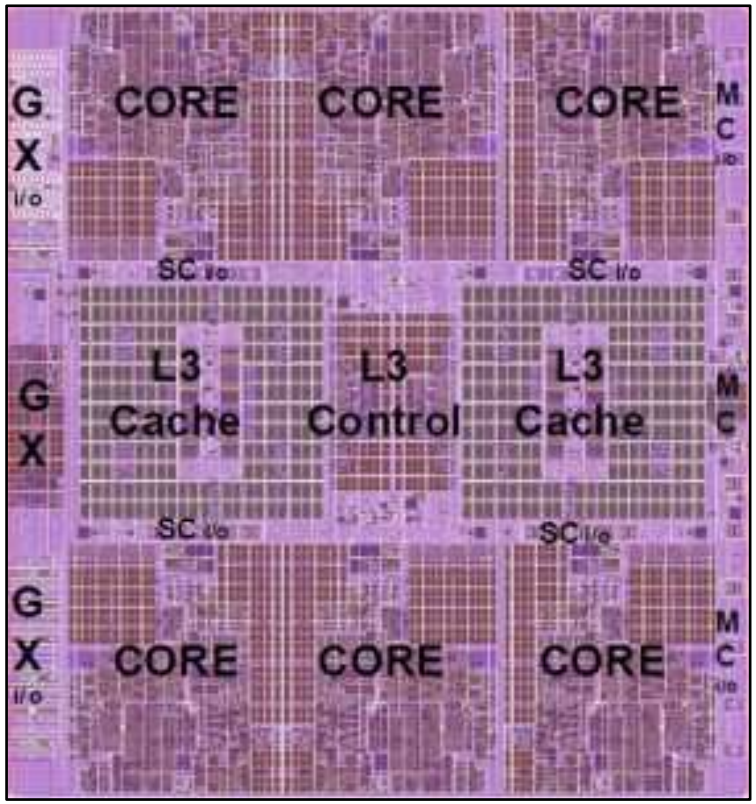
Synthese digitaler Schaltungen

am Beispiel eines synchronen Zählers mit D Flip-Flops

Dr. Matthias Fertig



IBM 6-Kern CPU für zEC12, 2.75 Milliarden Transistoren
Entwickelt für 166ps Taktperiode (6GHz)
L3-Bandbreite bis zu 2x80GB/s
32nm CMOS SOI Technologie
2012

Inhalt

- Hardwaresynthese
- Der Syntheseprozess
  - High-Level Synthese
  - Logik-Synthese
  - Layout-Synthese
- Praktisches Beispiel
  4-Bit Zähler
- IEEE Standard für VHDL Synthese

Hardwaresynthese

Generation 0 (1970-1980)
- Erste Arbeiten (z.B. C. Mead & L. Conway, 1979)
- Grundlagenarbeiten ohne industriellen Einsatz
- Erste HDLs (z.B. ISPS) und „Silicon Compiler“

Generation 1 (1980-1990)
- Grundlagenarbeiten zur High-Level Synthese (z.B. Paulin & Knight, Gajski et. al.)
- Anwendungsspezifische HDL für DSPs (Silage)

Generation 2 (1990-2003)
- HDL für RTL-Synthese (z.B. Verilog, VHDL)
- Erste kommerzielle Softwarepakete (Cadence, Synopsis, Mentor Graphics)

Generation 3 (2004-2009)
- C/C++-basierte Hardwarebeschreibung (z.B. SystemC)
- Anwendungsoptimiert (ASIC, FPGA, CPLD)*

Generation 4 (2010-heute)
- Large-Block Synthese (LBS)
- Single- und Multicycle

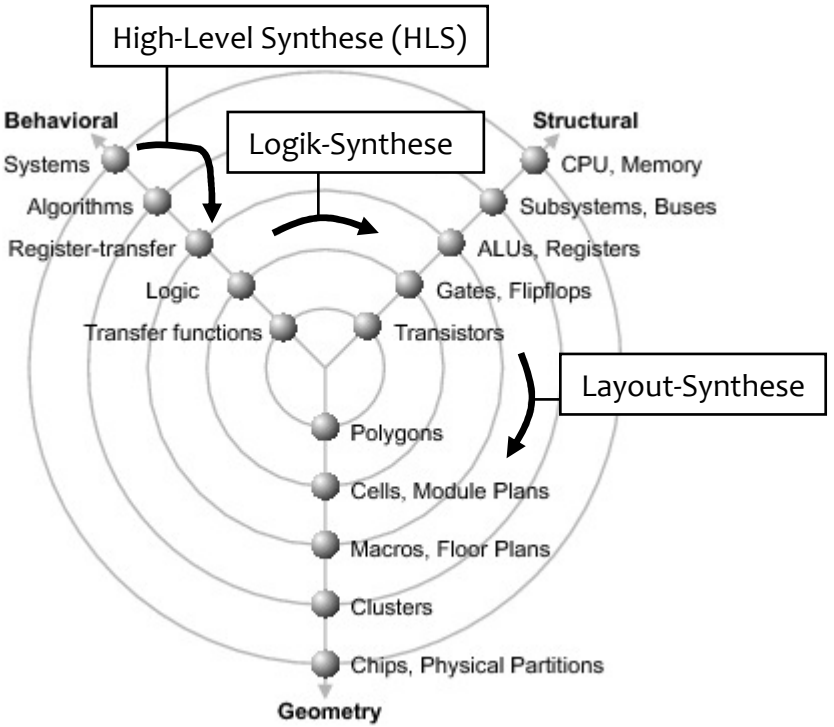
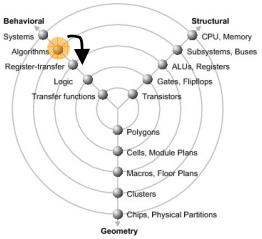


Abb.1 Hierarchienmodell der Hardwarebeschreibung nach Gajski und Kuhn (Y-Chart)

*ASIC - Application Specific Integrated Circuit
FPGA – Field Programmable Gate Array
CPLD – Complex Programmable Logic Device

