18.341 Projekt

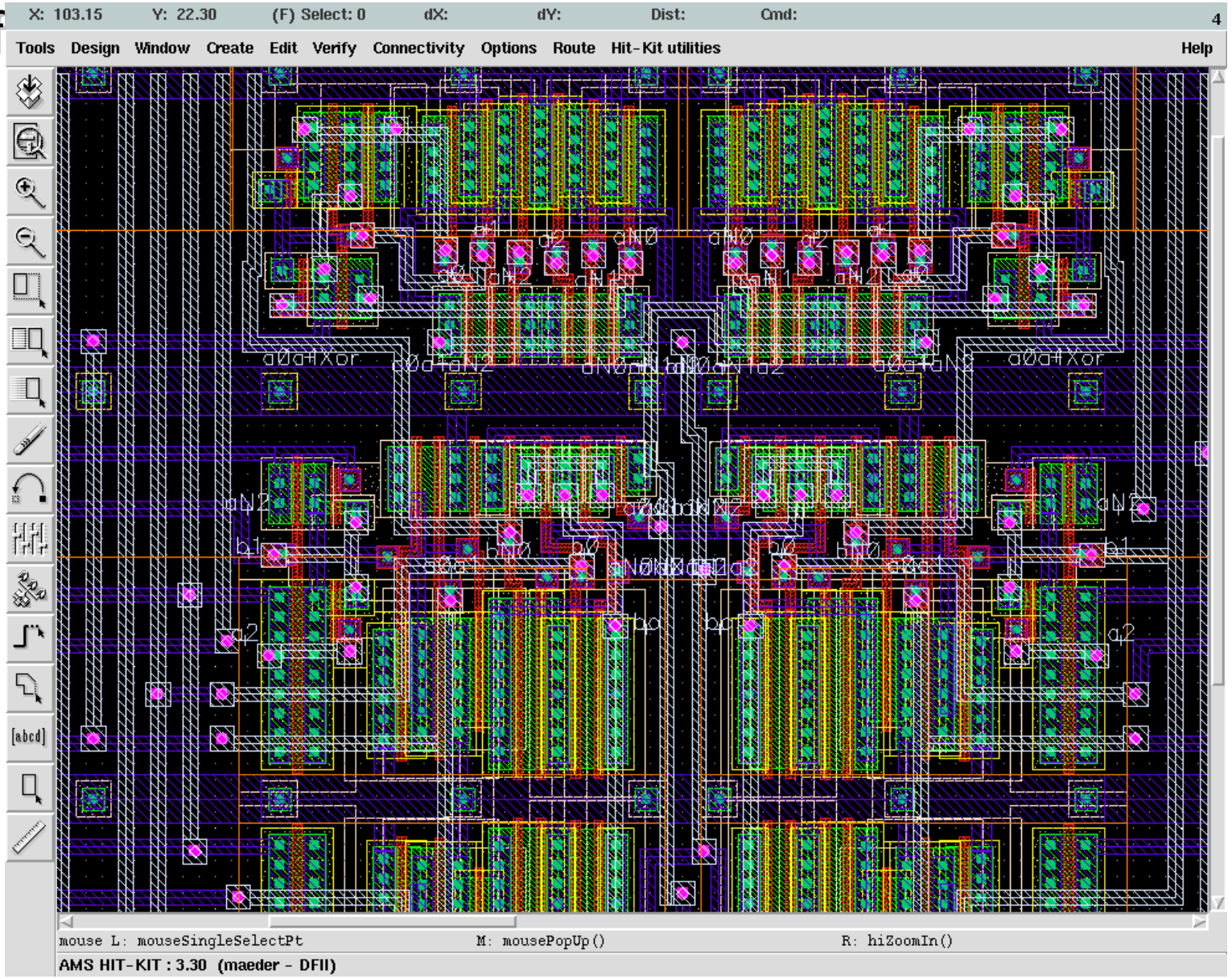
Systementwurf

—

Grundlagen des Schaltungs- und Systementwurfs

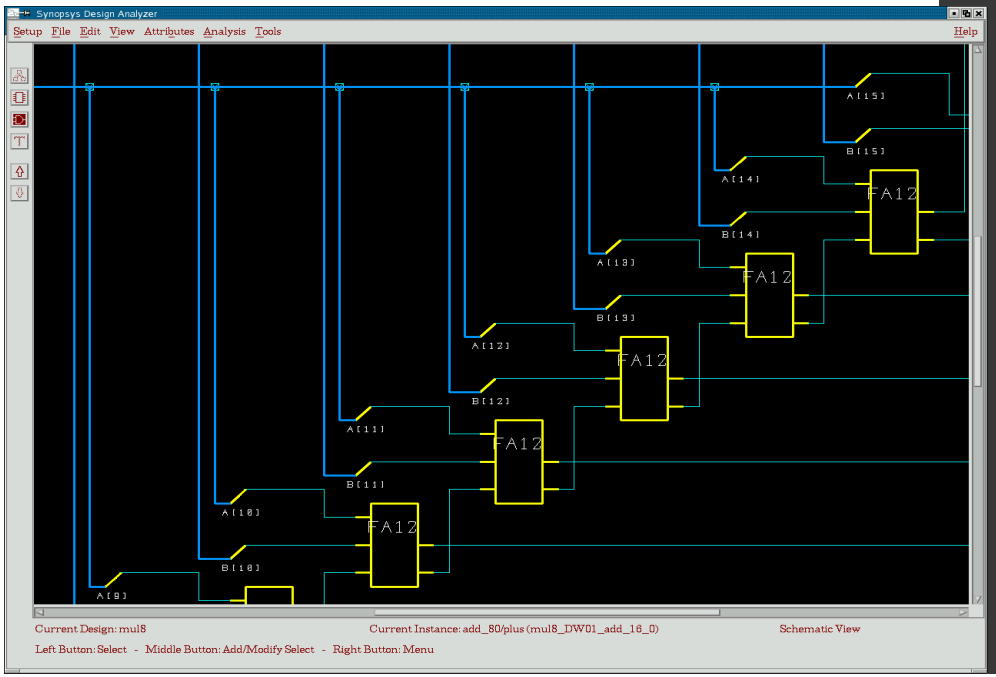
Hardwareentwurf

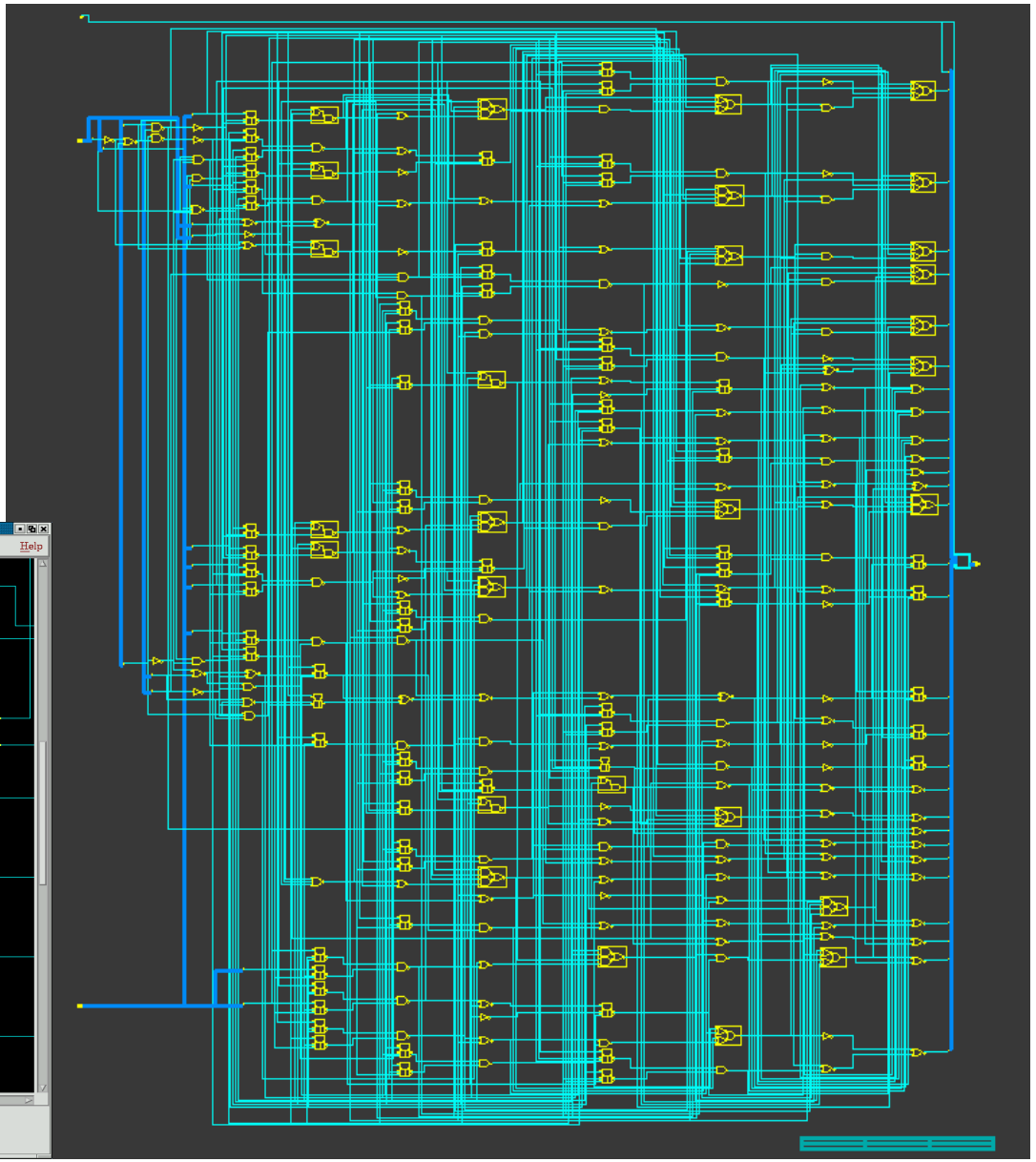
...so nicht



Hardwareentwurf

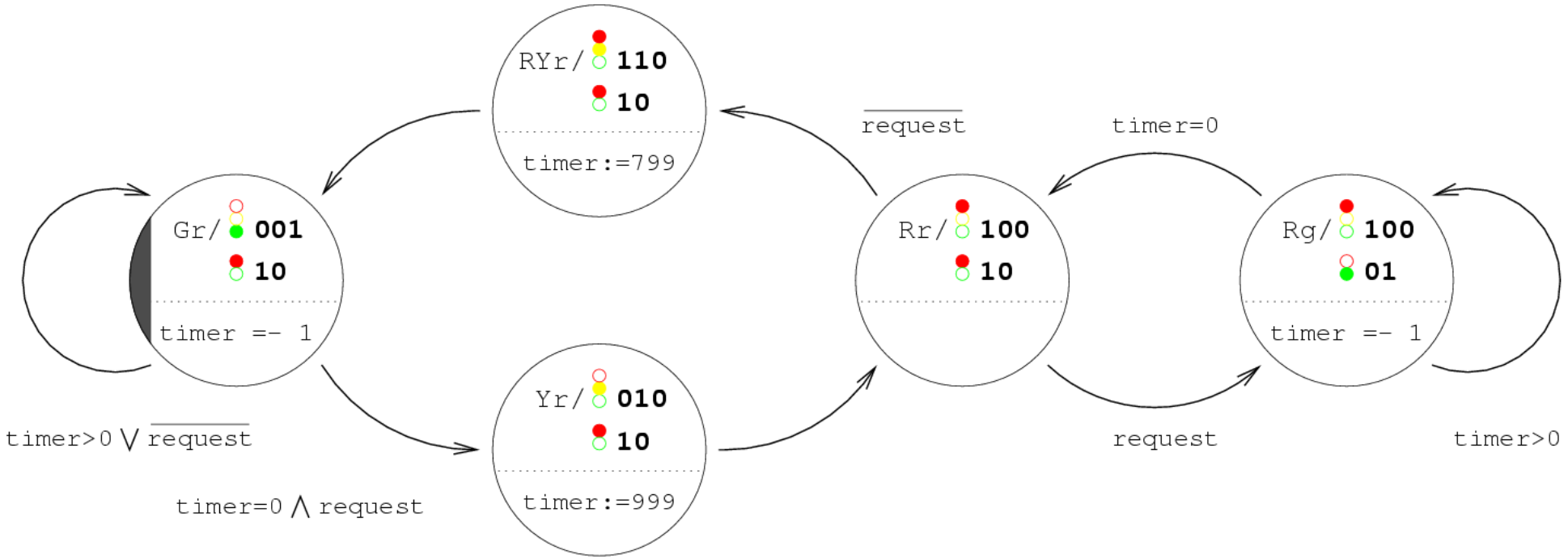
...so auch nicht





Hardwareentwurf

...sondern so



Hardwareentwurf

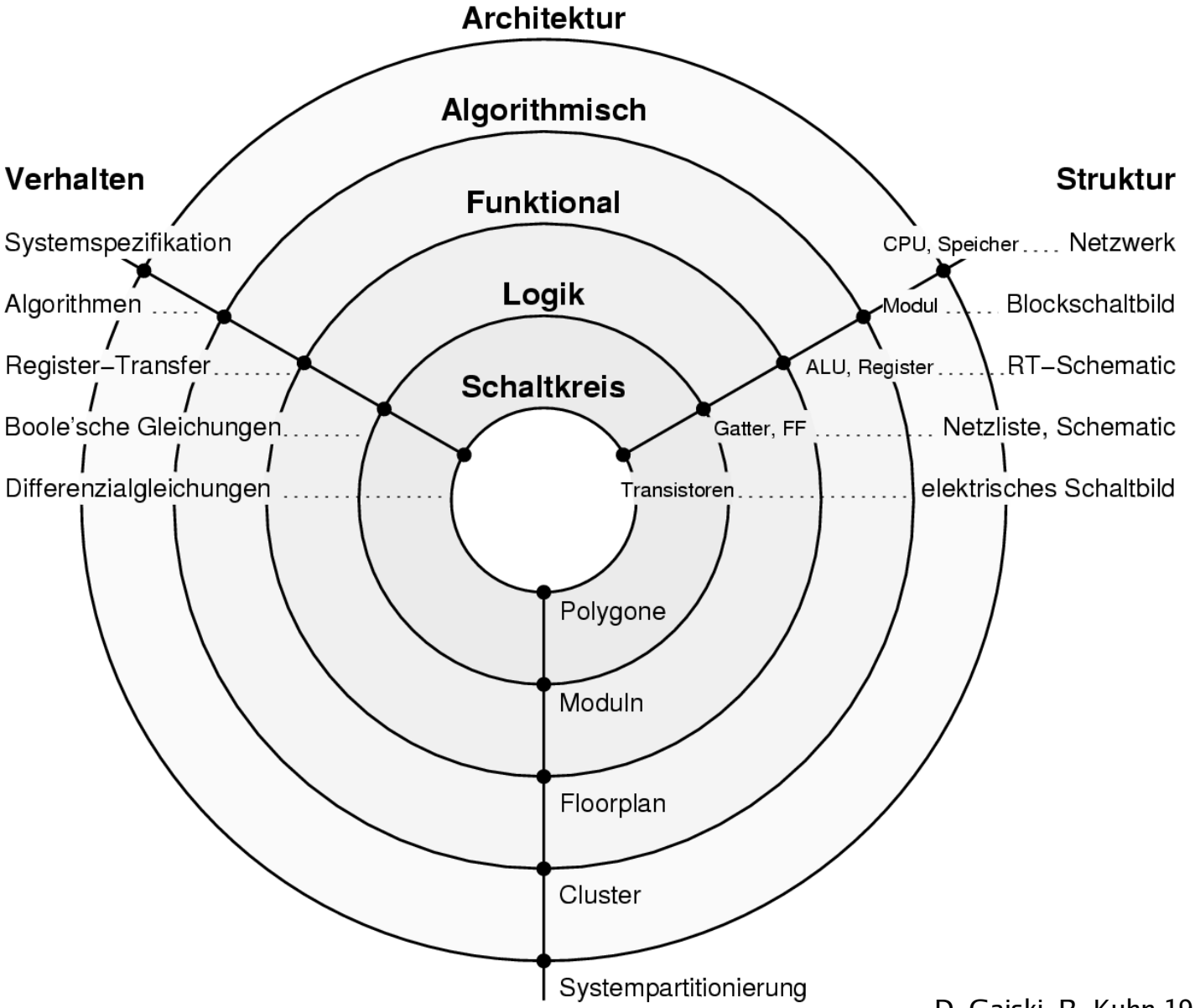
```
mainP: process (clk, rst)
    type stateTy is (Gr, Yr, Rr, Rg, RYr);
    variable   timer      : integer range 0 to maxWalkC;
    variable   state      : stateTy;
    variable   request    : boolean;
begin
  if rst = '0' then                                    -- async. reset
    lightCar  <= "001";
    lightWalk <= "10";
    state     := Gr;
    timer     := 0;
    request   := false;
  elsif rising_edge(clk) then                          -- clock
    case state is
    when Gr =>                                         -- Green + red
      lightCar  <= "001";
      lightWalk <= "10";
      if (reqWalk = '1')  then  request := true;       -- store reqest
      end if;
      if (timer > 0)      then  timer   := timer - 1;  -- no timeout
      elsif request       then  state   := Yr;         -- timeout and request
      end if;
    when Yr =>                                         -- Yellow + red
      lightCar  <= "010";
      lightWalk <= "10";
      timer     := maxWalkC-1;                         -- init. Timer
      ...
```

