RTR532 2018 Spring
«In the VHDL!»
Lecture 02

- VHDL signal data types and operations
- VHDL assignment constructions
- Example - counter
- Using entity as component
- Top-level VHDL entity and FPGA physical pins

VHDL «basic rules»

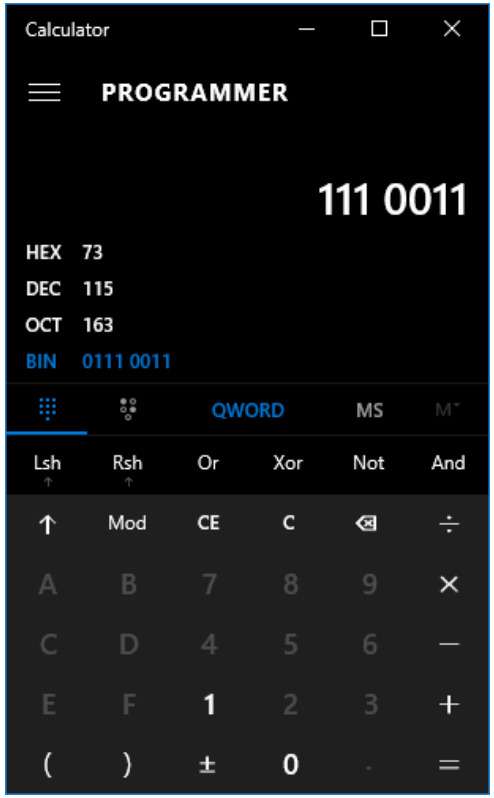
- Case Insensitive: signal = Signal = SIGNAL
- Line finished by **;**
- Bit defined with `'0'`
- Array defined with `"0000"`
- Comment with «--»
- White space does not matter (this is NOT python)
- One top-level VHDL file per project (entity name = project name)
- There is no one correct way to do it.

VHDL signal data types and operations

Digital signal (width, levels)

- How we represent numbers in digital ?
- 1 bit
- X bits

Dec	115
Hex	73
Binary	01110011



Data types are defined in **library**!

- **std_logic_1164**

`std_logic`	1 bit, ‘1’, ‘0’, ‘X’, ‘U’
`std_logic_vector(N-1 downto 0)`	N bit digital signal

- **numeric_std**

`unsigned(N-1 downto 0)`	X bit digital signal, interpreted as unsigned bit vector
`signed(N-1 downto 0)`	X bit digital signal, interpreted as signed bit vector
`integer`	Synthesis assigns a bit vector. Can be interpreted as both signed and unsigned

std_logic

```
signal tmp_output : std_logic; --no init

signal tmp_output : std_logic := '0'; --init as low
signal tmp_output : std_logic := '1'; --init as high
signal tmp_output : std_logic := 'Z'; --init as high impendance
signal tmp_output : std_logic := '-'; --init as dont care
```

Power-on initialization value

Name of signal

type

:= operator for initialization

std_logic_vector

Width 4 bits

```
signal tmp_output_4 : std_logic_vector(3 downto 0); -- no init

:= '0' & '0' & '0' & '0' ; --coupled bits
:= "0000"; --init as array
:= x"0"; --init as hex
:= (others => '0'); --init wide
:= (0 => '0', 1 => '0', others => 'Z');
```

x"0" – value in hex char
0123456789ABCDEF
1 char – 4 bits

Accessing single std_logic bit of std_logic_vector

```
tmp <= tmp_output_4(0);
tmp <= tmp_output_4(4); --is this OK?
tmp <= tmp_output_4(BIT); --BIT - integer type
```

Whiteboard exercise

- Define signal of type std_logic, name zuperSignal, init value of 1
- Define signal of type std_logic_vector, width 8 bits, name zuperVector, init value of 115 (dec)
- Assign 3rd bit of zuperVector to zuperSignal

Operations
(use IEEE.std_logic_1164.all;)

- Assign <=
```
b_out <= b_in; --assign value
```
You can assign only matching data types.