High-Level Synthese

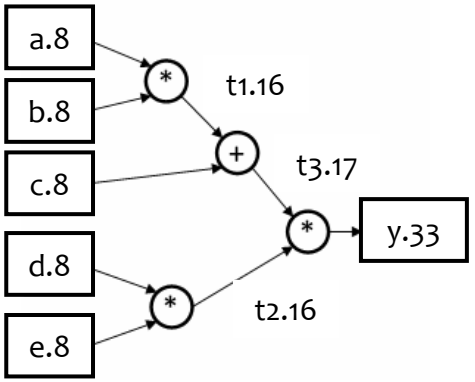


Ziel: **Register-Transfer-Level Beschreibung eines Verhaltens unter der Einhaltung von Randbedingungen.**

1. „Lexical Processing“
   - Umsetzen der HDL in internes Format zur weiteren Bearbeitung
2. „Algorithm Optimization“
   - Reduktion von identischen Teilausdrücken
   - Ersetzung von Konstanten
   - Auflösen von Schleifen
3. „Control-/Dataflow analysis“
   - Prüfung von Datenabhängigkeiten
   - Erstellung eines Kontroll- und Datenflussgraphen (CDFG)
4. „Library processing“
   - Lese Synthese-Bibliothek
   - Identifikation von Funktionsblöcken

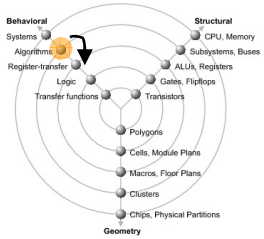
Beispiel:

```
variable a,b,c,d,e : unsigned(7 downto 0);
variable y : unsigned(32 downto 0);
y <= ((a*b)+c)*(d*e);
```

1. y.33 ← ((a.8 * b.8) + c.8) * (d.8 * e.8)
   (<Literal-Name>.<Bitbreite>)
2. -
3. 

4. Mult_8x8_16
   Add_16x16_17
   Mult_20x20_40

High Level Synthese (2)



5. „Resource Allocation“
   - Zuweisung von Ressourcen gemäß dem CDFG und der verfügbaren Funktionsblöcke (Zellen)
6. „Scheduling“
   - Zuweisung von Takten (Single-Cycle vs. Multi-Cycle vs. Pipelined)
   - Optimierung auf Timing, Leistung, Fläche (Sequentielle vs. Parallele Hardware)
7. „Functional binding“
   - Reduktion von Ressourcen
   - Instantiierung
8. „Register binding“
   - Instantiierung von Speicherelementen (z.B. D-Flip-Flop)
9. „Output processing“
   - z.B. Beschreibung des Verhaltens in HDL auf Register Transfer Ebene (RTL)

5. 2x Mult_8x8_16
   1x Add_16x16_17
   1x Mult_20x20_40

6.

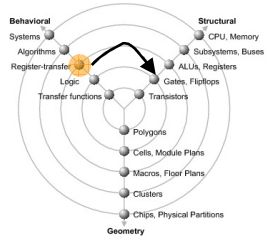
OP	#	C1	C2	C3
*16	2 1	a*b d*e	d*e	
+17	1		t1+c	
*40	1			t2*t3

7.

OP	#	C1	C2	C3
*16	0			
+17	1		t1+c	
*40	1	a*b	d*e	t2*t3

1. **Wieviele Speicherbits sind nötig ?**
2. **Benötigt eine Pipeline mehr oder weniger Speicher?**

Logik-Synthese



Ziel: Optimierung und Abbildung einer RTL-Beschreibung auf Gatterinstanzen einer Technologie unter Randbedingungen.

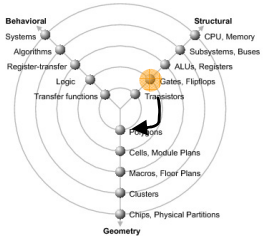
1. „Register Transfer Level (RTL) HDL“
   - Datenfluss mit Finite State Machine
   - Finite State Machine (FSM)
   - Transfer Funktion (KV-Diagramm)
2. „HDL Analysis“
   - Auflösen von Konstanten, Schleifen und arithmetischer Operatoren
   - Identifikation von Speicher (D-Flip-Flops)
3. „Boolean Minimization/Optimization“
   - Optimierung auf Fläche, Timing, Leistung
4. „Binding and Optimization on Gates“
   - Abbilden auf Standardzellen
   - Erstellen der Gatter-Netzliste
5. „Local optimization on nets“
   - Randbedingungen für I/Os (arrival times, slew rates, fan-out)
   - Elektrische Optimiere (z.B. RC-delay)

Beispiel:

1. signal A,B : bit;
   constant C : bit := 1;
   y <= C and (not A and B) or (not B and A);
2. y <= 1 and (not A and B) or (not B and A);
3. y <= (not A and B) or (not B and A);
4. 1x XOR
5. Annahme: a hat schlechte slew rate ($\Delta V/\Delta t$)
   2x Inverter für Eingang A

1. **Wie sieht die Netzliste aus ?**

Layout-Synthese



Ziel: Abbilden der Gatter-Netzliste auf Halbleiterstrukturen. Platzieren und Verdrahten von Instanzen.

1. „Partitioning“
   - Teile das Problem in Teilprobleme (Divide and Conquer)
   - Minimiere die Anzahl der partitions-überschreitenden Netze
2. „Placement“
   - Constraint-File (z.B. I/O Positionen, Cell-Margin für späteres Routing)
   - Minimiere die Fläche
   - Vermeide Überlappungen
3. „Routing“
   - „global routing“ (Planung der traversierten Regionen)
   - „detailed routing“ (Tracks and Vias)
4. „Compaction“
   - Optimierung der Abstände, ohne die Layout-Regeln der Technologie zu verletzen (z.B. Mindestbreiten, Abstände)

Praktisches Beispiel

Synthese eines synchronen digitalen Zählers

CAD-Software:

Alliance CAD (freie Software)