Inhalt

- Chipentwurf
  - Grundlagen
  - Begriffe
- VHDL
  - Typen, Objekte
  - sequenziell, konkurrent, hierarchisch
  - Simulation
- Beispiel Systementwurf
  - Datenhandschuh
  - Systemaufbau
  - Hardwareentwurf
  - VHDL-Simulation
  - Softwareentwurf
  - Systemintegration
    - Sensorintegration

Begriffe

- Abstraktion
  - Entwurfsebenen
- Sichtweisen
  - Verhalten
  - Struktur
- Entwurfsmethodik
  - iterativer Prozess
  - Alternativen
  - Versionen



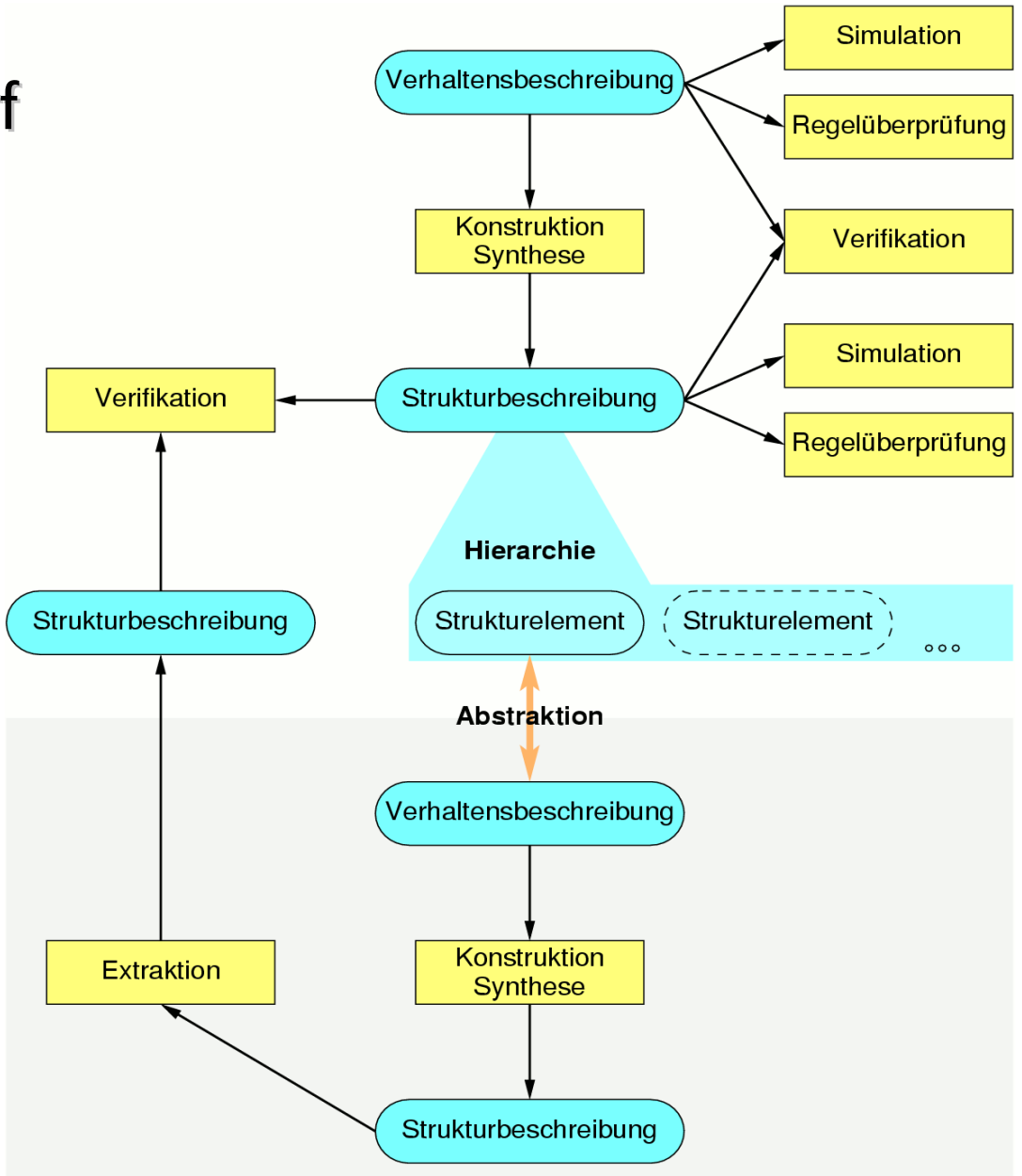
Y-Diagramm

D. Gajski, R. Kuhn 1983:
„New VLSI Tools“

Entwurfsmethodik

- Änderung in der Entwurfsmethodik
  - Struktur ⇒ Verhalten
  - grafische Eingabe ⇒ Hardwarebeschreibungssprache
- Entwurf auf höheren Abstraktionsebenen
- Automatische Transformationen bis zum Layout
  - Synthese
  - Datenpfad- / Makrozellgenerierung
  - Zellsynthese
  - Platzierung & Verdrahtung