- Initialize :=
```
signal b_sig : std_logic := '0';
```
- Concatenating &
```
b_out <= b_in & b_in; --join together (b_in 1bit,
                        b_out - 2bit wide!)
```
Signal widths must be defined correctly, else – error by software

- Logic operators NOT, AND, OR, NAND, NOR, XOR, XNOR
```
B_out <= not b_in;
b_out <= a_in and  b_in;
b_out <= a_in or   b_in;
b_out <= a_in nand b_in;
b_out <= a_in nor  b_in;
b_out <= a_in xor  b_in;
b_out <= a_in xnor b_in;
```

Operations (use numeric_std)

Summary of NUMERIC_STD

```
+  -  *  /   rem  mod
<  <=  >  >=  =  /=

UNSIGNED ■ UNSIGNED
UNSIGNED ■ NATURAL
NATURAL  ■ UNSIGNED
SIGNED   ■ SIGNED
SIGNED   ■ INTEGER
INTEGER  ■ SIGNED
```

```
sll srl rol ror

UNSIGNED ■ INTEGER
SIGNED   ■ INTEGER
```

```
not and or nand nor
xor xnor

UNSIGNED ■ UNSIGNED
SIGNED   ■ SIGNED
```

```
TO_INTEGER   [UNSIGNED          ] return INTEGER
TO_INTEGER   [SIGNED            ] return INTEGER
TO_UNSIGNED  [NATURAL,  NATURAL] return UNSIGNED
TO_SIGNED    [INTEGER,  NATURAL] return SIGNED
RESIZE       [UNSIGNED, NATURAL] return UNSIGNED
RESIZE       [SIGNED,   NATURAL] return SIGNED
```


Illustration from doulos inc.

