PLD (eg. PAL)

Typically 8 logic elements

Technology: AND-OR array
CPLD (eg. MAX)

Typically 64 Macrocells

Technology: AND-OR array

(larger MAX circuits uses MUX-tree technique)
Gates with many inputs?

PAL matrix has gates with so many inputs that you have to draw them in a "simplified way".

Everyone must lead to get "0" = slow when there are many inputs.
As bad with CMOS NOR

Everyone must lead to get "1" = slow when there are many inputs

(one of them must lead to get "0" = fast)

William Sandqvist  william@kth.se
Fast, but with high power dissipation

"Pull-Up" resistor provides "1" = Fast but "Power Hungry" when output is "0"

Just one must lead to get "0"?

= Fast

William Sandqvist  william@kth.se
Large programmable circuits

There is therefore a need for other techniques not based on gates with many inputs, in order to be able to build large programmable circuits in CMOS technology!

William Sandqvist  william@kth.se
FPGA (eg. Cyclone II)

Typically **50000** logic elements

Technology: **MUX** tree

William Sandqvist  william@kth.se
The Multiplexor  MUX

The multiplexer can select which input you are going to connect to the output.

MUX is now the "standard component" in the development of Digital Logic.

$$Z = SX + \overline{S}Y$$
Multiplexer MUX

To the right we have a MUX in rope technology - April 1-joke from Scientific American!

\[ Z = SX + S'Y \]

Did you recognize that it was a MUX that was the "secret" circuit at LAB1?

William Sandqvist  william@kth.se
Logical functions with MUX

How can the following functions be implemented with a 2:1 Multiplexer?

\[ Z = \overline{x_0} \quad \text{NOT} \]
\[ Z = x_1 \cdot x_0 \quad \text{AND} \]
\[ Z = x_1 + x_0 \quad \text{OR} \]
\[ Z = x_1 \oplus x_0 \quad \text{XOR} \]

\[ Z = SX + \overline{S}Y \]
Quickie Question …

How to connect the inputs of the MUX in order to implement an inverter?

Desired function: \( z = \overline{x} \)
Invertering NOT with MUX

Specification:

if input = '1' then result <= '0'
if input = '0' then result <= '1';

\[
Z = S \cdot X + \overline{S} \cdot Y = \\
= x_0 \cdot 0 + x_0 \cdot 1 = \overline{x_0} \quad NOT
\]

William Sandqvist  william@kth.se
Quickie Question …

How to connect the inputs of the MUX in order to implement an and gate?

Desired function: \( z = xy \)
**AND-function with MUX**

**Specification:**

$$Z = x_1 \cdot x_0 \quad \text{AND}$$

$$Z = SX + \overline{SY} = x_1 \cdot x_0 + \overline{x_1} \cdot 0 = x_1 \cdot x_0$$
OR-function with MUX

Specification:

\[
Z = x_1 + x_0 \quad OR
\]

\[
Z = x_1 x_0 + x_1 x_0 + x_1 x_0 =
\]

\[
= \{SX + \overline{SY}\} = x_1 (x_0 + \overline{x_0}) + \overline{x_1} \cdot x_0 =
\]

\[
= x_1 \cdot 1 + x_1 \cdot \overline{x_0}
\]

William Sandqvist  william@kth.se
**XOR-function with MUX**

**Specification:**

\[
Z = x_1 \oplus x_0 \quad XOR
\]

\[
Z = SX + \overline{S}Y = x_1 \cdot \overline{x}_0 + x_1 \cdot x_0 = x_1 \oplus x_0
\]

William Sandqvist  william@kth.se
Hierarchies of muxes

William Sandqvist  william@kth.se
Choose any of the inputs as address inputs ...  

\[ f = \overline{z}x + \overline{xy} + zy \]

... And minimize / implement function that occur? For each input. Draw new Karnaugh diagrams if necessary.

