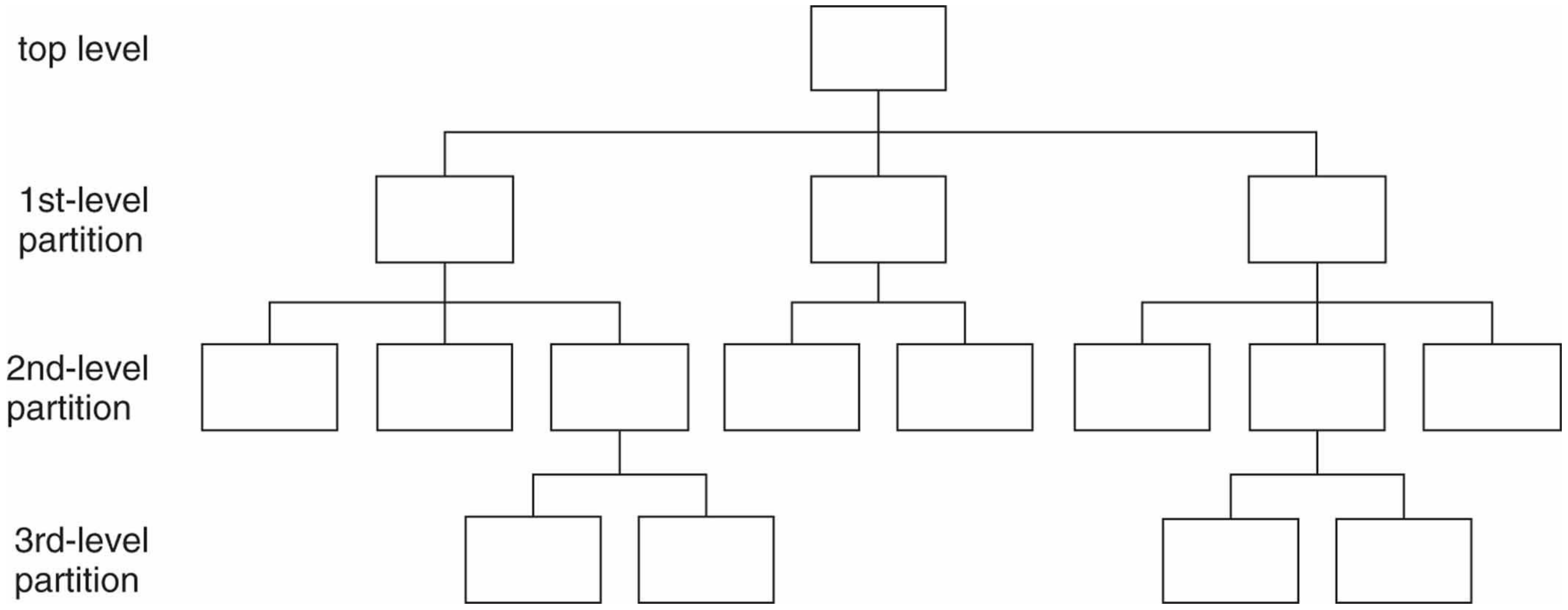
Turun yliopisto
University of Turku

HDL Based Design
ME620138

Hierarchical Design

- Top-down design

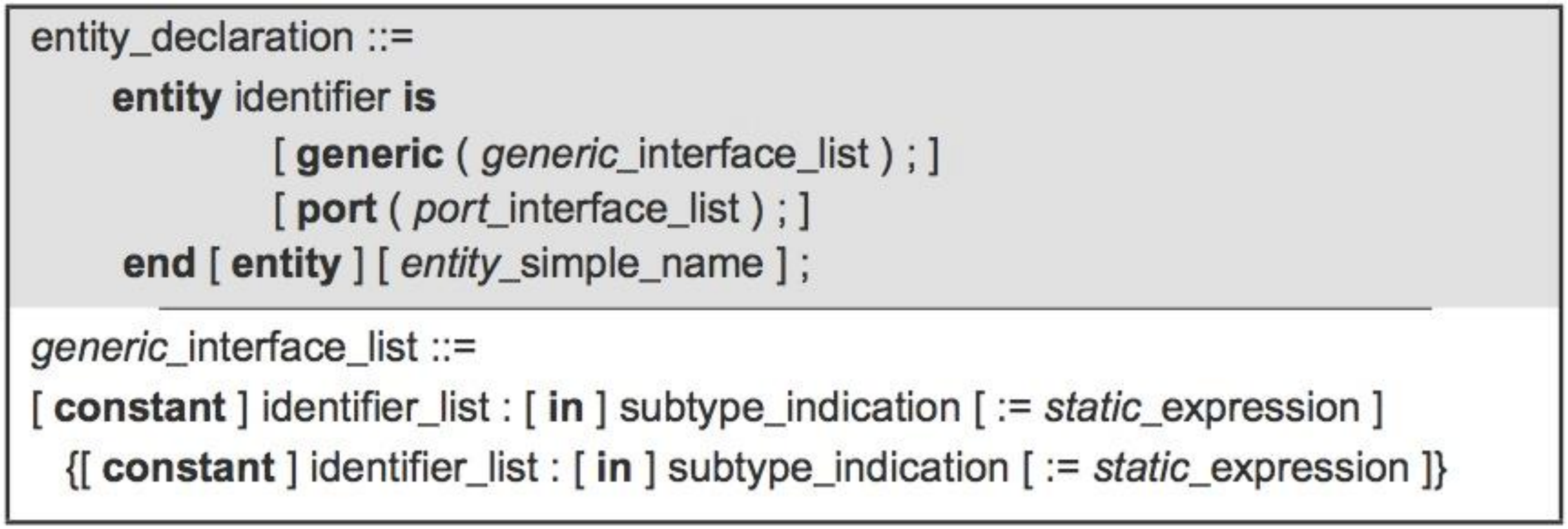


- Each module may itself be partitioned to further reduce its complexity

GENERICS

Generics

- Generics allow a design to be described so that its structure can be altered easily during the instantiation of the module



entity_declaration ::=

entity identifier **is**

[**generic** (*generic*_interface_list) ;]

[**port** (*port*_interface_list) ;]

end [**entity**] [*entity*_simple_name] ;

