

Oberseminar Informationstechnik

Project Work

- **ONE system ACCOUNT for ALL lab courses (e.g. Schaltkreis- und Systementwurf, Prozessorentwurf, this seminar)**
- **Please REGISTER yourself for EACH LAB COURSE**

Website (Home)

<https://tu-dresden.de/ing/elektrotechnik/iee/hpsn>

Website (Information, Links, **Anmeldung**)

<https://>

tu-dresden.de/ing/elektrotechnik/iee/hpsn/studium/materialien

Login via IDM (**ZIH-Login**) **required**. Typical Login ID: e.g. **s1234567**

Register for **`OSM`**

Room: **TOE 201**

Single-Person projects. One student – one project work.

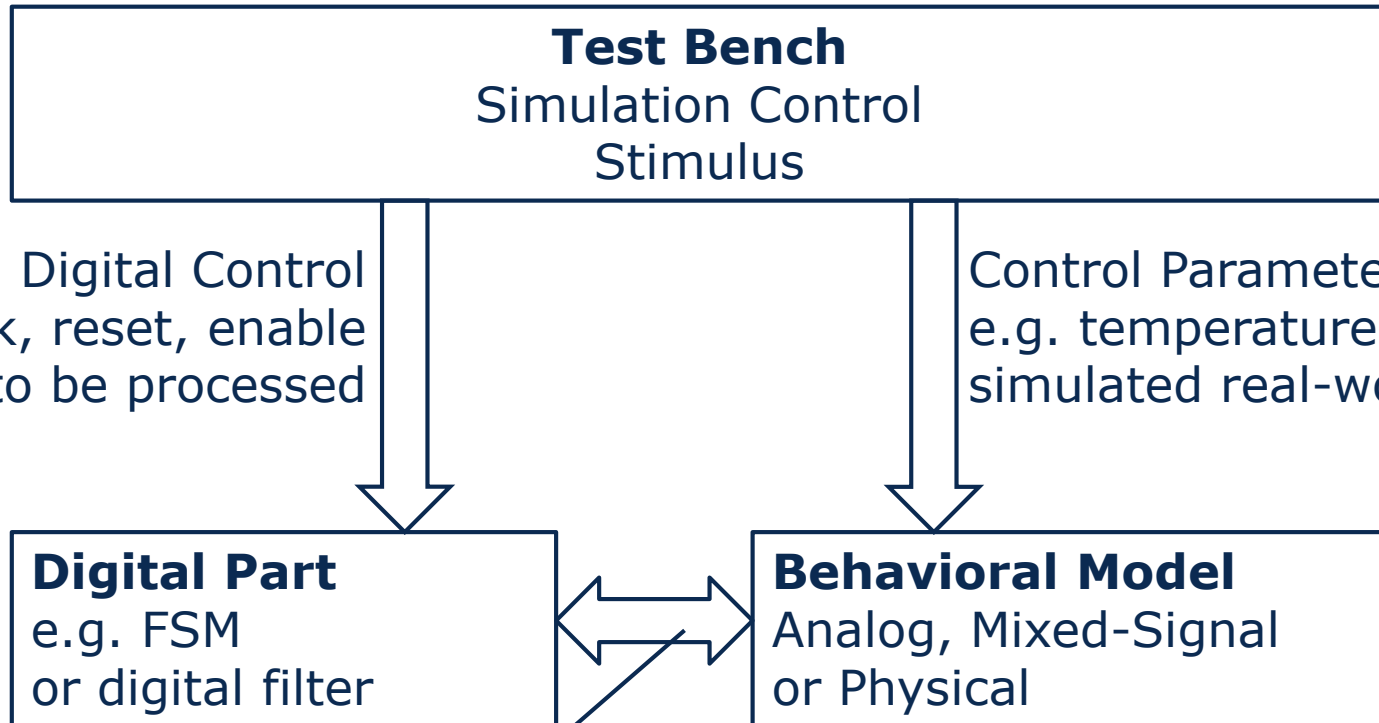
Topics

- **Choice of topics presented in the lecture**
- **You can also make your own project proposal**