An \((n + 1)\)-input function could be implemented with a MUX having \(n\) select-inputs!
Shannon decomposition

Claude Shannon
mathematician / electrical engineer  (1916 –2001)
Shannon decomposition

A Boolean function \( f(x_n, \ldots, x_1, x_0) \) can be divided to

\[
 f(x_n, \ldots, x_1, x_0) = x_0 \cdot f(x_n, \ldots, x_1, 1) + \overline{x_0} \cdot f(x_n, \ldots, x_1, 0)
\]

The function can then be implemented with a multiplexer.
Recursively

All Boolean functions $f(x_n, \ldots, x_1, x_0)$ can be broken down (recursively) to

\[
f(x_n, \ldots, x_1, x_0) = x_0 \cdot f_1(x_n, \ldots, x_1, 1) + \overline{x_0} \cdot f_0(x_n, \ldots, x_1, 0) \\
f_1(x_n, \ldots, x_1) = x_1 \cdot f_{11}(x_n, \ldots, x_2, 1) + x_1 \cdot f_{10}(x_n, \ldots, x_2, 0)
\]

and so on. $f_{0000} \ldots$

William Sandqvist william@kth.se
Proof

\[ f(x_n, \ldots, x_1, x_0) = x_0 \cdot f(x_n, \ldots, x_1, 1) + \overline{x_0} \cdot f(x_n, \ldots, x_1, 0) \]

Right hand side:
- If \( x_0 = 1 \) so is the right term zero. Then the \( f \) is equal to the left term.
- If \( x_0 = 0 \) so is the left term zero. Then the \( f \) is equal to the right term.

Left hand side:
- If \( x_0 = 1 \) so is \( f \) equal to \( f(x_n, \ldots, x_1, 1) \) (= left term on right side)
- If \( x_0 = 0 \) so is \( f \) equal to \( f(x_n, \ldots, x_1, 0) \) (= right term on right side)

LHS = RHS
One can see $xyz$ as an address, to the squares in the Karnaugh map. With values $1/0$ from the squares to the inputs of the MUX the function $f$ is realized.
Look-up-table (LUT)

A LUT with \( n \) inputs can realize all combinational functions with \( n \) inputs.

Two-input LUT

William Sandqvist  william@kth.se
LUT for XOR-gate

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Two-input LUT

William Sandqvist  william@kth.se
A simple FPGA cell

The simplest FPGA cell consists of a single Look-Up-Table - LUT, a D flip-flop and a bypass mux. D flip-flop is a memory circuit for synchronization – it will come later in the course. With the bypass mux one can exclude the D flip-flop when not needed.

D-flipflop will be explained soon in this course
The functions that are stored in a LUT are usually numbered after the number that is made up of the 1's in the truth table / Karnaugh map.

\[ f(x_3, x_2, x_1, x_0) = "011010011100101110" = f_{6996} \]
LUT function number

\[ f(x_3, x_2, x_1, x_0) = "0110100110010110" = f_{6996} \]

Now you know which function that has the number 6996!

With a LUT, all functions are realized, so none of them are more difficult to make than any other!

Odd parity!
No mergings are possible.

William Sandqvist  william@kth.se
Decoder

Mostly used as address decoder
Only one output is active when the 'enable' (en) is active
The active output is selected with $a_1a_0$

<table>
<thead>
<tr>
<th>en</th>
<th>$a_1$</th>
<th>$a_0$</th>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

2-to-4 decoder
The demultiplexer has basically the same function as the decoder, but is drawn differently ... The input is connected to a selected output

<table>
<thead>
<tr>
<th>$l_0$</th>
<th>$s_1$</th>
<th>$s_0$</th>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Now en is named $I_0$!
### Read-only-memory (ROM)

<table>
<thead>
<tr>
<th>Sel&lt;sub&gt;0&lt;/sub&gt;</th>
<th>0/1</th>
<th>0/1</th>
<th>...</th>
<th>0/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sel&lt;sub&gt;1&lt;/sub&gt;</td>
<td>0/1</td>
<td>0/1</td>
<td>...</td>
<td>0/1</td>
</tr>
<tr>
<td>a&lt;sub&gt;0&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a&lt;sub&gt;1&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a&lt;sub&gt;m&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sel&lt;sub&gt;2m&lt;/sub&gt;</td>
<td>0/1</td>
<td>0/1</td>
<td>...</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Decoder

En

d<sub>n-1</sub> d<sub>n-2</sub> ... d<sub>0</sub>

Programable bits

Threestate buffers

William Sandqvist  william@kth.se
Encoder

Encoders have the opposite function as a decoder, ie it translates $2^N$ bit input to a $N$-bit code.

