

# VHDL und FPGAs – Eine Einführung

Jörg Bornschein [joerg@capsec.org](mailto:joerg@capsec.org)

LABOR

23. Februar 2008



**AND**



x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

**OR**



x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

**NAND**  
(Not AND)



x	y	f
0	0	1
0	1	1
1	0	1
1	1	0

**NOR**  
(Not OR)



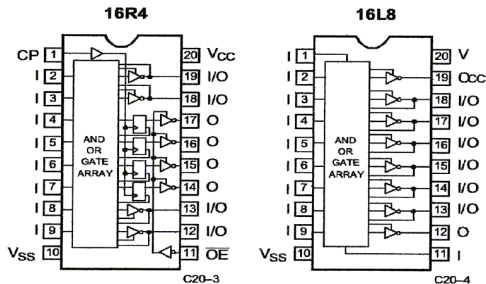
x	y	f
0	0	1
0	1	0
1	0	0
1	1	0



# Der Plan



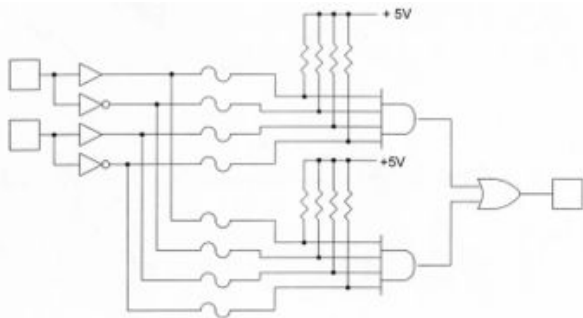
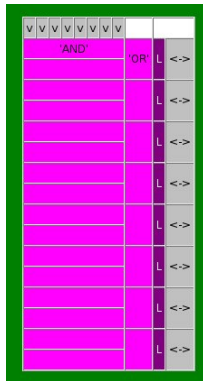
# PAL (Programmable Array Logic)



- typ. 10+10 Ein-/Ausgänge pro Bauteil
- Ausgänge sind Logikfunktion der Eingänge
- Fuse-basiert => nur einmal programmierbar



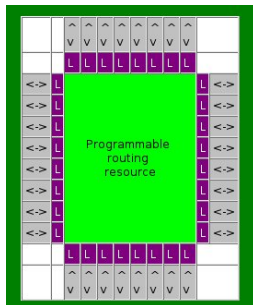
# PAL (Programmable Array Logic)