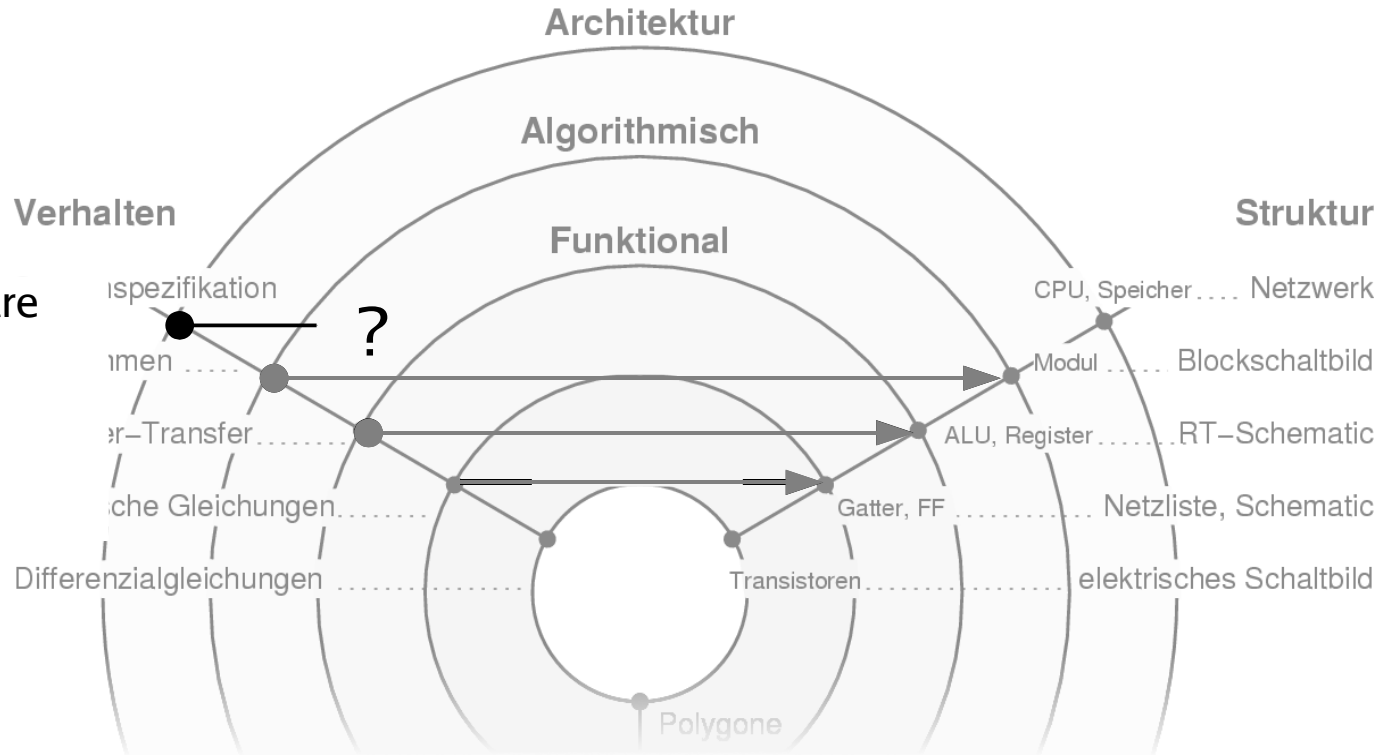
Hierarchischer Entwurf
„top-down“



Synthesewerkzeuge

- Einordnung in der Hierarchie
- Synthesewerkzeuge

Hardware-Software CoSynthese



VHDL

- VHSIC Hardware Description Language
  Very High Speed Integrated Circuit
- Entwicklung
  - 1983 vom DoD initiiert
  - 1987 IEEE Standard
  - Überarbeitungen VHDL'93, VHDL'02
  - Erweiterungen Hardwaremodellierung/Zellbibliotheken
    Hardwaresynthese
    mathematische Typen und Funktionen
    analoge Modelle und Simulation

VHDL – sequenziell

- Sequenzielle Programmiersprache (Pascal)
- Typen, Untertypen, Alias-Deklarationen
  - skalar `integer, real, character, boolean, bit,` Aufzählung
  - komplex `line, string, bit_vector,` Array, Record
  - Datei- `text,` File
  - Zeiger- Access
- Objekte `constant, variable, file`
- Operatoren

```
and, or, nand, nor, xor, xnor     =, /=, <, <=, >, >=
sll, srl, sla, sra, rol, ror      +, -, &    +, -
*, /, mod, rem                    **, abs, not
```

VHDL – sequenziell

- Anweisungen
  - Zuweisung `:=, <=`
  - Verzweigung `if, case`
  - Schleifen `for, while, loop, exit, next`
  - Zusicherungen `assert, report`
  - ...
- Sequenzielle Umgebungen
  - Prozesse `process`
  - Unterprogramme (rekursiv) `procedure, function`

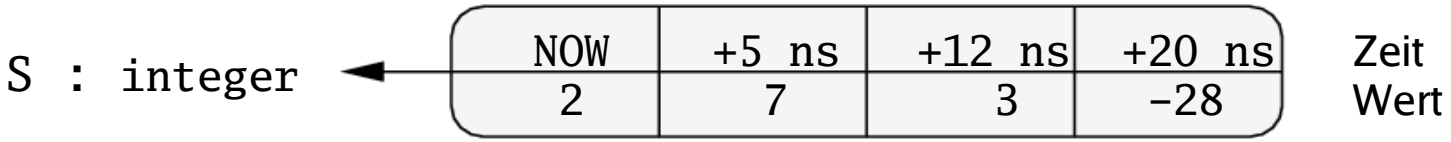
```
...
type      list_T;
type      list_PT          is  access list_T;
type      list_T           is  record key   : integer;
                                      link  : list_PT;
                           end record list_T;
constant  input_ID         : string   := "inFile.dat";
file      dataFile         : text;
variable  dataLine         : line;
variable  list_P, temp_P   : list_PT  := null;
...
procedure readData is
  variable  keyVal         : integer;
  variable  rdFlag         : boolean;
begin
  file_open (dataFile, input_ID, read_mode);
  L1: while not endfile(dataFile) loop
      readline(dataFile, dataLine);
      L2: loop
      read(dataLine, keyVal, rdFlag);
      if rdFLag    then    temp_P := new list_T'(keyVal, list_P);
                           list_P := temp_P;
                   else    next L1;
      end if;
      end loop L2;
  end loop L1;
  file_close(dataFile);
end procedure readData;
```

VHDL – konkurrent

- Konkurrenter Code (ADA'83)
  Modelliert die gleichzeitige Aktivität der Hardwareelemente
  - Mehrere Prozesse
  - Prozeduraufrufe
  - Signalzuweisung, bedingt (if), selektiv (case) `<=`
  - Zusicherung `assert`
- Synchronisationsmechanismus für Programmlauf / Simulation
  - Objekt `signal`
  - Signale verbinden konkurrent arbeitende „*Teile*“ miteinander
  - Entsprechung in Hardware: Leitung

VHDL – Simulation

- Semantik der Simulation im Standard definiert: *Simulationszyklus*
- konkurrent aktive Codefragmente
  - Prozesse, konkurrente Anweisungen, Hierarchien
  - Signale verbinden diese Codeteile (Prozesse)
- Signaltreiber: Liste aus Wert-Zeit Paaren



- Simulationsereignis:
  - Werteänderung eines Signals
  - (Re-) Aktivierung eines Prozesses nach Wartezeit