- The information is concentrated

Eg. Keyboard with 16 (2^4) keys

4-bit HEX-code for the pressed key

$2^n$ inputs

$w_0$ $w_1$ $w_{2n-1}$

$y_0$ $y_1$ $y_{n-1}$

$n$ outputs
Priority Encoder

A Priority Encoder gives back the address of the input with the lowest (or highest) indices that are set to a one (or zero depending on what you are looking for). If all inputs are 0, the output $f = 0$, else $f$ has the value $= 1$.

What if you press several keys at the same time?

<table>
<thead>
<tr>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$f$</th>
<th>$a_1$</th>
<th>$a_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Now it will be well-defined what happens if several inputs are active.
Ex 8.4 7-4-2-1 code

Codeconverter 7-4-2-1-code to BCD-code.

When encoding the digits 0 ... 9 sometimes a code with weights 7-4-2-1 was used instead of the binary code with the weights 8-4-2-1.

In cases where a digit of the code word could be selected in various ways, the code word containing the least number of ones is chosen.

(a variant of the 7-4-2-1 code is used today to store the bar code)

You will construct such an encoder at exercise 3.
Code-converter

Code-converters translates from one code to another. Typical examples are:

- Binary to BCD (Binary-Coded Decimal)
- Binary to Gray-code
- BCD or BIN to 7-segment decoder
Ex 8.5 One of the segments ”g”

7-segment decoder consists of 7 different combinatorial circuits, one for each segment. One should look at Karnaughmaps for all segments simultaneously. There could be groupings that are common to several segments!

The optimal 7-segment decoder has probably already been invented!
Graycode or Binarycode?

Wind direction indicator usually use Gray code to provide safe decoding.

<table>
<thead>
<tr>
<th>Binär-kod</th>
<th>Gray-kod</th>
<th>Binär-kod</th>
<th>Gray-kod</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1101</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>10</td>
<td>1110</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>1111</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>12</td>
<td>1010</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>13</td>
<td>1011</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>14</td>
<td>1001</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>15</td>
<td>1000</td>
</tr>
</tbody>
</table>

Tabell med Binär-kod och Gray-kod.

William Sandqvist  william@kth.se
Binary Code disadvantage

Binary code, adjacent code words
1-2 double change
3-4 triple change
5-6 double change
7-8 quadruple change!
9-A double change
B-C quadruple change!
D-E double change
F-0 quadruple change!

But can two bits change at exactly the same time?

- Safe Data Registration use **Graycode**
- Data Processing **Binarycode**

William Sandqvist  william@kth.se
Graycode

By changing the order of the code words, one can find codes where it is never more than one bit at a time that changes in the transitions from one codeword to the next. Such codes are called Gray Codes.

0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100
1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000
can number the corners of a Boolean space

With Gray codes one can number the "hyper-corners" in a Boolean space.

William Sandqvist  william@kth.se
can solve the “Towers of Hanoi”

Gray code is also very helpful for those who want to solve the game "Towers of Hanoi“, constructed by the mathematician Edouard Lucas in 1883.

According to legend, three monks move 64 golden discs, of different sizes, from one bar to another. The discs should be moved one by one, and they must always be placed so that a smaller disc ports on a larger disc.

When the monks are done with their work the earth will go under - it will take $2^{64}$ moves, so the whole thing will probably take a while ...
Conversion Binary-Gray

Binary $\rightarrow$ Gray:
If Binary bit $b_n$ and bit $b_{n-1}$ are different, the Graycode bit $g_{n-1}$ is "1", else "0".

Gray $\rightarrow$ Binary (the most common transformation direction):
If Binary bit $b_n$ and Graycode bit $g_{n-1}$ are different the Binary bit $b_{n-1}$ is "1", else "0".
Logic Gate for the conversion

XOR-gate is ”1” if the inputs are different!

<table>
<thead>
<tr>
<th>Binär-kod</th>
<th>Gray-kod</th>
<th>Binär-kod</th>
<th>Gray-kod</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

4 bit code converter
Gray cote to Binary code

\[ f = a \oplus b \]

\[
\begin{array}{ccc}
 a & b & f \\
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 0 \\
\end{array}
\]
VHDL-introduction

VHDL is a language used to specify the hardware

- HDL - Hardware Description Language
- VHSIC - Very High Speed Integrated Circuit
  Used mostly in Europe
- Verilog is also a language used to specify the hardware
  Used mostly in the United States

