flip-Flops



A. Koch

```
always @(posedge CP)
MIDDLE = INPUT; // Fehler!
```

```
always @(posedge CP)
begin
MIDDLE = #1 INPUT; // Fehler!
```

```
always @(posedge CP)
begin
MIDDLE <= INPUT;
```

```
always @
OUTPU
Zeit: 0, INPUT = 00, MIDDLE = xx, OUTPUT = xx
```

```
Zeit: 11, INPUT = 00, MIDDLE = 00, OUTPUT = xx
```

```
Zeit: 20, INPUT = ff, MIDDLE = 00, OUTPUT = xx
```

```
Zeit: 31, INPUT = ff, MIDDLE = ff, OUTPUT = 00
```

```
always @
MIDDLE
Zeit: 40, INPUT = aa, MIDDLE = ff, OUTPUT = 00
```

```
always @
OUTPUT = #1 MIDDLE;
Zeit: 51, INPUT = aa, MIDDLE = aa, OUTPUT = ff
```

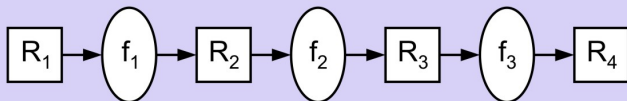
end

end

Beispiel-Pipeline: Rahmenmodul



A. Koch



```
module pipeline #(
    parameter    Low  = 10,           // Takt low
                High = 5,           // Takt high
);
    reg         CLOCK;               // Takt

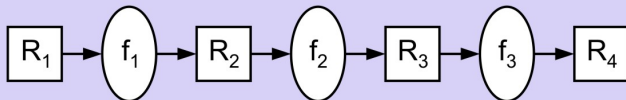
    reg [7:0] R1,                    // Register 1
          R2,                        // Register 2
          R3,                        // Register 3
          R4;                       // Register 4

    integer    l;                   // Hilfsvariable
    ...
endmodule // pipeline
```

Beispiel-Pipeline: Takterzeugung



A. Koch



```
// Ein-Phasen-Takt  
always begin  
  #Low CLOCK <= 1;  
  #High CLOCK <= 0;  
end
```

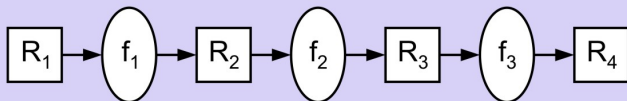
```
// Takt low  
// Takt high
```

Beispiel-Pipeline: Kombinatorische Logik

Führt eigentliche Rechnung aus



A. Koch



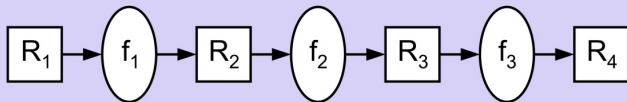
```
// Logik zwischen R1 und R2  
function [7:0] f1 (input [7:0] IN);  
    f1 = 2 * IN;  
endfunction
```

```
// Logik zwischen R2 und R3  
function [7:0] f2 (input [7:0] IN);  
    f2 = IN + 5;  
endfunction
```

```
// Logik zwischen R3 und R4  
function [7:0] f3 (input [7:0] IN);  
    f3 = IN * IN;  
endfunction
```

Testrahmen

Hier in einem Modul (kürzer), besser: saubere Trennung in eigenem Modul



A. Koch

```
// Test-Ausgaben
always @(*)
  $display ("%4.0f...%b...%d...%d...%d...%d",
    $time, CLOCK, R1, R2, R3, R4);

// Ueberschrift, Test-Eingaben
initial begin
  $display ("Zeit...CLOCK...R1...R2...R3...R4\n");

  @(negedge CLOCK) R1 <= 1; // R1 eingeben
  @(negedge CLOCK) R1 <= 2; // R1 eingeben
  @(negedge CLOCK) R1 <= 3; // R1 eingeben
  @(negedge CLOCK) R1 <= 4; // R1 eingeben

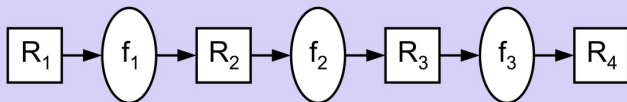
  for (l=1;l<=5;l=l+1) // Pipeline leeren
    @(posedge CLOCK);
  $finish;
end
```

Beispiel-Pipeline: Ablaufsteuerung

Hier in einem Modul (kürzer), besser: saubere Trennung in eigenem Modul



A. Koch



```
// Pipeline steuern und Funktionen berechnen
```

```
always @(posedge CLOCK) begin
```

```
  R2 <= f1(R1);
```

```
  R3 <= f2(R2);
```