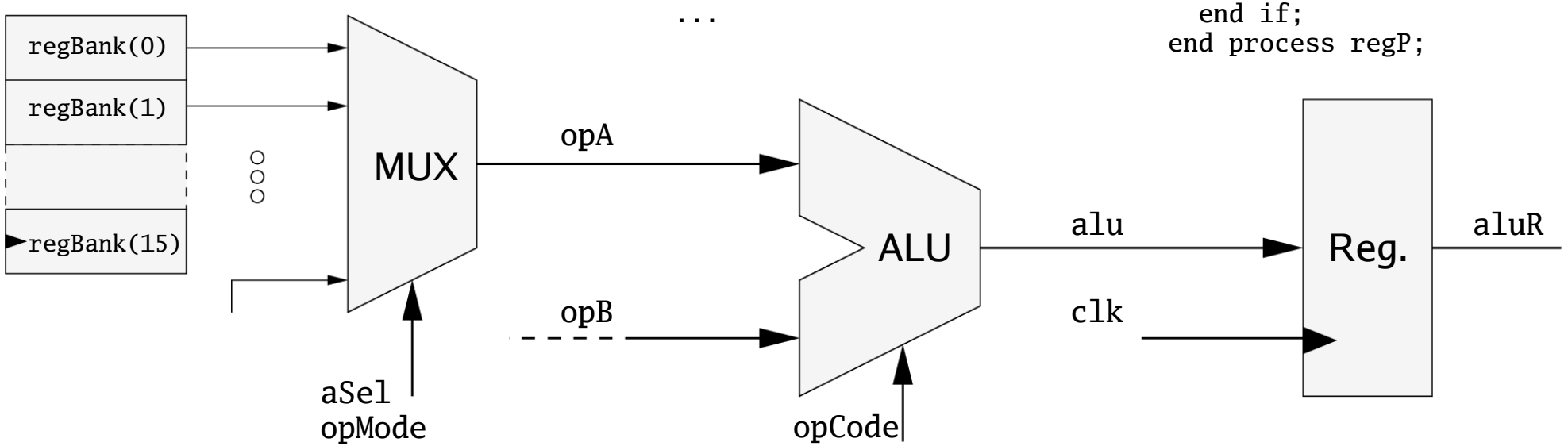
Ereignisgesteuerte Simulation

- Beispiel: Datenpfad

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation

1. Simulationsereignis

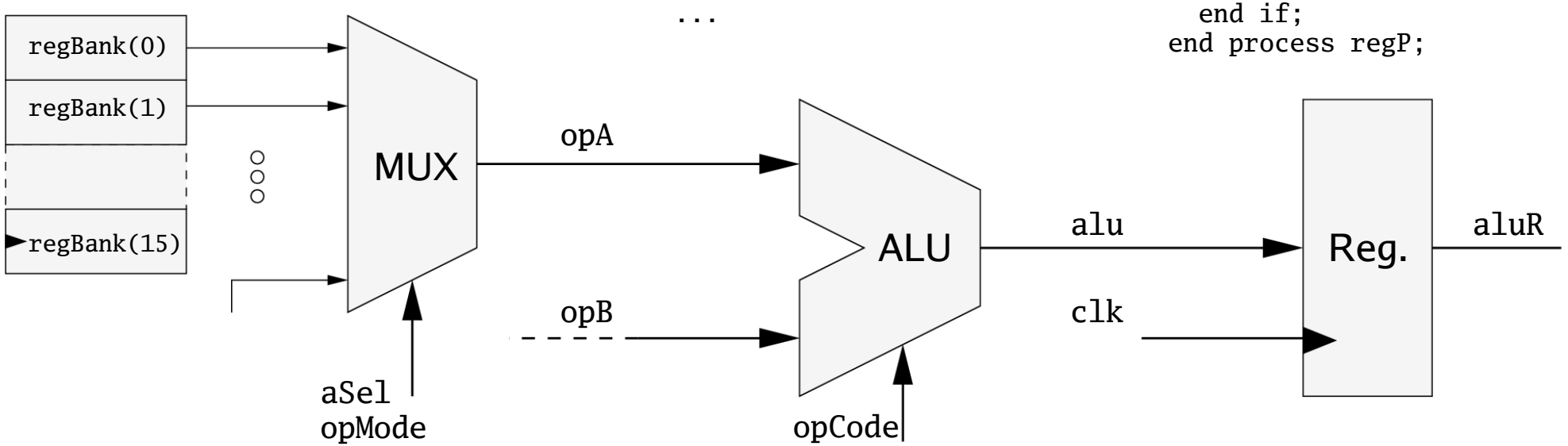
Liste aller Ereignisse

```
NOW       aSel      1
          opMode    regM
          opCode    opcAdd
+10 ns    clk       '1'
...
```

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



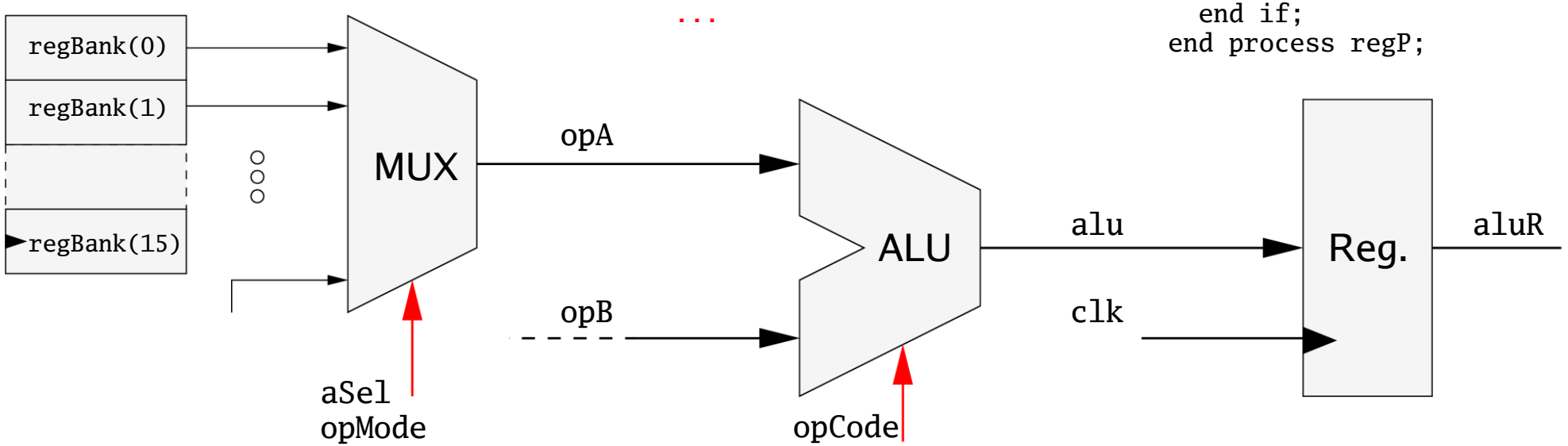
Ereignisgesteuerte Simulation

1. Simulationsereignis
2. Prozessaktivierung

```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

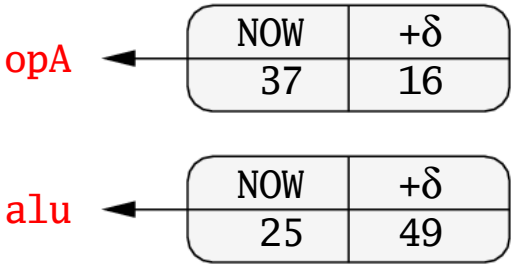
```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation

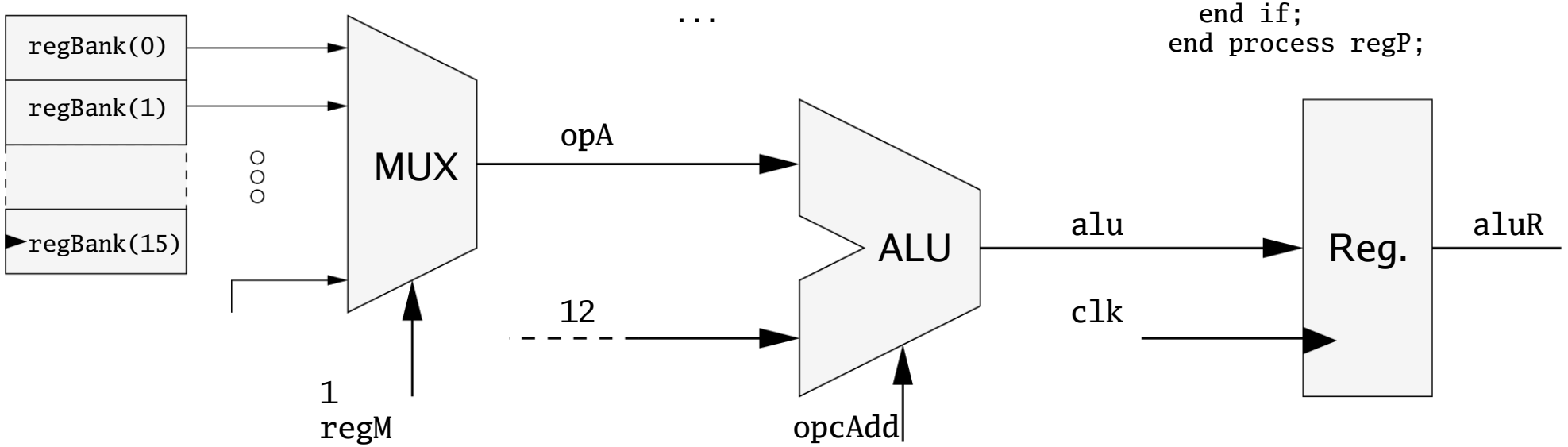
1. Simulationsereignis
2. Prozessaktivierung
3. Aktualisierung der Signaltreiber




```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation

Zeitschritt: +δ

Simulationsereignisse

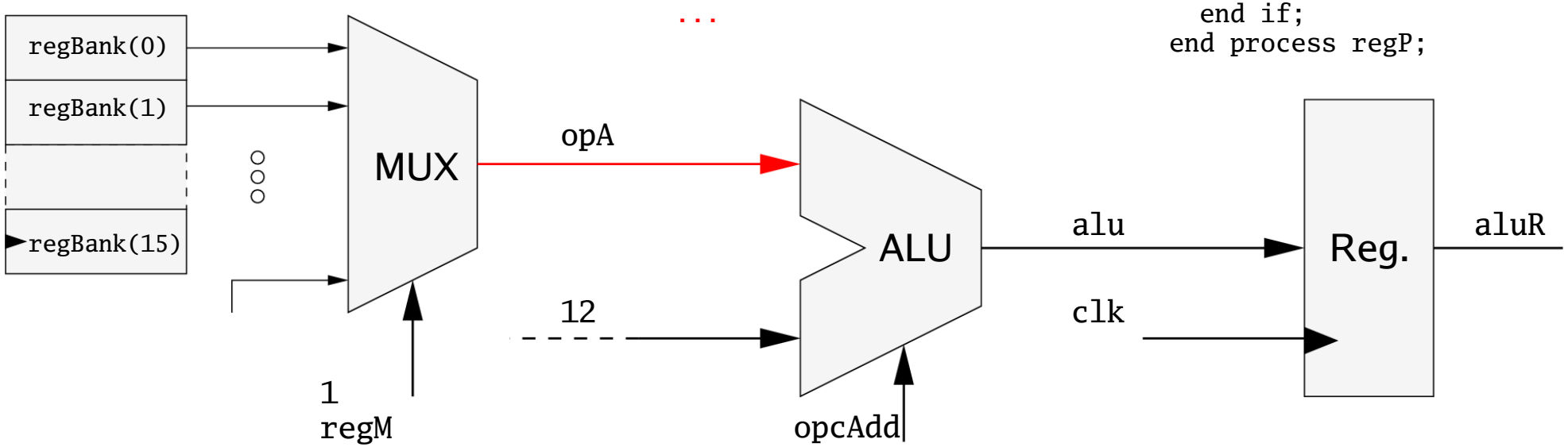
+δ	opA	16
	alu	49
+10 ns	clk	'1'
...		

Signaltreiber