Pierre et Marie Curie Universität, Paris

Source Code verfügbar unter GNU General Public Licence (GPL)

http://www-oc.lip6.fr/recherche/cian/alliance/

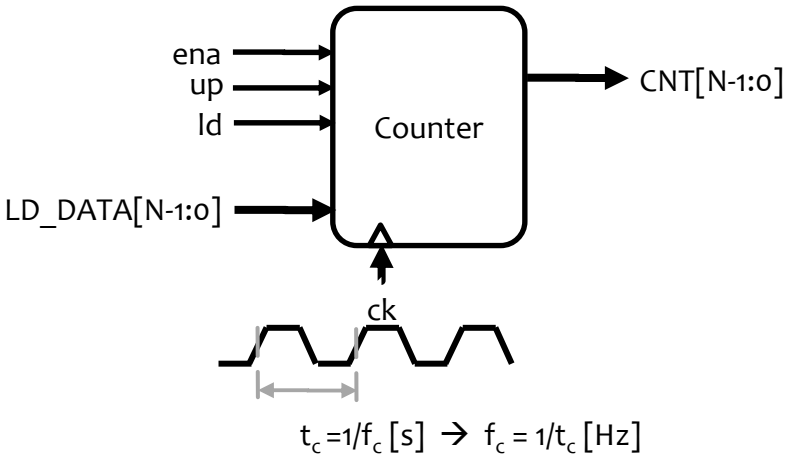
erfolgreich verwendet für

875k Zellen superscalaren Mikroprozessor

400k Zellen IEEE Gigabit High Speed Link (HSL) Router

Spezifikation eines N-Bit Zählers

- Eingängssignale (Bitbreite)
  - ck (1)
  - up (1)
  - ena (1)
  - ld (1)
  - LD_DATA (N)
- Ausgangssignale
  - CNT (N)
- Blockdiagramm

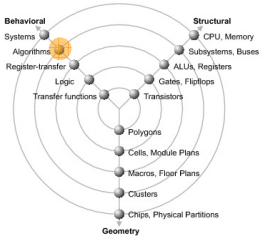


- Beschreibung mit Pseudocode

```
IF ( RISING_EDGE(CK)  )
THEN
   IF ( TRUE == LD ) THEN
      CNT ← LD_DATA
   ELSE
      IF( TRUE == ENA ) THEN
        CNT ← CNT +  UP
      END
   END
END
```

1. Wie groß ist t_c bei 1GHz und 5GHz?
2. Ist ENA synchron oder asynchron?
3. Wie ändert sich der Pseudocode bei einem asynchronen ENA?

HDL-Beschreibung



Algorithmische Beschreibung in VHDL

emacs@t41p

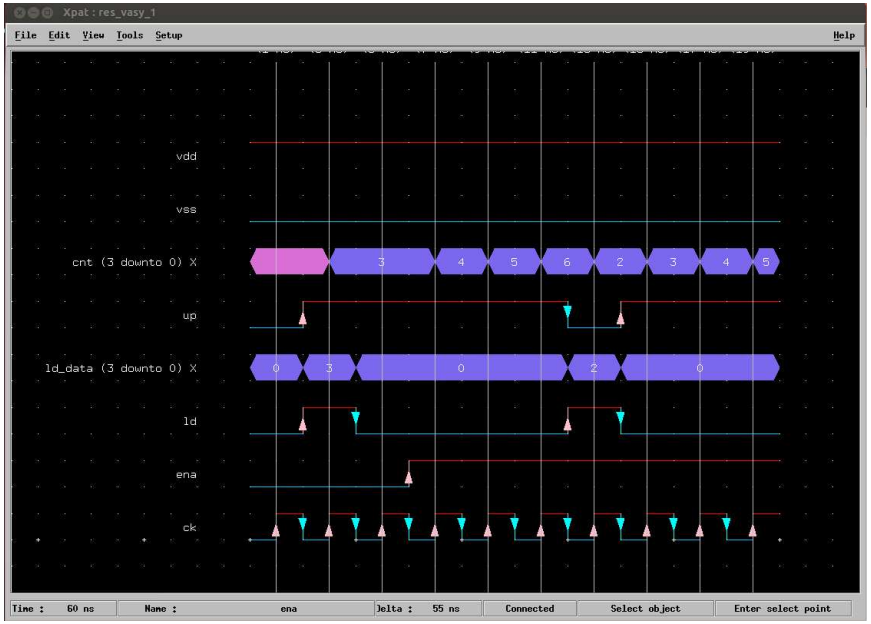
File Edit Options Buffers Tools VHDL Help

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity counter4 is
    generic (
      WIDTH: integer:=4
      );
    port (
      ck : in std_logic;
      ena : in std_logic;
      ld : in std_logic;
      ld_data : in std_logic_vector(WIDTH-1 downto 0);
      up : in std_logic;
      cnt : out std_logic_vector(WIDTH-1 downto 0)
      );
end counter4;

architecture beh of counter4 is
  signal up_s : unsigned;
begin
  up_s <= unsigned(up);

  process(ck)
    variable cnt_v : unsigned(WIDTH-1 downto 0);
  begin
    if not ck'stable and ck='1' then
      if ld='1' then
        cnt_v :=  unsigned(ld_data);
      elsif ena='1' then
        cnt_v :=  cnt_v + up_s;
      end if;
    end if;
    cnt <= std_logic_vector(cnt_v);
  end process;

end beh;
```

--:--- counter4.vhdl All L21 (VHDL/es)----------

menu-bar buffer C-d

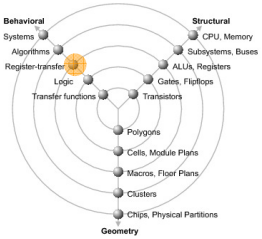
Simulation des Zählers



1. „ld“ aktiv: lade den Wert 3
2. „ena“ deaktiv: kein Inkrement
3. „ena“ aktiv: Zähldauer drei Takte
4. „ena“ aktiv und „ld“ aktiv:
   lade den Wert 2 hat Prioriät!

High-Level Synthese

„Lexical Processing“



Eingabe:

- Algorithmische HDL-Beschreibung

VHDL-Analyse (abh. vom Synthesewerkz.):

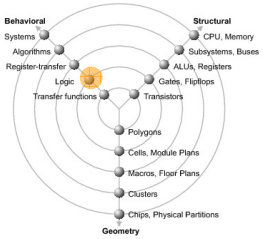
- Auflösen der GENERIC Statements
- Eingänge VDD und VSS
- Umsetzung von STD_LOGIC auf BIT
- Auflösen der Addition in bool‘sche Gleichungen
- Identifikation/Ableiten von Speicherelementen

Ausgabe:

- VHDL auf Register-Transfer-Ebene (RTL)

Wie leitet das Synthesetool Speicherelemente ab ?

