FYS4220
Real time and embedded data systems

Introduction to Tools & VHDL

Ketil Røed

Autumn 2015

Motivation

- Doing your first FPGA design. (Lab 1).

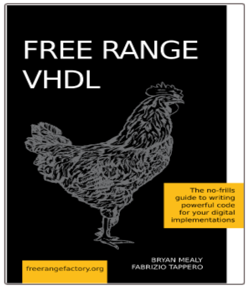
- Some keywords for this lecture
  - Entity & architecture
  - Signal declaration & assignment
  - Description models (Structural, behavioral)
    - Component and port map
    - non-procedural and procedural (data-flow and process)

(Some slides are based on material from 2011 lectures by J.K. Bekkeng)

VHDL

“It is well worth noting that VHDL and other similar hardware design languages are used to create most of the digital integrated circuits found in the various electronic gizmos that overwhelm our modern lives.”

Mealy and Teppero, “Free range VHDL”


FREE RANGE VHDL
The no-frills guide to writing powerful code for your digital implementations
BRYAN MEALY
FABRIZIO TAPPERO
freerangefactory.org

VHDL

- VHDL: VHISC Hardware Description Language
  - VHISC: Very High Speed Integrated Circuits

- Introduced by the US Department of Defence (DoD) in 1981

- Initially a specification and modeling language - documenting behavior of ASICs
  - Alternative to complex manuals

- The idea of simulating this documentation became attractive and logic simulators where developed in the late 1980s.

- The next step was the development of logic synthesis tools to read VHDL and output a definition of the physical implementation of the circuit

- Today it is an industry standard (IEEE 1987) for specifying, verifying and designing digital electronics (library IEEE; use IEEE.std_logic_1164.all;)

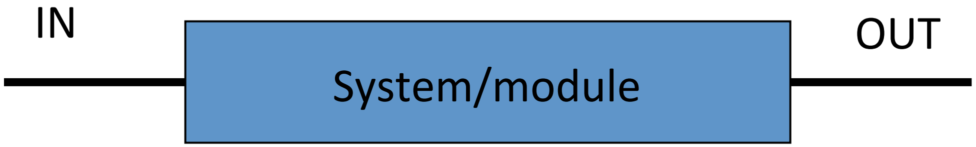
- Revised VHDL standards: VHDL 93, 2000, 2002, 2007, 2008

VHDL

- VHDL can be looked at as a model of a digital system
  - Powerful alternative to schematic based design

- A change (transition) on the input may lead to a new system state and consequently a change of the output after a given time delay

- Describes and simulates concurrent events

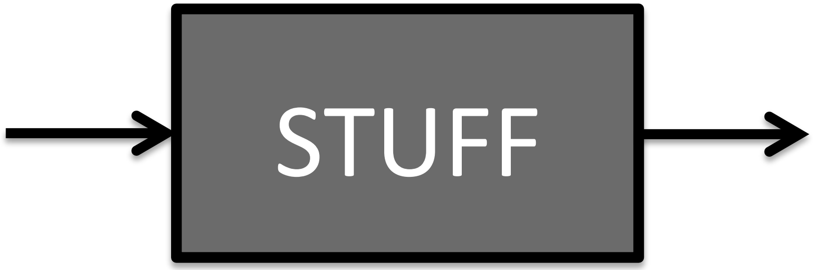


Representation of a digital system

Important remarks

- VHDL is a HARDWARE DESCRIPTION LANGUAGE
- You are designing actual hardware!
- A VHDL model is translated into actual hardware and mapped onto a CPLD/FPGA
- Due to the nature of hardware components which are always running, VHDL is a highly concurrent language.
  - Simultaneously execution of several tasks
- This is in contrast to other high level languages where the code is executed (sequentially) by a processor (predefined hardware).
- Execution of a VHDL code means that the VHDL model is being simulated (by software running on a computer)

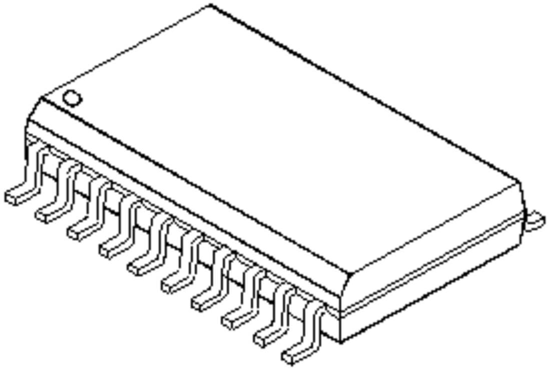
Design approach



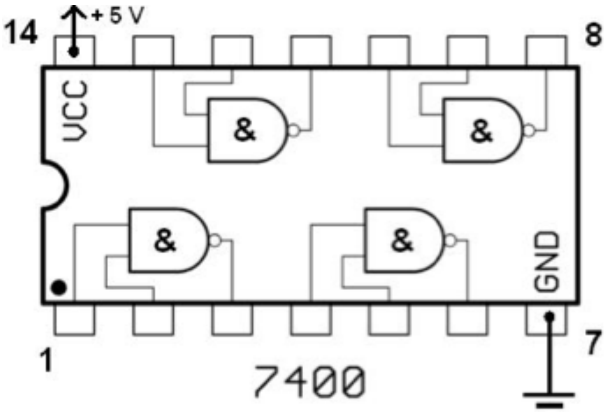
- VHDL descriptions of circuits are based on the black-box approach.
- The two main parts of any hierarchical design are the black box and the stuff that goes into the black box (e.g. other black boxes)
- In VHDL the black box is referred to as **entity** and the stuff that goes inside it is referred to as the **architecture**.
- Allows to use hierarchical structure (modularity) and the reuse of previously written code.
- A module is referred to by its inherently simple black box representation rather than by the details of its inside circuitry

Compared with an IC

- The **entity** describes the interface to the outside world (connection pins of package)



- The **architecture** describes the functionality of the circuit inside the entity (package)



Entity & architecture templates

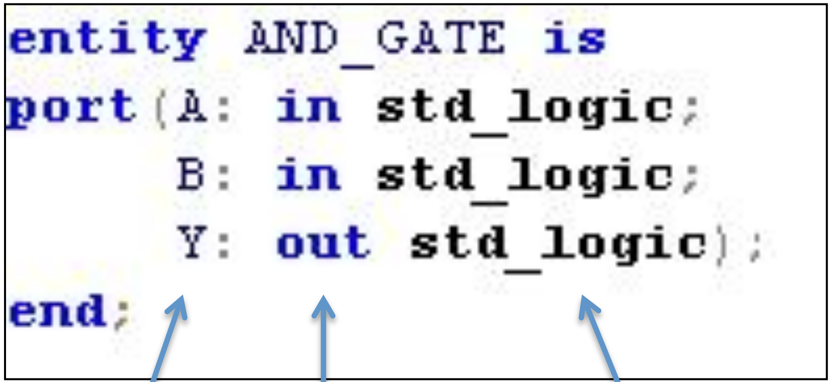
```
entity  model_name is
port
(
      list of inputs and outputs
);
end  model_name;
```

Same name as the file, e.g. **module_name.vhd**

```
architecture architecture_name of model_name is
begin
      ...
      VHDL concurrent statements
      ....

end architecture_name ;
```

Entity declaration

- Abstract the functionality of a circuit description to a higher level.
- Provides a wrapper for the lower-level circuitry



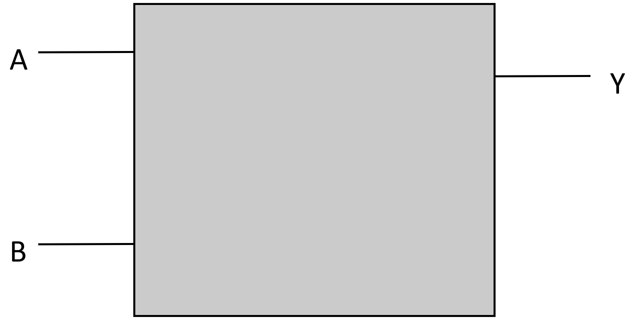
```
entity AND_GATE is
port (A:  in std_logic;
      B:  in std_logic;
      Y:  out  std_logic);
end;
```

Unique Identifier name

direction

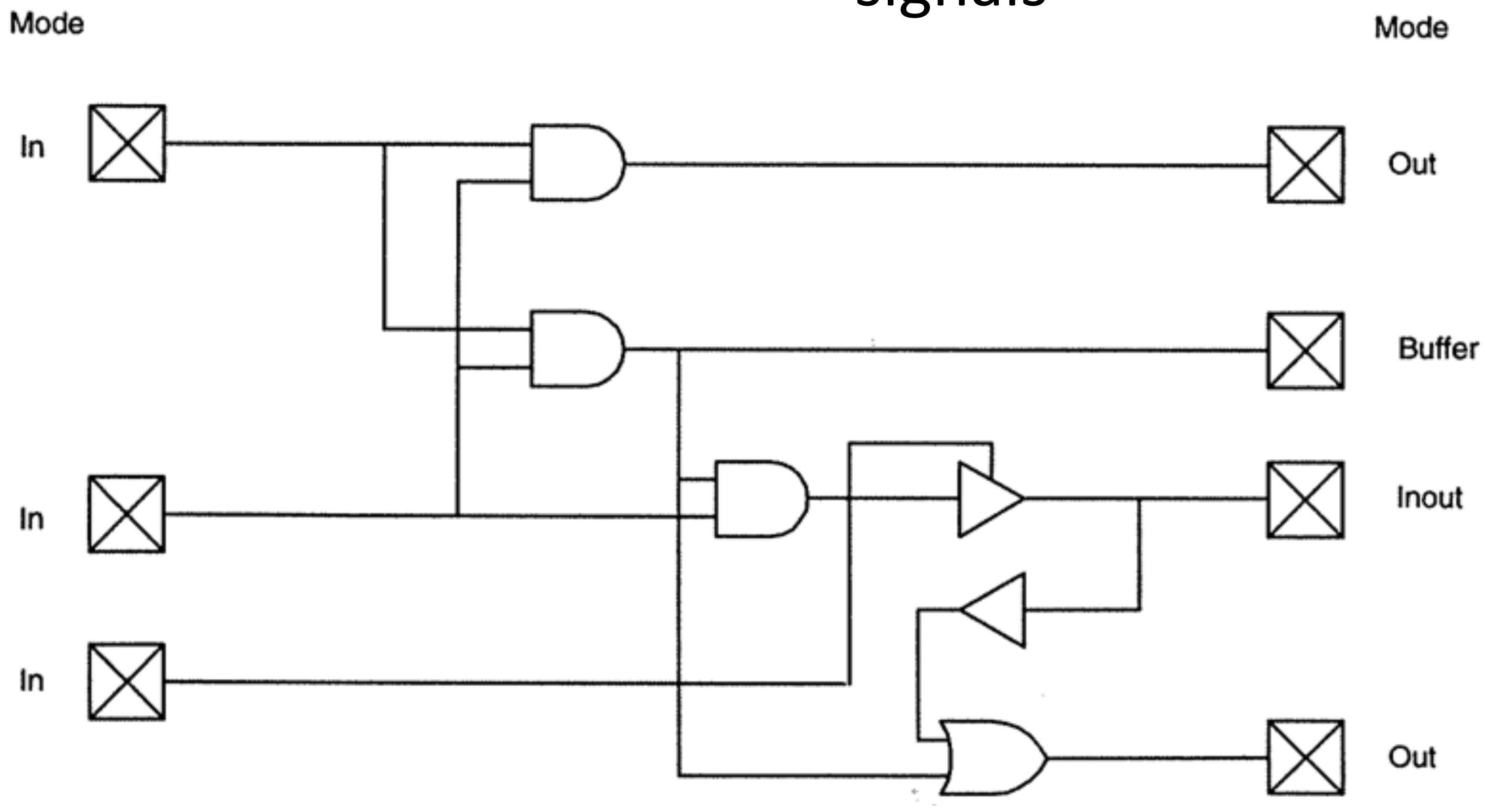
Type of data

Entity AND_GATE is



Direction (mode)

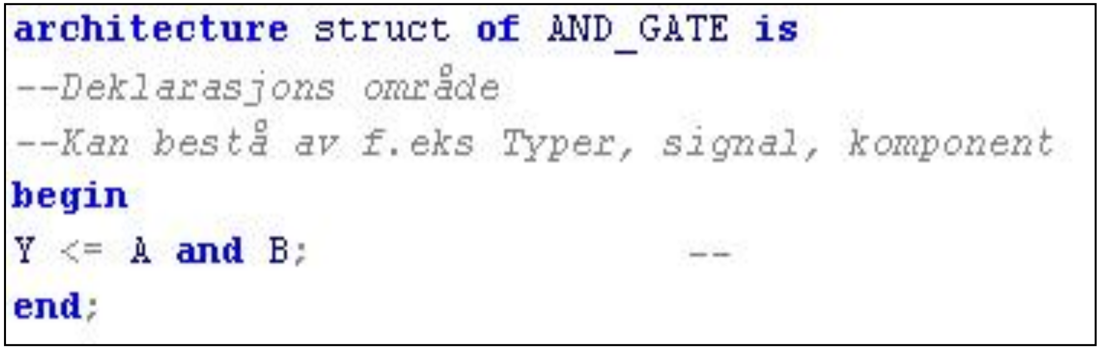
- **In** – flow into the entity
- **Out** – flow out of the entity, ***no*** feedback
- **Buffer** - flow out of the entity, feedback allowed
- **Inout** - for bi-directional signals



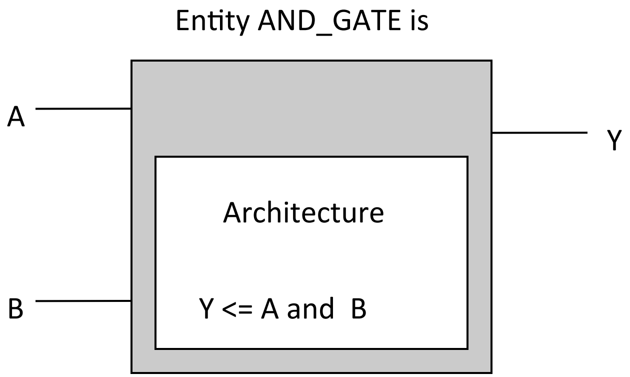
Architecture

- While **entity** describes the interface or external representation of the circuit, the **architecture** describes what the circuit actually does
  - Defines structure/behavior of the entity

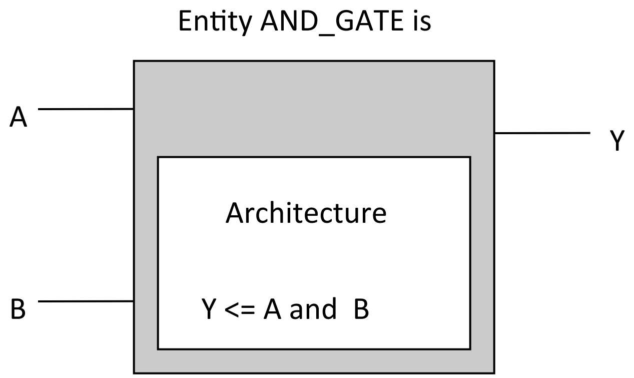
- Entity declarations are generally easy while describing the operation of a circuit can become very complex