Conversion between types

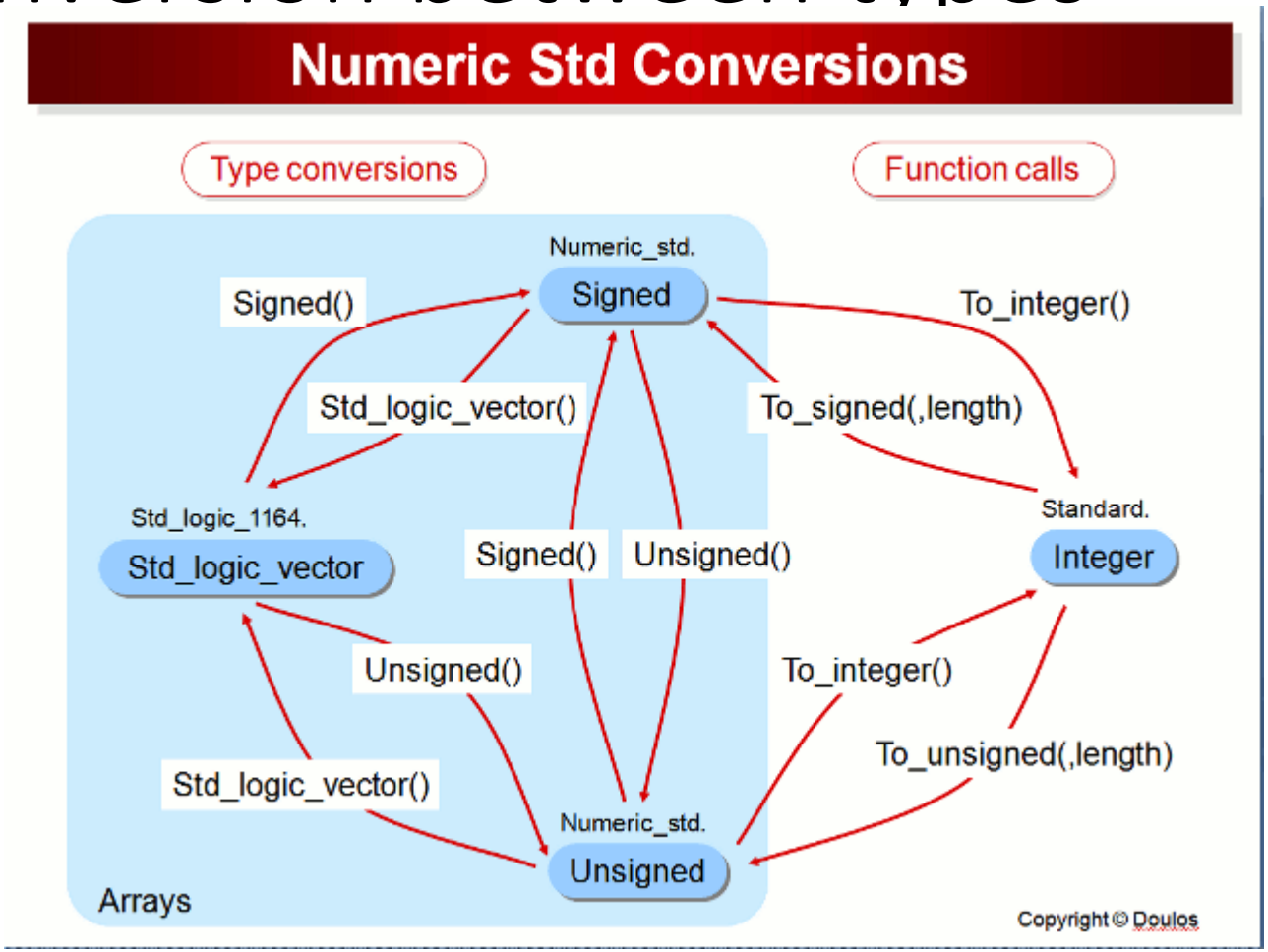


Illustration from doulos inc.

Whiteboard exercise

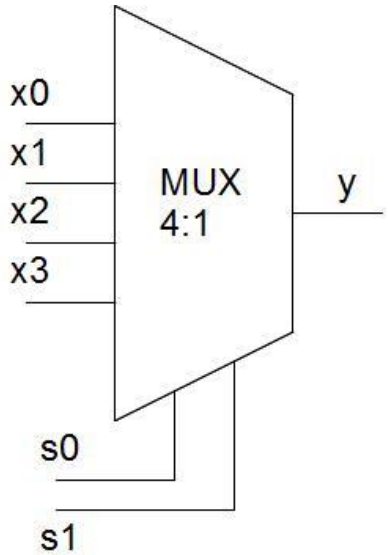
- Convert integer dec 15 to unsigned std_logic_vector of length N=24 bits

VHDL assignment constructions

Four – asynchronous

Two – synchronous

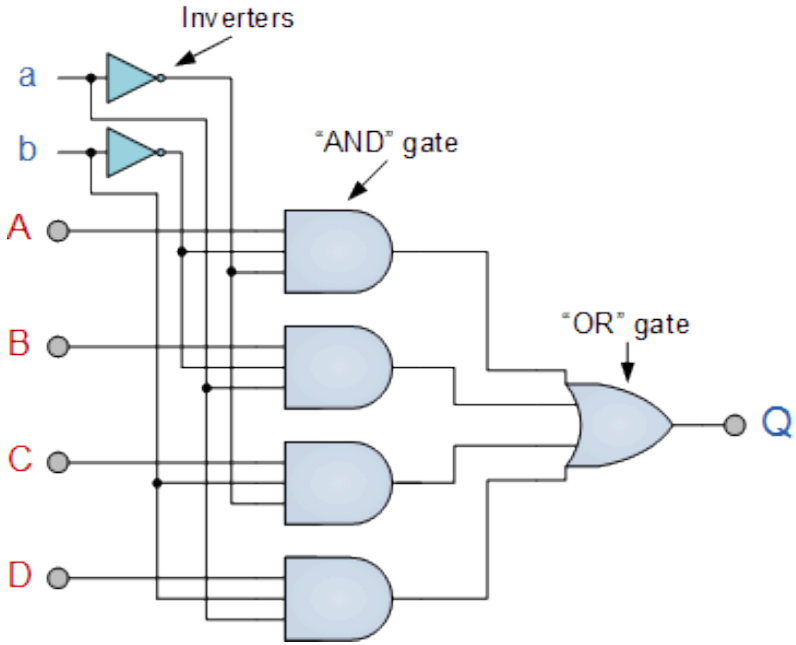
Example 4:1 mux (x – dont care)



s1	s0	x3	x2	x1	x0	y
0	0	x	x	x	0	0
0	0	x	x	x	1	1
0	1	x	x	0	x	0
0	1	x	x	1	x	1
1	0	x	0	x	x	0
1	0	x	1	x	x	1
1	1	0	x	x	x	0
1	1	1	x	x	x	1

```
entity mux_4_1 is
port
(
   x     : in std_logic_vector(3 downto 0);
   s     : in std_logic_vector(1 downto 0);
   y     : out std_logic
);
end mux_4_1;
```

MUX 4:1 with logic gates



Let us use logic gates to implement this!

```
y <= (x(0) and (not s(1)) and (not s(0)) ) or
     (x(1) and (not s(1)) and (     s(0)) ) or
     (x(2) and (     s(1)) and (not s(0)) ) or
     (x(3) and (     s(1)) and (     s(0)) );
```

NOT EFFECTIVE – hard to debug, easy mistakes!