```
emacs@t41p
File  Edit  Options  Buffers  Tools  Help
--
-- Generated by VASY
--
ENTITY counter4 IS
PORT(
  ck     : IN BIT;
  rst    : IN BIT;
  ena    : IN BIT;
  ld     : IN BIT;
  ld_data        : IN BIT_VECTOR(3 DOWNTO 0);
  up     : IN BIT;
  cnt    : OUT BIT_VECTOR(3 DOWNTO 0);
  vdd    : IN BIT;
  vss    : IN BIT
);
END counter4;

ARCHITECTURE VBE OF counter4 IS

  SIGNAL rtlsum_0       : BIT_VECTOR(3 DOWNTO 0);
  SIGNAL rtlcarry_0     : BIT_VECTOR(3 DOWNTO 0);
  SIGNAL rtlexts_0      : BIT_VECTOR(3 DOWNTO 0);
  SIGNAL up_s    : BIT_VECTOR(0 TO 0);
  SIGNAL p26_2_cnt_v    : REG_VECTOR(3 DOWNTO 0) REGISTER;
  SIGNAL p26_2_reddef_6 : BIT_VECTOR(3 DOWNTO 0);
  SIGNAL p26_2_reddef_5 : BIT_VECTOR(3 DOWNTO 0);
  SIGNAL p26_2_reddef_4 : BIT_VECTOR(3 DOWNTO 0);
BEGIN

  rtlcarry_0(0) <= '0';
  rtlsum_0 <= ((p26_2_cnt_v XOR rtlexts_0) XOR rtlcarry_0);
  rtlcarry_0(3 downto 1) <= (((p26_2_cnt_v(2 downto 0) AND rtlexts_0(2 downto 0)) OR (p2→
 AND rtlcarry_0(2 downto 0))) OR (rtlexts_0(2 downto 0) AND rtlcarry_0(2 downto 0)
));
  rtlexts_0 <= ("000" & up_s(0));
  cnt <= p26_2_cnt_v;
  LABEL0 : BLOCK  ((ck = '1') AND NOT(ck'STABLE) )
  BEGIN
    p26_2_cnt_v <= GUARDED ((ld AND ((rst AND p26_2_reddef_6(3)) OR (NOT(rst) AND p26_2→
 (NOT(ld) AND ((ena AND NOT(rst) AND p26_2_reddef_4(3)) OR (NOT(ena AND NOT(rst)
) AND p26_2_reddef_6(3))))) & ((ld AND ((rst AND p26_2_reddef_6(2)) OR (NOT(rst)
 AND p26_2_reddef_5(2)))) OR (NOT(ld) AND ((ena AND NOT(rst) AND p26_2_reddef_4(2)
--:---  counter4.vbe   Top L9     (Fundamental)----------------------------------------
Truncate long lines enabled
```



Logik Synthese

„Boolean Minimization/Optimization“

Eingabe:

- VHDL auf Register-Transfer-Ebene (RTL)
- Optimierungsgrad
  - Fläche
  - Laufzeit

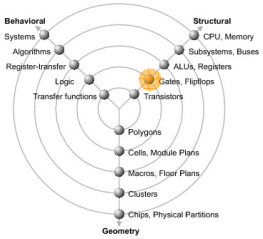
Logische Optimierung:

- Logischen Tiefe
- Gatteranzahl
- Literale
- Schaltkreisfläche

Ausgabe:

- Optimierte VHDL
  - Sequentielle Logik (FSM)
  - „random logic“ („AND-OR“ Logik)

```
mfertig@t41p: ~/vhdl/AllianceCAD/counter4-opt.l

                        BOOlean Minimization

            Alliance CAD System 5.0 20110203,  boom 5.0
            Copyright (c) 2000-2014,     ASIM/LIP6/UPMC
            Author(s):                Ludovic Jacomme
            E-mail       : alliance-users@asim.lip6.fr

      --> Parse BEH file counter4.vbe

      --> Check figure counter4

      --> Parse parameter file counter4.boom

      --> Optimization parameters
          Algorithm : simulated annealing
          Keep aux  : no
          Area      :  50 %
          Delay     :  50 %
          Level     :   3

      --> Initial cost
          Surface  : 100250
          Depth    : 14
          Literals : 79

      --> Translate Abl to Bdd
          Total Bdd nodes 58

      --> Optimization %

      --> Final cost
          Surface  : 43750
          Depth    : 7
          Literals : 33