William Sandqvist  william@kth.se
The entity describes the ports to the outside of the circuit.
The circuit as a block.
**Architecture**

\[ C_{in} \rightarrow \text{FA} \rightarrow S \]
\[ A \rightarrow \text{FA} \rightarrow S \]
\[ B \rightarrow \text{FA} \rightarrow S \]
\[ \rightarrow C_{out} \]

**architecture** behave of fulladder **is**

begin
\[ S <= A \text{ xor } B \text{ xor } C_{in}; \]
\[ C_{out} <= (A \text{ and } B) \text{ or } (A \text{ and } C_{in}) \text{ or } (B \text{ and } C_{in}); \]
end behave;

**Architecture** describes the function inside the circuit.

William Sandqvist  william@kth.se
Why VHDL?

VHDL is used to

• be able to verify that you have the right functioning by *simulating* the circuit
• be able to describe large structures in a simple way and then generate the circuit by synthesis
• enable structured descriptions of a circuit

VHDL increases the level of abstraction!

William Sandqvist  william@kth.se
Basics in VHDL

There are two types of VHDL code

• VHDL for synthesis: The code is to be input to a synthesis tool which converts it into an implementation (for example, on an FPGA)
• VHDL for modeling and simulation: The code is used to describe a system in an early stage. Since the code can be simulated you can check on the intended operation is correct.
VHDL Hierarchie

Package

Generics

Entity

Ports

Architecture

Concurrent Statements

Architecture

Concurrent Statements

Architecture (structural)

Process

Sequential Statements

William Sandqvist  william@kth.se
Entity

- The primary abstraction level in VHDL is called **entity**
- In a behavioral description one defines the entity through their responses to signals and inputs
- A behavioral model is the same as a "black box“
  The inside is not visible from the outside
  - The entity's behavior is defined by the black box functionality

Input → **Behavioral Entity** → Output
Entity

- An entity describes a component's interface with the outside world.
- The PORT-declaration indicates if it is an input or output.
- An *entity* is a symbol of a component.

```vhdl
ENTITY xor_gate IS
  PORT( x, y: IN bit;
        q: OUT bit);
END xor_gate;
```

Use English names for variable names in the code!
VHDL  Port

• PORT declaration establishes the interface between the component and the outside world.

• A port declaration contains three things:
  – The name of the port
  – The direction of the port
  – The port's datatype

• Example:

```vhdl
ENTITY test IS
  PORT( name : direction data_type);
END test;
```
The most common data types

• Scalars (signals/variables)
  – bit (’0’,’1’)
  – std_logic (’U’,’0’,’1’,’X’,’Z’,’L’,’H’,’W’,’-’)
  – integer
  – real
  – time

• Vectors (multivalued signals / variables)
  – bit_vector – vector of bit’s
  – std_logic_vector – vector of std_logic
An *architecture* describes the function of the component.
An entity can have many architectures, but only one can be active at a time.
An architecture corresponds to the component circuit diagram or behavior.

\[
\begin{align*}
X & \quad | \quad q \\
 y & \quad | \\
\end{align*}
\]

Code for simulation

```
ARCHITECTURE behavior OF xor_gate IS
BEGIN
  q <= a xor b after 5 ns;
END behavior;
```

means arrow

<=
VHDL-Example: 4/1 multiplexer

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Multiplexer_41 IS
    PORT(ce_n : IN std_logic; -- Chip En(active low)
         data_in : IN std_logic_vector(3 DOWNTO 0);
         sel : IN std_logic_vector(1 DOWNTO 0);
         data_out : OUT std_logic); -- TriState Output
END ENTITY Multiplexer_41;
VHDL-Example: 4/1 multiplexer

ARCHITECTURE RTL OF Multiplexer_41 IS
BEGIN
  PROCESS (ce_n, data_in, sel)
  BEGIN
    IF ce_n = '1' THEN
      data_out <= 'Z';
    ELSE
      CASE sel IS
        WHEN "00" => data_out <= data_in(0);
        WHEN "01" => data_out <= data_in(1);
        WHEN "10" => data_out <= data_in(2);
        WHEN "11" => data_out <= data_in(3);
        WHEN OTHERS => null;
      END CASE;
    END IF;
  END PROCESS;
END ARCHITECTURE RTL;

William Sandqvist william@kth.se
Synthesis tool Quartus II

Will be used in LAB 3

William Sandqvist  william@kth.se
More on VHDL

- The study material on the synthesis shows a number of VHDL constructs and the resulting hardware.