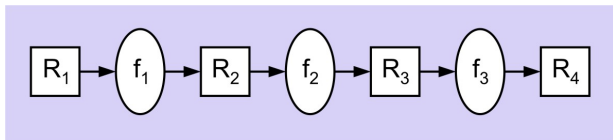
```
  R4 <= f3(R3);
```

```
end
```

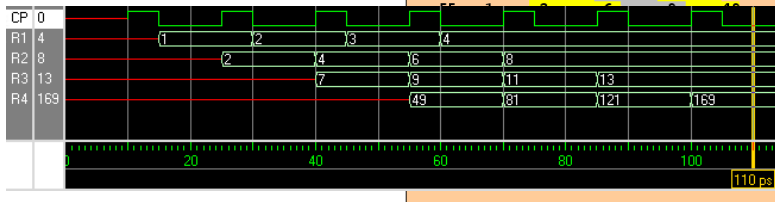
Beispiel-Pipeline: Ergebnisse



A. Koch



Zeit	CLOCK	R1	R2	R3	R4
10	1	x	x	x	x
15	0	x	x	x	x
15	0	1	x	x	x
25	1	1	x	x	x
25	1	1	2	x	x
30	0	1	2	x	x
30	0	2	2	x	x
40	1	2	2	x	x
40	1	2	4	7	x
45	0	2	4	7	x
45	0	3	4	7	x
55	1	3	4	7	x
55	1	3	4	7	x





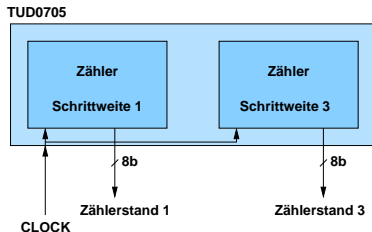
A. Koch

Busse

Beispiel-Chip TUD0705: Großer Verkaufserfolg!



A. Koch



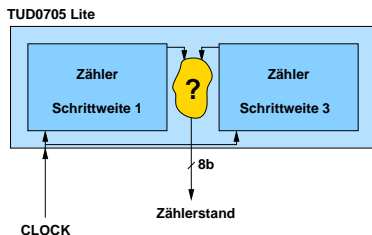
- Zwei synchrone 8b-Zähler
- Schrittweiten 1 und 3
- Beide parallel auslesbar

Problem: Zu teuer!



- Wo Geld sparen?
- Anforderung: Es wird nur jeweils **einer** der beiden Werte gebraucht
- Ausgang-Pins sparen (kosten extra)
- Beide Zähler über die **gleichen** Pins nach aussen leiten

A. Koch

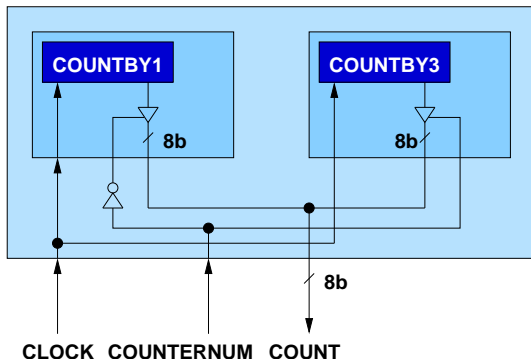


➡ Wie beide Werte auseinanderhalten?

Idee: Nicht gebrauchten Zählerausgang hochohmig schalten



TUD0705 Lite



A. Koch

- Neuer Steuereingang COUNTERNUM
 - Bei COUNTERNUM=0 Wert des **ersten** Zählers ausgeben
 - Bei COUNTERNUM=1 Wert des **zweiten** Zählers ausgeben

Verilog: Modellierung der Zähler



- Beliebige Schrittweite
- Hochohmig-schaltbarer Ausgang

A. Koch

```
module COUNTER
#(
  parameter stepsize = 1      // Schrittweite
)
(
  input  wire    CLOCK,
  input  wire    SELECT, // Wert ausgeben?
  output wire [7:0] OUT
);

reg [7:0] COUNT = 0;      // Nur für Simulation!

always @(posedge CLOCK)
  COUNT <= COUNT + stepsize;

assign OUT = (SELECT) ? COUNT : 8'bz; // Tri-State Treiber

endmodule
```

Verilog: Modellierung des Gesamt-Chips



```
module TUD0705Lite
```

```
(  
  input  wire    CLOCK,  
  input  wire    COUNTERNUM,  
  output wire [7:0] COUNT  
);
```

```
COUNTER #(1) COUNTBY1(CLOCK, COUNTERNUM == 0, COUNT);  
COUNTER #(3) COUNTBY3(CLOCK, COUNTERNUM == 1, COUNT);
```

```
endmodule
```