      --> Post treat figure counter4

      --> Drive BEH file counter4_o

:
```



Logik Synthese

„Functional Binding“

Eingabe:

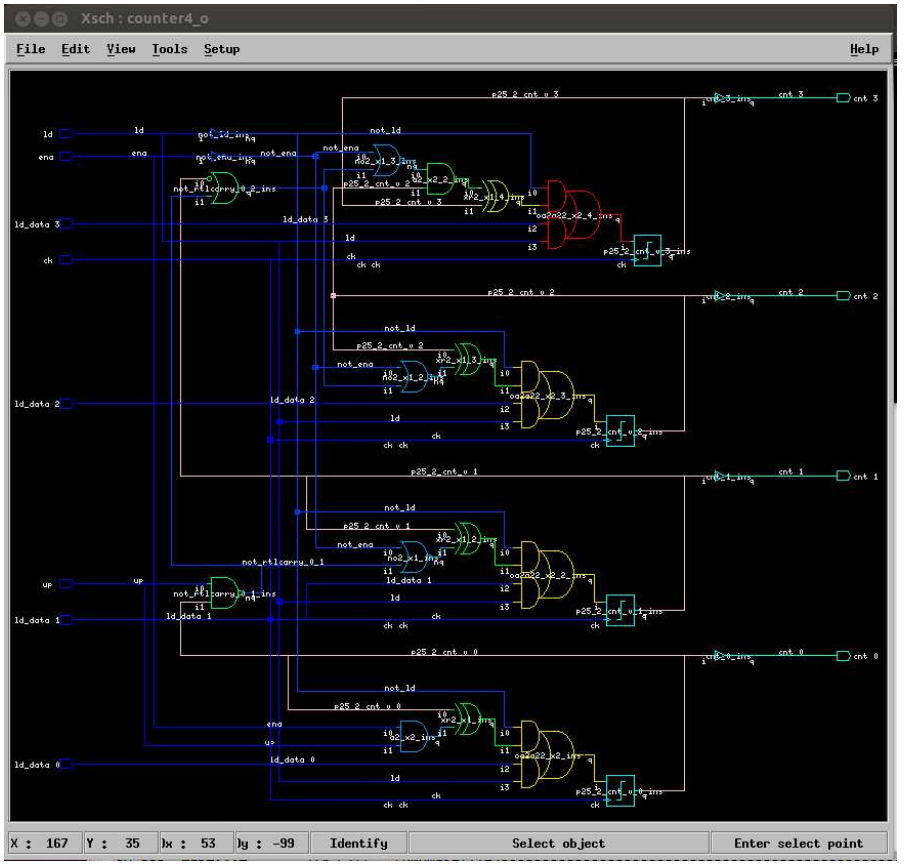
- Optimierte VHDL
- Technologie-Bibliothek
  http://www.vlsitechnology.org/html/cells/sxlib013/lib_gif_index.html

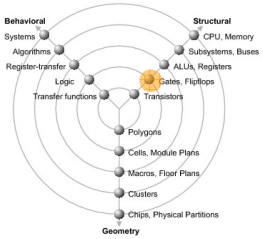
Synthese der Gatter-Netzliste:

- Instantiierung der Gatter- und Speicherelemente
- Berechnung der Durchlaufzeiten aus der Summe der Gatterlaufzeiten

Ausgabe:

- Gatter-Netzliste





Logik Synthese

„Local Optimization on Nets“

Eingabe:

- Synthetisierte Gatter-Netzliste
- Bedingungen für Ein- und Ausgänge
  - Signal Ankunfts- u. Anstiegszeiten
  - Fan-out u. kapazitive Lasten
- Technologie-Bibliothek
  http://www.vlsitechnology.org/html/cells/sxlib013/lib_gif_index.html

Elektrische Optimierung:

- Einfügen von Invertern (Buffer) und Ersetzen von Logik-Gattern zur Optimierung der Signalanstiegszeiten (RC-delays)

Ausgabe:

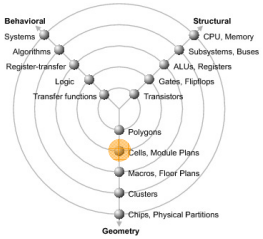
- Optimierte Gatter-Netzliste

```
mfertig@t41p: ~/vhdl/AllianceCAD/counter4-opt.beh

                    Local optimization on Nets

          Alliance CAD System 5.0 20110203, loon 5.0 [2003/12/07]
          Copyright (c) 2000-2014,                ASIM/LIP6/UPMC
          Author(s):                            Fran<E7>ois Donnet
          E-mail        :            alliance-users@asim.lip6.fr

      MBK_IN_LO      : vst
      MBK_OUT_LO     : vst
      MBK_TARGET_LIB  : /usr/share/alliance/cells/sxlib

Reading parameter file 'counter4.lax'...
50% area - 50% delay optimization
Reading file 'counter4_o.vst'...
Controlling file 'counter4.lax'...
Reading lib '/usr/share/alliance/cells/sxlib'...
Capacitances on file 'counter4_o.vst'...
Delays on file 'counter4_o.vst'...2610 ps
Area on file 'counter4_o.vst'...49250 lamda<B2> (with over-cell routing)
Details...
       buf_x2: 4 (8%)
       sff1_x4: 4 (36%)
       oa2a22_x2: 4 (18%)
       xr2_x1: 4 (18%)
       no2_x1: 3 (6%)
       a2_x2: 2 (5%)
       inv_x2: 2 (3%)
       na2_x1: 1 (2%)
       on12_x1: 1 (2%)
       Total: 25
Worst RC on file 'counter4_o.vst'...78 ps
Inserting buffers on critical path for file 'counter4.vst'...2 buffers inserted -> 2572 ps
Improving RC on critical path for file 'counter4.vst'...2541 ps
Improving all RC for file 'counter4.vst'...
Worst RC on file 'counter4.vst'...78 ps
Area on file 'counter4.vst'...51250 lamda<B2> (with over-cell routing)
Details...
       buf_x2: 6 (11%)
       sff1_x4: 4 (35%)
       oa2a22_x2: 4 (17%)
       xr2_x1: 4 (17%)
       no2_x1: 3 (5%)
:
```

```
Identify
Instance :
> Name     oa2a22_x2_4_ins
> Model    oa2a22_x2

[2541 ps]