Structure

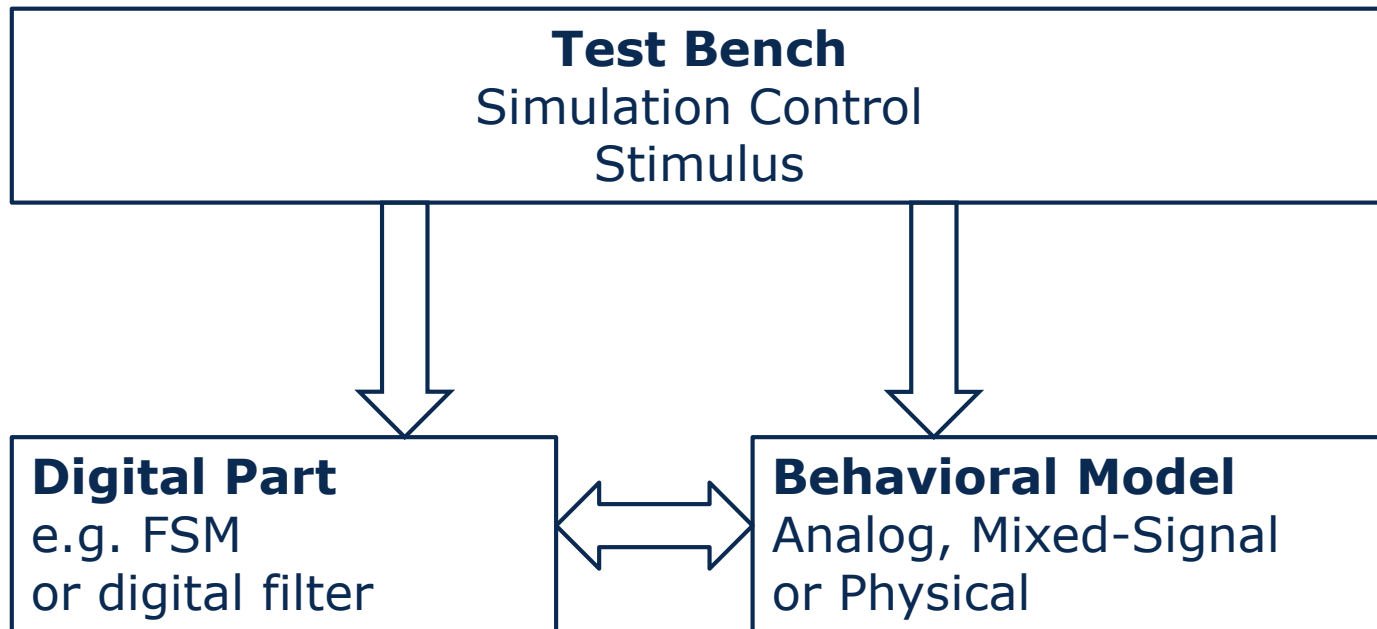
- **Self-study your topic**
- **Give a presentation to the course: Elaborate the problem and present approaches to the solution.**
- **Develop and implement the solution in VHDL, simulate it.**
- **Hand in a report.**

Presentation and Report are evaluated to yield your credits.



Digital Interface:

Control lines operating real stuff represented by the model.
Feedback lines providing digital information on the model state

