# Meisenantworten … vielen Dank für die Hinweise!

TECHNISCHE UNIVERSITÄT DARMSTADT

- Autonome Züge? *Haben wir technischen Seite her schon. Aber die Juristen verlanden, dass Lokführer alle zwei Minuten den Tot-Mann-Knopf drücken…*
- Elektronische Meisenstatistik. *Habe ich verlinkt, siehe Lehre-Wiki. Eure Mühe sollen alle teilen. Auf Anfrage kriegt Ihr auch Eure eigene Wiki-Seite (sagt mir nur wie ich Euch anonym login+passwort zukommen lassen kann).*
- "Wenn ich das Buch esse, bestehe ich dann die Prüfung?" Ja, aber nur wenn Du es in einem Happen isst, das Video auf YouTube stellen lässt und den Darwin-Award gewinnst. Dann wirst Du u.U. post-hum bestehen…
- Mettbrötchen für Chris' Ernährungs- und Traingsplan. *Ernährung: Mettbrötchen-only. Traingsplan: Geheimnis, nur IAS Studenten (Hiwis, BSc/MSc/PhD Theses, IP, etc) und Mitarbeiter werden normalerweise von ihm eingeweiht…und eine Ausnahme: den 3 Studierenden mit der besten TGDI Abschlussnote werden Chris und Herke ein Probetraining gewähren!*

# Meisenantworten … vielen Dank für die Hinweise!

TECHNISCHE UNIVERSITÄT DARMSTADT

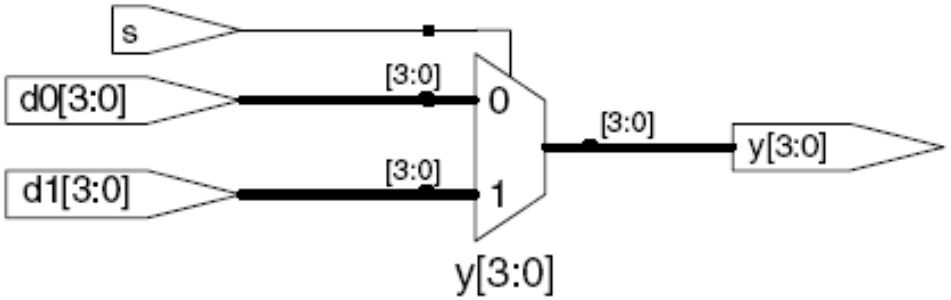
- Grösserer Font für SystemVerilog code? *OK, yessir … or yes, ma'm*!
- Organisatorische Verbesserung. *Habe ich an die Übungsleiter (Chris & Herke) weitergeleitet*.

- **Chris: Wir haben einen Tutor der eine Sprechstunde anbietet, das haben wir letzte Woche auch ueber Moodle angekuendigt.**
- Übung und Vorlesung besser synchronisieren. *Müssen leider den FB Standards folgen… kann ich leider nicht ändern* ☹!
- Prozentzahlen auf 2. Nachommastelle runden. *Done*!

- **Weitere fünf Meisenfragen werden von der Meise immer noch verdaut…**

# Bedingte Zuweisung

```
module mux2 (input  logic[3:0] d0, d1,
             input  logic s,
             output logic[3:0] y);
   assign y = s ? d1 : d0;
endmodule
```