Close
```

Welche Bedeutung hat die gezeigte Signallaufzeit für den synchronen Zähler?

Layout Synthese



„Placement“

Eingabe:

- Optimierte Gatter-Netzliste
- Position der Ein- und Ausgänge
- Partitionierung oder Fläche
- Abstand („Margin“) für Standardzellen

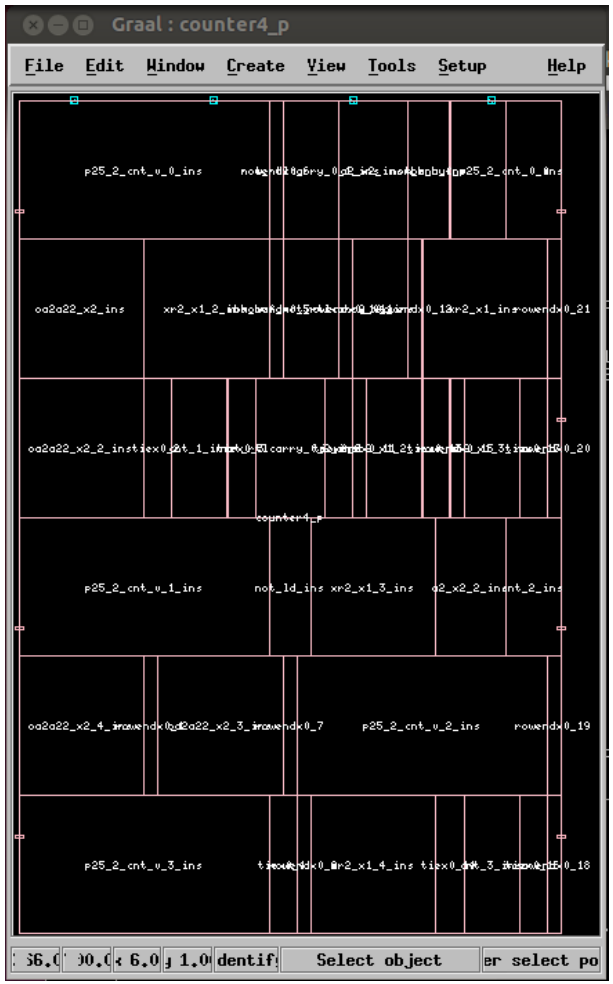
```
## I/O Placement
TOP (
# IOs are ordered
# from left to right
   (IOPIN ena.0 );
   SPACE 10;
   (IOPIN up.0 );
   SPACE 10;
   (IOPIN ld.0 );
   SPACE 10;
   (IOPIN ck.0 );
)
LEFT (
# IOs are ordered
# from top to bottom
   (IOPIN ld_data(3).0 );
   (IOPIN ld_data(2).0 );
   (IOPIN ld_data(1).0 );
   (IOPIN ld_data(0).0 );
)
RIGHT (
# IOs are ordered
# from top to bottom
   (IOPIN cnt(3).0 );
   (IOPIN cnt(2).0 );
   (IOPIN cnt(1).0 );
   (IOPIN cnt(0).0 );
)