*generic*_interface_list ::=

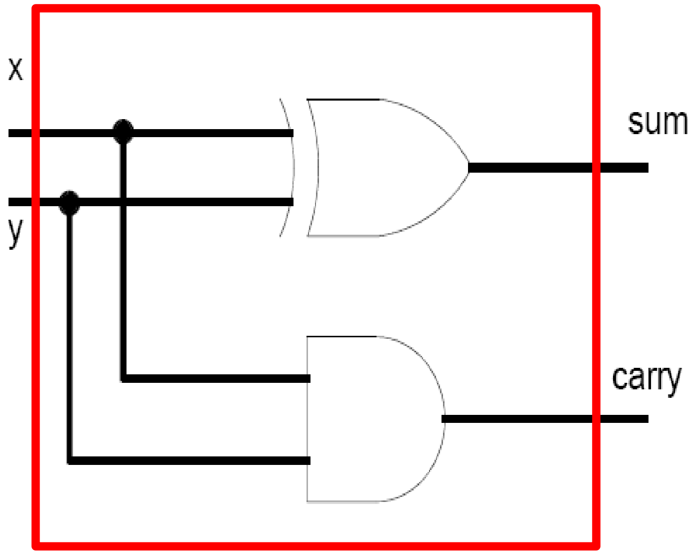
[**constant**] identifier_list : [**in**] subtype_indication [:= *static*_expression]

{[**constant**] identifier_list : [**in**] subtype_indication [:= *static*_expression]}

Generic Example

```
entity half_adder is
generic (Tpd: time := 10 ns)
port( x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
     sum <= x xor y after Tpd;
     carry <= x and y after Tpd;
end myadder;
```



- Generics are visible in the entity in which it is declared as well as in the corresponding architecture body