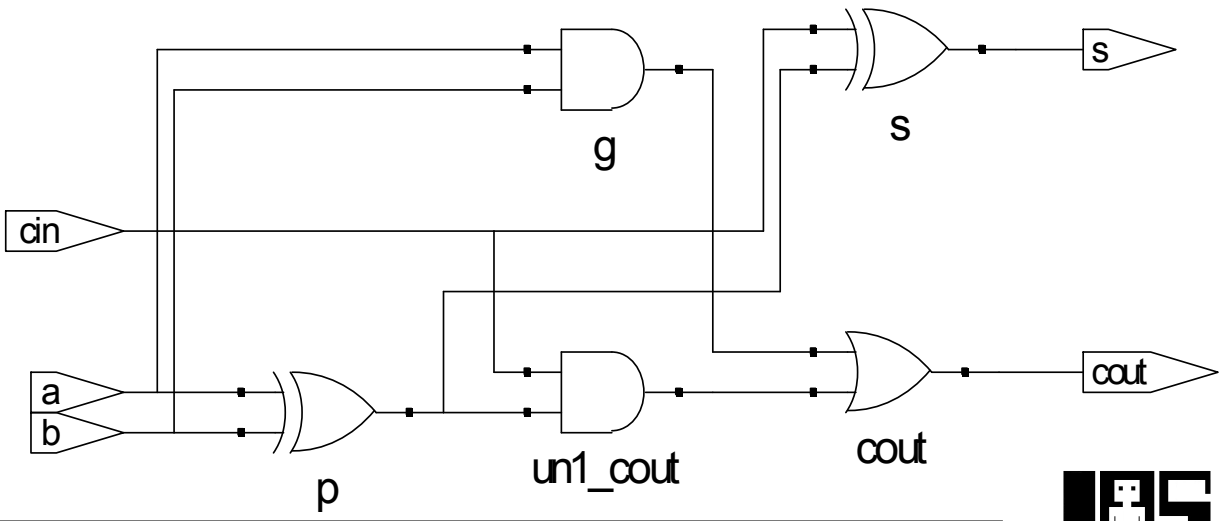

`? :` ist ein ternärer Operator, da er drei Operanden miteinander verknüpft: `s`, `d1`, und `d0`. So wie in C/C++!

# Interne Verbindungsknoten oder Signale

```
module fulladder(input logic a, b, cin, output logic s, cout);
  logic p, g;          // interne Verbindungsknoten ("Drähte")

  assign p = a ^ b;
  assign g = a & b;

  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```



# Bindung von Operatoren (Präzedenz)

TECHNISCHE UNIVERSITÄT DARMSTADT

Bestimmt Ausführungsreihenfolge

Höchste

| | |
|---|---|
| ~ | NOT |
| *, /, % | Multiplikation, Division, Modulo |
| +, - | Addition, Subtraktion |
| <<, >> | Schieben (logisch) |
| <<<, >>> | Schieben (arithmetisch) |
| <, <=, >, >= | Vergleiche |
| ==, != | gleich, ungleich |
| &, ~& | AND, NAND |
| ^, ~^ | XOR, XNOR |
| \|, ~\| | OR, NOR |
| ?: | Ternärer Operator |

Niedrigste

# Zahlen

TECHNISCHE UNIVERSITÄT DARMSTADT

Syntax: *N'Bwert*

*N* = Breite in Bits, B = Basis

*N'B* ist optional, sollte der Konsistenz halber aber immer geschrieben werden

wenn weggelassen: Dezimalsystem

| Zahl | Bitbreite | Basis | entspricht Dezimal | Darstellung im Speicher |
|---|---|---|---|---|
| 3’b101 | 3 | binär | 5 | 101 |
| ‘b11 | Nicht vorgegeben | binär | 3 | 00…0011 |
| 8’b11 | 8 | binär | 3 | 00000011 |
| 8’b1010_1011 | 8 | binär | 171 | 10101011 |
| 3’d6 | 3 | dezimal | 6 | 110 |
| 6’o42 | 6 | oktal | 34 | 100010 |
| 8’hAB | 8 | hexadezimal | 171 | 10101011 |
| 42 | Nicht vorgegeben | dezimal | 42 | 00…0101010 |

# Operationen auf Bit-Ebene: Beispiel 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};

/* wenn y ein 12-bit Signal ist, hat die Anweisung diesen
 Effekt: */
y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0
```