OCP_MARGIN = 0.15
OCP_ROWS = 6
```

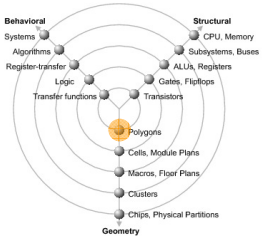
Platzierung (Placement):

- Minimiere Schaltkreis-Fläche
- Vermeide Überlappungen der Instanzen

Ausgabe:

- Layout der beim „Functional Binding“ ausgewählten Gatter





Layout Synthese

„Routing“

Eingabe:

- Platziertes Layout
- Design-Regeln der Technologie
- Anzahl der Verdrahtungsebenen

Routing:

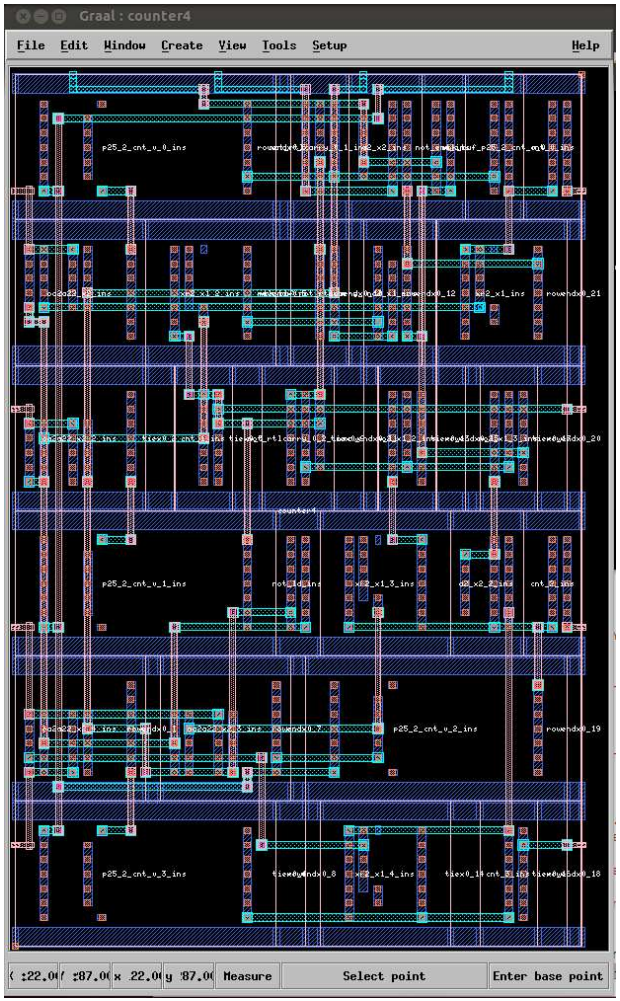
- Verdrahtung des platzierten Layouts

Ausgabe:

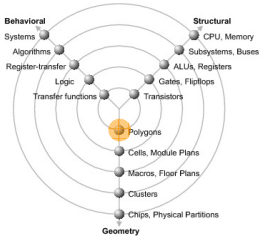
- Verdrahtetes Layout

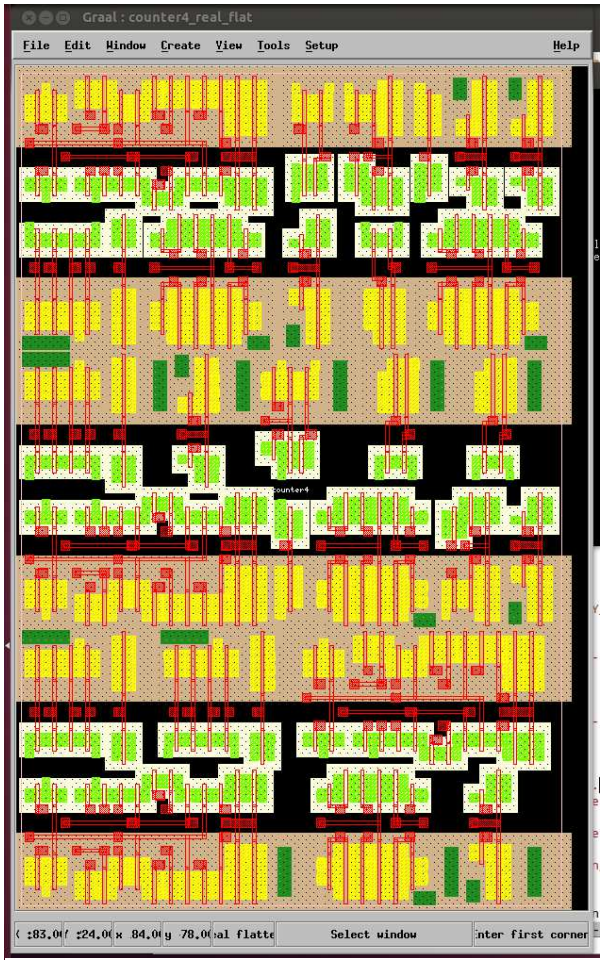
Abschliessende Verifikation:

- Layout versus Schematic (LVS)
- Design Rule Check (DRC)



Layout des 4-Bit Zählers





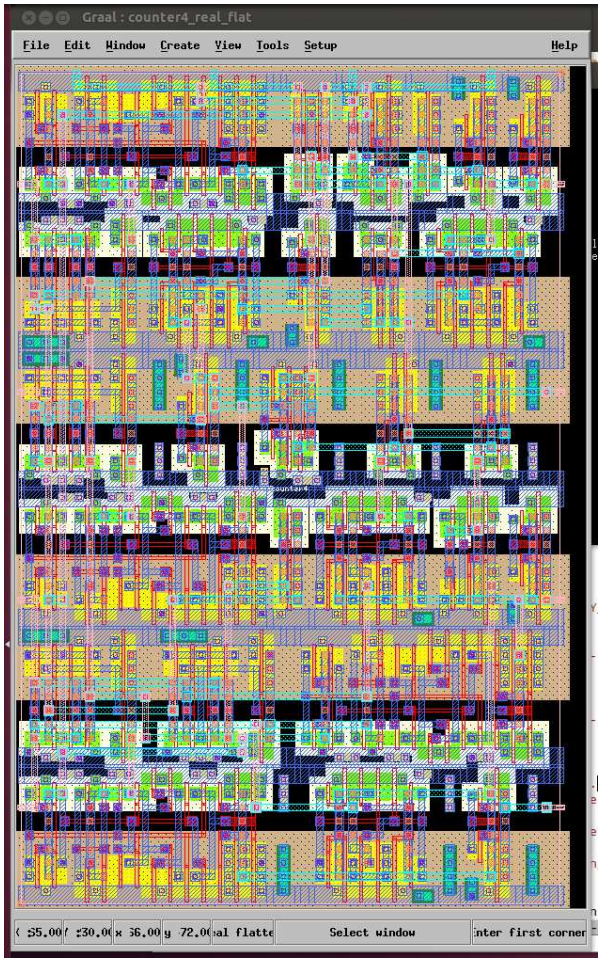
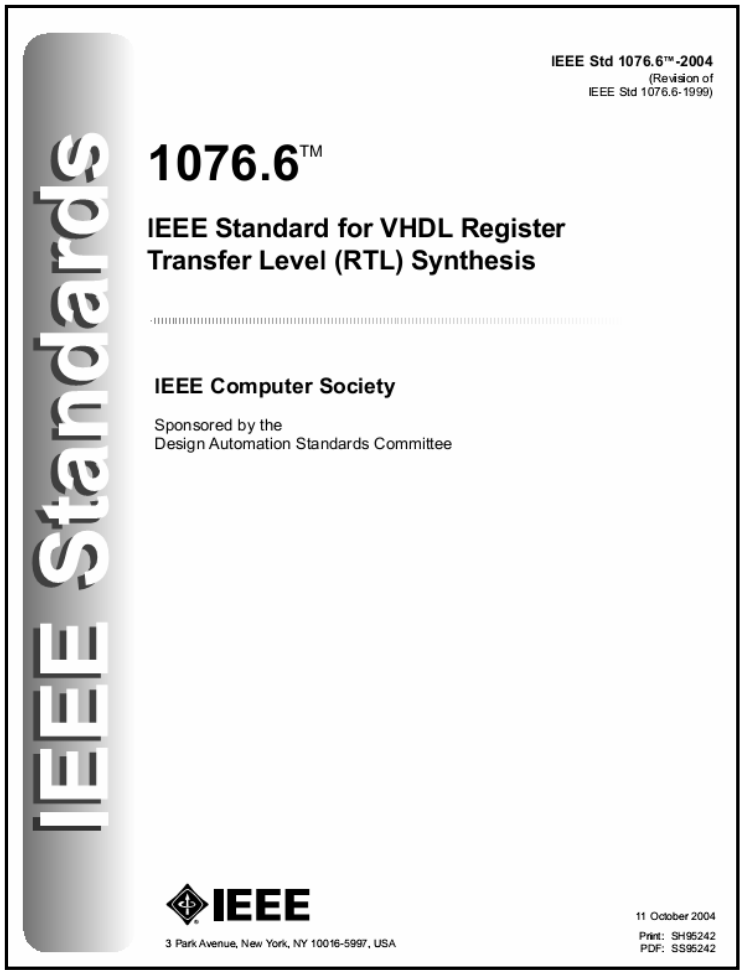


Abb.1 CMOS-Transistoren

Abb.2 Layout des 4-Bit Zählers

IEEE Standard für Synthese



in Kapitel 4:

- Synthetisierbare Standard-Typen

in Kapitel 6:

- flankengetriggerte sequentielle Logik
- levelsensitive sequentielle Logik
- RAM und ROM

in Kapitel 7:

- HDL-Pragmas zur Kontrolle des Synthesetools

in Kapitel 8:

- Syntax

Literatur

[1] **The Designers Guide to VHDL**, Peter J. Asheden,
Morgan Kaufmann Publishers, ISBN-1-55860-674-2

[2] **Digital Design Principles & Practices**, John F. Wakerly,
Prentice Hall, ISBN 0-13-089896-1
(enthält Xilinx Student Edition für FPGA/CPLD designs)

[3] **Algorithms for VLSI Design Automation**, Sabih H. Gerez
Wiley Verlag, 2004, ISBN 0-471-98489-2

[4] **CMOS VLSI Design: A Circuits and Systems Perspective**
Addison Wesley Pub Co Inc, 4. Auflage (März 2010)
ISBN-13: 978-0321547743

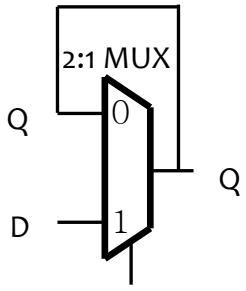
Aktuelle Informationen im Internet:

http://www.vlsitechnology.org/
http://www.edacafe.com
http://www.semiwiki.com/
https://twitter.com/EDANews

Achtung Synthese!

- Latch oder Multiplexer?