1/4 When - Else

<expression> can be
- Constant values
- Other signals
- Operations with signals

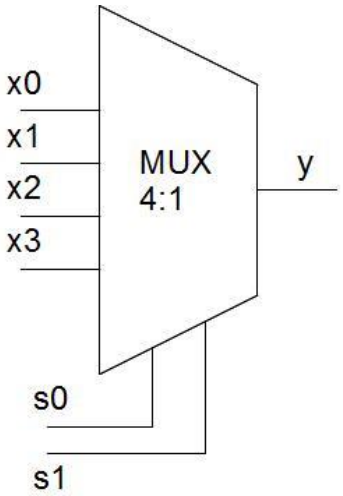
```
b_out <=      <expression> when <condition> else
              <expression> when <condition> else
              <expression>;
```

<condition> can be
- Comparison (>, <, =, etc)

```
b_out <=      b_in when sel_carry = '0' else
              a_in when sel_carry = '1' else
              '0';

b_out <=      (b_in and a_in) when sel_carry = '0' else
              (b_in or  a_in) when sel_carry = '1' else
              '0';

--when-else is a single long line statement!
```

MUX 4:1 When-Else

```
y <= x(0) when s = "00" else
     x(1) when s = "01" else
     x(2) when s = "10" else
     x(3) when s = "11" else
     '0';
```



2/4 With-Select

<expression> can be
- Constant values
- Other signals
- Operations with signals

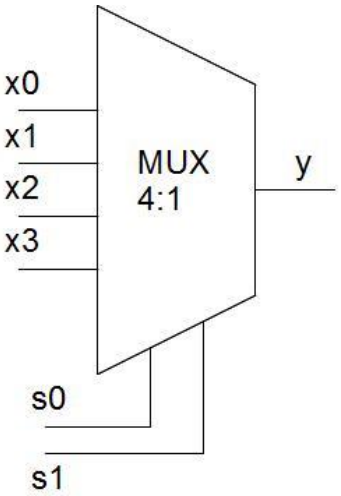
```
with <signal_name> select
        b_out <= <expression> when <choice1>,
                 <expression> when <choice2>,
                 <expression> when others;

with sel_carry select
        b_out <= (b_in and a_in)  when '0',
                 (b_in or  a_in)  when '1',
                 '0'              when others;
```

<choice1> - exact for <signal_name>

--with-select is a single long line *statement!*

MUX 4:1 With-select

```
with s select
    y <= x(0) when "00",
         x(1) when "01",
         x(2) when "10",
         x(3) when "11",
         '0'  when others;
```



What is VHDL Process ?

- VHDL process is «evaluated» only when signals in sensitivity list changes (high to low, etc.)
- Process allows to use sequential statements
  - **If**, **case**, variable, ...

Process name - optional

```
process_name : process(sensitivity list)
begin
...
end process;
```

Sensitivity list (a, b, ...)

Sequential statements

3/4 If-Else *(only inside process)*

```
process_name : process(sensitivity list)
begin
  if (condition) then
    <expression>
  elsif (condition) then
    <expression>
  else
    <expression>
  end if;
end process;
```

If ... Then

If first condition is NOT true, go to next if

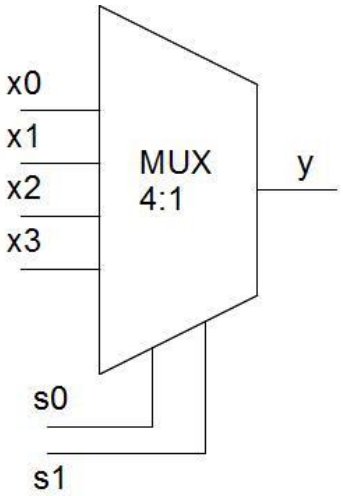
If all conditions are NOT true, go to else

Elsif – short of **else if**

MUX 4:1 If-else

```
process(s, x)
begin
    if (s = "00") then
      y <= x(0);
    elsif (s = "01") then
      y <= x(1);
    elsif (s = "10") then
      y <= x(2);
    elsif (s = "11") then
      y <= x(3);
    else
      y <= '0';
    end if;
end process;
```

```
-- or process(s)
To change output
only when S changes
```



4/4 Case (only inside process)

```
process_name : process(sensitivity list)
begin
  case (signal_name) is
    when choice1 =>
      <expressions>
    when choice2 =>
      <expressions>
    when others  =>
      <expressions>
 end case;

end process;
```