A. Koch

Für Input-Wire **SELECT** direkt Ausdruck angeben, statt:

```
wire SELECTBY1, SELECTBY3;
```

```
assign SELECTBY1 = COUNTERNUM == 0;
```

```
assign SELECTBY3 = COUNTERNUM == 1;
```

```
counter #(1) COUNTBY1(CLOCK, SELECTBY1, COUNT);
```

```
counter #(3) COUNTBY3(CLOCK, SELECTBY3, COUNT);
```

Verilog: Modellierung des Testrahmens



A. Koch

```
module TESTFRAME;  
  
wire [7:0] COUNT;  
reg      CLOCK;  
reg      COUNTERNUM;  
  
TUD0705Lite DUT(CLOCK, COUNTERNUM, COUNT);
```

```
always begin // Takt erzeugen  
  CLOCK = 0;  
  #10;  
  CLOCK = 1;  
  #10;  
end
```

```
initial begin // Stimuli  
  COUNTERNUM = 0; // 1. Zähler  
  #60;  
  COUNTERNUM = 1; // 2. Zähler  
  #60;  
  COUNTERNUM = 0; // 1. Zähler  
  #60;  
  $finish;  
end
```

```
always @(COUNT) // Ausgaben überwachen  
  $display("%2.0f: COUNTERNUM=%b COUNT=%d",  
    $time, COUNTERNUM, COUNT);  
endmodule
```

Ergebnisse: Textausgabe



A. Koch

```
always begin // Takt erzeugen  
    CLOCK = 0;  
    #10;  
    CLOCK = 1;  
    #10;  
end
```

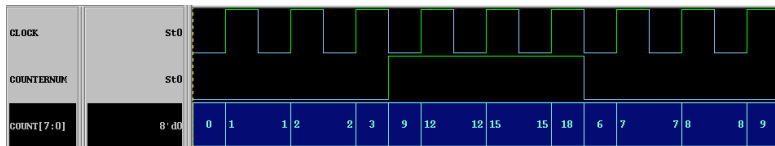
```
initial begin // Stimuli  
    COUNTERNUM = 0; // 1. Zähler  
    #60;  
    COUNTERNUM = 1; // 2. Zähler  
    #60;  
    COUNTERNUM = 0; // 1. Zähler  
    #60;  
    $finish;  
end
```

```
0 : COUNTERNUM=0 COUNT= 0  
10 : COUNTERNUM=0 COUNT= 1  
30 : COUNTERNUM=0 COUNT= 2  
50 : COUNTERNUM=0 COUNT= 3  
60 : COUNTERNUM=1 COUNT= 9  
70 : COUNTERNUM=1 COUNT= 12  
90 : COUNTERNUM=1 COUNT= 15  
110 : COUNTERNUM=1 COUNT= 18  
120 : COUNTERNUM=0 COUNT= 6  
130 : COUNTERNUM=0 COUNT= 7  
150 : COUNTERNUM=0 COUNT= 8  
170 : COUNTERNUM=0 COUNT= 9
```

Ergebnisse: Waves



A. Koch



```
0: COUNTERNUM=0 COUNT= 0
10: COUNTERNUM=0 COUNT= 1
30: COUNTERNUM=0 COUNT= 2
50: COUNTERNUM=0 COUNT= 3
60: COUNTERNUM=1 COUNT= 9
70: COUNTERNUM=1 COUNT= 12
90: COUNTERNUM=1 COUNT= 15
110: COUNTERNUM=1 COUNT= 18
120: COUNTERNUM=0 COUNT= 6
130: COUNTERNUM=0 COUNT= 7
150: COUNTERNUM=0 COUNT= 8
170: COUNTERNUM=0 COUNT= 9
```




- Mehrere Quellen/Senken auf einer Leitung: **Bus**
- Mehrere Senken: Unkritisch, fan-out immer konfliktfrei
- Quellen realisierbar durch **Tri-State-Treiber**
 - 0, 1, **Z** (hochohmig)
- Nur **eine** Quelle darf gleichzeitig aktiv sein
- Andere **hochohmig** schalten
- Benötigt Steuerung: **Welche** Quelle soll aktiv sein?
- Verschiedenste Möglichkeiten
- Hier gezeigt: **Slave-Mode**
 - Quellen wird von **aussen** mitgeteilt, ob Sie aktiv sein dürfen