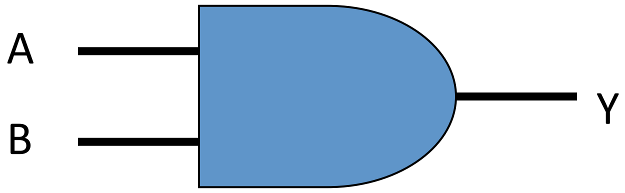


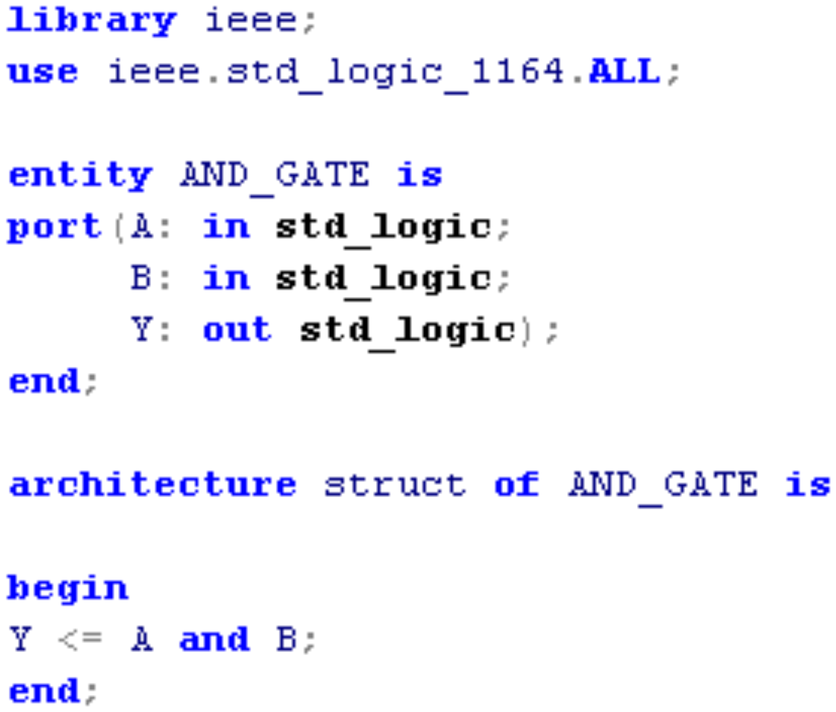
```
architecture struct of AND_GATE is
--Deklarasjons område
--Kan bestå av f.eks Typer, signal, komponent
begin
Y <= A and B;                    --
end;
```



Example of basic AND-gate







```
library ieee;
use ieee.std_logic_1164.ALL;

entity AND_GATE is
port(A: in std_logic;
     B: in std_logic;
     Y: out std_logic);
end;

architecture struct of AND_GATE is

begin
Y <= A and B;
end;
```

Main VHDL object types

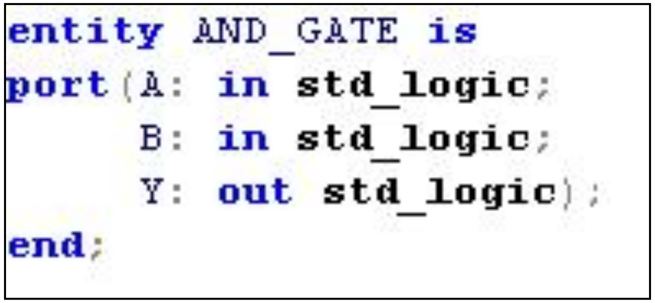
- **Signals**
  - communication between components
  - Software representation of a wire (real physical signal)
  - Signal assignments are associated with a delay
- **Variables**
  - Convenient mechanism for local storage
  - E.g. loop counters, intermediate values
  - All assignments take place immediately (no delay)
- **Constants**
  - Is like a variable object but the value cannot change

Declaration and assignment

- Needs to be declared before they are used
- Declaration of
  - **Signals** are done inside the scope of the architecture (cannot be declared inside a process)
  - **Variables** can only be done inside a process
  - **Constants** can be done in architecture and inside process
- Assignment of a new value to a
  - **Signal** is done using the operator “<="
  - **Variable** is done using the operator “:=”
- Variable assignments are executed instantaneously while signal assignments are executed after a certain time

Identifier

- Identifier is the name of an object (e.g. signal)
- Names can be constructed using:
  - a b c … z (alphabetic letters)
  - 0 1 2 … 9 (numbers)
  - _ (underscore)



```
entity AND_GATE is
port(A: in std_logic;
     B: in std_logic;
     Y: out std_logic);
end;
```

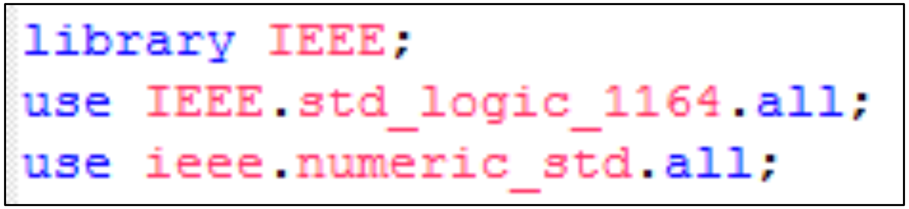
- With the following reservations:
  - The first character must be a letter
  - The last character cannot be an under score
  - Successive underscores are not allowed
  - Cannot use VHDL reserved words (e.g. **and**, or, **port, constant**)
- VHDL is case insensitive
- nextstate ⇔ NEXTSTATE ⇔ nExTsTaTe

Arithmetic & logical operators

Arithmetic operators

- + Addition
- - Subtraction
- * Multiplication
- / Division

Use with care, creates much logic



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

Logical operators:

- and, nand, or, nor, not, xor, xnor
- IEEE 1164 uses these operators in **std_logic**

Logical operators

- **and, or, not, nand, nor, xor og xnor** are predefined for **bit**, **boolean, std_(u)logic,**
- Logical operators (except **not**) do not have precedence in VHDL, therefore parenthesis is demanded in multi level logic:
  - X <= A **or** B **and** C gives an error in VHDL (equal precedence of operators)
  - A **or** (B **and** C)
  - (A **or** B) **and** C

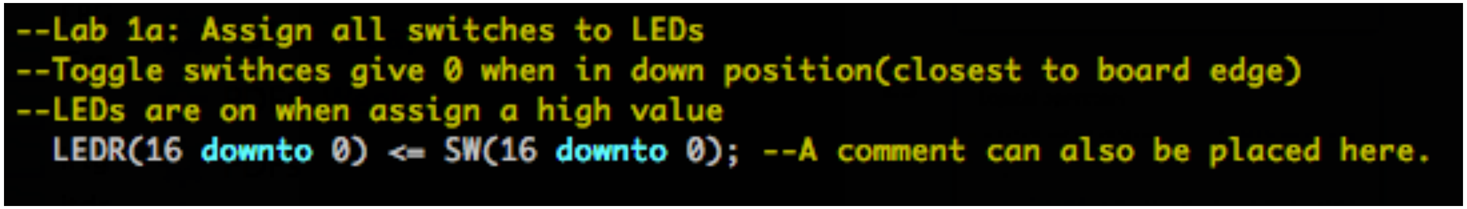
    } Correct for VHDL
  - not A and B This is the same as:
  - (not A) and B ***not*** has higher precedence than ***and***, hence ***not A*** is evaluated before the ***and*** term.

General recommendation is to be **generous** with parenthesis, it will make equations more readable and less prone to errors.

And you will not have to bother with precedence.

Comments

-- In **VHDL**, when a double **dash** (--) is used, any text to the right will be treated as a **comment** and will not be interpreted by the compiler.



```
--Lab 1a: Assign all switches to LEDs
--Toggle swithces give 0 when in down position(closest to board edge)
--LEDs are on when assign a high value
  LEDR(16 downto 0) <= SW(16 downto 0); --A comment can also be placed here.
```

“Research has shown that using lots of appropriate comments is actually a sign of high intelligence. “

According to Mealy and Teppero, “Free range VHDL”

Relational operators

- equality =
- inequality /=
- Size operators < , <= , > , >=

- The operands must both be of the same type, and the result is a Boolean value (true/false)

Example:

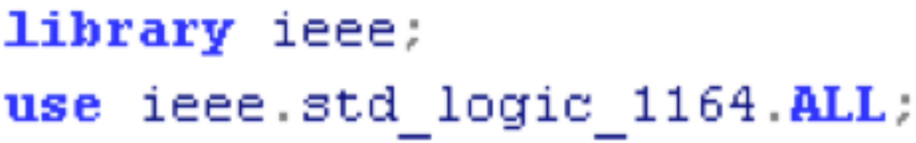
```
signal a : std_logic;

      ..........

if a = 1 then
```

Gives an error, becasue ***a*** is ***std_logic***, while 1 is an ***integer***

Data type **std_ulogic / std_ulogic_vector**

- IEEE standards for representation of digital signals

- Available after declaration of:



```
library ieee;
use ieee.std_logic_1164.ALL;
```

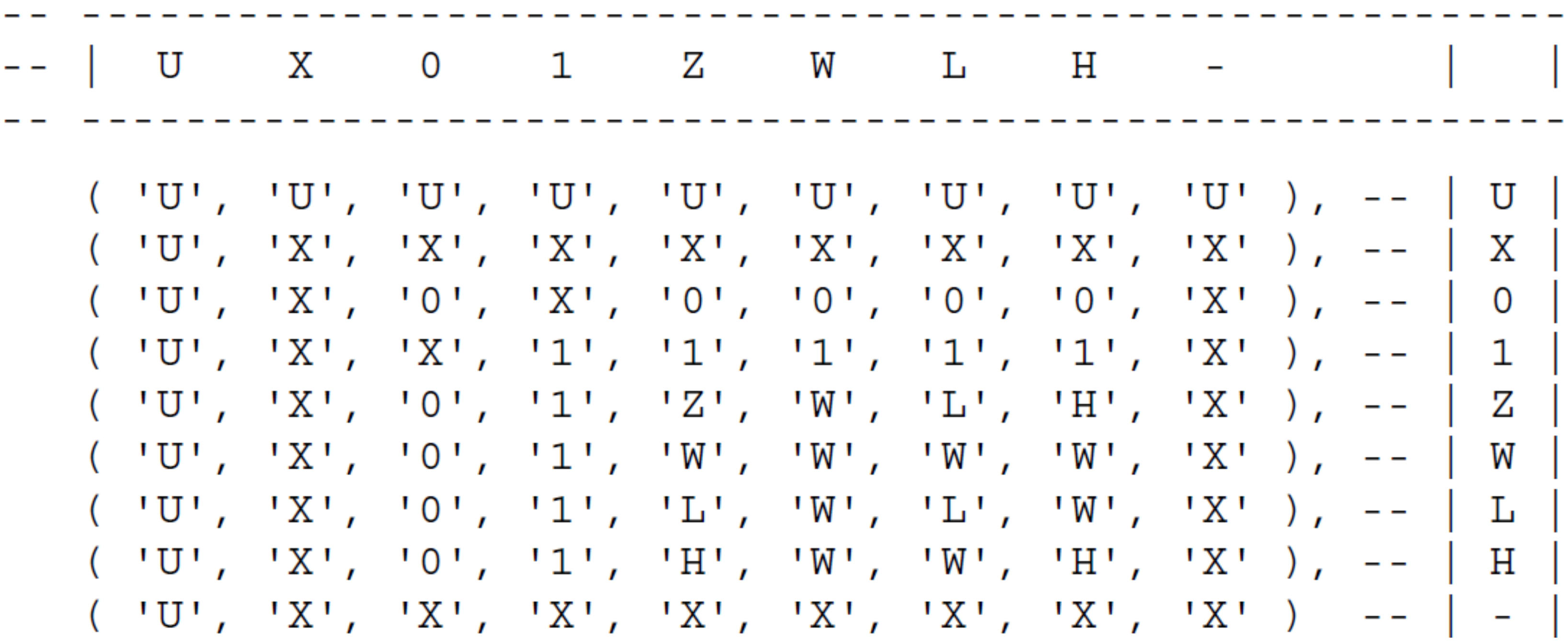
- Normally assuming values 0 and 1

- However, the desire to model three-state drivers, pull-up and pull-down outputs, high impedance state the **std_ulogic** type includes 9 different values

’U’	Uninitialized		
’X’	Forcing Unknown		
’0’	**Forcing 0**		For synthesis of logic
’1’	**Forcing 1**		
’Z’	**High Impedance**		
’W’	Weak unknown		
’L’	Weak 0	– pull down	
’H’	Weak 1	-- pull up	
’-’	Don’t care		

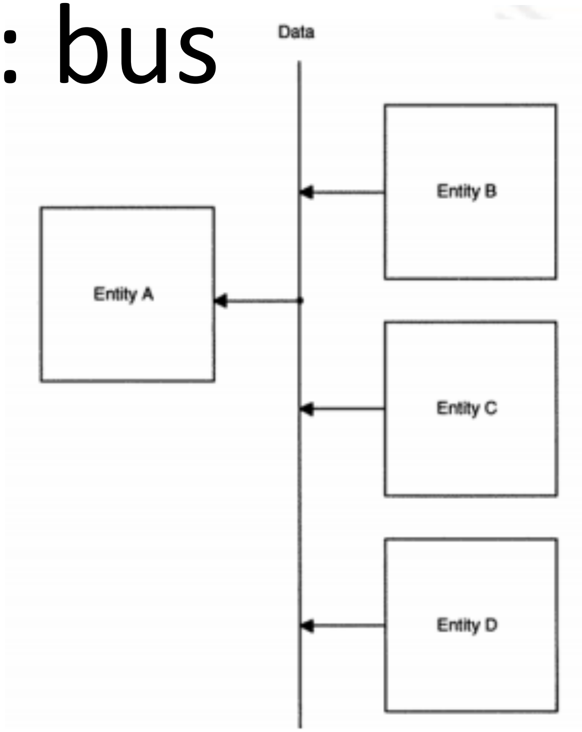
Std_logic 1164 resolution function

The sub type **std_logic** is ”resolved” **std_ulogic**. When two or more drivers are connected together the value is determined by a ”resolution 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', 'X' ), -- | 0 |
    ( 'U', 'X', 'X', '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' )  -- | - |
```

Ex. Multiple drivers: bus



- The resolution function is used to simulate a data bus.

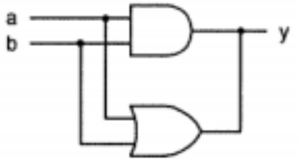
- Useful that the simulator can indicate an unknown value if two or more entities write to the same bus line at the same time with opposite logic values

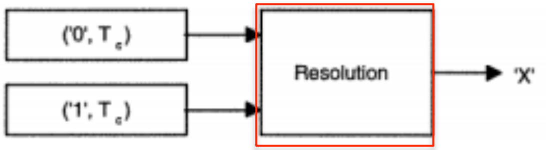
- If one module writes to a bus, the outputs of the other module's bus line must be in tri-state (high impedance)

- The unknown value 'X' has no meaning for synthesis.

~~y <= 'X'~~