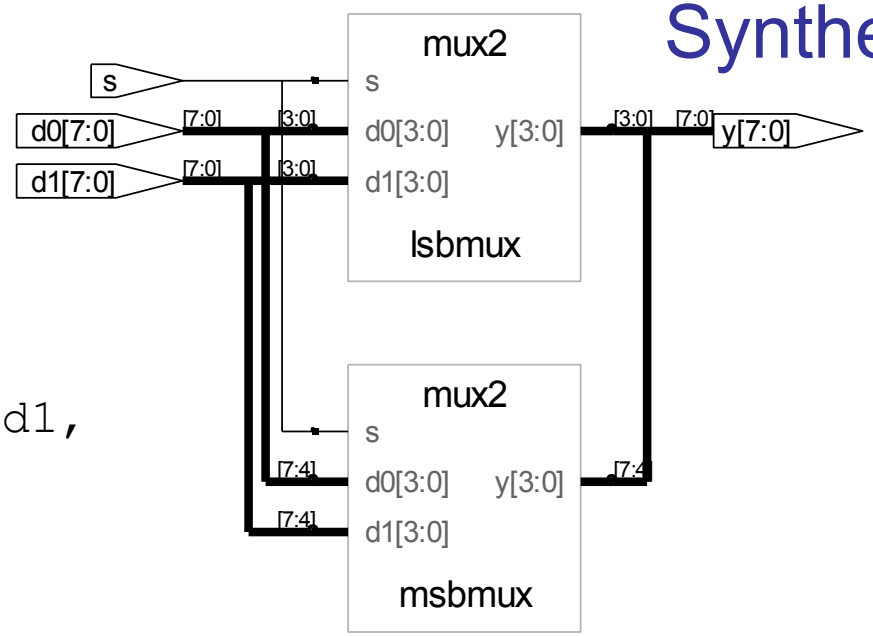
Unterstriche (_) in numerischen Konstanten dienen nur der besseren Lesbarkeit, sie werden von SystemVerilog ignoriert

LEHRE WIKI TEST IN DREI FOLIEN

# Operationen auf Bit-Ebene: Beispiel 2

TECHNISCHE UNIVERSITÄT DARMSTADT

## Synthese




## SystemVerilog

```
module mux2_8
    (input  logic[7:0] d0, d1,
     input  logic[1:0] s,
     output logic[7:0] y);

  mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
  mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```

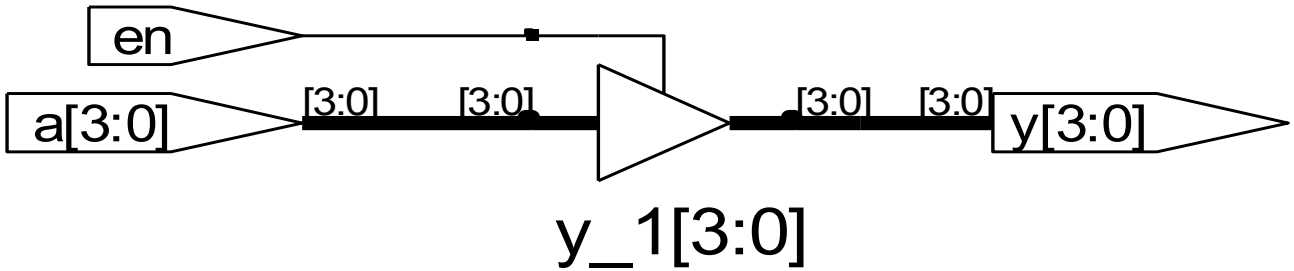
# Hochohmiger Ausgang: Z

TECHNISCHE UNIVERSITÄT DARMSTADT

## SystemVerilog:

```
module tristate (input logic[3:0] a,
                 input logic[1:0] en,
                 output tri[3:0] y);
   assign y = en ? a : 4'bz;
endmodule
```

## Synthese:



# LEHRE WIKI FRAGE

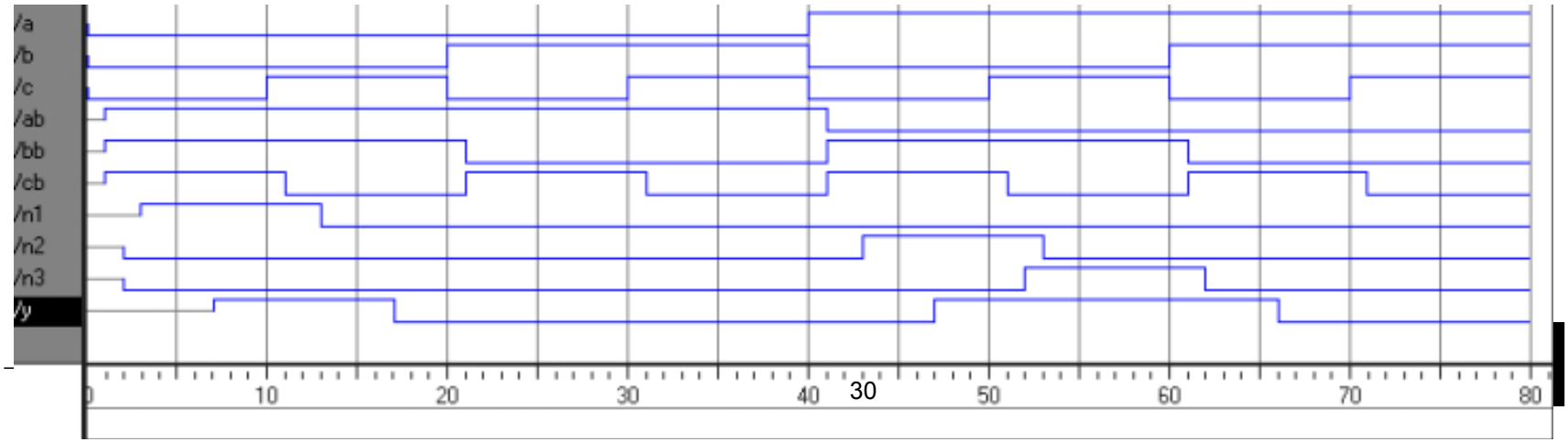
- Bitte jetzt auf LEHRE WIKI eine Frage beantworten!

# Verzögerungen: # *Zeiteinheiten*

TECHNISCHE UNIVERSITÄT DARMSTADT

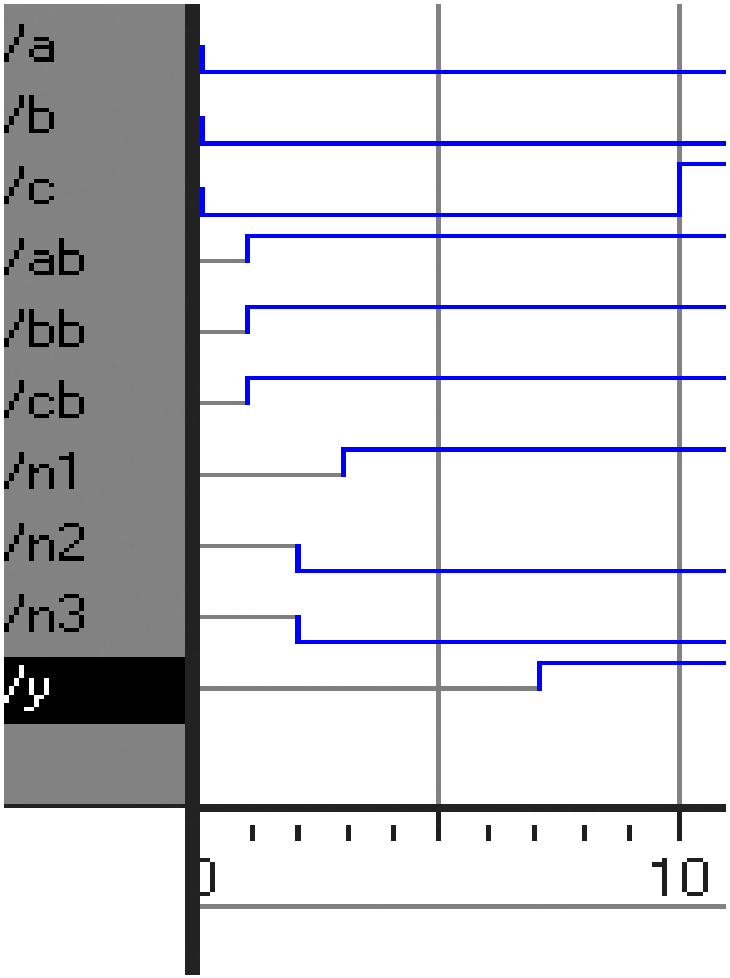
```
`timescale 1ns/1ps
module example (input  logic a, b, c, output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} = ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```

Nur für die Simulation,
*#n* werden für die Synthese
**ignoriert**!