Up Counter

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

entity Counte3 is
port( clk: in std_logic;
         reset: in std_logic;
         count: out std_logic_vector(3 downto 0));
end Counte3;

architecture behavioural of Counte3 is
        signal counting: std_logic_vector(3 downto 0);
begin
        process (clk,reset)
        begin
                if reset = ‘0' then  counting <= “0000”;
                elsif (rising_edge(clk)) then counting <= counting + 1;
                end if;
        end process;
        count <= counting;
end behavioural;
```

Turun yliopisto
University of Turku

Up Counter

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

entity Counte3 is
generic( w: natural := 4)
port( clk: in std_logic;
         reset: in std_logic;
         count: out std_logic_vector(w-1 downto 0));
end Counte3;

architecture behavioural of Counte3 is
        signal counting: std_logic_vector(w-1 downto 0);
begin
        process (clk,reset)
        begin
                if reset = ‘0' then  counting <= (others => 0);
                elsif (rising_edge(clk)) then counting <= counting + 1;
                end if;
        end process;
        count <= counting;
end behavioural;
```

Port Mapping with Generics

```
entity Counte3 is
    generic( w: natural := 4)
    port(  clk: in std_logic;
           reset: in std_logic;
           count: out std_logic_vector(w-1 downto 0));
    end Counte3;
```

```
c1:   entity work.counter (behavioural)
      generic map (16)
      port map (m_clk, m_reset, m_count);
```

```
c2:   entity work.counter (behavioural)
      generic map (12)
      port map (m_clk, m_reset, m_count);
```

```
c3:   entity work.counter (behavioural)
      generic map (w => 16)
      port map (m_clk, m_reset, m_count);
```

Port Mapping with Generics (VHDL-2008)

```
entity Counte3 is
    generic( type data_type)
    port(  clk: in std_logic;
           reset: in std_logic;
           count: out data_type);
    end Counte3;
```

```
c1:   entity work.counter (behavioural)
      generic map (data_type => std_logic_vector(3 downto 0))
      port map (m_clk, m_reset, m_count);
```

```
generic ( type T; constant init_val : T );
signal v : T := init_val;
```

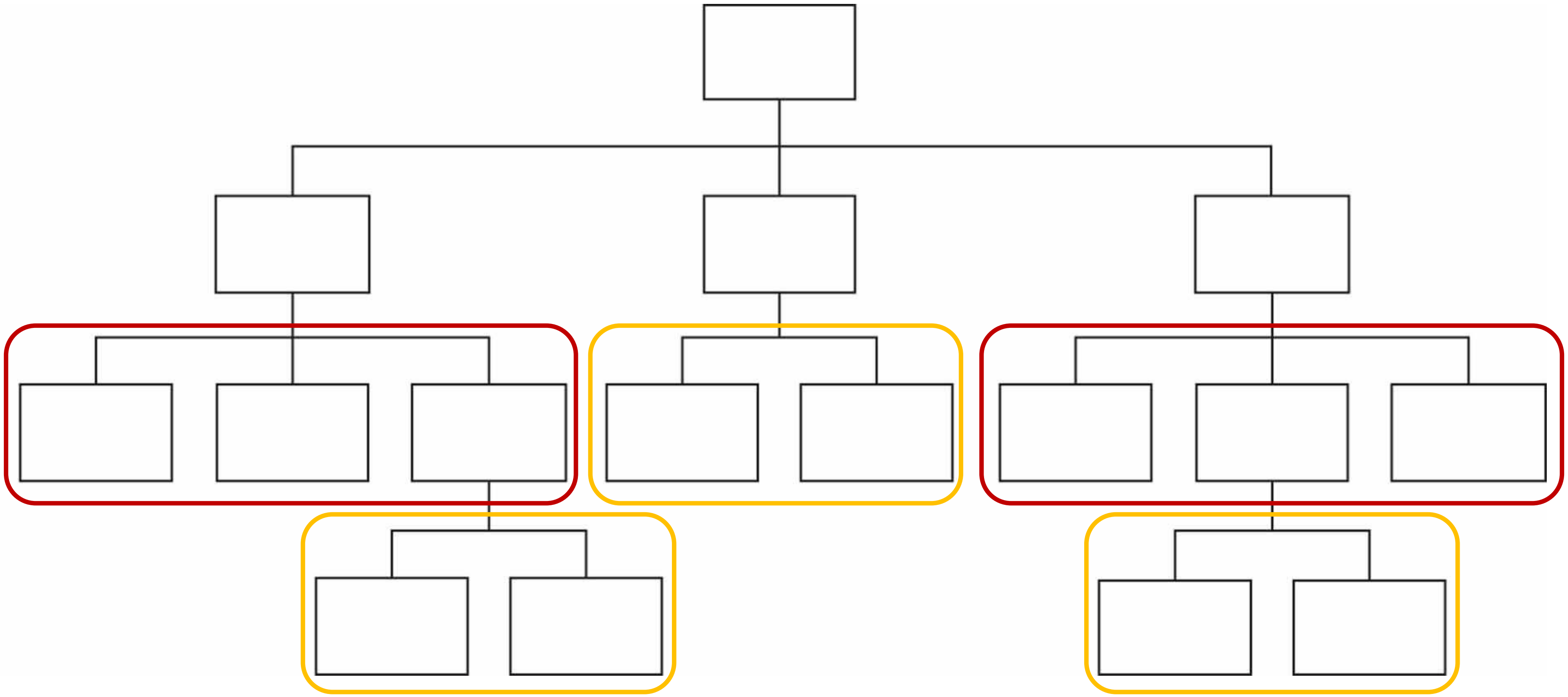
Port Mapping with Generics (VHDL-2008)