```
architecture multiply_driven_signal of my_design is
begin
    y <= a and b;
    y <= a or b;
end multiply_driven_signal;
```





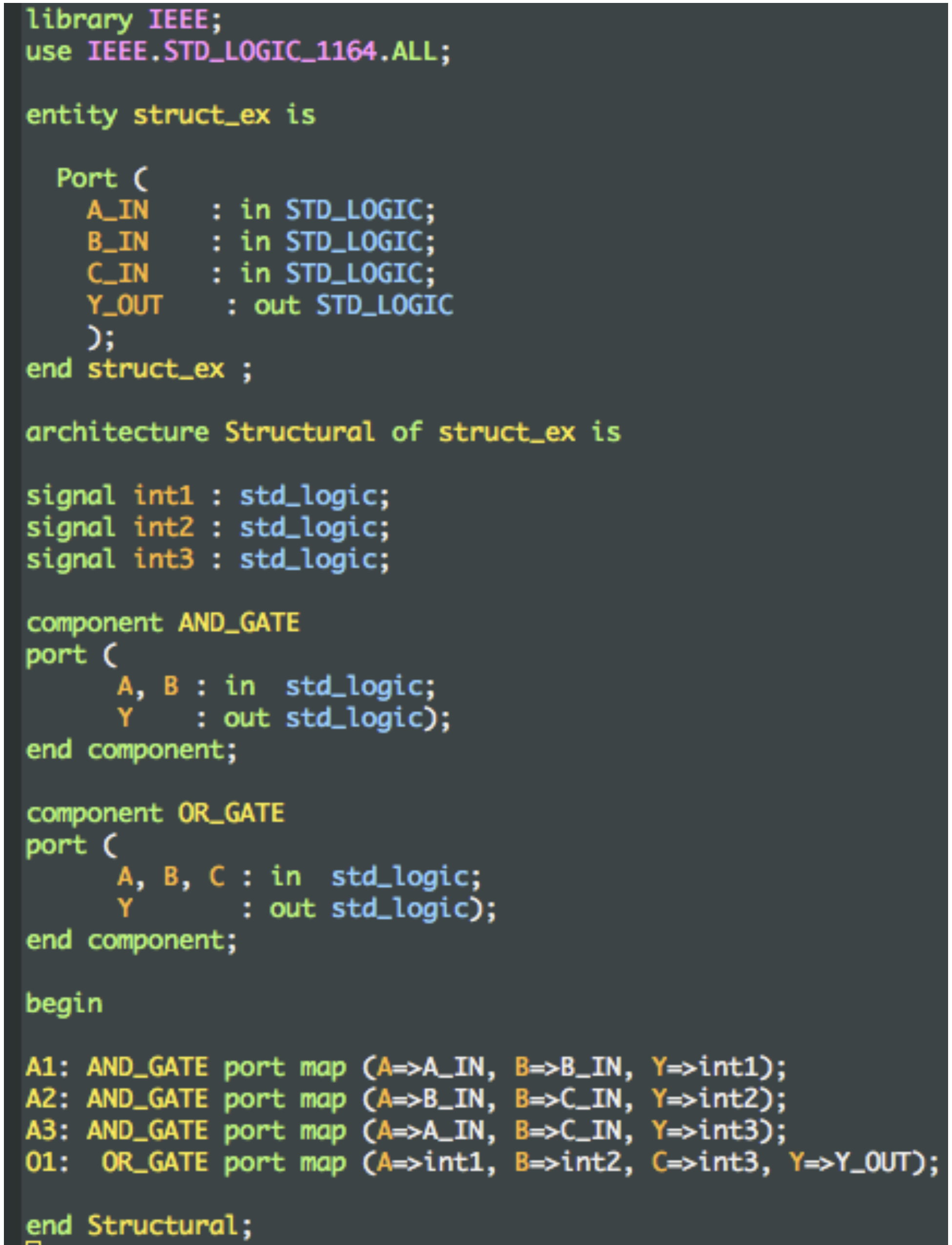
Conflicting assignment must be resolved

Description models

- Structural
  - Interconnection of components (black boxes)
  - Concurrent execution of statements
  - Can be used to create hierarchy in the code
  - Keywords: ***component*** & ***port map***

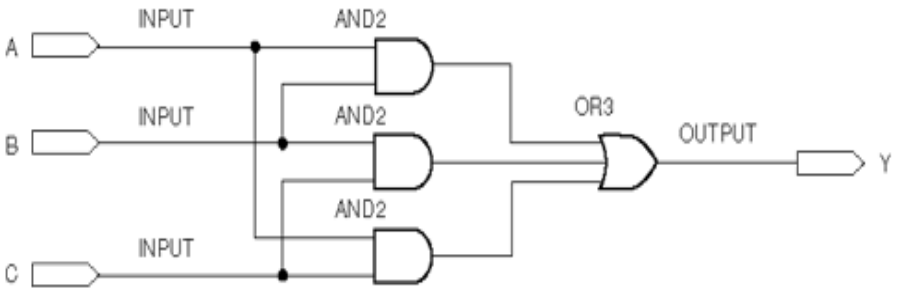
- Behavioral
  - Models how the circuit outputs will react to the circuit inputs
  - Both concurrent and sequential execution of statements
  - Keywords: ***data flow*** & ***process***

Example of structural model



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

entity struct_ex is

  Port (
    A_IN      : in STD_LOGIC;
    B_IN      : in STD_LOGIC;
    C_IN      : in STD_LOGIC;
    Y_OUT      : out STD_LOGIC
    );
end struct_ex ;

architecture Structural of struct_ex is

signal int1 : std_logic;
signal int2 : std_logic;
signal int3 : std_logic;

component AND_GATE
port (
      A, B : in  std_logic;
      Y    : out std_logic);
end component;

component OR_GATE
port (
      A, B, C : in  std_logic;
      Y       : out std_logic);
end component;

begin

A1: AND_GATE port map (A=>A_IN, B=>B_IN, Y=>int1);
A2: AND_GATE port map (A=>B_IN, B=>C_IN, Y=>int2);
A3: AND_GATE port map (A=>A_IN, B=>C_IN, Y=>int3);
O1:  OR_GATE port map (A=>int1, B=>int2, C=>int3, Y=>Y_OUT);

end Structural;
```

Can not directly connect together the input/output of a component to another component's output/input! Must use an internal signal (such as int1 in this example), unless a connection to a port is made



Behavioral model styles

- Non-procedural (Data-flow)
  - Concurrent execution

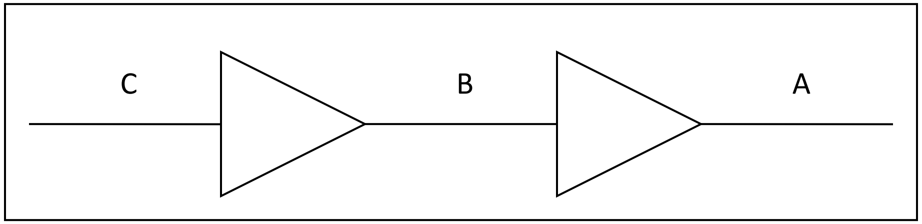
- Procedural (more in next lecture)
  - Algorithmic
    - step-by-step calculations / sequence of operations
  - Sequential execution of statements within process
  - Execution of a process is equivalent to a single concurrent statement.

- All VHDL processes execute concurrently

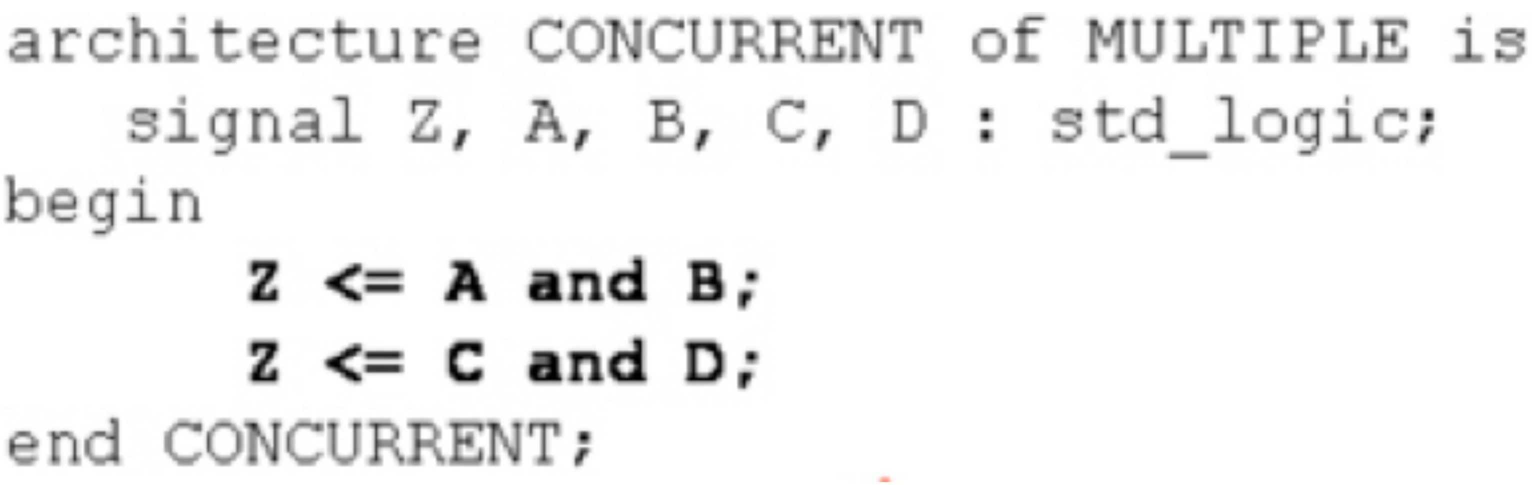
Data flow (non-procedural)

- Concurrent statements
  - Executed in parallel (order or statements is irrelevant)
- Event driven
  - Executed only if transition/change on input

```
------------------------
B  <= C;  --executed if transition on C
A  <= B;  --executed if transition on B

is equivalent to

A  <= B;
B  <= C;
------------------------
```



Data flow (non-procedural)

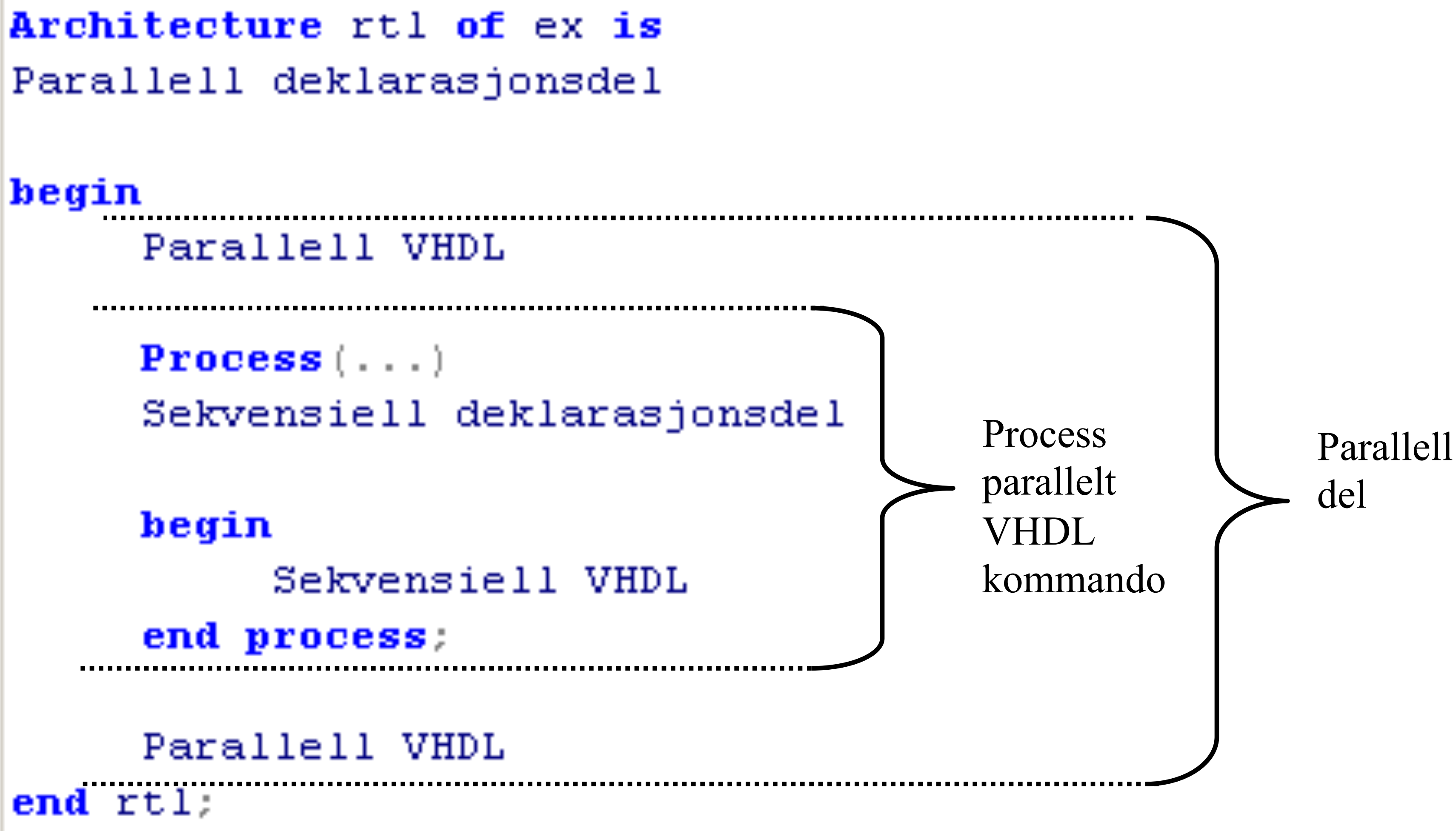


```
architecture CONCURRENT of MULTIPLE is
   signal Z, A, B, C, D : std_logic;
begin
      Z <= A and B;
      Z <= C and D;
end CONCURRENT;
```