# Verzögerungen

```
module example(input logic a, b, c,
                output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} =
                 ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```




Nur für die Simulation,
*#n* werden für die Synthese
**ignoriert**!

# Sequentielle Schaltungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beschreibung basiert auf Verwendung fester “Redewendungen”
  - Idiome

- Feststehende Idiome für
  - Latches
  - Flip-Flops
  - Endliche Zustandsautomaten (FSM)

- Vorsicht beim Abweichen von Idiomen
  - Wird möglicherweise noch richtig simuliert
  - Könnte aber fehlerhaft synthetisiert werden

➔ Haltet Euch an die Konventionen!

# `always` / `always_ff`-Anweisung

TECHNISCHE UNIVERSITÄT DARMSTADT

## Allgemeiner Aufbau:

**Verilog**

```
always @ (sensitivity list)
  statement;
```

**SystemVerilog**

```
always_ff @ (sensitivity list)
        statement;
```

**Interpretation**:

Wenn sich die in der `sensitivity list` aufgezählten Werte ändern, wird die Anweisung `statement` ausgeführt.

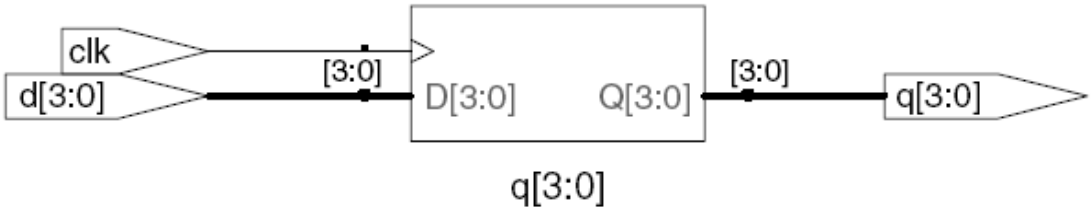
Werte: In der Regel Signale, manchmal noch erweitert

# D Flip-Flop

```
module flop (input  logic clk,
             input  logic[3:0] d,
             output logic[3:0] q);

  always_ff @ (posedge clk)
    q <= d;                    // gelesen als "q übernimmt d"

endmodule
```




Verilog: Jedes Signal, an das innerhalb von einer `always`-Anweisung zugewiesen wird, muss als `reg` deklariert sein

- Im Beispiel: `q`

Wichtig: So ein Signal wird bei der Synthese nicht zwangsläufig in ein Hardware-Register abgebildet!

# Rücksetzbares D Flip-Flop

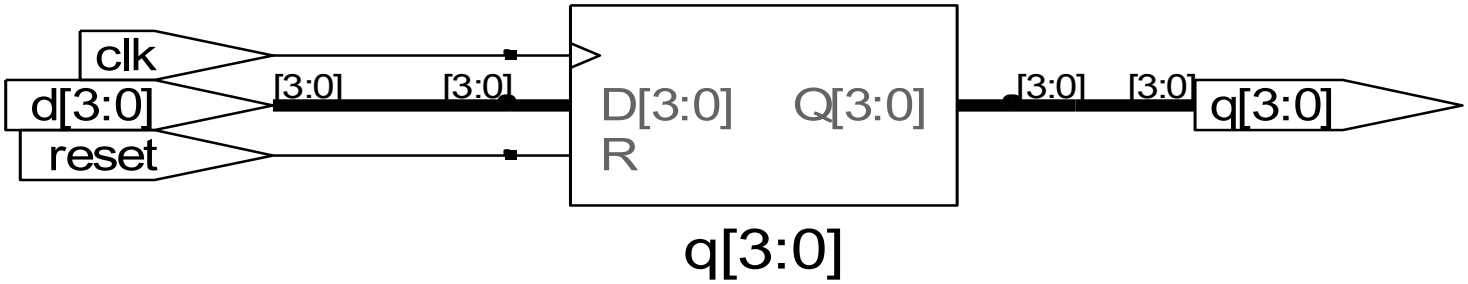
TECHNISCHE UNIVERSITÄT DARMSTADT

```
module flopr (input logic clk,
              input logic reset,
              input logic[3:0] d,
              output logic[3:0] q);