NOTE

- To use types in generics requires one to be careful!
- For example, arithmetic operators are not defined for all the available types

```
entity counte3 is
generic( type data_type )
port(    . . .
         Dout : out data_type);
architecture
. . .
Dout <= Dout + 1;
end architecture;
```

Turun yliopisto
University of Turku

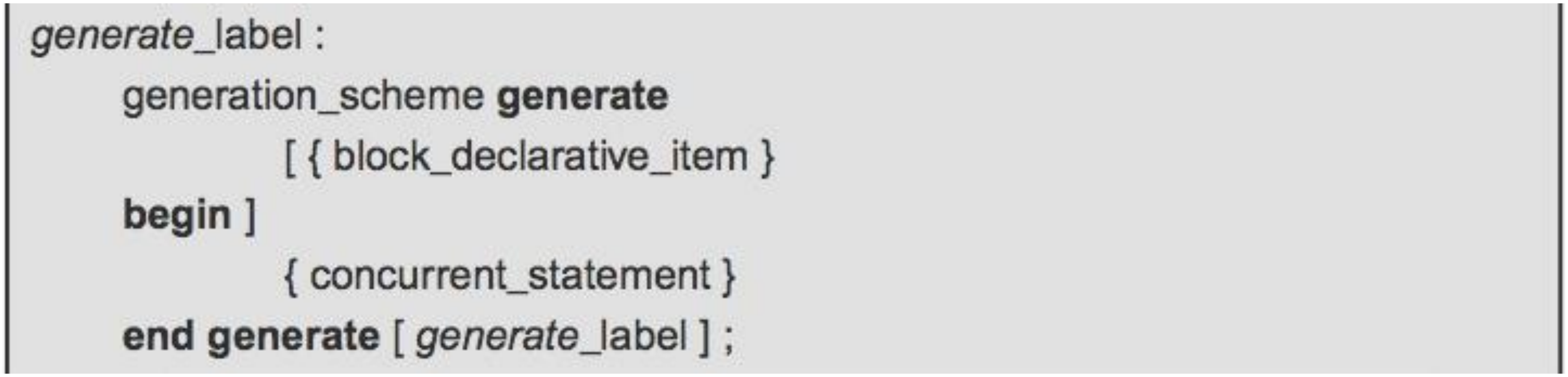


Turun yliopisto
University of Turku

GENERATE

Generate

- With generate statement you can replicate a component during instantiation of the design



```
generate_label :
    generation_scheme generate
            [ { block_declarative_item }
    begin ]
            { concurrent_statement }
    end generate [ generate_label ] ;
```

- Generate *label* is mandatory
- Generation scheme:
  - For loop
  - If statement