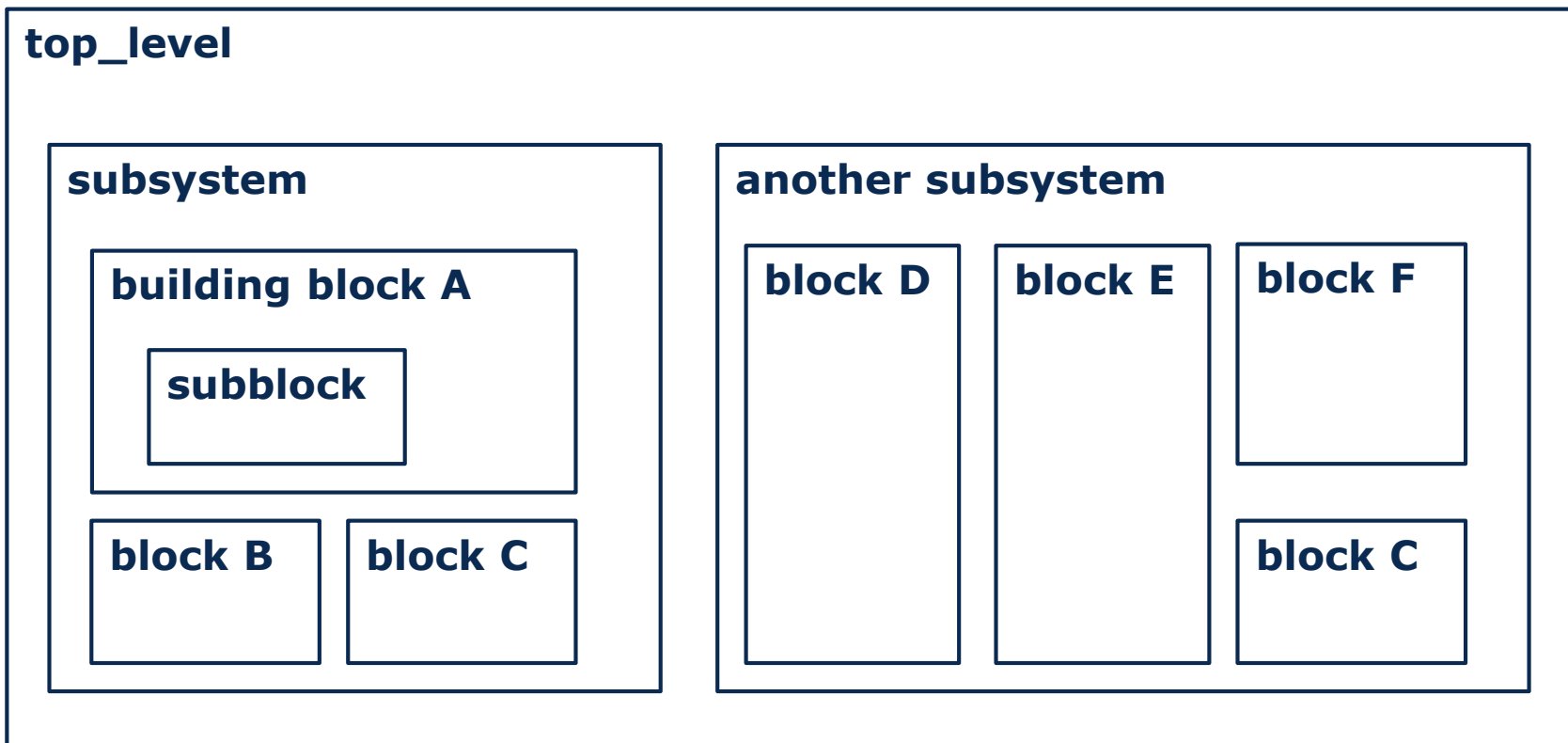


synchronuous digital logic
combinational + sequential
synthesizable RTL description

abstract simplified model
explicitly calculates states of
analog and physical quantities

Project Work	Digital Controller	Mixed Signal Behavioural Model
PLL		
Analog PLL	PFD, Divider, Binary Search	Loop Filter, Oscillator
Digital PLL	PID controller	Oscillator with temperature and voltage dependencies
Digital DLL	PID controller	Digitally adjustable Delay Line, external Delay
ADC/DAC		
SAR ADC	SAR register / FSM	SC charge redistribution network
SD-ADC	Decimation filter implementation	SD-modulator
Class D PA	Digital FIR design + implementation	analog RLC network
Power		
Step-Down DCDC voltage mode regulation	PID controller + digital PWM	analog RLC network, simple ADC
Step-Down DCDC current mode regulation	PID controller + digital PWM	analog RLC network, simple ADC
Step-Down DCDC discontinuous mode pwm control	PID controller + digital PWM	analog RLC network, simple ADC
DCDC Current Mode / controlled LED supply	digital controller, PWM, interface	analog RLC network, simple ADC
Neuro		
Polychronuous Spiking Neural Network	AER busses and arbiters	Neurons, Synapses, Axons
Leaky IAF neurons	configuration registers, spike decoders	Neurons, Synapses
Time Division Multiplexed Perceptron	Multiplexers, Counters, FSM	Current mode MDAC, simple non-linear neuron core
Other		
Brushless DC Motor	Sequencer, controller	Electro-mechanical motor model
<i>Your Own Idea</i>	?	?