Simplified programmable logic device



# CPLD (Complex Programmable Logic Device)

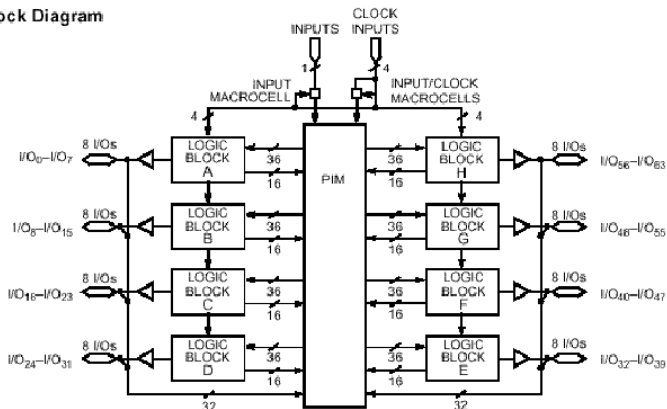


- typ. 100 Ein-/Ausgänge
- typ. 100 MHz Taktfrequenz
- reprogrammierbar

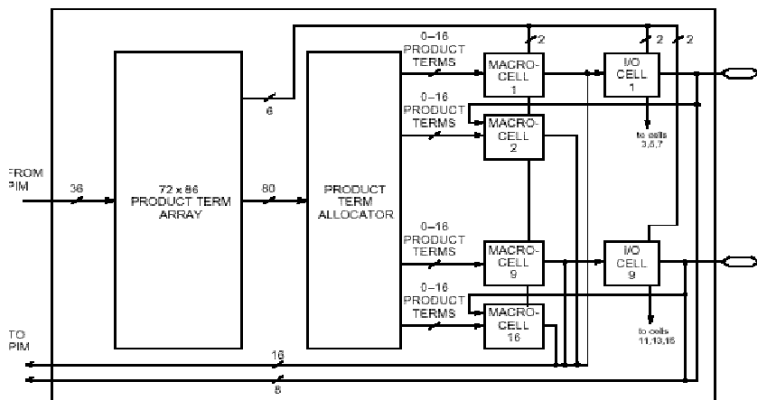


## CPLD (Cypress 374i)

### Logic Block Diagram



# CPLD (Cypress 374i)



flash370i-3

Figure 3. Logic Block for CY7C372i and CY7C374i (Register Intensive)





# CPLD (Cypress 374i)

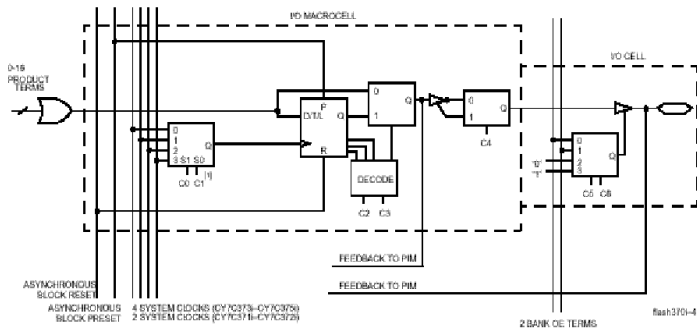
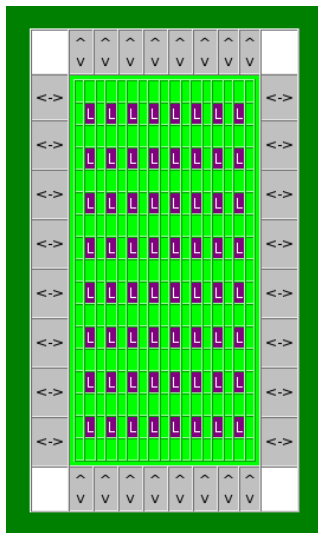


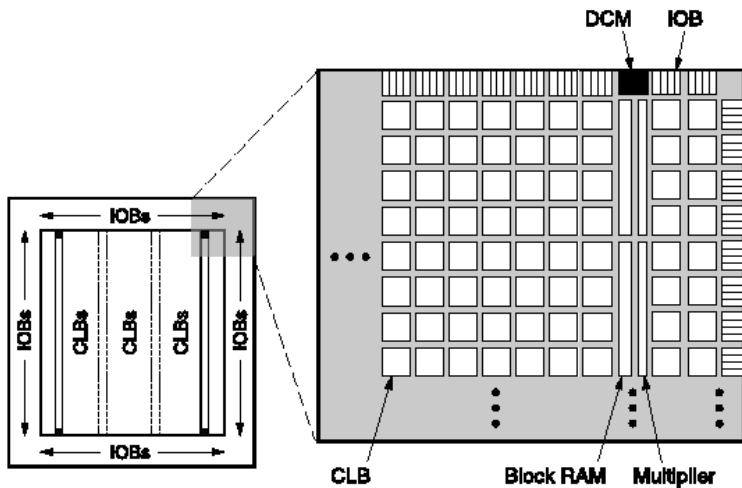
Figure 6. I/O Macrocell



# FPGA (Field Programmable Gate Array)



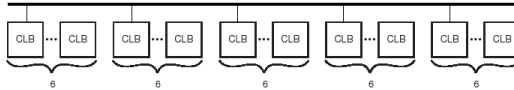
# FPGA (Field Programmable Gate Array)



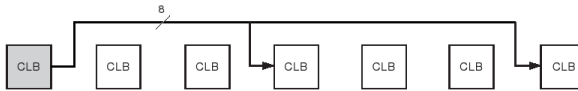
DS099-1\_01\_032703



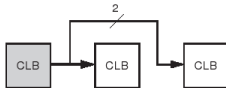
# FPGA (Field Programmable Gate Array)