Memory Element using Generate

```
entity NbitRegister is
generic ( width : natural:=32 );
port ( Din: in std_logic_vector( width-1 downto 0),
        Dout: out std_logic_vector( width-1 downto 0),
        clk, rst : std_logic);
end NbitRegister;
architecture generated is
    component Dff is
        port ( clk : in std_logic;
               d : in std_logic;
               q : out std_logic );
    end component Dff;
begin
    generate_registers:
    for i in (width-1) downto 0 generate
    begin
        dffn: component Dff
            port map (Din(i), clk, rst, Dout(i) );
    end generate generate_registers;
end generated
```

Concurrent statement

Memory Element using Generate

```
Entity NbitFullAdder is
generic ( width : natural:=8);
port( a, b : in std_logic_vector (width-1 downto 0);
         s : out std_logic_vector (width-1 downto 0);
        cin : in std_logic;
        cout : out std_logic );
end NbitFullAdder;
```

```
architecture dataflow of NbitFullAdder is
  signal c : std_logic_vector (width-1 downto 0);
begin
    array : for i in width-1 downto 0 generate
    -- remember component declarations
  begin
    first : if i=0 generate
    begin
      cell : component full_adder
        port map (a(i), b(i), cin, s(i), c(i));
    end generate first;
    other : if i/=0 generate
    begin
      cell : component full_adder
        port_map (a(i), b(i), c(i-1), s(i), c(i));
    end generate other;
  end generate array;
cout <= c(15);
end architecture dataflow;
```

Memory Element using Generate

```
Entity NbitFullAdder is
generic ( width : natural:=8);
port( a, b : in std_logic_vector (width-1 downto 0);
         s : out std_logic_vector (width-1 downto 0);
        cin : in std_logic;
        cout : out std_logic );
end NbitFullAdder;
```

```
architecture dataflow of NbitFullAdder is
  signal c : std_logic_vector (width-1 downto 0);
 -- remember component declarations
begin
  array : for i in width-1 downto 0 generate
  begin
        first_and_others:
        if i=0 generate
                cell : component full_adder
                        port map (a(i), b(i), cin, s(i), c(i));
        else generate
                        cell : component full_adder
                        port_map (a(i), b(i), c(i-1), s(i), c(i))
        end generate first_and_others;
  end generate array;
cout <= c(15);
end architecture dataflow;
```