```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation

Zeitschritt: +10 ns

Simulationsereignisse

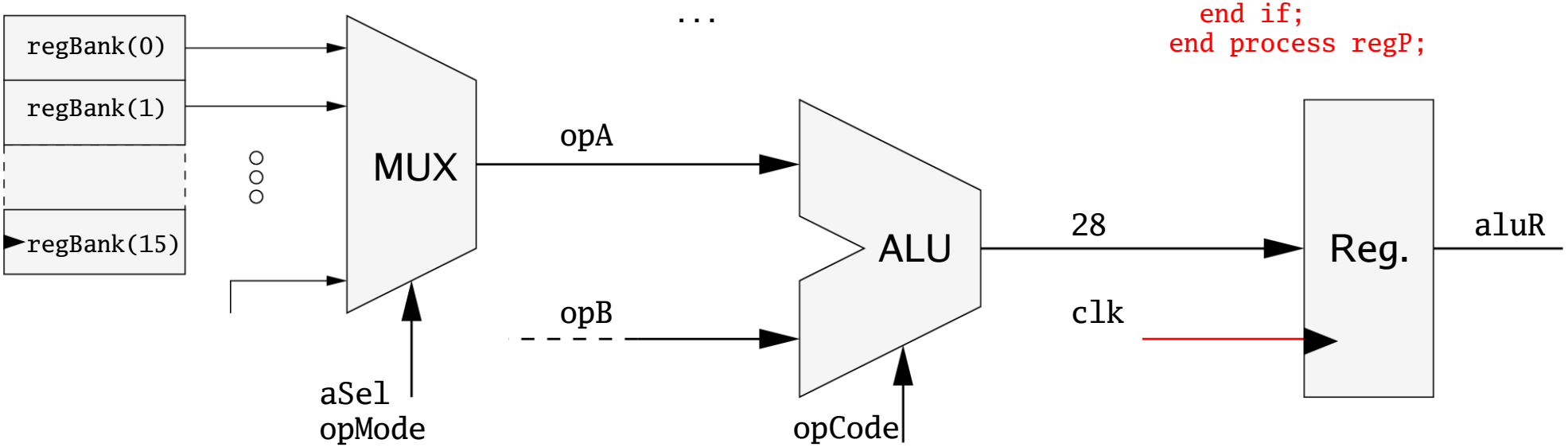
```
+10 ns   clk      '1'
...
```

Signaltreiber


```
opA <= regBank(aSel)
         when opMode=regM else
       dataBus;
```

```
with opCode select
alu <= opA  +  opB  when opcAdd,
       opA  -  opB  when opcSub,
       opA and opB  when opcAnd,
       ...
```

```
regP: process (clk)
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



VHDL – Simulation

- Kausalität
  1. Simulationsereignis — $Zyklus_n$
  2. Aktivierung des konkurrenten Codes
  3. Signalzuweisungen in der Abarbeitung

  ---

  4. Erneute Signaländerung — $Zyklus_{n+1}$
- Trennung der Zyklen
  - Simulation ist unabhängig von der *sequenziellen Abarbeitungsreihenfolge* durch den Simulator, auch bei *mehreren Events* in einem Zyklus, bzw. *mehrfachen Codeaktivierungen* pro Event !

VHDL – Simulation

- Prozesse sind ständig aktiv ⇒ Endlosschleife

Typen: 1. Sensitiv zu Signalen bis Prozessende

```
alu_P: process(a, b, add)
begin
  if add  then  x <= a+b;
          else  x <= a-b;
  end if;
end process alu_P;
```

2. `wait`-Anweisungen bis wait

```
producer_P: process
begin
  pReady  <= false;
  wait until cReady;
  channel <= ...
  pReady  <= true;
  wait until not cReady;
end process producer_P;

consumer_P: process
begin
  cReady <= true;
  wait until pReady; ...
```

VHDL – Simulation

- Signalzuweisungen im sequenziellen Kontext
  - sequenzieller Code wird nach Aktivierung bis zum Prozessende/`wait` abgearbeitet
  - Signalzuweisungen werden (frühestens) im folgenden Simulationszyklus wirksam

⇒ *eigene Zuweisungen sind in dem Prozess nicht sichtbar*

```
process ...
  ...
  if swap = '1' then
    b <= a;
    a <= b;
  end if;
```

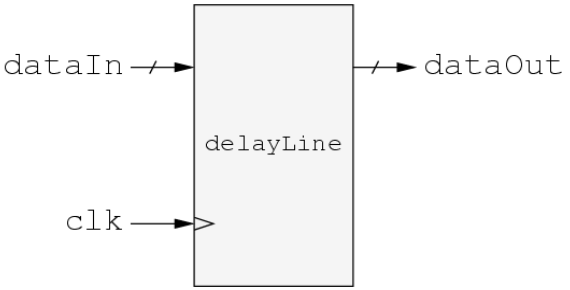
```
process ...
  ...
  num <= 5;
  ...
  if num > 0 then
    ...
```

VHDL – Entwurf

- Entwurfsspezifische Eigenschaften
  - Strukturbeschreibungen / Hierarchie
    Instanziierung von Teilentwürfen `component` `configuration`
  - Schnittstellen `entity`
  - Versionen und Alternativen `architecture` `configuration`
- zusätzliche Features
  - Bibliotheken `library`
  - Design- und Code-Reuse `package`

VHDL – Entity

- Entity als zentrale Entwurfseinheit
  - Beschreibung der Schnittstelle „black-box“
  - mit Parametern `generic`
  - und Ein- / Ausgängen `port`




```
entity delayLine is
generic ( bitWid  :     integer range 2 to 64 := 16;
          delLen  :     integer range 2 to 16 := 16);
   port ( clk     : in  std_logic;
          dataIn  : in  signed (bitWid-1 downto 0);
          dataOut : out signed (bitWid-1 downto 0));
end entity delayLine;
```

VHDL – Architektur

- Architektur als Implementation einer Entity
  - mehrere Architekturen sind möglich ⇒ Alternativen
  - lokale Deklarationen
  - konkurrente Anweisungen + Prozesse + Instanzen