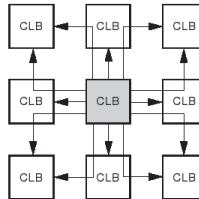
(a) Long Line



(b) Hex Line



(c) Double Line



(d) Direct Lines



## Example

```
-----  
-- Entity declaration  
ENTITY HalbAddierer  
    Port (  din1    : in  bit;  
           din2    : in  bit;  
           dout    : out bit;  
           cout    : out bit );  
END HalbAddierer;  
  
-----  
-- Implementation  
ARCHITECTURE rtl of HalbAddierer is  
    dout <= din1 XOR din2;  
    cout <= din1 AND DIN2;  
END rtl;
```



- – ist Kommentarzeichen (bis EOL)
- Case inSENSiTiVE
- Zahlen: '\_' ist Tausender-Trennzeichen
- Zahlen: b-adische Darstellung basis#digits#;
- Compiler/Synthesewerkzeuge verwalten Entities in 'Libraries' (default: work)

## Example

```
max <= 16#FF#;      -- 255 Dezimal  
c   <= 299_792_458; -- m/s, Lichtgeschw.
```



## Example

```
ENTITY MeinAddierer
  GENERIC( width : natural := 8 );
  PORT( data1  : in  bit_vector( width-1 downto 0 );
        data2  : in  bit_vector( width-1 downto 0 );
        sum    : out bit_vector( width-1 downto 0 );
        carry  : out bit );
END MeinAddierer;
```

- beschreibt das Interface einer Komponente
- mit Generics parametrisierbar
- Modes: in / out / inout / buffer



## Example

```
ARCHITECTURE gates OF MeinXOR IS
```

```
BEGIN
```

```
    dout = din1 XOR din2;
```

```
END ENTITY;
```

```
ARCHITECTURE TruthTable OF MeinXOR IS
```

```
BEGIN
```

```
    dout <= '0' WHEN din1='0' AND din2='0' ELSE
```

```
        '1' WHEN din1='1' AND din2='0' ELSE
```

```
        '1' WHEN din1='0' AND din2='1' ELSE
```

```
        '0';
```

```
END TruthTable;
```

- beinhaltet die Implementation
- mehrere ARCHITECTUREs möglich





# Concurrent Statements

- Signal Assignment
- When-Else Assignment
- With-Select-When-Assignment
- Instantiation
- Process (Umschliessen sequentielle Anweisungen)

Jedes Statement kann optional ein Label vorangestellt haben. Dies hilft bei der Identifikation einzelner Komponente in der Simulationsphase.



# Signal Assignment mit <=

```
[ label: ] target <= [ delay_mechanism ] waveform ;

delay_mechanism
  transport
  reject time_expression
  inertial

waveform
  waveform_element [, waveform_element]
  unaffected

waveform_element
  value_expression [ after time_expression ]
  null [ after time_expression ]
```

## Example

```
dout1 <= din1 XOR din2;
clk <= '0', '1' after 20 ns;
```



# Conditional Signal Assignment

Conditional Signal Assignment: signal  $\leq$  waveform WHEN boolean ELSE waveform WHEN ...

## Example

```
ENTITY Mux IS
    PORT( d0, d1, d2, d3 : IN  bit;
          sel0, sel1      : IN  bit;
          dout            : OUT bit );
END ENTITY;

ARCHITECTURE rtl OF Mux is
BEGIN
    dout <= d0 WHEN sel1='0' AND sel0='0' ELSE
            d1 WHEN sel1='0' AND sel0='1' ELSE
            d2 WHEN sel1='1' AND sel0='0' ELSE
            d3;
END rtl;
```



# Selected Signal Assignment

Verhaelt sich zu 'Conditional Signal Assingment' wie 'switch-case' zu 'if-then-else'. Die Auswahlfunktion ist nur von einer Expression abhaenig.