Hierarchical Design



Hierarchy corresponds to the Instatiation

Parallel Statement Inside the Architecture:

```
architecture RTL of Example is  
    -- signal declarations go here  
begin  
    S <= A xor B xor Ci;  
  
    process (A, B) is  
    begin  
        AB <= A and B;  
    end process;  
  
    OR3_i : OR3  
        port map (I1 => AB, I2 => AC, I3 => BC, O => Co);  
  
end architecture RTL;
```

instantiation

architecture STRUCTURE of FOO is

```
component FULL_ADDER is  
  port( A : in std_logic;  
        B : in std_logic;  
        Ci: in std_logic;  
        S : out std_logic;  
        Co: out std_logic);  
end component FULL_ADDER;
```

```
  -- signal declarations  
begin  
  ADD_BIT1_i : FULL_ADDER  
    port map (A => OP_A(1),  
              B => OP_B(1),  
              Ci => CARRY(0),  
              S => RES(1),  
              Co => CARRY(1)  
            );  
end architecture STRUCTURE;
```

```
entity FULL_ADDER is  
  port( A : in std_logic;  
        B : in std_logic;  
        Ci: in std_logic;  
        S : out std_logic;  
        Co: out std_logic);  
end entity FULL_ADDER;
```

subblock

instance of subblock

port S of FULL_ADDER is connected to
bit 1 of signal RES in block FOO

configuration – *optionally* select particular architectures

configuration name

entity

```
configuration STRUCTURE_CFG of FOO is
```

```
for STRUCTURE
```

select architecture for FOO

```
    for ADD_BIT1_i : FULL_ADDER  
        use entity work.FULL_ADDER(NET) ;  
    end for;
```

```
end for;
```

```
end configuration STRUCTURE_CFG;
```

configuration – *optionally* select particular architectures

in architecture **STRUCTURE** of entity **FOO** ...

```
configuration STRUCTURE_CFG of FOO is
```

```
  for STRUCTURE
```

... instance **ADD_BIT1_i** ...

```
    for ADD_BIT1_i : FULL_ADDER  
      use entity work.FULL_ADDER (NET) ;  
    end for;
```

```
  end for;
```

```
end configuration STRUCTURE_CFG
```

... is an instance of
entity **FULL_ADDER**
architecture **NET**

configuration – *optionally* select particular architectures

- for an entity: select the architecture
- for an instance:
 - use particular architecture
 - use specific configuration
 - can replace the component with different entity
 - can map ports, even interchange ports
- configuration specification also within architecture
- no config -> default binding, last architecture

configuration – *optionally* select particular architectures

- Very wide range of means to select between choices in the hierarchy
- Most features ***not supported by synthesis*** tools
- Restrict yourself to **one top-level configuration** or **no configurations** at all

Binary Arithmetic

- **IEEE 1164 logic** is the most prominent enumeration type

`std_logic`

single-bit digital signals

`std_logic_vector`

multi-bit digital busses

`signed, unsigned`

`std_logic_vector` in numerics

Binary Arithmetic

- **IEEE 1164 logic** is the most prominent enumeration type

`std_logic`

single-bit digital signals

`std_logic_vector`

no numerical interpretation

`signed, unsigned`

2's complement numerics

numerics

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BAR is...
architecture FOO of BAR is...
    
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BAR is ... end entity;

architecture FOO of BAR is
    signal L : std_logic_vector(15 downto 0);
    signal U : unsigned(15 downto 0);
    signal S : signed(15 downto 0);
...
    U <= unsigned(L);
    S <= signed(L);
    L <= U when X='1' else S;

    U <= to_unsigned( 173, 16);
    S <= to_signed( -212, 16);
    X <= A( to_integer(U(3 downto 0)));
```

library and use statements

assignment to subtype with cast

direct assignment to base type

conversion functions

... back to

Parallel Statements Inside Architectures

```
architecture RTL of Example is  
    -- signal declarations go here  
begin  
  
    S <= A xor B xor Ci;  
  
    process (A, B) is  
    begin  
        AB <= A and B;  
    end process;  
  