Case starts…

If signal_name = choice1, then…

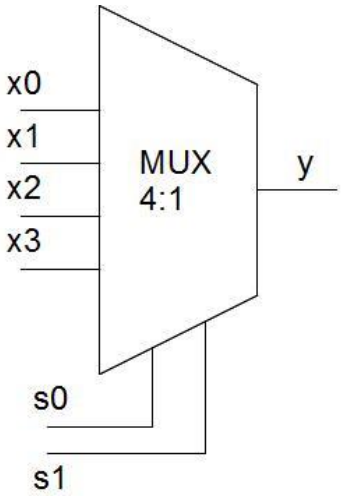
Execute this, when signal does not match any choices

End Case

MUX 4:1 Case

```
process(s, x)
begin
   case (s) is
      when "00" =>
         y <= x(0);
      when "01" =>
         y <= x(1);
      when "10" =>
         y <= x(2);
      when "11" =>
         y <= x(3);
      when others =>
         y <= '0';
   end case;
end process;
```

```
-- or process(s)
To change output
only when S changes
```



Synchronous assignment

- VHDL functions to detect edge of signal
  - rising_edge(sig) ,
  - falling_edge(sig)
- Option 1 - use clock signal rising_edge in process with
  - if-else
  - case
- Option 2 – run asnynchronous signal through process with rising edge

```
process(clk)
begin
  if rising_edge(clk) then
      y_sync <= y;
  end if;
end process;
```

Whiteboard exercise:
How will this be implemented in the FPGA?

If-Else, Case MUX 4:1, synchronous

Synchronous If-else

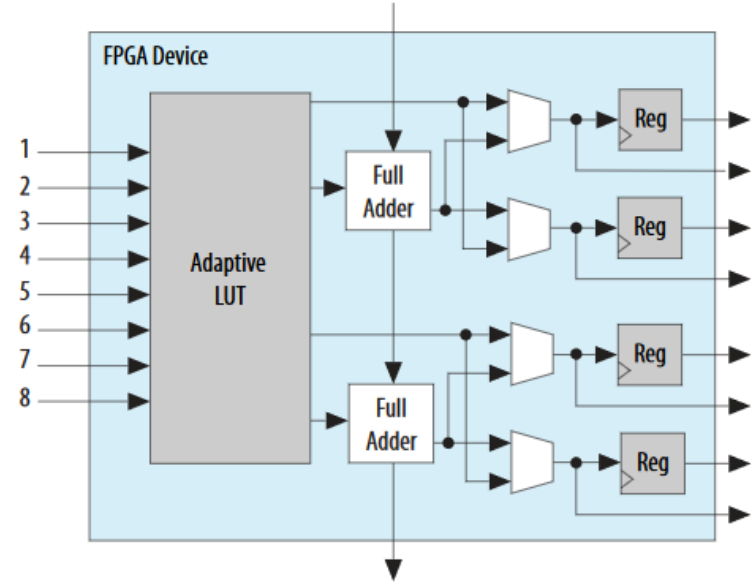
```
process(clk)
begin

  if rising_edge(clk) then

    if (s = "00") then
      y <= x(0);
    elsif (s = "01") then
      y <= x(1);
    elsif (s = "10") then
      y <= x(2);
    elsif (s = "11") then
      y <= x(3);
    else
      y <= '0';
    end if;

  end if;

end process;
```

Synchronous case

```
process(clk)
begin

  if rising_edge(clk) then

    case (s) is
      when "00" =>
        y <= x(0);
      when "01" =>
        y <= x(1);
      when "10" =>
        y <= x(2);
      when "11" =>
        y <= x(3);
      when others =>
        y <= '0';
    end case;

  end if;

end process;
```

Example (async to sync)

```
entity mux_4_1 is
port
(
  clk     : in std_Logic;
  x       : in std_logic_vector(3 downto 0);
  s       : in std_logic_vector(1 downto 0);
  y       : out std_logic

);
end mux_4_1;

architecture beh of mux_4_1 is
     signal y_async : std_logic : ='0';

begin
     y_async <= x(0) when s = "00" else
                x(1) when s = "01" else
                x(2) when s = "10" else
                x(3) when s = "11" else
                '0';

     process(clk)
     begin
     if rising_edge(clk) then
        y <= y_async;
     end if;
     end process;

end beh;
```



Implements mux 4:1 in the LUT using WHEN-ELSE

Implements flip flop (register, trigger)