## Example

```
WITH sel SELECT          -- sel ist vom typ bit_vector
  dout <= d0 WHEN "00",
         d1 WHEN "01",
         d2 WHEN "10",
         d3 WHEN "11";
```



# Component Instantiation

## Example

```
ARCHITECTURE rtl OF ALU IS
SIGNAL sum, prod : bit_vector(7 downto 0);
BEGIN
  adder: ENTITY work.Adder
    PORT MAP( din1  => alu_in1,
              din2  => alu_in2,
              dout  => sum );

  multiplier: ENTITY work.Multiplier
    PORT MAP( din1  => alu_in1,
              din2  => alu_in2,
              dout  => prod );

  alu_out <= sum WHEN alu_operation='1' ELSE
            <= prod;
END rtl;
```



Bis jetzt hatten wir nur kombinatorische Logik. D.h. Nach dem elaborieren (rekursives Auflösen aller Komponenten) entspricht unser 'Programm' einem booleschen Gleichungssystem. Das ist zwar auch turing-mächtig (Satz von Cook!), aber mit Sicherheit noch nicht das was wir wollen. Wir wollen Speicherelemente:

- FlipFlops
- Latches



# Process Statements

## Example

```
ENTITY FlipFlop IS
    PORT( d    : IN  bit;
          q    : OUT bit;
          clk  : IN  bit );
END ENTITY;

ARCHITECTURE rtl OF FlipFlop is
BEGIN
    proc: PROCESS(clk) IS
    BEGIN
        IF clk'event AND clk='1' THEN
            q <= d;
        END IF,
    END PROCESS;
END
```



- Sensivity List für die Simulation
- Anweisungen werden (scheinbar) sequentiell ausgeführt.
- Signalzuweisungen werden AM ENDE DER PROCEDUR AUSGEFUEHRT
- Synthesewerkzeug generiert daraus kombinatorische Logik + Register.





## Example

```
proc: PROCESS(clk, clr) IS -- sinnlose komische Logik
BEGIN
    IF clr='1' THEN          -- asynchron-Clear
        dout <= '0'
    ELSIF clk'event AND clk='0' THEN -- falling edge
        CASE sel is
            WHEN '0' => dout <= din1 XOR (din2 AND din3);
            WHEN '1' => dout <= din1 OR din3;
        END CASE;
    END IF,
END PROCESS;
```



- IF-THEN-ELSE
- CASE-IS-WHEN
- NULL (primär für CASE)
- LOOP / WHILE-LOOP
- WAIT
- ASSERTION / REPORT (nicht synthesefähig)



# Priority-Encoder / Register

## Example

```
proc: PROCESS(din1, din2)    -- nochmal quatsch-Logik
BEGIN
    IF din='1' AND din2='1' THEN    -- kein Register,
        dout1 <= '1';                -- aber priority-encoder
    ELSIF din='1' THEN
        dout1 <= din2;
    ELSE
        dout1 <= '0';

    IF din1='1' THEN                -- einfache Logik,
        dout <= din2;                -- aber Register
    END IF;
END PROCESS;
```



# Variablen und Loop-Statements

## Example

```
orproc: PROCESS(vec) --vec:bit_vector(width-1 downto 0)
VARIABLE result : bit;
BEGIN
    result := '0';      --Achte := ist Zuweisungsoperator
    FOR i IN width-1 DOWNTO 0 LOOP
        result := result OR vec(i);
    END LOOP;

    dout <= result;
END PROCESS;
```

Schleifen und Variablen werden zu synthese-Zeit in kombinatorische Logik verwandelt.



## Example

```
clkgen: PROCESS IS
BEGIN
    clk <= '1', '0' after 20 ns;
    wait for 40 ns;
END PROCESS;
```

```
resetgen: PROCESS IS
BEGIN
    reset <= '1';
    wait for 100 ns;
    report "Reset done"
    reset <= '0';
    wait;                -- forever
END PROCESS;
```

Nicht synthesefähig – aber ständiger Begleiter fuer Test-Benches.