Multiple drivers for one signal

Conflicting assignment must be resolved

Code structure

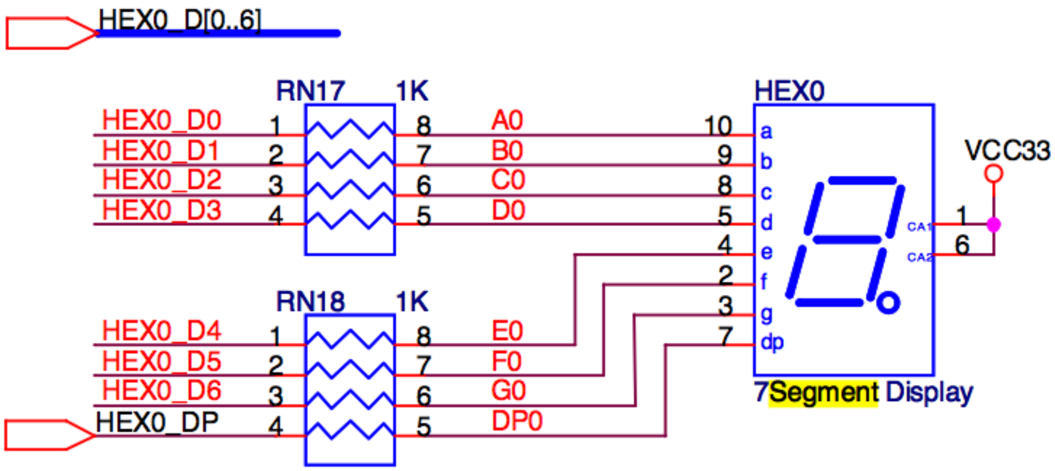


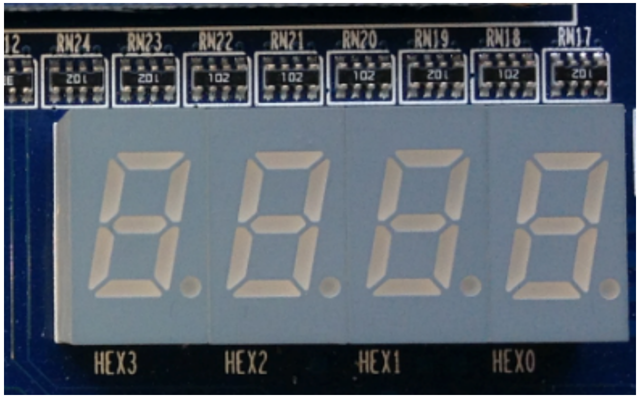
```
Architecture rtl of ex is
Parallell deklarasjonsdel

begin
     Parallell VHDL

     Process(...)
     Sekvensiell deklarasjonsdel

     begin
          Sekvensiell VHDL
     end process;

     Parallell VHDL
end rtl;
```

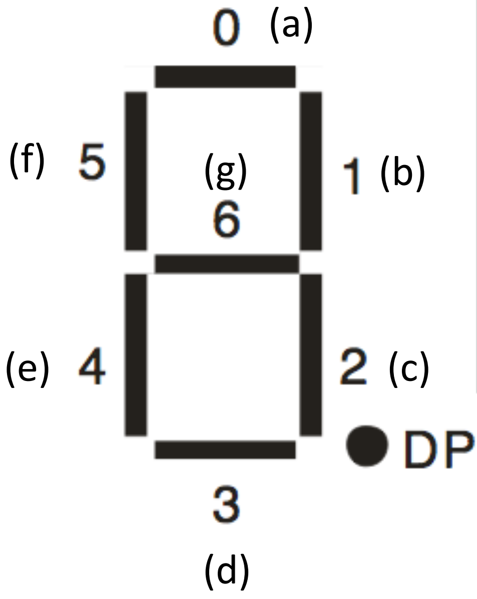
Process parallelt VHDL kommando

Parallell del

Ex.: 7-segment decoder



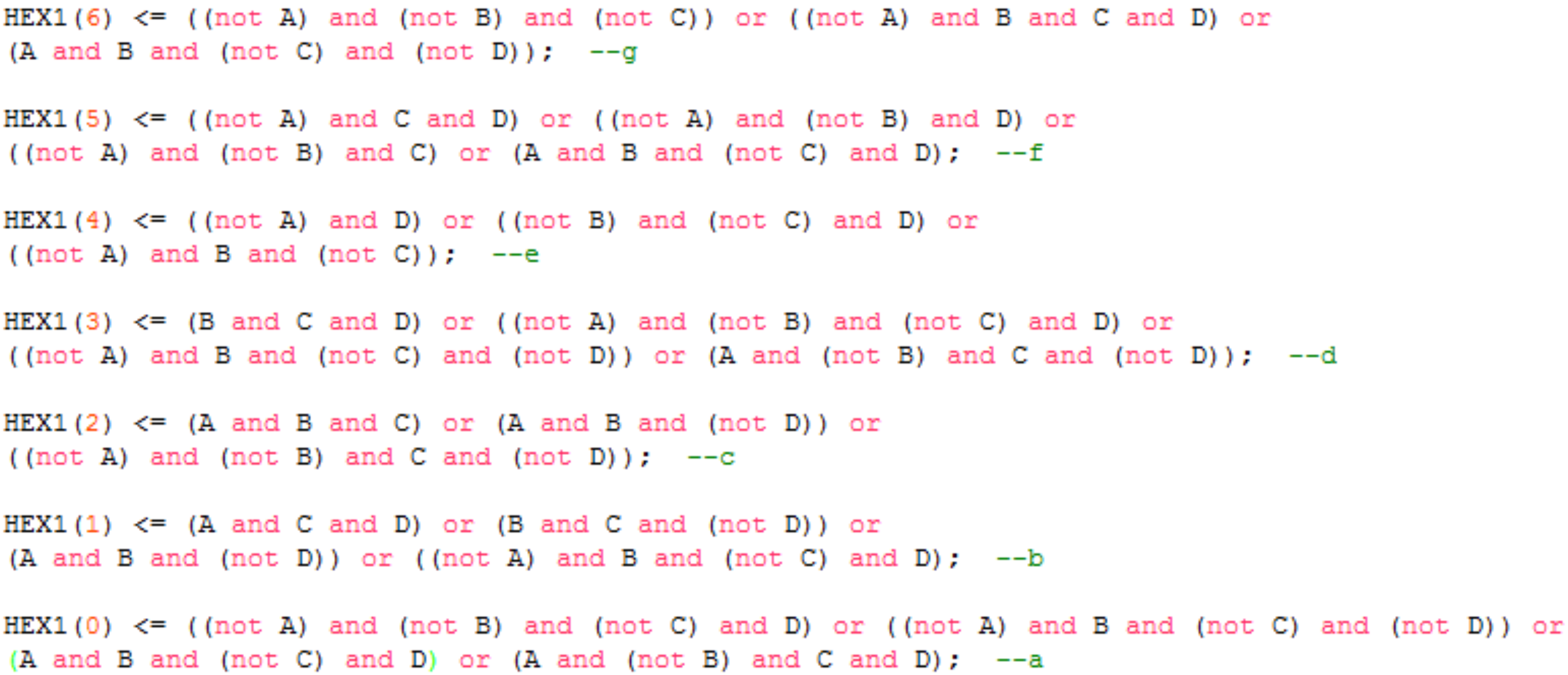




			0	1	2	3	4	5	6
Digit	**gfedcba**	**abcdefg**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0x3F	0x7E	**on**	**on**	**on**	**on**	**on**	**on**	off
1	0x06	0x30	off	**on**	**on**	off	off	off	off
2	0x5B	0x6D	**on**	**on**	off	**on**	**on**	off	**on**
3	0x4F	0x79	**on**	**on**	**on**	**on**	off	off	**on**
4	0x66	0x33	off	**on**	**on**	off	off	**on**	**on**
5	0x6D	0x5B	**on**	off	**on**	**on**	off	**on**	**on**
6	0x7D	0x5F	**on**	off	**on**	**on**	**on**	**on**	**on**
7	0x07	0x70	**on**	**on**	**on**	off	off	off	off
8	0x7F	0x7F	**on**	**on**	**on**	**on**	**on**	**on**	**on**
9	0x6F	0x7B	**on**	**on**	**on**	**on**	off	**on**	**on**
A	0x77	0x77	**on**	**on**	**on**	off	**on**	**on**	**on**
b	0x7C	0x1F	off	off	**on**	**on**	**on**	**on**	**on**
C	0x39	0x4E	**on**	off	off	**on**	**on**	**on**	off
d	0x5E	0x3D	off	**on**	**on**	**on**	**on**	off	**on**
E	0x79	0x4F	**on**	off	off	**on**	**on**	**on**	**on**
F	0x71	0x47	**on**	off	off	off	**on**	**on**	**on**

‘0’ → ON
‘1’ → OFF

Ex.: 7-segment decoder



```
HEX1(6) <= ((not A) and (not B) and (not C)) or ((not A) and B and C and D) or
(A and B and (not C) and (not D));  --g

HEX1(5) <= ((not A) and C and D) or ((not A) and (not B) and D) or
((not A) and (not B) and C) or (A and B and (not C) and D);  --f

HEX1(4) <= ((not A) and D) or ((not B) and (not C) and D) or
((not A) and B and (not C));  --e

HEX1(3) <= (B and C and D) or ((not A) and (not B) and (not C) and D) or
((not A) and B and (not C) and (not D)) or (A and (not B) and C and (not D));  --d

HEX1(2) <= (A and B and C) or (A and B and (not D)) or
((not A) and (not B) and C and (not D));  --c

HEX1(1) <= (A and C and D) or (B and C and (not D)) or
(A and B and (not D)) or ((not A) and B and (not C) and D);  --b

HEX1(0) <= ((not A) and (not B) and (not C) and D) or ((not A) and B and (not C) and (not D)) or
(A and B and (not C) and D) or (A and (not B) and C and D);  --a
```

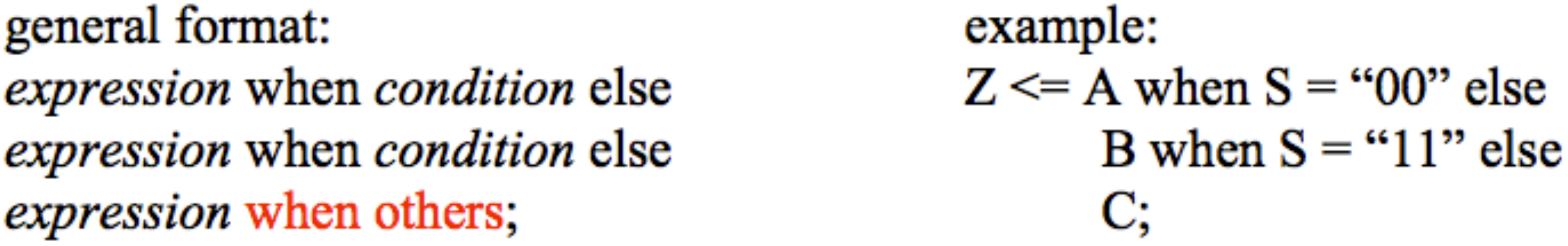
The use of logical operators and concurrent signal assignment is no longer comfortable.

VDHL

Non-procedural (data-flow)	Procedural (sequential)
• **Process statement** • **When else statement** • **With select statement** • **Signal declaration** • Block statement	• **If- then- else statement** • **Case statement** • **Variable declaration** • **Variable assignment** • Loop statement • Return statement • Null statement • Wait statement
Allowed in both non-procedural and procedural part	
• **Signal assignment** • **Declaration of types and constants** • Function and procedure calls • Assert statements	

When-else / with-select (non-procedural)

Conditional signal assignment



general format:
expression when *condition* else
expression when *condition* else
expression when others;

example:

```
Z <= A when S = “00” else
     B when S = “11” else
     C;
```

Selected signal assignment

general format:
with *selection* select
expression when *condition*,
expression when *condition*,
expression when others;

example:

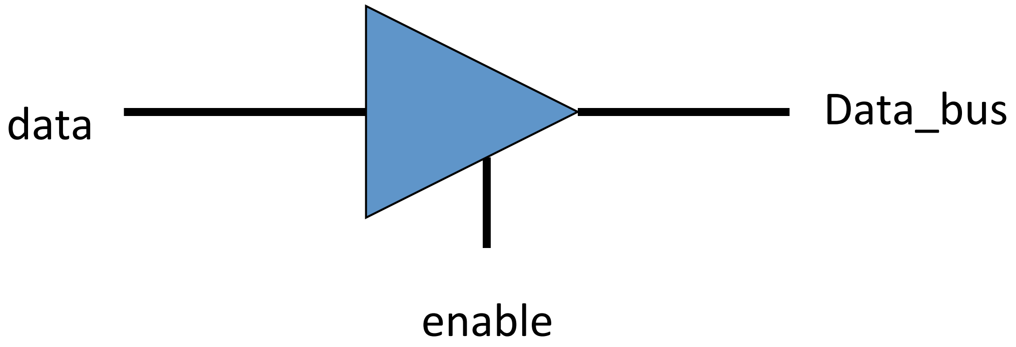
```
with S select
     Z <= A when “00” ,
          B when “11” ,
          C when others;
```

Ex. *when – else*: Tri-state buffer

- The output buffer can be put into a high impedance (’Z’) state, such that only one entity writes to the bus
    - Three possible signal levels: ’0’, ’1’, ’Z’

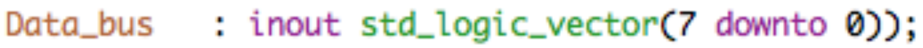
- FPGAs and CPLDs have three-state buffers on the outputs (the signals defined as ***port*** in the entity)

- However, many programmable logic devices can not have three-state buffers internally on the circuit (on internal signals)