Turun yliopisto
University of Turku

Generate

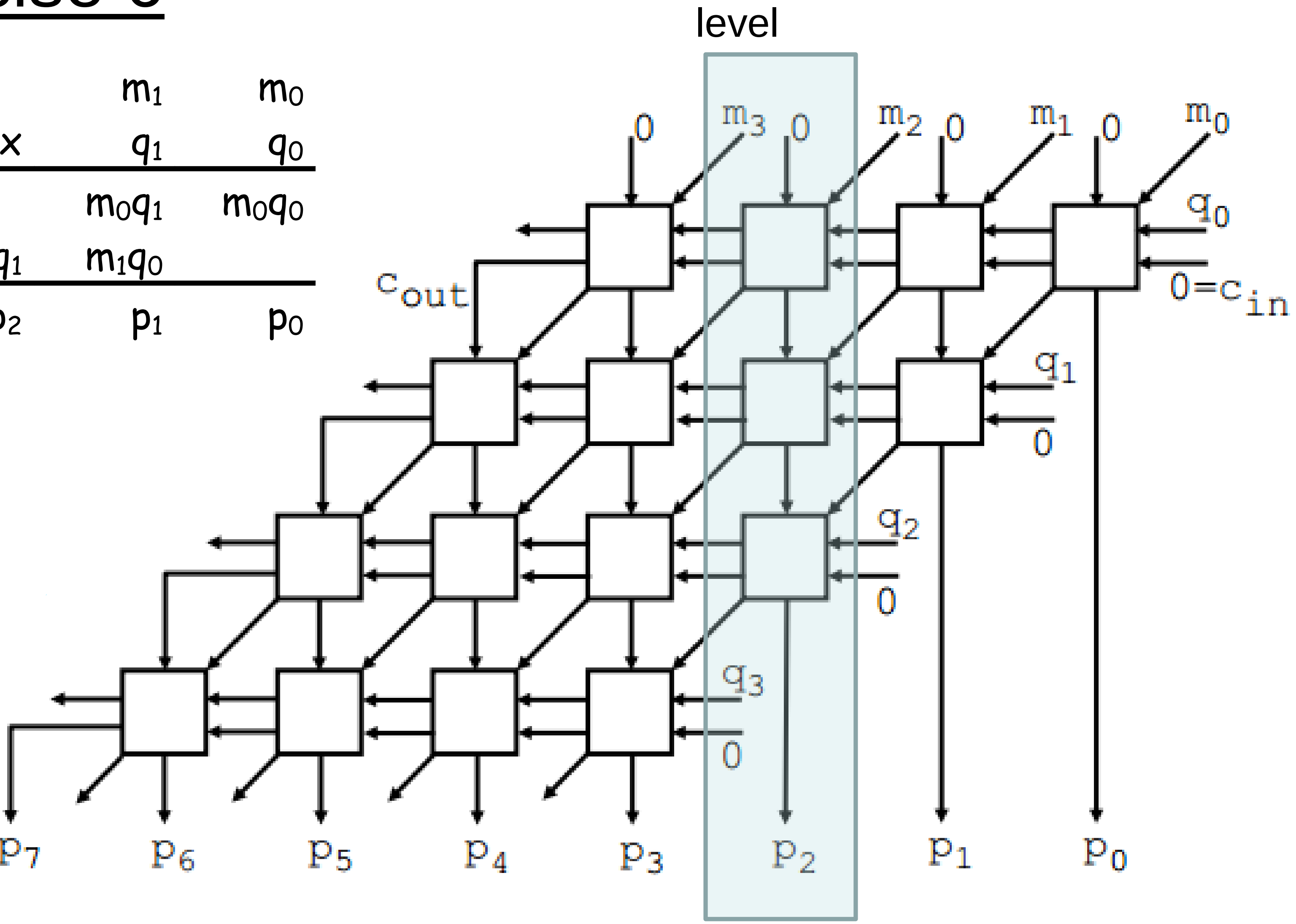
- For generate
  - All elements are equivalent
  - discrete range must be specified
  - Loop cannot be terminated
- If generate
  - Conditional creation of elements
  - **elsif** or **else** alternatives are allowed only in VHDL-2008
- Case generate
  - Conditional creation of elements
  - Only available in VHDL-2008

Else and elsif
Does not work with
Questa 10.2 nor 10.4

Exercise 6

Exercise 6

$$\begin{array}{rrrrr}
 & & & m_1 & m_0 \\
 & & \times & q_1 & q_0 \\
\hline
 & & & m_0q_1 & m_0q_0 \\
+ & & m_1q_1 & m_1q_0 & \\
\hline
 & p_3 & p_2 & p_1 & p_0
\end{array}$$



Turun yliopisto
University of Turku

Exercise 6

$$
\begin{array}{cccc}
 & & m_1 & m_0 \\
 & \times & q_1 & q_0 \\
\cline{2-4}
 & & m_0q_1 & m_0q_0 \\
+ & m_1q_1 & m_1q_0 & \\
\hline
p_3 & p_2 & p_1 & p_0
\end{array}
$$

$$
\begin{array}{cccccccccc}
 & & & & 1 & 0 & 1 & 1 & (11) & \text{Kerrottava (multiplicand)} \\
 & & & \text{x} & 1 & 1 & 0 & 1 & (13) & \text{Kertoja (multiplier)} \\
\cline{4-8}
 & & & & 1 & 0 & 1 & 1 & & \\
 & & + & 0 & 0 & 0 & 0 & & & \\
\cline{3-8}
 & & & 0^{1)} & 1 & 0 & 1 & 1 & & \\
 & + & 1 & 0 & 1 & 1 & & & & \\
\cline{2-8}
 & ^{1)} & 1^{1)} & 1 & 0 & 1 & 1 & 1 & & \\
+ & 1 & 0 & 1 & 1 & & & & & \\
\cline{1-8}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & (143) & \text{Tulo (product)}
\end{array}
$$