```
architecture behavior of delayLine is
  type    delArray_T is array (1 to delLen) of signed (bitWid-1 downto 0);
  signal  delArray   :  delArray_T;
begin
  dataOut <= delArray(delLen);
  reg_P: process (clk)
  begin
    if rising_edge(clk) then delArray <= dataIn & delArray(1 to delLen-1);
    end if;
  end process reg_P;
end architecture behavior;
```

VHDL – strukturell

- Hierarchie
  - repräsentiert Abstraktion
  - zur funktionalen Gliederung
- Instanziierung von Komponenten
  1. Komponentendeklaration und `component`
  2. Instanziierung in der Architecture
  3. Bindung an Paar: Entity+Architecture `configuration`
- Komponente als Zwischenstufe mit anderen Bezeichnern und Schnittstellen (Ports und Generics)

VHDL – strukturell

- Konfiguration zur Bindung: Komponente ⇔ Entity+Architecture
  - lokal, innerhalb der Architektur
  - als eigene Entwurfseinheit
  - „default“-Regeln: identische Bezeichner, Deklarationen
- Strukturierende Anweisungen
  - Gruppierung `block`
  - bedingte und/oder wiederholte Ausführung konkurrenten Codes oder Instanziierungen `generate`

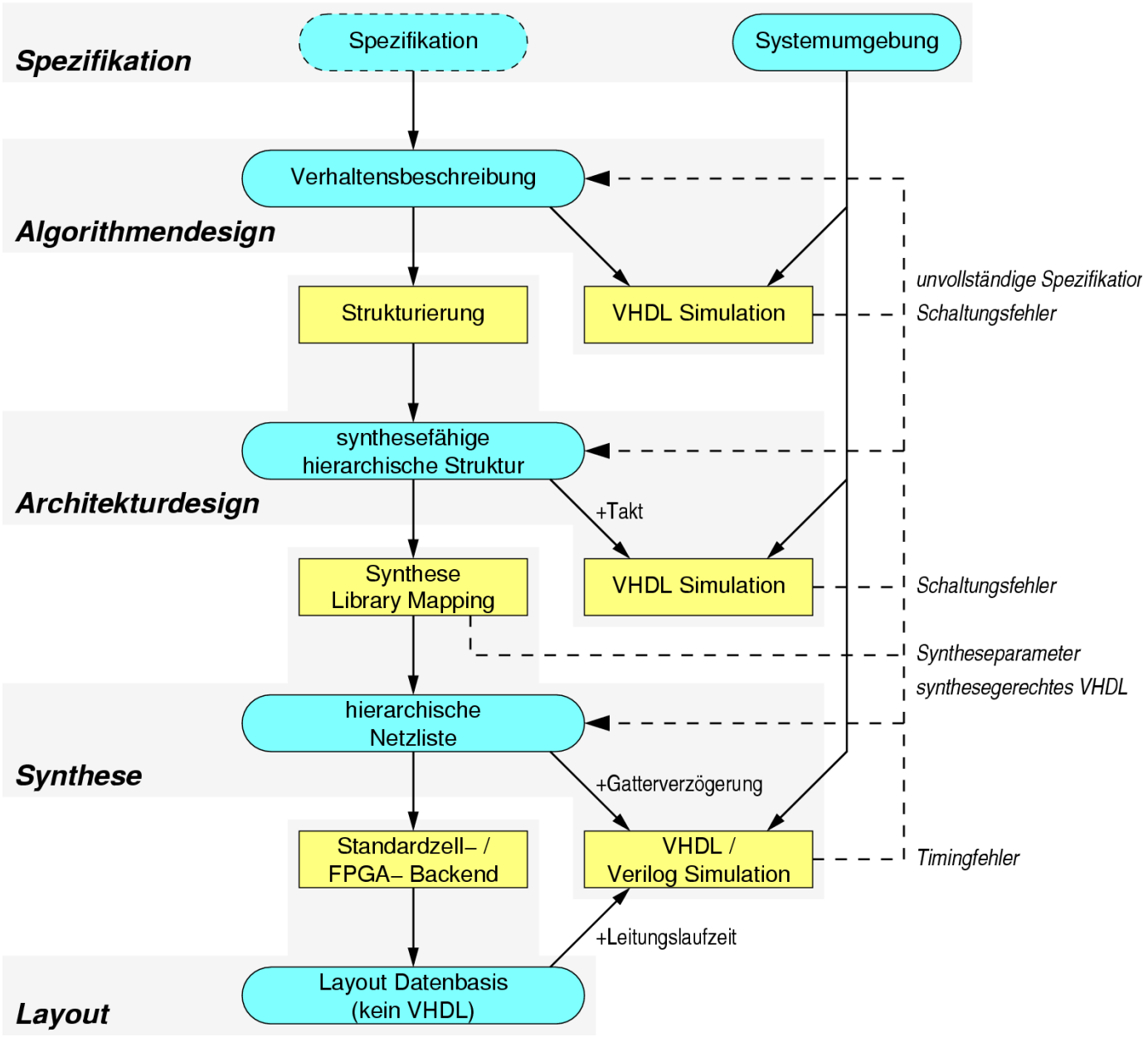
```
architecture structure of delayLine is
  component reg is
  generic ( width  :     integer range 2 to 64);
  port    ( clk    : in  std_logic;
            dIn    : in  signed(width-1 downto 0);
            dOut   : out signed(width-1 downto 0));
  end component reg;
  type    delReg_T is array (0 to delLen) of signed(bitWid-1 downto 0);
  signal  delReg   :  delReg_T;
begin
  delReg(0) <= dataIn;
  dataOut   <= delReg(delLen);
  gen_I: for i in 1 to delLen generate
    reg_I: reg generic map (width => bitWid)
                   port map (clk, delReg(i-1), delReg(i));
  end generate gen_I;
end architecture structure;