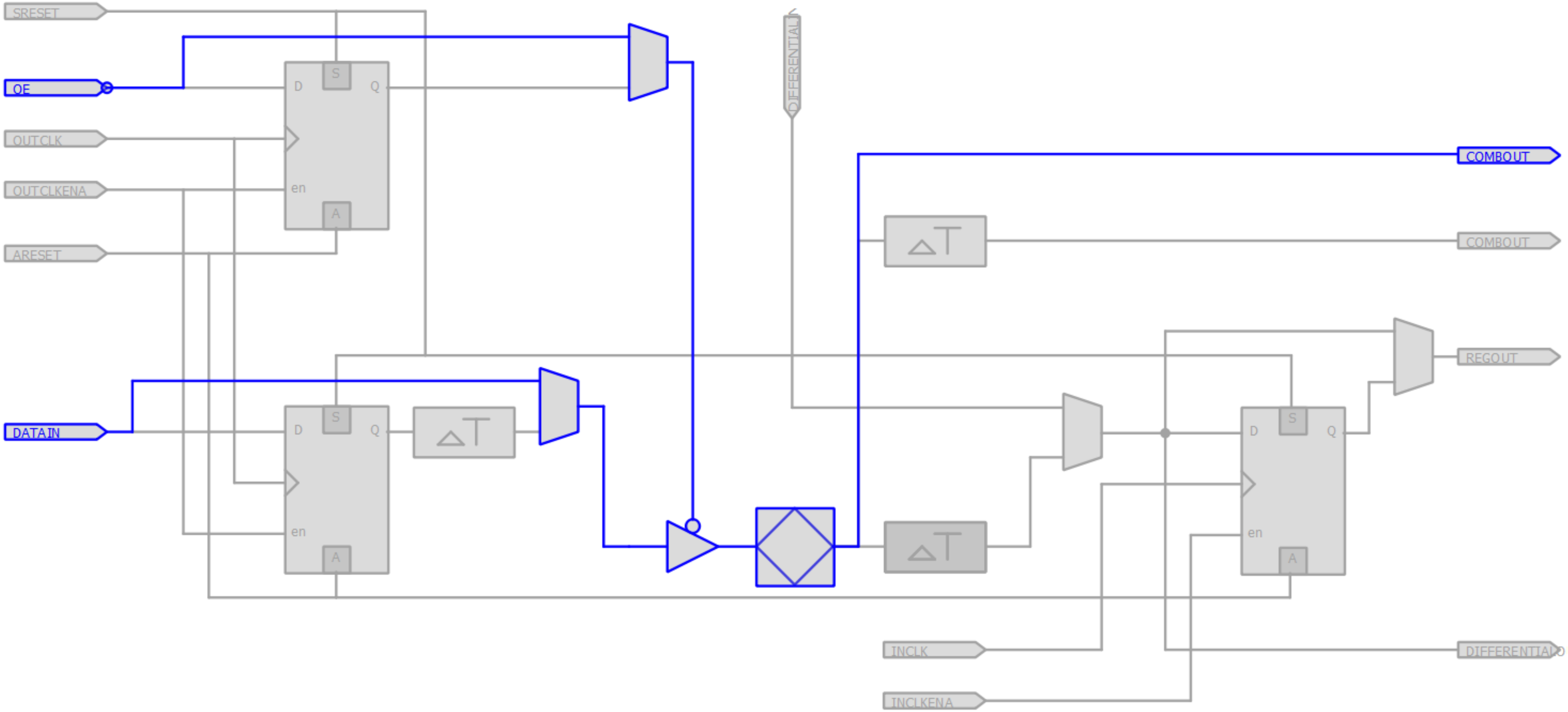
When – else

Data_bus <= data when enable = '1' else (others => ‘Z’);

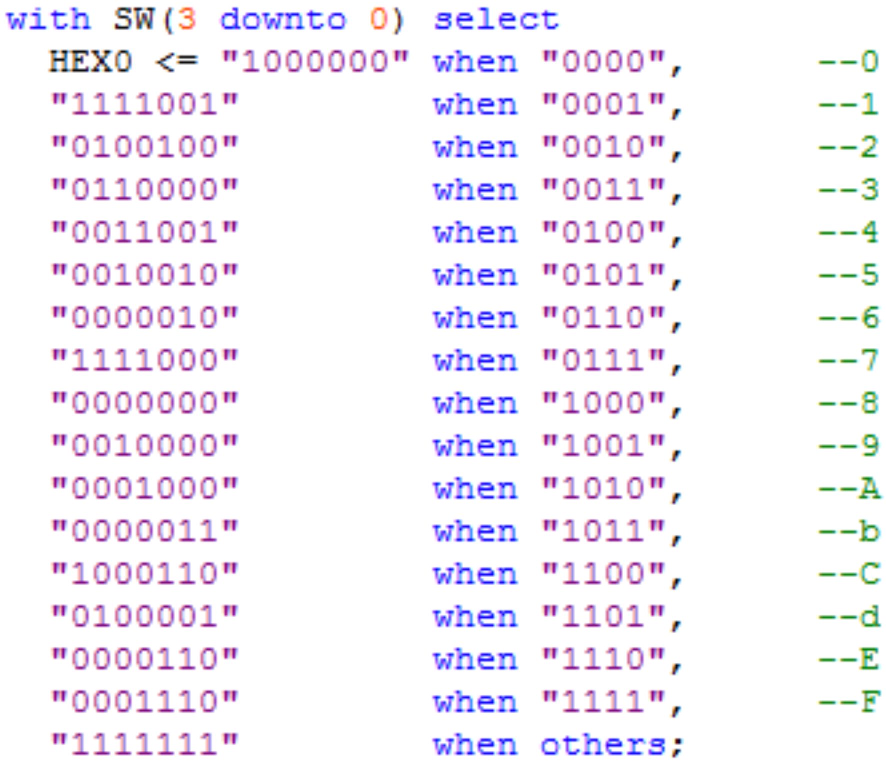


```
Data_bus    : inout std_logic_vector(7 downto 0));
```

Ex. *when – else*: Tri-state buffer

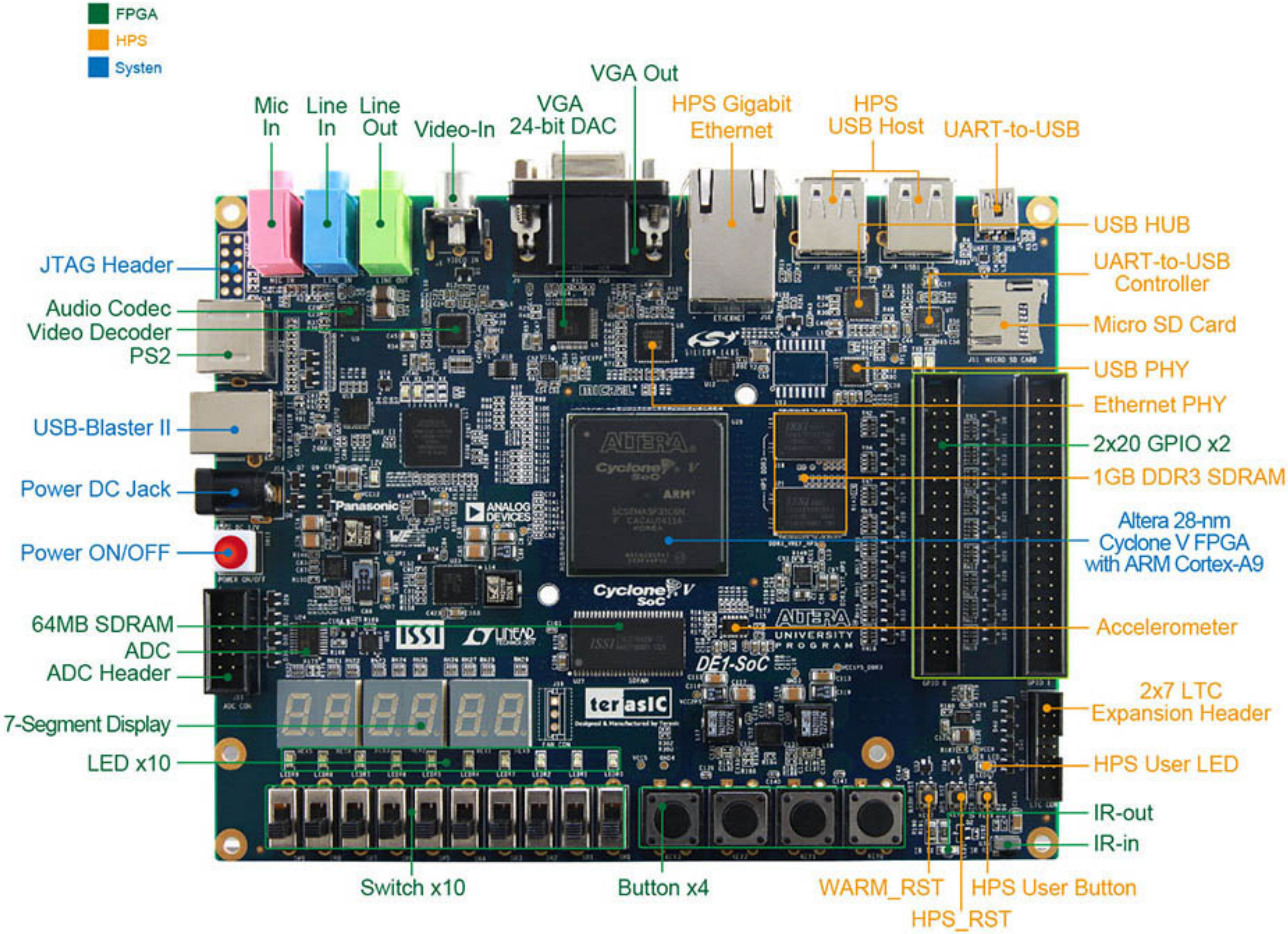


Ex.: 7-segment decoder



```
with SW(3 downto 0) select
  HEX0 <= "1000000" when "0000",       --0
  "1111001"         when "0001",       --1
  "0100100"         when "0010",       --2
  "0110000"         when "0011",       --3
  "0011001"         when "0100",       --4
  "0010010"         when "0101",       --5
  "0000010"         when "0110",       --6
  "1111000"         when "0111",       --7
  "0000000"         when "1000",       --8
  "0010000"         when "1001",       --9
  "0001000"         when "1010",       --A
  "0000011"         when "1011",       --b
  "1000110"         when "1100",       --C
  "0100001"         when "1101",       --d
  "0000110"         when "1110",       --E
  "0001110"         when "1111",       --F
  "1111111"         when others;
```

DE1-SoC board

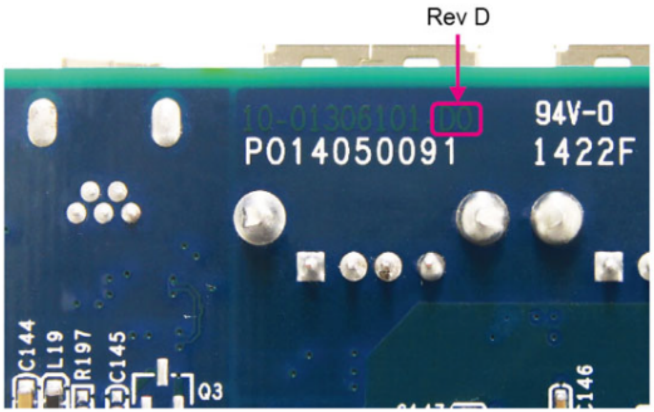


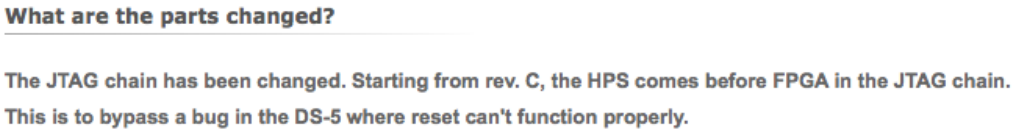
User Manual: **DE1-SoC User Manual(rev.C/rev.D Board)**

TERASIC: DE1-SoC webpage

DE1-SoC board revisions

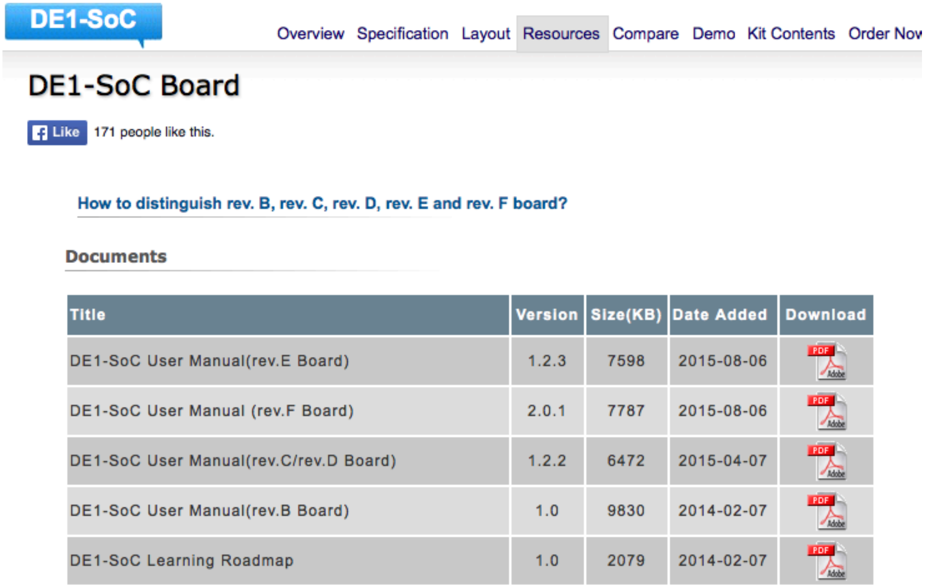
rev. D





What are the parts changed?

The JTAG chain has been changed. Starting from rev. C, the HPS comes before FPGA in the JTAG chain. This is to bypass a bug in the DS-5 where reset can't function properly.



DE1-SoC Board

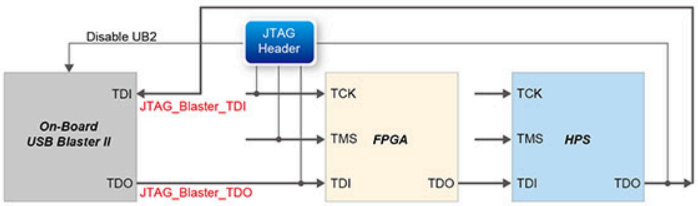
171 people like this.

How to distinguish rev. B, rev. C, rev. D, rev. E and rev. F board?