```
ALat : process ( ENABLE, D, Q)
begin
  if ENABLE = '1' then
    Q <= D;
  else
    Q <= Q ;
  end if;
end process;  -- ALat
```



wird zum Multiplexer falls

```
attribute COMBINATIONAL of ALat: process is TRUE;
```

- Vermeide „Read before Write“

```
StrangeProc : process (A, B, D)
  variable C : std_logic ;
begin
  Y <= C or D ;
  C := A and B ;
end process ;
```

Ungültiges Statement! Synthesetool sollte eine Fehlermeldung erzeugen!

- Sichere Zustandsautomaten
  - Implementierung des others statements wenn FSM_COMPLETE Attribut TRUE
  - Unerreichbare Zustände erzeugen Fehler falls FSM_COMPLETE Attribut TRUE
  - FSM_STATE kann auch die Werte BINARY, GRAY, ONE_HOT, ONE_COLD annehmen

```
type StateType is (S0, S1, S2, S3, S4);
signal state, next: StateType;
attribute FSM_STATE of state : signal is
   "0000 0011 0110 1100 1001" ;
attribute FSM_COMPLETE of state : signal is TRUE;
. . .
NextStateProc : process
begin
  wait until Clk = '1' ;
  if nReset = '0' then
    state <= S0
  else
    case state is
      when S0 =>          state <= S1;
      when S1 =>          state <= S2;
      when S2 =>          state <= S3;
      when S3 =>          state <= S4;
      when S4 =>          state <= S0;
      when others =>      state <= S0;
    end case;
  end if ;
end process;
```

Sequentielle Logik

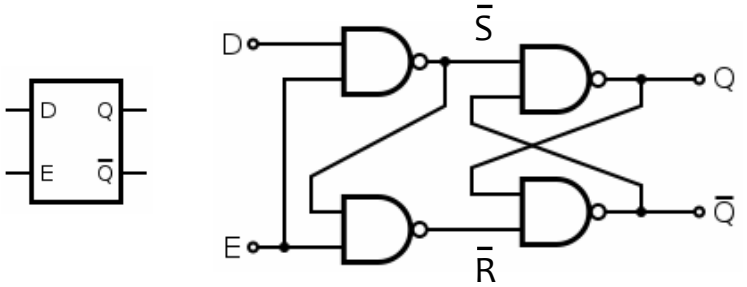
Endliche Zustandsautomaten (FSMs)

- Simple Moore
- Moore
- Mealy

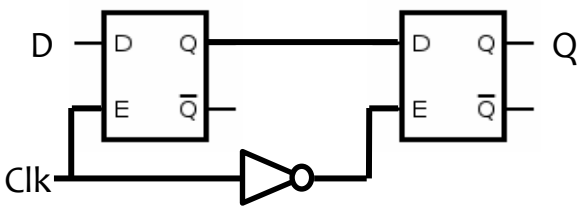
Aufbau sequentieller Logik

- Eingangssignale
- Ausgangssignale
- Kombinatorische Logik
- Speicher

- Digitale Speicherelemente
  - Gated Latch (level-sensitiv)



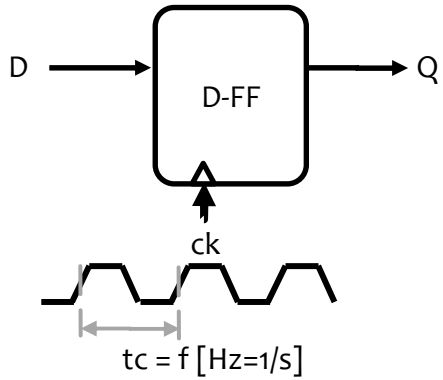
  - Master-Slave D Flip-Flop (flanken-getriggert)



1. **Warum ist der Zustand $\overline{S}$=$\overline{R}$=0 verboten?**
2. **Ist das gezeigte Flip-Flop positiv oder negativ flankengetriggert ?**

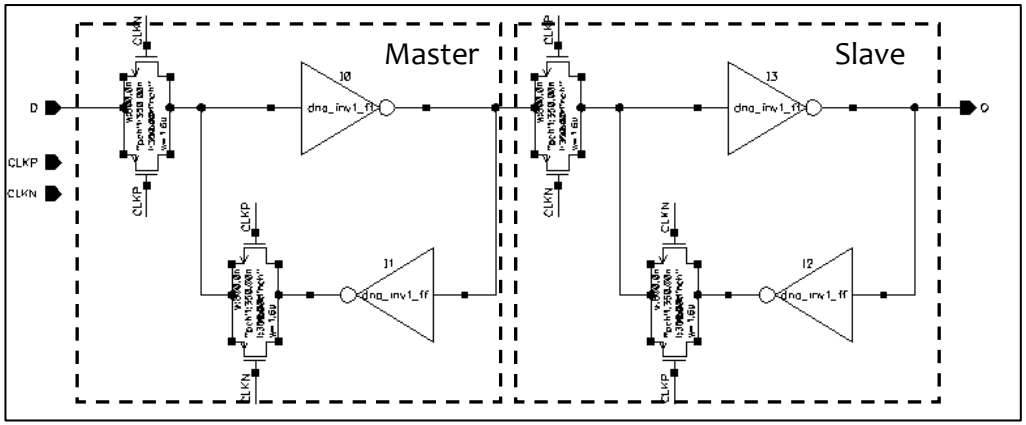
D Flip-Flop

- Eingängssignale
  - Ck (1)
  - D (1)
- Ausgangssignale
  - Q (1)
- Blockdiagramm



- Verhaltensbasierte Beschreibung (Pseudocode)

```
IF ( ck=1 and not ck‘stable  )
THEN
            D=Q
END
```



Master-Slave D Flip-Flop in der Anwendung (z.B. IBM Microprocessors)