configuration delayLineStr of delayLine is
for structure
  for gen_I
    for all: reg use entity work.reg(behavior);
    end for;
  end for;
end for;
end configuration delayLineStr;
```

VHDL – Synthese

- VHDL deckt Abstraktion von Algorithmen- bis zur Gatterebene ab
  - eine Sprache als Ein- und Ausgabe der Synthese
- Synthese - allgemein üblich: RT-Ebene
  - Abbildung von Register-Transfer Beschreibungen auf Gatternetzlisten
  - erzeugt neue Architektur, Entity bleibt
- Simulation
  - System-/Testumgebung als VHDL-Verhaltensmodell
  - Simulation der Netzliste durch Austausch der Architektur

VHDL-basierter Entwurfsablauf

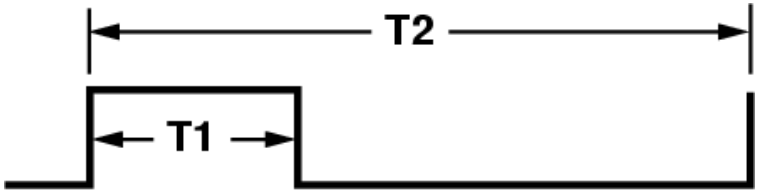


Beispiel

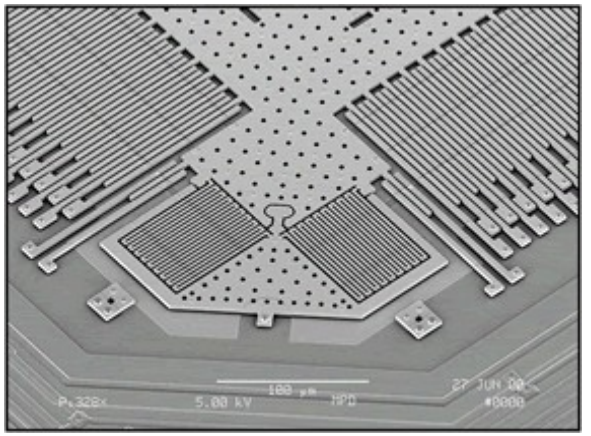
- Datenhandschuh
  - Biege- und Beschleunigungssensoren

  Krümmung der Finger
  Lage und Wege ⇒ Werte Integrieren
- MEMS Beschleunigungssensor

  Analog Devices ADXL202
  - 2-Achsen
  - pulsbreitenmoduliertes Signal



A(*g*) = (T1/T2 – 0.5)/12.5%
0*g* = 50% DUTY CYCLE
T2(s) = $R_{SET}(\Omega)/125M\Omega$



Beispiel – Softwareimplementation

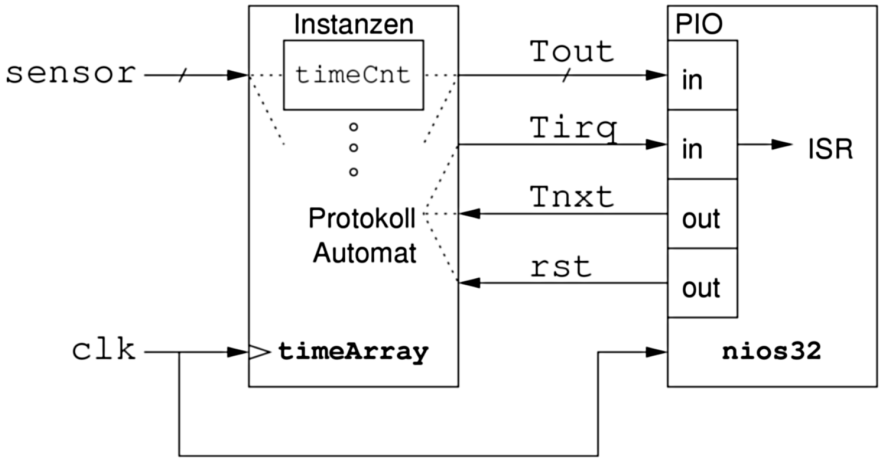
- Softwarelösung
  - Abtastung mit Prozessor, timer-Interrupt,...
- Probleme
  - Genauigkeit T2: 1 ms, Auflösung: 14-bit ⇒ Abtastrate 16 Mhz
  - Erweiterbarkeit Datenhandschuh mit 7x2-Sensoren

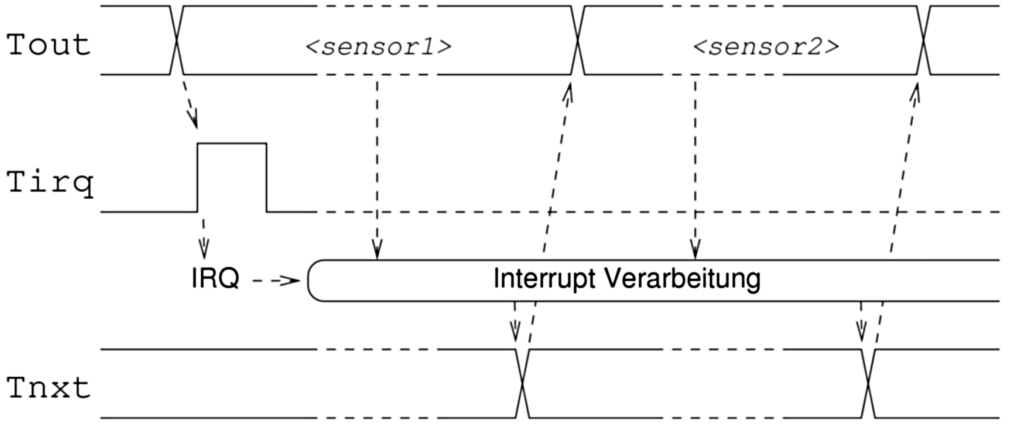
⇒ Hardware „überdimensionieren“ nur für Abtastung?

- typisches Beispiel für „Datenreduktion“
  - pro Sensor jede Millisekunde T1 (und T2), je 14-bit

Beispiel – CoDesign

- Systemdesign
  - Hardware
    - Sensoren abtasten
    - T1, T2 messen
  - Software
    - Normierung
    - Beschleunigung
    - Strecken ?
- Schnittstelle
  - Interruptgetrieben
  - parallel IO



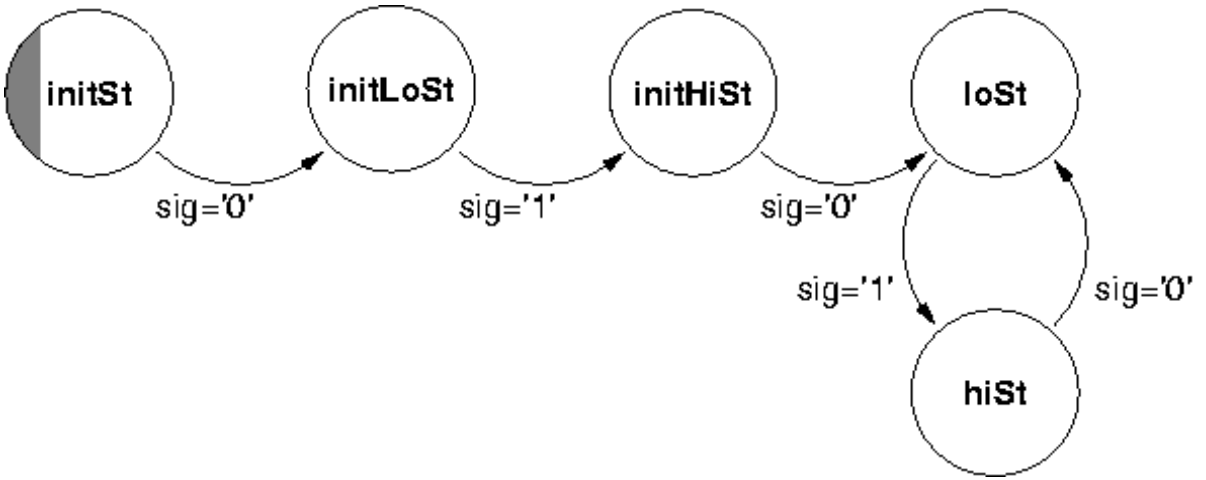


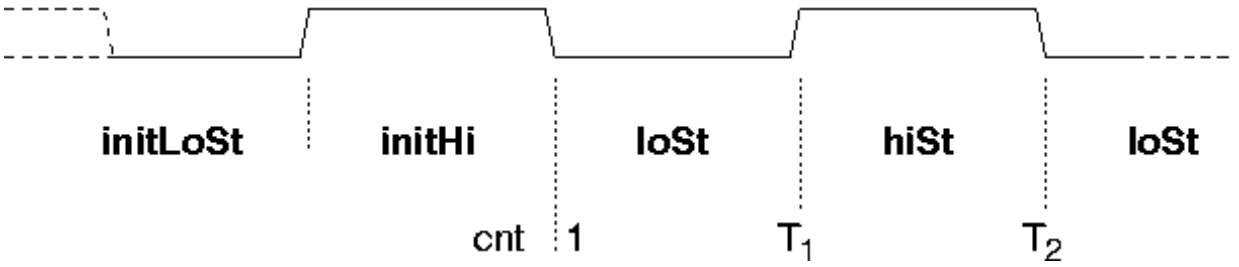
Beispiel – CoDesign

- Hardware
  - `timeCnt` Abtastung eines Sensors (33MHz)
    T1, T2 als 16-bit unsigned Wort
  - `timeArray` Instaziierung der `timeCnts`
    Implementation der Protokollschnittstelle
- Software, prototypisch
  - `iniISR` je Sensor: Maximum und Minimum von T1, T2 bestimmen
  - Normalisierung berechnet Skalierungsfaktor
  - `runISR` Sensorwerte berechnen 16-bit signed

Hardwareentwurf

- `timeCnt`
  - endlicher Automat
  - Zähler `cnt`
  - Register `T1, T2`
  - Flag `Tflag`





Hardwareentwurf

• Codierung der Schaltung		`timeCnt`
• Simulation in einer Testumgebung	(VHDL)	`timeCntTB`
• Aufbau der VHDL-Hierarchie und entsprechende Simulationen		`timeArray` `timeArrayTB`
• SOPC Builder für Prozessor		`nios32`
• Aufbau des Gesamtsystems		`niosBoard`

Software + Systemintegration

- Software entwickeln
  - Softwaretest setzt Hardware voraus !
    ⇒ Aufbau der Systems
  - Entwicklung kleiner Testprogramme die später um weitere Funktionen ergänzt werden
- Problem: Debugging der Schnittstelle
  - Software — Testausgaben
  - zusätzliche Hardware — Signale nach Außen führen
    LEDs
    ...