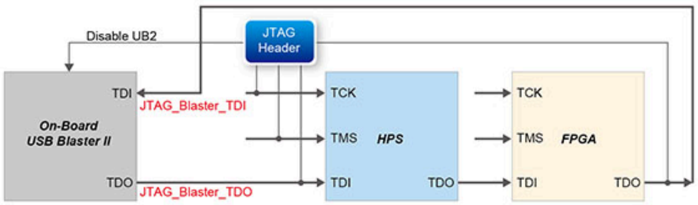
Documents

Title	Version	Size(KB)	Date Added	Download
DE1-SoC User Manual(rev.E Board)	1.2.3	7598	2015-08-06	PDF
DE1-SoC User Manual (rev.F Board)	2.0.1	7787	2015-08-06	PDF
DE1-SoC User Manual(rev.C/rev.D Board)	1.2.2	6472	2015-04-07	PDF
DE1-SoC User Manual(rev.B Board)	1.0	9830	2014-02-07	PDF
DE1-SoC Learning Roadmap	1.0	2079	2014-02-07	PDF

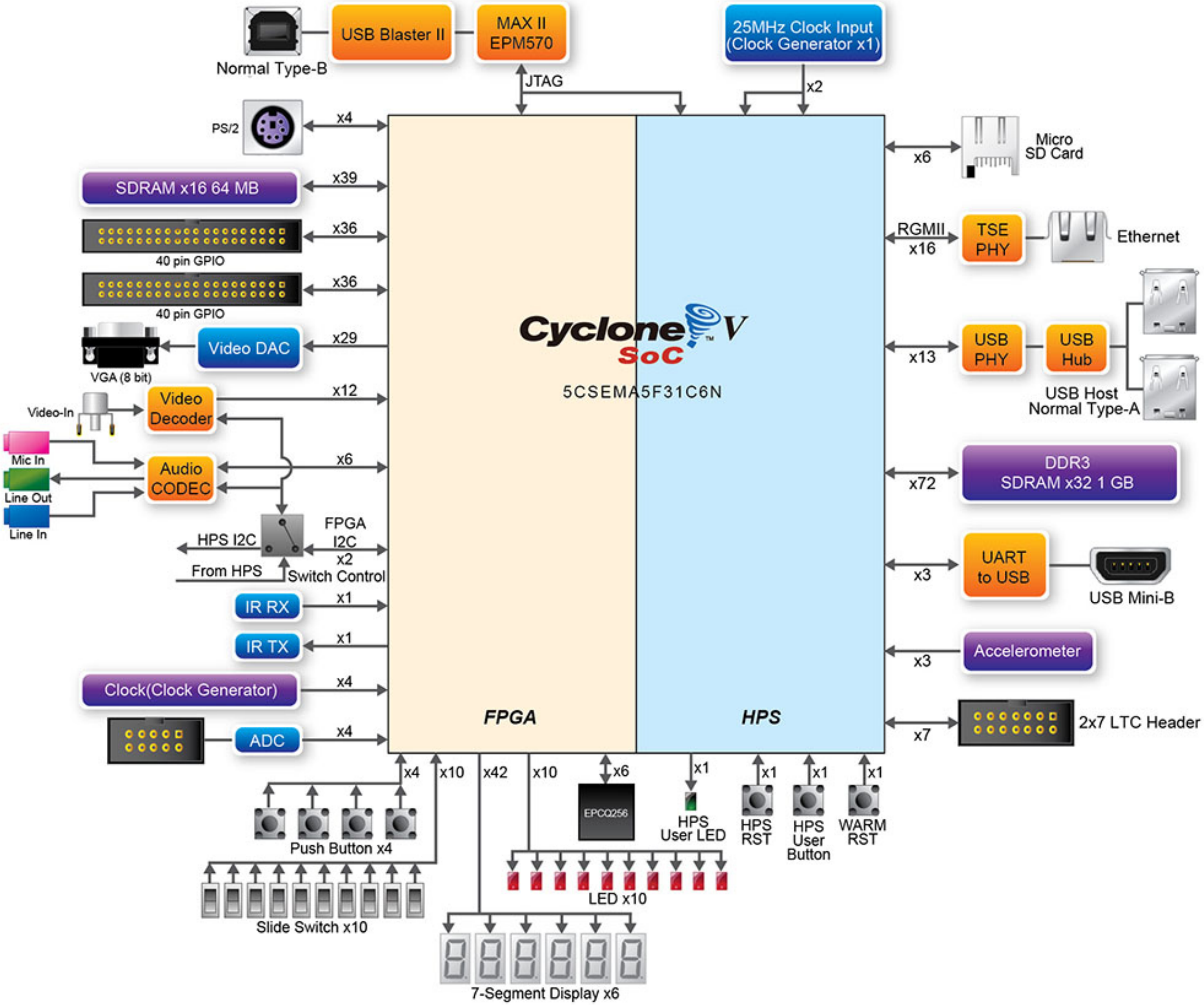
rev. B JTAG chain: USB Blaster II ---> FPGA ---> HPS ---> USB Blaster II



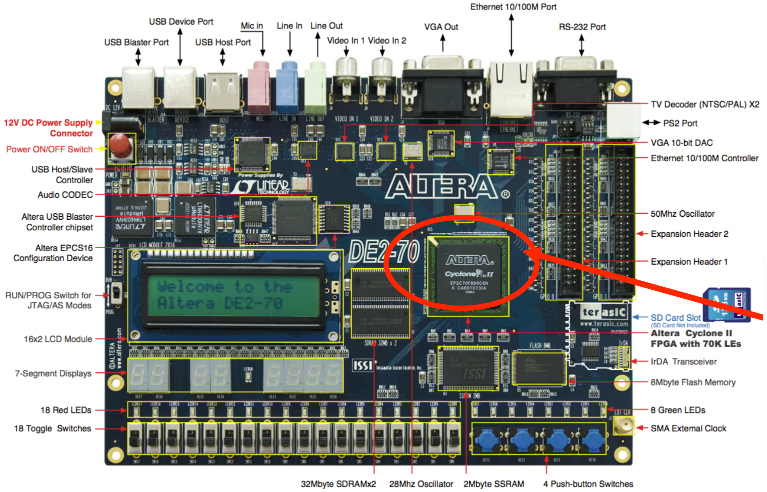
JTAG chain starting from rev. C: USB Blaster II ---> HPS ---> FPGA ---> USB Blaster II

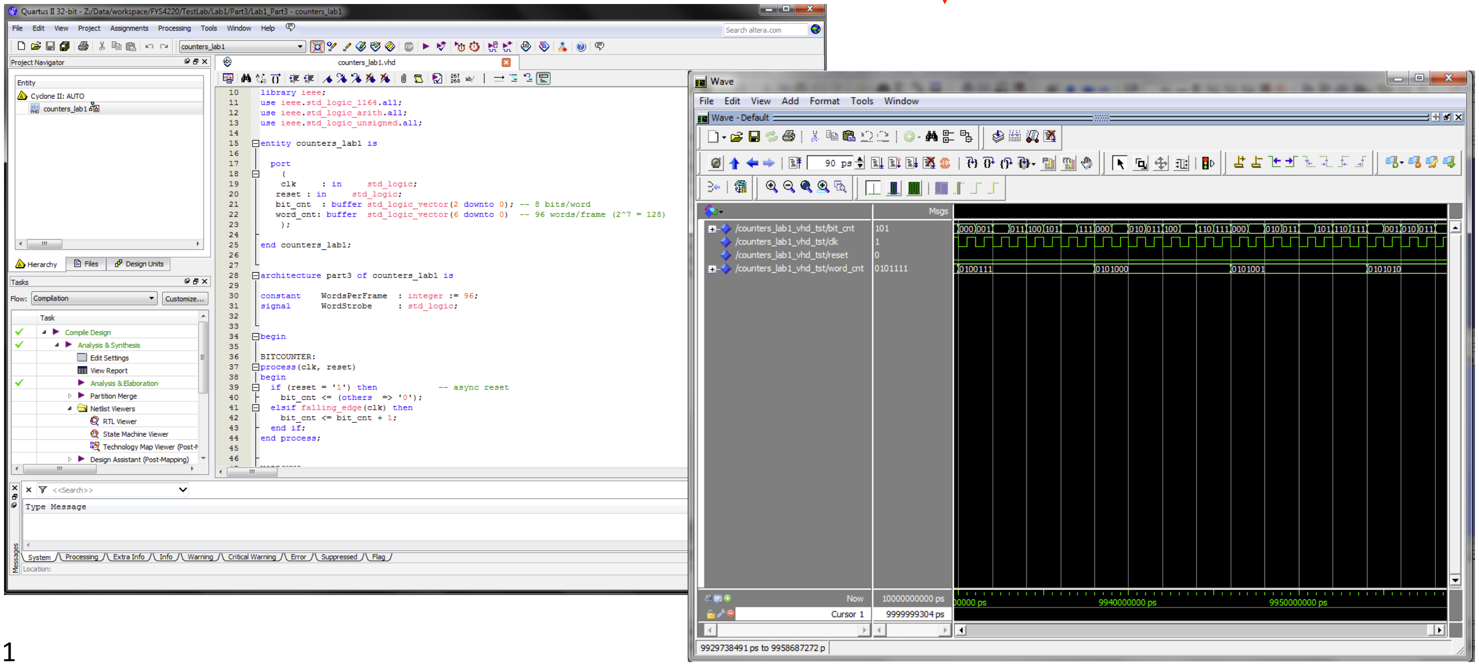


DE1-SoC board



Design using FPGA/CPLD



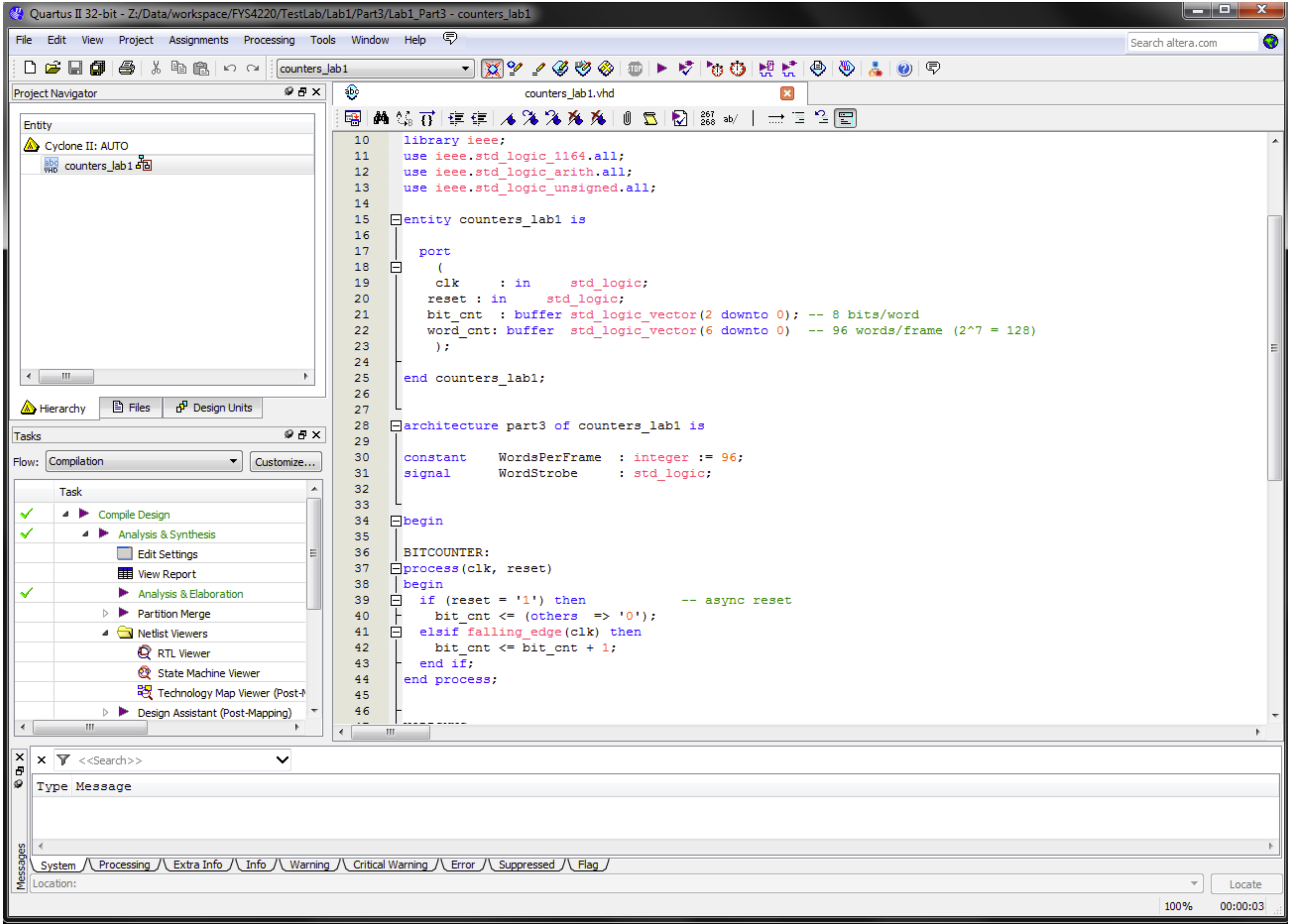


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counters_lab1 is

  port
    (
    clk     : in     std_logic;
   reset : in     std_logic;
   bit_cnt  : buffer std_logic_vector(2 downto 0); -- 8 bits/word
   word_cnt: buffer  std_logic_vector(6 downto 0)  -- 96 words/frame (2^7 = 128)
    );

end counters_lab1;


architecture part3 of counters_lab1 is

constant    WordsPerFrame  : integer := 96;
signal      WordStrobe     : std_logic;


begin

BITCOUNTER:
process(clk, reset)
begin
  if (reset = '1') then           -- async reset
    bit_cnt <= (others  => '0');
  elsif falling_edge(clk) then
    bit_cnt <= bit_cnt + 1;
  end if;
end process;
```

```
/counters_lab1_vhd_tst/bit_cnt    101
/counters_lab1_vhd_tst/clk        1
/counters_lab1_vhd_tst/reset      0
/counters_lab1_vhd_tst/word_cnt   0101111
Now       10000000000 ps
Cursor 1  9999999304 ps
9929738491 ps to 9958687272 p
```

Ref: JKB slides 2011

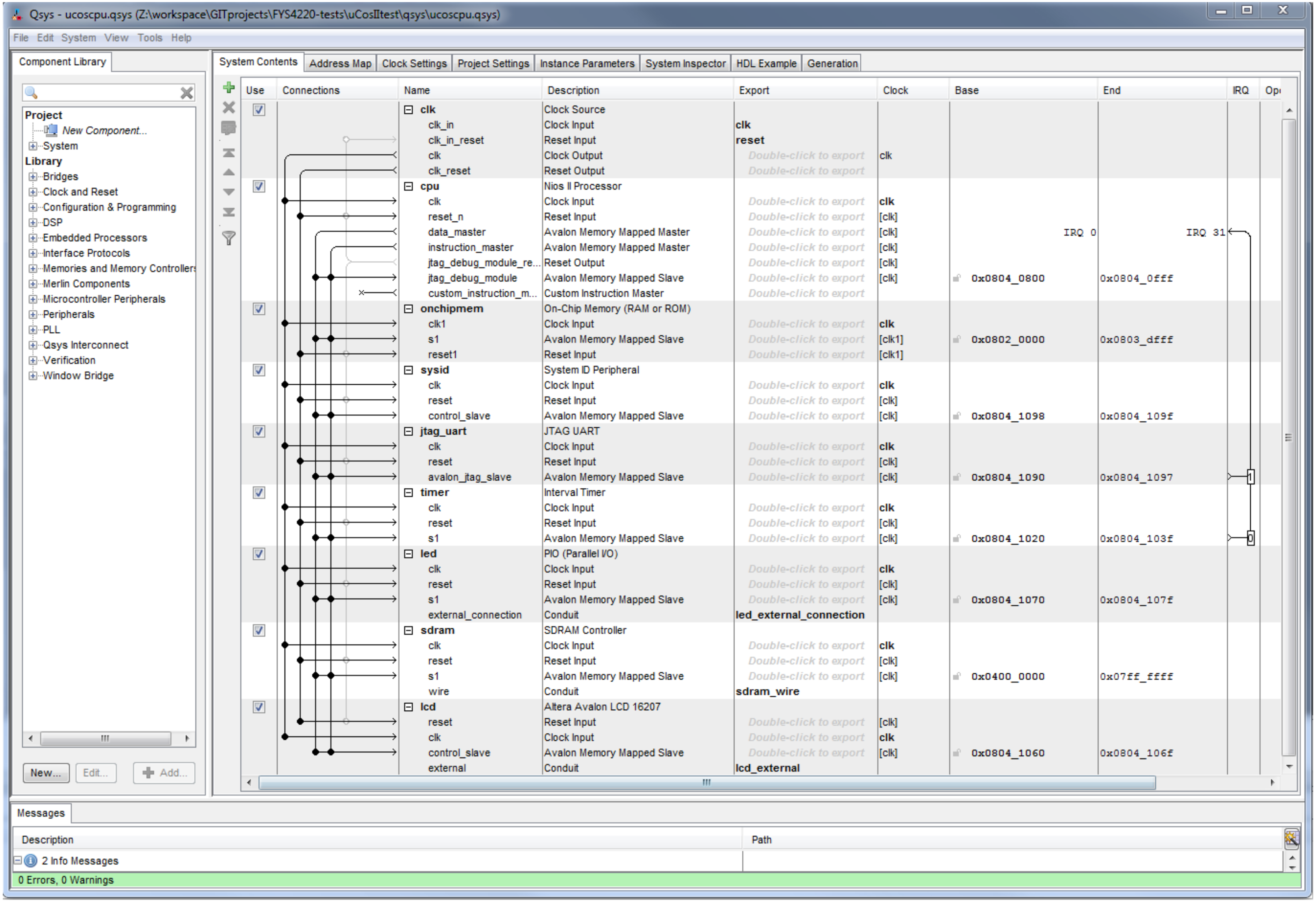
Quartus II Development software



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counters_lab1 is

  port
    (
    clk     : in     std_logic;
   reset : in     std_logic;
   bit_cnt  : buffer std_logic_vector(2 downto 0); -- 8 bits/word
   word_cnt: buffer  std_logic_vector(6 downto 0)   -- 96 words/frame (2^7 = 128)
    );

end counters_lab1;

architecture part3 of counters_lab1 is

constant    WordsPerFrame  : integer := 96;
signal      WordStrobe     : std_logic;


begin

BITCOUNTER:
process(clk, reset)
begin
  if (reset = '1') then            -- async reset
    bit_cnt <= (others   => '0');
  elsif falling_edge(clk) then
    bit_cnt <= bit_cnt + 1;
  end if;
end process;
```

```
Type Message
```