## Example

```
subtype small_int is integer range -128 to 127;
subtype bit_index is natural range 0 to 31;

type machine_state is (unknown, working, sleeping, error);
[...]
```

```
variable next_state : machine_state := error;
```

```
type resistance is range 0 to 1E9
  units
    ohm;
    kohm = 1000ohm;
  end units resistance;
```

(Dies ist nur eine kleine Auswahl)



## Example

```
ARCHITECTURE rtl OF SRAM IS
  TYPE MemoryType : ARRAY( 0 TO 16#FFFF# ) OF BIT;
  SIGNAL mem : MemoryType;
BEGIN
  dout <= mem(addr);

  eraseproc: PROCESS(erase) IS
  BEGIN
    mem <= (0 TO 16#FF# => '1', others => '0');
  END
END rtl
```



## Example

```
fsm: PROCEDURE(clk,reset) IS
TYPE states IS (a, b, c);
TYPE transition_matrix IS ARRAY (states) of states;
CONSTANT next_state : transition_matrix :=
    ( a=> b, b=>c, c=>b);
VARIABLE state : states;
BEGIN
    IF reset='1' THEN
        state := a;
    ELSIF clk'event AND clk='1' THEN
        state := next_state(state);
    END IF;
END PROCESS;
```





## Example

```
type std_ulogic is ('U','X','0','1','Z','W','L','H','-' );

CONSTANT resolution_table : stdlogic_table :=
-- 'U','X','0','1','Z','W','L','H','-'
  (('U','U','U','U','U','U','U','U','U'), -- 'U'
   ('U','X','X','X','X','X','X','X','X'), -- 'X'
   ('U','X','0','X','0','0','0','0','0','X'), -- '0'
   ('U','X','X','1','1','1','1','1','1','X'), -- '1'
   ('U','X','0','1','Z','W','L','H','X'), -- 'Z'
   ('U','X','0','1','W','W','W','W','X'), -- 'W'
   ('U','X','0','1','L','W','L','W','X'), -- 'L'
   ('U','X','0','1','H','W','W','H','X'), -- 'H'
   ('U','X','X','X','X','X','X','X','X','X')); -- '-'
```



## Example

```
LIB ieee;
USE ieee.std_logic_1164.all;

ENTITY BusDriver IS
    PORT( en      : IN      std_logic;
          dread   : OUT     std_logic;
          dwrite  : IN      std_logic;
          bussig  : INOUT   std_logic );
END ENTITY;

ARCHITECTURE rtl OF BusDriver IS
    dread    <= bus;
    bussig   <= dwrite WHEN en='1' ELSE
                'Z';
END ENTITY;
```

(Fast) immer STD\_LOGIC anstelle von BIT benutzen!



## Example

```
LIB ieee;
USE ieee.std_logic_1164.all;

ENTITY cpu IS
    PORT( clk      : IN      std_logic;
          int_req   : IN      std_logic_vector(7 downto 0);
          mem_we     : OUT     std_logic;
          mem_addr   : OUT     std_logic_vector(31 downto 0);
          mem_data   : INOUT   std_logic_vector(31 downto 0);
          mem_rdy    : IN      std_logic );
END ENTITY;

[...]
```

```
interrupt: PROCESS(clk)
BEGIN
    IF clk'event AND clk='1' THEN
        IF int /= "00000000" THEN -- handle interrupt condition.
```



# USE ieee.numeric\_std.all

## Example

```
USE IEEE.NUMERIC_STD.ALL;

ENTITY alu IS
  PORT( op      : IN  alu_operation;
        oper1   : IN  std_logic_vector(31 downto 0);
        oper2   : IN  std_logic_vector(31 downto 0);
        result  : OUT std_logic_vector(31 downto 0) );
END ENTITY;

alu: PROCESS(clk) IS
BEGIN
  IF op = add THEN
    result <= std_logic_vector(signed(oper1)+signed(oper2));
  ELSE op = multiply THEN
    result <= std_logic_vector(signed(oper1)*signed(oper2));
  [..]
```



- Typ fuer Ganz-Zahlen mit definierbarer Breite
- konvertiert(castet) problemlos nach std\_logic\_vector
- erlaubt arithmetische Operationen
- voll synthesefaehig



# USE ieee.numeric\_std.all

Example