- The following images contain additional materials (optionally material).

- The book gives many examples and detailed explanations of VHDL.
Optionally material on VHDL
signal declaration

Signal-declaration is used inside architectures to declare internal (local) signals:

```plaintext
signal a,b,c,d : bit;
signal a,b,sum : bit_vector(31 downto 0);
```

Signal assignment is used to describing the behavior:

```plaintext
sum <= a + b; -- assignment without delay
```
VHDL description various styles

- **Structural**
  similar to how to connect components

- **Sequential**
  similar to how to write computer programs

- **Data Flow**
  Concurrent assignments

William Sandqvist  william@kth.se
Sequential or Parallel Code

• There are two types of code execution in VHDL: sequential and parallel

• Hardware can then be modeled in two different ways

• VHDL is supporting two different levels of abstraction.

• **Sequential code** describes the hardware from a programmer's point of view and are executed in the order it is written.

• The **parallel code** is executed regardless of the order it is in and is asynchronous.
Sequential style

XOR-gate

\[ x \quad y \quad q \]

\[
\text{process}(x, y) \\
\text{begin} \\
\quad \text{if } (x /= y) \text{ then} \\
\quad \quad q <= '1'; \\
\quad \text{else} \\
\quad \quad q <= '0'; \\
\quad \text{end if;} \\
\text{end process;}
\]

means not!
Data flow style

XOR-gate

\[ q \leq a \text{ xor } b; \]

Or as "behavioural dataflow style"

\[ q \leq '1' \text{ when } a \neq b \text{ else } '0'; \]

William Sandqvist william@kth.se
Structural style

\[
\begin{align*}
&u1: \text{not\_gate} \text{ port map } (x,xi); \\
&u2: \text{not\_gate} \text{ port map } (y,yi); \\
&u3: \text{and\_gate} \text{ port map } (xi,y,t3); \\
&u4: \text{and\_gate} \text{ port map } (yi,x,t4); \\
&u5: \text{or\_gate} \text{ port map } (t3,t4,q);
\end{align*}
\]
• A component must be declared before it can be used

ARCHITECTURE test OF test_entity
COMPONENT and_gate
  PORT ( in1, in2 : IN BIT;
      out1 : OUT BIT);
  END COMPONENT;
... more statements ...

• Required, unless it is already in a library somewhere
Instantiation

Component instantiation connects the component interface with the signals in the architecture.

ARCHITECTURE test OF test_entity
COMPONENT and_gate
  PORT ( in1, in2 : IN BIT;
       out1 : OUT BIT);
END COMPONENT;
SIGNAL S1, S2, S3 : BIT;
BEGIN
  Gate1 : and_gate PORT MAP (S1,S2,S3);
END test;
generate

- Generate-statement connects many of the same elements

ENTITY adder IS
  GENERIC (N:integer)
  PORT (a,b:IN bit_vector(N-1 downto 0);
        sum:OUT bit_vector(N-1 downto 0));
END adder;
ARCHITECTURE structural OF adder IS
  COMPONENT full_adder
    PORT (a,b,cin:IN bit;cout,s:OUT bit);
  END COMPONENT;
signal c:bit_vector(N-2 downto 0);
BEGIN
  G0:for i in 1 to N-2 generate
    U0:full_adder PORT MAP (a(i),b(i),c(i-1),c(i),s(i));
  end generate;  -- G0
  U0:full_adder PORT MAP (a(0),b(0),'0',c(0),s(0));
  UN:full_adder PORT MAP (a(N-1),b(N-1),c(N-2),OPEN,s(N-1));
END structural;

Generate an n-bit adder!
generate n-bit adder

Five lines of code generates the ripple-carry n-bit adder from F5!
Test Benches

- To test if your design works so you have to create a test bench. It has three functions:
  - Generating stimuli for simulation
  - Apply these stimuli to an entity to be tested
  - Comparing the output values with expected values

You will use a test bench at LAB 3. A test bench program can try tirelessly through all input combinations!

William Sandqvist  william@kth.se
The ENTITY is empty!

The circuit under test is used as a component of the test bench program.

Here are the test signals generated:
Testbench

A test bench can mark when the desired events occur during the execution.

Or mark when unwanted events occur

The result of a run with a test bench can be saved in a file, as proof that everything is ok - or as a troubleshooting aid if it did not go well.