Quartus system integration tool (QSYS)



Qsys - ucoscpu.qsys (Z:\workspace\GITprojects\FYS4220-tests\uCosIItest\qsys\ucoscpu.qsys)

Project
- New Component...
- System

Library
- Bridges
- Clock and Reset
- Configuration & Programming
- DSP
- Embedded Processors
- Interface Protocols
- Memories and Memory Controllers
- Merlin Components
- Microcontroller Peripherals
- Peripherals
- PLL
- Qsys Interconnect
- Verification
- Window Bridge

System Contents | Address Map | Clock Settings | Project Settings | Instance Parameters | System Inspector | HDL Example | Generation

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Op
☑		⊟ clk	Clock Source						
		clk_in	Clock Input	clk					
		clk_in_reset	Reset Input	reset					
		clk	Clock Output	Double-click to export	clk				
		clk_reset	Reset Output	Double-click to export					
☑		⊟ cpu	Nios II Processor						
		clk	Clock Input	Double-click to export	clk				
		reset_n	Reset Input	Double-click to export	[clk]				
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]	IRQ 0	IRQ 31		
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		jtag_debug_module_re...	Reset Output	Double-click to export	[clk]				
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_0800	0x0804_0fff		
		custom_instruction_m...	Custom Instruction Master	Double-click to export					
☑		⊟ onchipmem	On-Chip Memory (RAM or ROM)						
		clk1	Clock Input	Double-click to export	clk				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0802_0000	0x0803_dfff		
		reset1	Reset Input	Double-click to export	[clk1]				
☑		⊟ sysid	System ID Peripheral						
		clk	Clock Input	Double-click to export	clk				
		reset	Reset Input	Double-click to export	[clk]				
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_1098	0x0804_109f		
☑		⊟ jtag_uart	JTAG UART						
		clk	Clock Input	Double-click to export	clk				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_1090	0x0804_1097	1	
☑		⊟ timer	Interval Timer						
		clk	Clock Input	Double-click to export	clk				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_1020	0x0804_103f	0	
☑		⊟ led	PIO (Parallel I/O)						
		clk	Clock Input	Double-click to export	clk				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_1070	0x0804_107f		
		external_connection	Conduit	led_external_connection					
☑		⊟ sdram	SDRAM Controller						
		clk	Clock Input	Double-click to export	clk				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0400_0000	0x07ff_ffff		
		wire	Conduit	sdram_wire					
☑		⊟ lcd	Altera Avalon LCD 16207						
		reset	Reset Input	Double-click to export	[clk]				
		clk	Clock Input	Double-click to export	clk				
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0804_1060	0x0804_106f		
		external	Conduit	lcd_external					

Messages

Description	Path
⊟ 2 Info Messages	

0 Errors, 0 Warnings

Development software

- Quartus II Web Edition and Modelsim-Altera Starter Edition software can be downloaded for free from the Altera web page

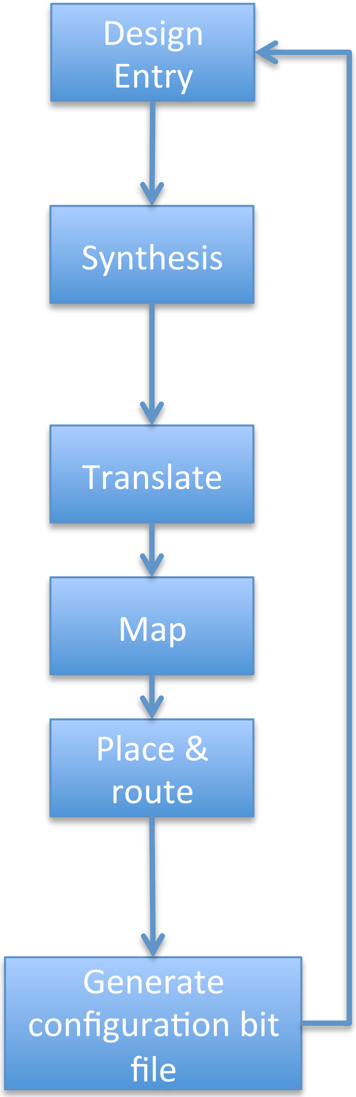
- Contains also QSYS and NIOSII EDS

- Latest version is v15.0 (14.0 installed in lab.)

QSYS: Quartus system integration tool
EDS: Embedded Design suite

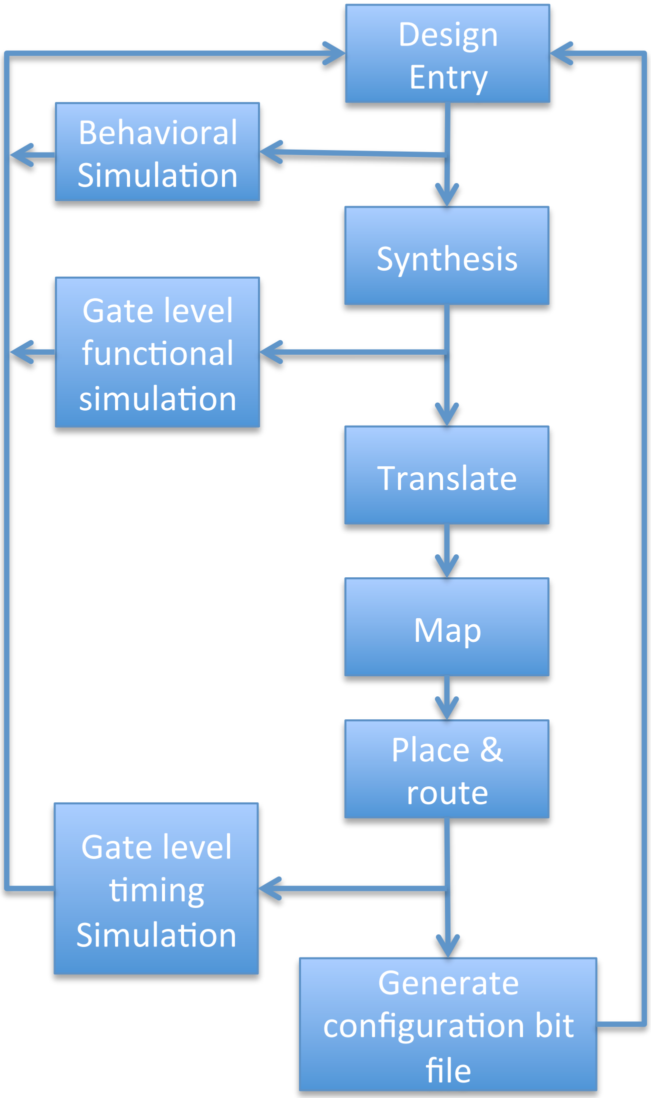
General design flow steps

- Design entry
  - Register Transfer Level (RTL) description of design (schematic or HDL)
- Synthesis
  - Checks code syntax, converts abstract form of desired circuit behavior into a design implementation of basic gate level primitives (netlist), i.e. circuit logic elements (gates, flip-flops, etc). A netlist is a text-based representation of a logic diagram.
- Translate
  - Merges netlist and constraints (e.g. physical port assignment, timing) into device specific design file.
- Map
  - Fits the design into specific device resources (LUT, FF, RAM etc)
- Place and route
  - Decides where in the die the resources will be placed and wires them together (accounts for timing constraints)
- Generate configuration bit file
  - That can be downloaded to the FPGA



General simulation steps

- Behavioral simulation
  - Simulation to verify RTL behavioral code (no timing and resource information)
- Gate level functional simulation
  - Run simulation on gate level description generated by the synthesizer.
  - Can discover improper coding that works at RTL level but which violates synthesis coding conventions.
    - E.g. omitting a signal in the sensitivity list.
- Gate level timing simulation
  - Gate level simulation with propagation delays

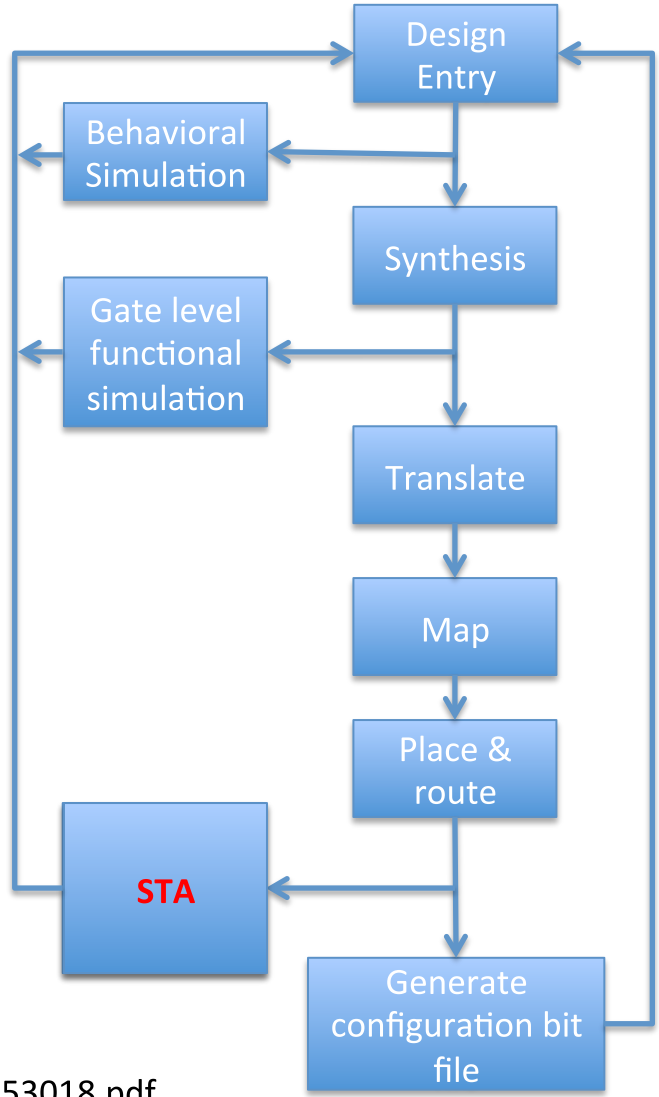

Design Entry
Behavioral Simulation
Synthesis
Gate level functional simulation
Translate
Map
Place & route
Gate level timing Simulation
Generate configuration bit file

Static Timing Analysis

- Gate level timing simulation of an entire design can be slow and should be avoided.

- In fact, not supported for Cylcone/Arria/Stratix V devices.

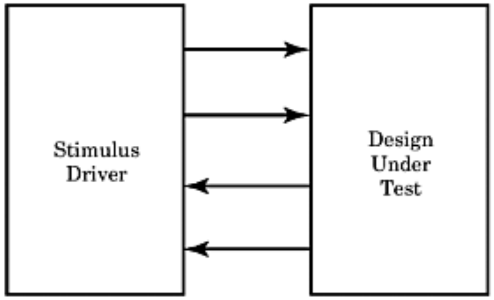
- Instead, use Static Timing Analysis (STA)
  - method of computing the expected timing of a digital circuit without requiring simulation
  - Considers timing of paths from e.g. **register to register**, **input port to register**, **register to output port**, **purely combinational paths**.
  - No need for test vectors
  - However, does not check functionality of design. => combine STA with behavioral simulation (RTL).