  // synchroner Reset
  always_ff @ (posedge clk)
    if (reset) q <= 4'b0;
    else       q <= d;

endmodule
```

LEHRE WIKI TEST IN DREI FOLIEN

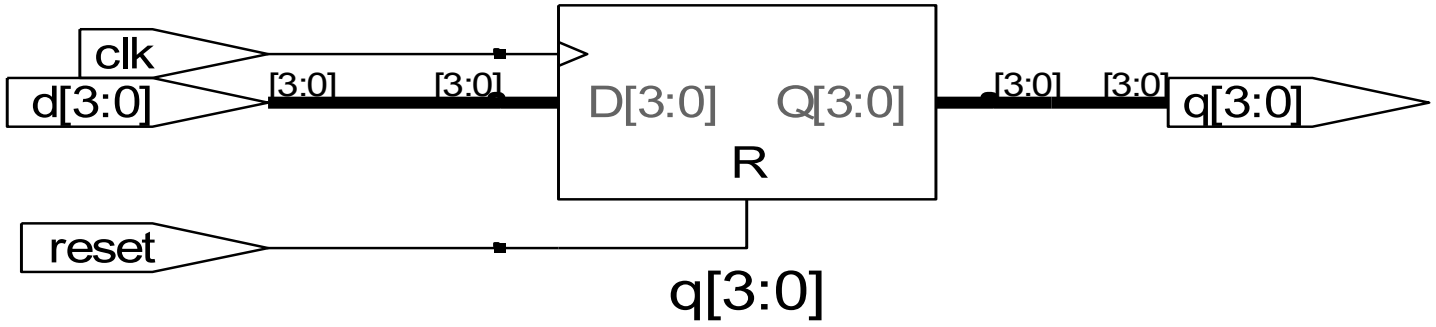


# Rücksetzbares D Flip-Flop

```
module flopr (input logic clk,
              input logic reset,
              input logic [3:0] d,
              output logic [3:0] q);

  // asynchroner Reset
  always_ff @ (posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else       q <= d;

endmodule
```



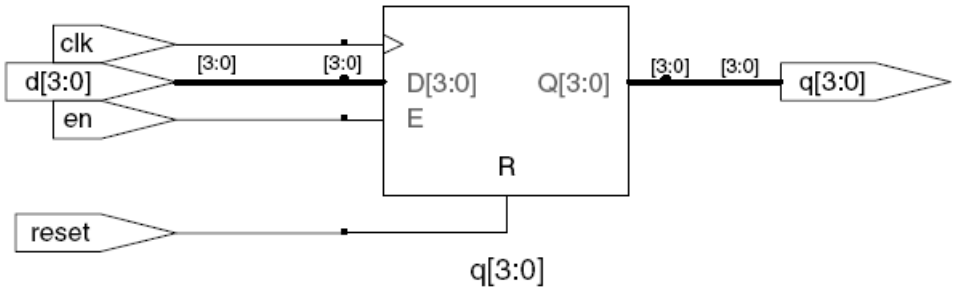
# Rücksetzbares D Flip-Flop mit Taktfreigabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module flopren (input logic clk,
                input logic reset,
                input logic en,
                input logic[3:0] d,
                output logic[3:0] q);

  // asynchroner Reset mit Clock Enable
  always_ff @ (posedge clk, posedge reset)
    if      (reset) q <= 4'b0;
    else if (en)    q <= d;

endmodule
```



# LEHRE WIKI FRAGE

- Bitte jetzt auf LEHRE WIKI eine Frage beantworten!

# Always_latch / always_comb-Anweisung

TECHNISCHE UNIVERSITÄT DARMSTADT

**Allgemeiner Aufbau:**

In SystemVerilog kann die `sensitivity list` automatisch erstellt werden.

```
always_comb begin
 tmp = a & b;
end
```