    OR3_i : OR3  
        port map (I1 => AB, I2 => AC, I3 => BC, O => Co);  
  
end architecture RTL;
```

Parallel Statements work concurrently.
They correspond to separate pieces of hardware.

Signal Assignment Statements

```
architecture RTL of Example is
  -- signal declarations go here
begin
```

```
S1 <= A xor B;
```

```
S2 <= B when A='1' else C;
```

assignment with an expression

conditional assignment

```
end architecture RTL;
```

whenever the right-hand side changes...

... the left-hand side receives a new value

Operators in Signal Assignments

- **bit-wise logic** *and* **Boolean logic**

result type = argument type

AND, OR, NAND, NOR, XOR, XNOR, NOT

- **comparisons, result type: Boolean**

= /= < > <= / >=
equal not equal less than greater than less/greater or equal

- **numeric, result type: numeric**

+, - addition, subtraction, unary sign

*, / multiplication, division

mod, rem modulo division, sign of right (mod) or left (rem) operand

** exponentiation

abs absolute value

other **Parallel Statements Inside Architectures**

```
architecture RTL of Example is  
    -- signal declarations go here  
begin  
    S <= A xor B xor Ci;
```

```
process (A, B) is           process  
begin                       statement  
    AB <= A and B;  
end process;
```

```
OR3_i   : OR3  
    port map (I1 => AB, I2 => AC, I3 => BC, O => Co);  
  
end architecture RTL;
```

Process Statement

```
process ( A, B, C) is
```

```
begin
```

```
  if A = "010" then
```

```
    S1 <= B;
```

```
  else
```

```
    S1 <= C;
```

```
  end if;
```

```
end process;
```

sensitivity list

inside the process:
sequential statements

all processes run in
parallel

```
label: process ( A) is
```

```
begin
```

```
  S2 <= A(1) and not A(0);
```

```
end process label;
```

a process can be
named with a label

Sequential Signal Assignments in Process

```
process ( A, B, C) is  
begin  
    S1 <= B;  
    if A = '1' then  
        S1 <= C;  
end if;  
end process;
```

default assignment
one good practice for
combinational logic

the last executed
sequential assignment
supercedes all others

Sequential Statement: if-then-else

```
if condition then  
    sequential_statements;  
elsif condition then  
    sequential_statements;  
else  
    sequential_statements;  
end if;
```

expression yielding a Boolean value

optional ELSIF branch.

optional ELSE branch.

required END IF;

Sequential Statement: case

expression of arbitrary type

case expression is

when choice => equential_statements;

when choice => equential_statements;

when choice => equential_statements;

end case;

choice value of same type as expression

Sequential Statement: case

As of the VHDL standard,
all possible choices must be included, ...

```
case SEL is  
  when "01" => Z <= A;  
  when "10" => Z <= B;  
  when others => Z <= 'X';  
end case;
```

... unless the others clause is used as the last choice

Sequential Statement: case

```
case SEL is  
  when "01" => Z <= A;  
  when "10" => Z <= B;  
  when others => null;  
end case;
```

**null statement – explicitly do
nothing in this branch**

Sequential Statement: case

```
case INT is  
  when 0           => Z <= A;  
  when 1 to 3     => Z <= B;  
  when 4 | 6 | 8  => Z <= A;  
end case;
```

range

selection: List of values

Not allowed in case statement:

ranges and selections must not overlap

```
case INT is
  when 1 to 3 => Z <= B;
  when 2|4|6 => Z <= A;
  when others => Z <= 'X';
end case;
```

ranges are not allowed for vectors / arrays

```
case VEC is
  when "000" => Z <= A;
  when "001" to "011" => Z <= B;
  when others => Z <= 'X';
end case;
```

Sequential Statement: for loop

Loop variable implicitly declared with type of loop range

```
Z <= "0000";  
for I in 0 to 3 loop  
  if (A = I) then Z(I) <= '1';  
  end if;  
end loop;
```

Example for combinational 1-hot decoder

Sequential Statement: for loop

```
Z <= "0000";  
for I in 0 to 3 loop  
  if (A = I) then  
    Z(I) <= '1';  
    exit;  
  end if;  
end loop;
```

End the loop with exit