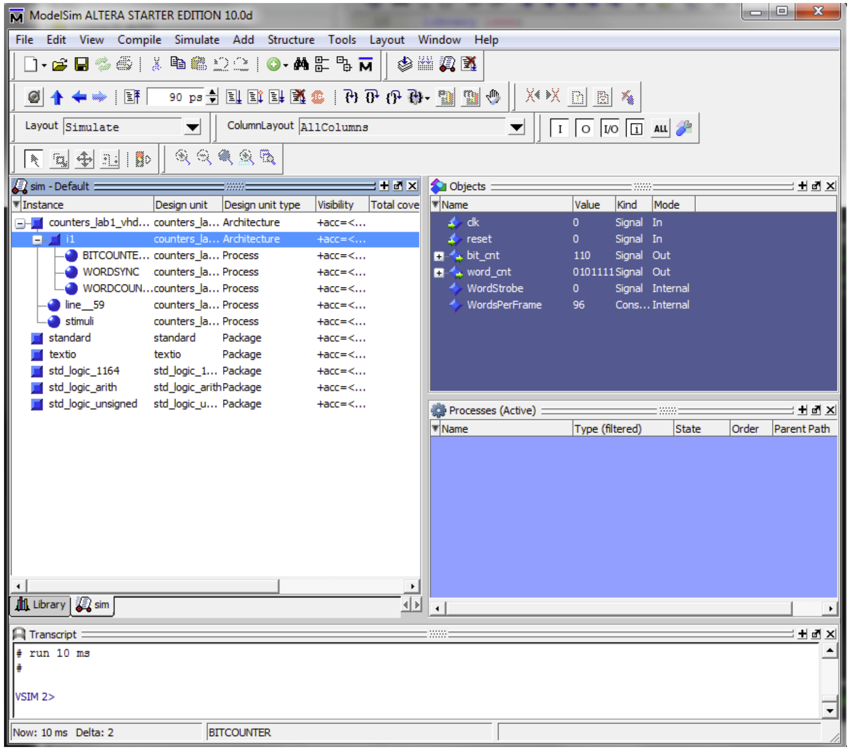


TimeQuest Timing Analyzer: http://www.altera.com/literature/hb/qts/qts_qii53018.pdf

How to simulate VHDL code

- Use a simulation tool like e.g. ModelSim
- Test bench to apply stimuli/test inputs to the VHDL code
- Visual inspection through graphical output (waveform)
- Self-checking test benches (add code to check and verify result)




ModelSim ALTERA STARTER EDITION 10.0d
File Edit View Compile Simulate Add Structure Tools Layout Window Help
90 ps
I O I/O i ALL
Library
sim

Layout Simulate   ColumnLayout AllColumns

sim - Default

Instance	Design unit	Design unit type	Visibility	Total cove
counters_lab1_vhd...	counters_la...	Architecture	+acc=<...	
i1	counters_la...	Architecture	+acc=<...	
BITCOUNTE...	counters_la...	Process	+acc=<...	
WORDSYNC	counters_la...	Process	+acc=<...	
WORDCOUN...	counters_la...	Process	+acc=<...	
line__59	counters_la...	Process	+acc=<...	
stimuli	counters_la...	Process	+acc=<...	
standard	standard	Package	+acc=<...	
textio	textio	Package	+acc=<...	
std_logic_1164	std_logic_1...	Package	+acc=<...	
std_logic_arith	std_logic_arith	Package	+acc=<...	
std_logic_unsigned	std_logic_u...	Package	+acc=<...	

Objects

Name	Value	Kind	Mode
clk	0	Signal	In
reset	0	Signal	In
bit_cnt	110	Signal	Out
word_cnt	0101111	Signal	Out
WordStrobe	0	Signal	Internal
WordsPerFrame	96	Cons...	Internal

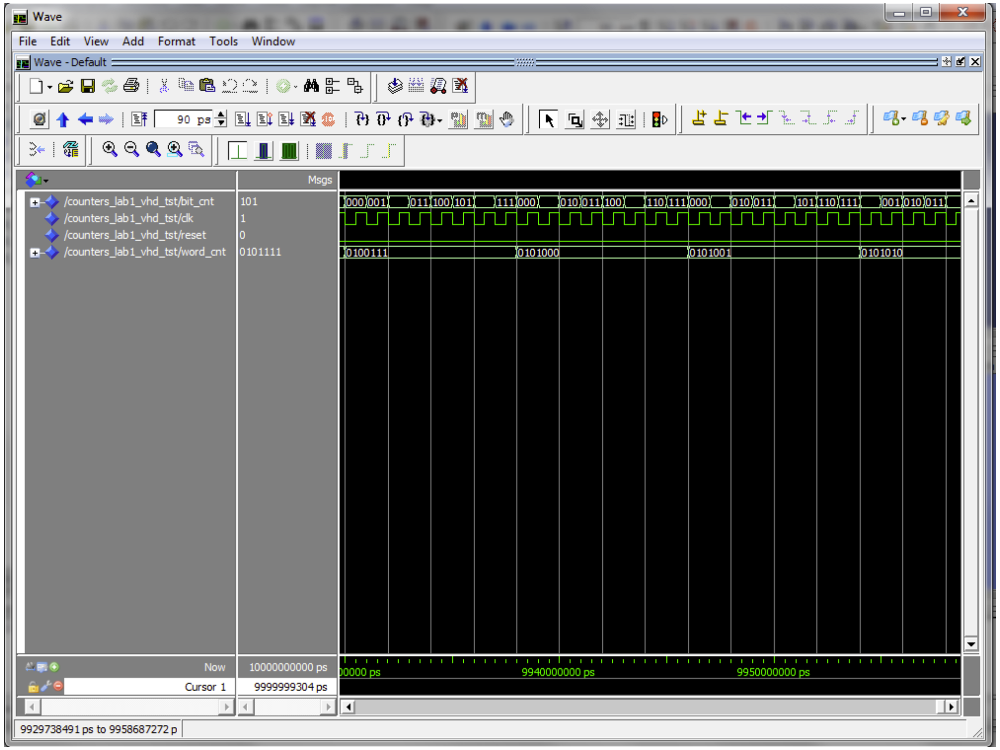
Processes (Active)

Name	Type (filtered)	State	Order	Parent Path

Transcript

```
# run 10 ms
#

VSIM 2>
```

Now: 10 ms Delta: 2   BITCOUNTER

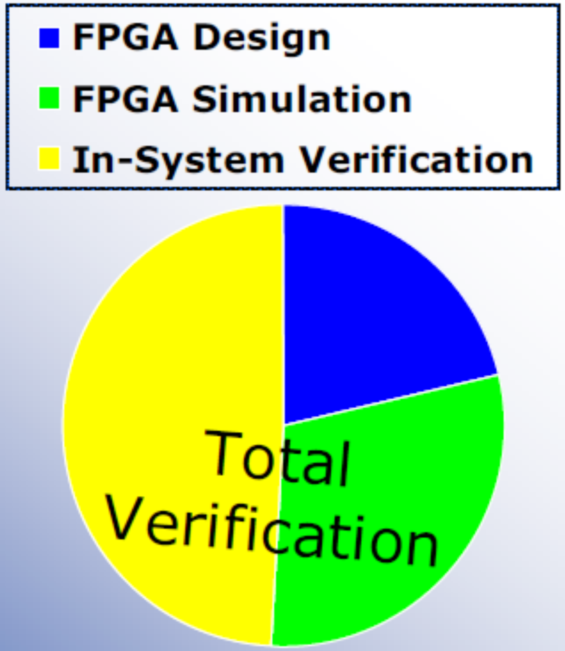


	Msgs
/counters_lab1_vhd_tst/bit_cnt	101
/counters_lab1_vhd_tst/clk	1
/counters_lab1_vhd_tst/reset	0
/counters_lab1_vhd_tst/word_cnt	0101111
Now	10000000000 ps
Cursor 1	9999999304 ps

9929738491 ps to 9958687272 p

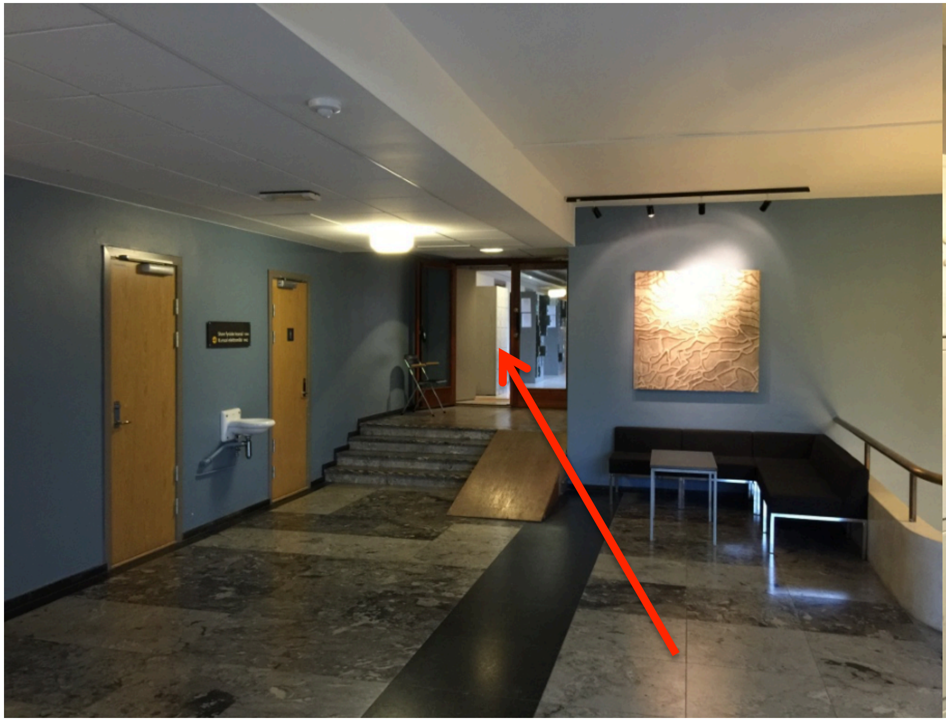
Why focus on Verification

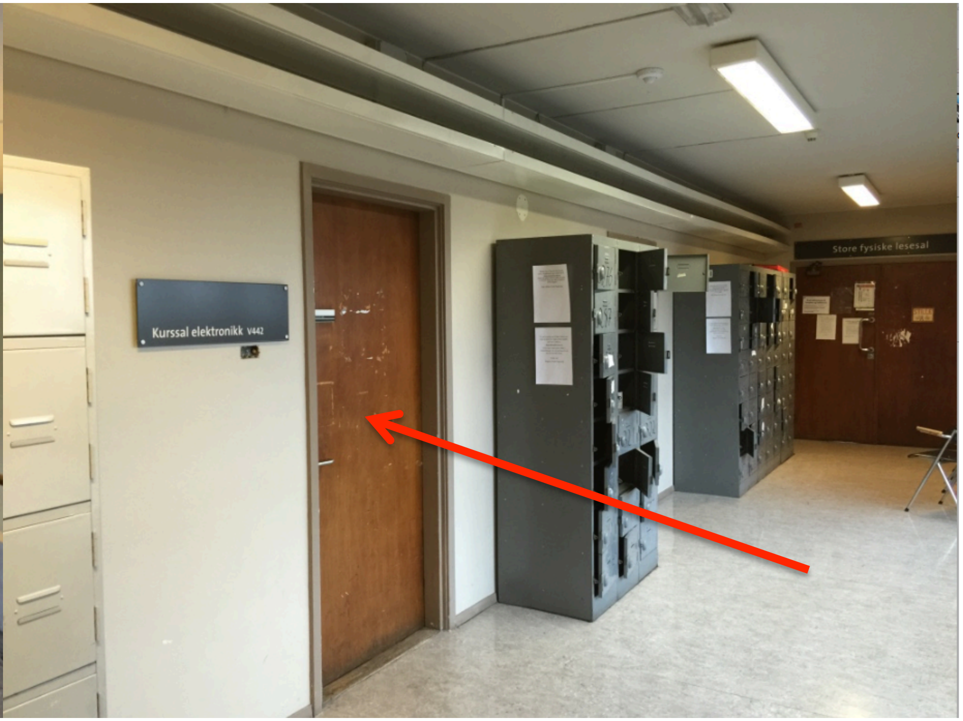
Consider a) FPGA development and
b) Further work related to FPGA quality



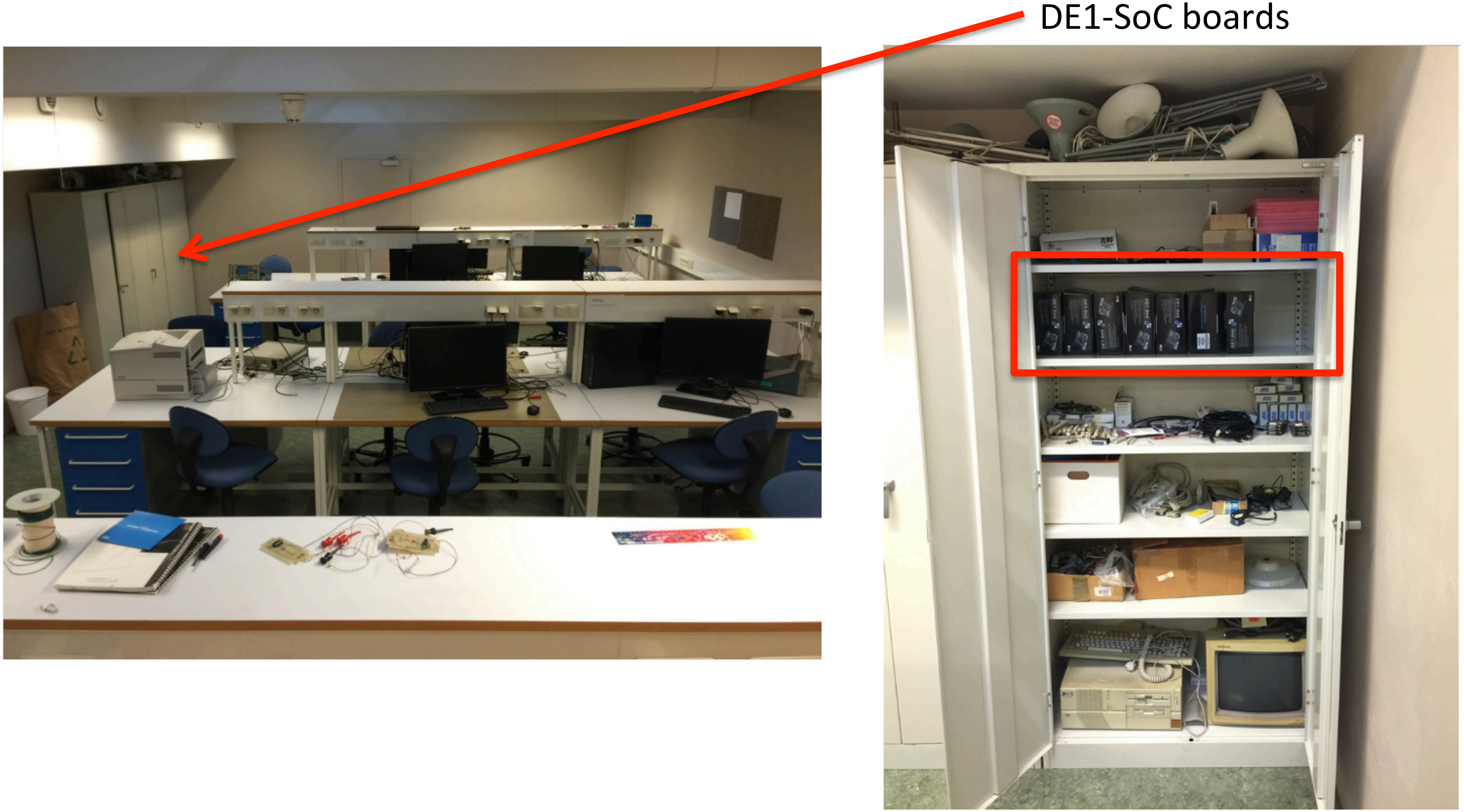
Average Design & Functional Verification tasks
- as seen in some reasonably structured projects.

Room V442, 4th floor of Physics building





Please keep the lab. Clean and tidy



DE1-SoC boards