Logikpegelsensitivität kann durch

```
always_latch
 if (en) tmp = a & b;
```

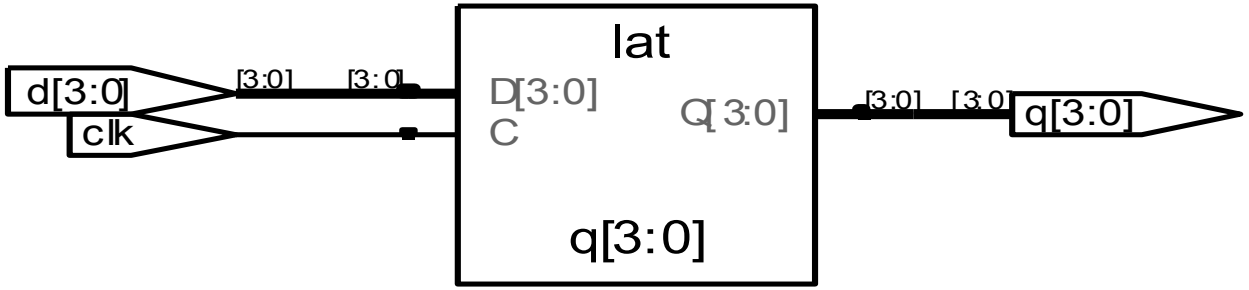
realisiert werden. Wichtig: **Nur** in SystemVerilog!

# Latch

```
module latch (input  logic clk,
              input  logic[3:0] d,
              output logic[3:0] q);

  always_latch
       if (clk) q <= d;

endmodule
```




Achtung: In dieser Veranstaltung werden Latches nur selten (wenn überhaupt) gebraucht werden.

Sollten sie dennoch in einem Syntheseergebnis auftauchen, ist das in der Regel auf Fehler in Ihrer HDL-Beschreibung zurückzuführen (z.B. Abweichen von Idiomen)!

# Weitere Anweisungen zur Verhaltensbeschreibung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Dürfen nur innerhalb von `always_XXXXX`-Anweisungen benutzt werden:
  - `if` / `else`
  - `case`, `casez`

- Erinnerung (nur für Verilog):
  - In Verilog: Alle Zuweisungsziele innerhalb einer `always`-Anweisung müssen als reg deklariert werden!
  - Selbst, wenn sie keine echten Hardware-Register beschreiben
  - In SystemVerilog nicht mehr relevant.

# Kombinatorische Logik als `always_comb`-Block

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module gates (input  logic[3:0] a, b,
              output logic[3:0] y1, y2, y3, y4, y5);
  always_comb        /* wann immer sich irgendein
                     gelesenes Signal ändert */
    begin            // bei mehr als einer Anweisung: begin/end
      y1 = a & b;    // AND
      y2 = a | b;    // OR
      y3 = a ^ b;    // XOR
      y4 = ~(a & b);// NAND
      y5 = ~(a | b); // NOR
    end
endmodule
```

Hätte einfacher durch fünf `assign`-Anweisungen beschrieben werden können.

# Kombinatorische Logik mit `case`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module sevenseg(input  logic[3:0] data, output logic[6:0] segs);
  always_comb// kombinatorische Logik …
    case (data)                //                abc_defg
      0: segs = 7'b111_1110;
      1: segs = 7'b011_0000;
      2: segs = 7'b110_1101;
      3: segs = 7'b111_1001;
      4: segs = 7'b011_0011;
      5: segs = 7'b101_1011;
      6: segs = 7'b101_1111;
      7: segs = 7'b111_0000;
      8: segs = 7'b111_1111;
      9: segs = 7'b111_1011;
      default: segs = 7'b000_0000; // alle Fälle abgedeckt!
    endcase
endmodule
```

So einfach nicht als `assign` formulierbar

# Kombinatorische Logik mit `case`

TECHNISCHE UNIVERSITÄT DARMSTADT

- Um kombinatorische Logik zu beschreiben, muss ein `case`-Block alle Möglichkeiten abdecken
  - Entweder explizit angeben
  - Oder einen `default`-Fall angeben
    - Tritt in Kraft, wenn sonst keine andere Alternative passt
    - Im Beispiel verwendet

# Kombinatorische Logik mit `casez`

```
module priority_casez (input  logic[3:0] a,
                       output logic[3:0] y);

  always_comb begin // kombinatorische Logik …
     casez(a)
      4'b1???: y = 4'b1000;  // ? = don't care
      4'b01??: y = 4'b0100;
      4'b001?: y = 4'b0010;
      4'b0001: y = 4'b0001;
      default: y = 4'b0000;  // alle Fälle
                             // abgedeckt
     endcase
   end
endmodule
```