Whiteboard example

- Signal lcd – 1 bit std_logic
- Signal counter – 8 bit std_logic_vector, that is being incremented 0 to 255 non stop
- Assume the signals are defined
- Write a when-else construct to set signal «lcd»
  - low when «counter» is 0 to 64
  - high when «counter» is 65 to 200
  - high impendance when «counter» is 201 to 255

Quartus example – counter l02_counter.vhd (in l02_quartus project)

counter 8b (signal name «cnt_signal», type unsigned)

reset input

clock input

counter value output

counter 50% value output

Using entity as component

Delay Pulse Module entity

- Entity (to be used)

```
entity delayed_pulse_module is
port
(

clk      : in std_logic;
i        : in std_logic;
o_del    : out std_logic

);

end delayed_pulse_module;
```

Followed by architecture….

- Component definition goes in Architecture (before BEGIN)
- Component port mapping goes in Architecture (after BEGIN)

Delayed pulse module (DPM) izsaukšana mūsu arhitektūrā

```
architecture behavioral of advanced_mux is

  component delayed_pulse_module
  port
  (
    clk : in std_logic;
    i : in std_Logic;
    o_del: out std_logic
  );
  end component;

begin

  INST_DEL_MOD_1 : delayed_pulse_module
  port map
  (
     clk            => clk,
     i              => sel,
     o_del          => sel_carry
  );

end behavioral;
```

Entity replaced with component keyword

Unique name for THIS instance of the component (you can use one component multple times)

component name

Advanced_mux signals or inputs/outputs

Port mapping operator =>

Quartus example – integrate l02_counter.vhd in l02.vhd

Top-level VHDL entity and FPGA physical pins

What, why ?

- From one side – VHDL top level entity
- From other side – FPGA physical pins

Connecting them with Quartus Pin Planner

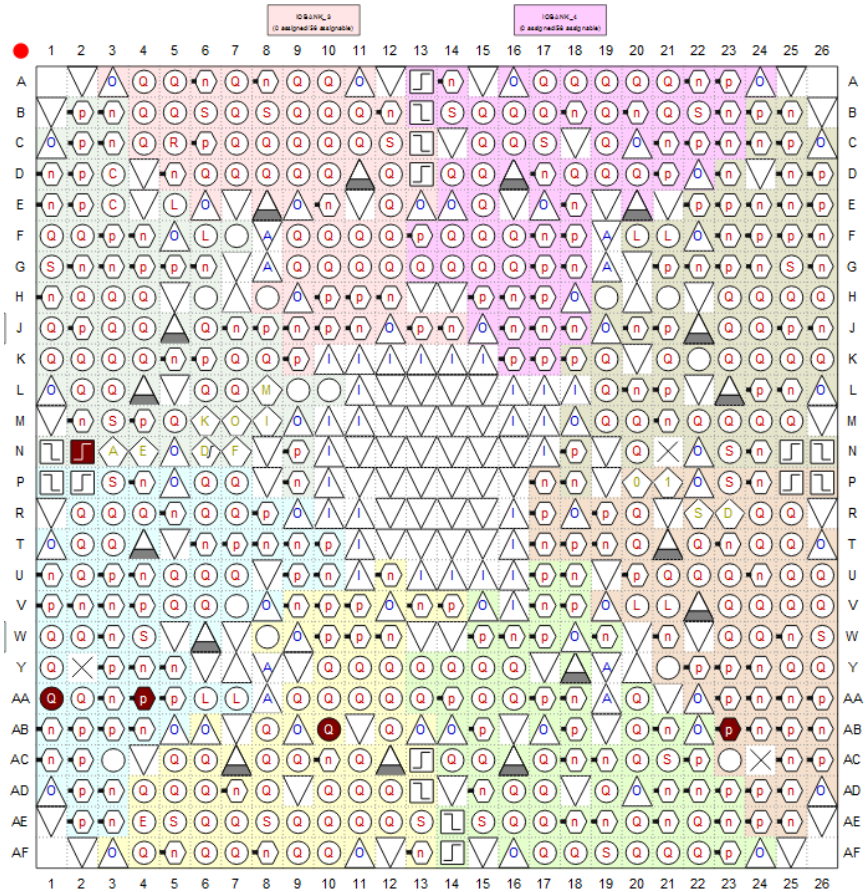
- Pin Planner – software tool
- .QSF (Quartus Settings File) – part of Quartus project

Which pins to use ?

- FPGA pinout in datasheet
- Which pins are connected where ?
- [C5G User Manual]

Top View - Wire Bond

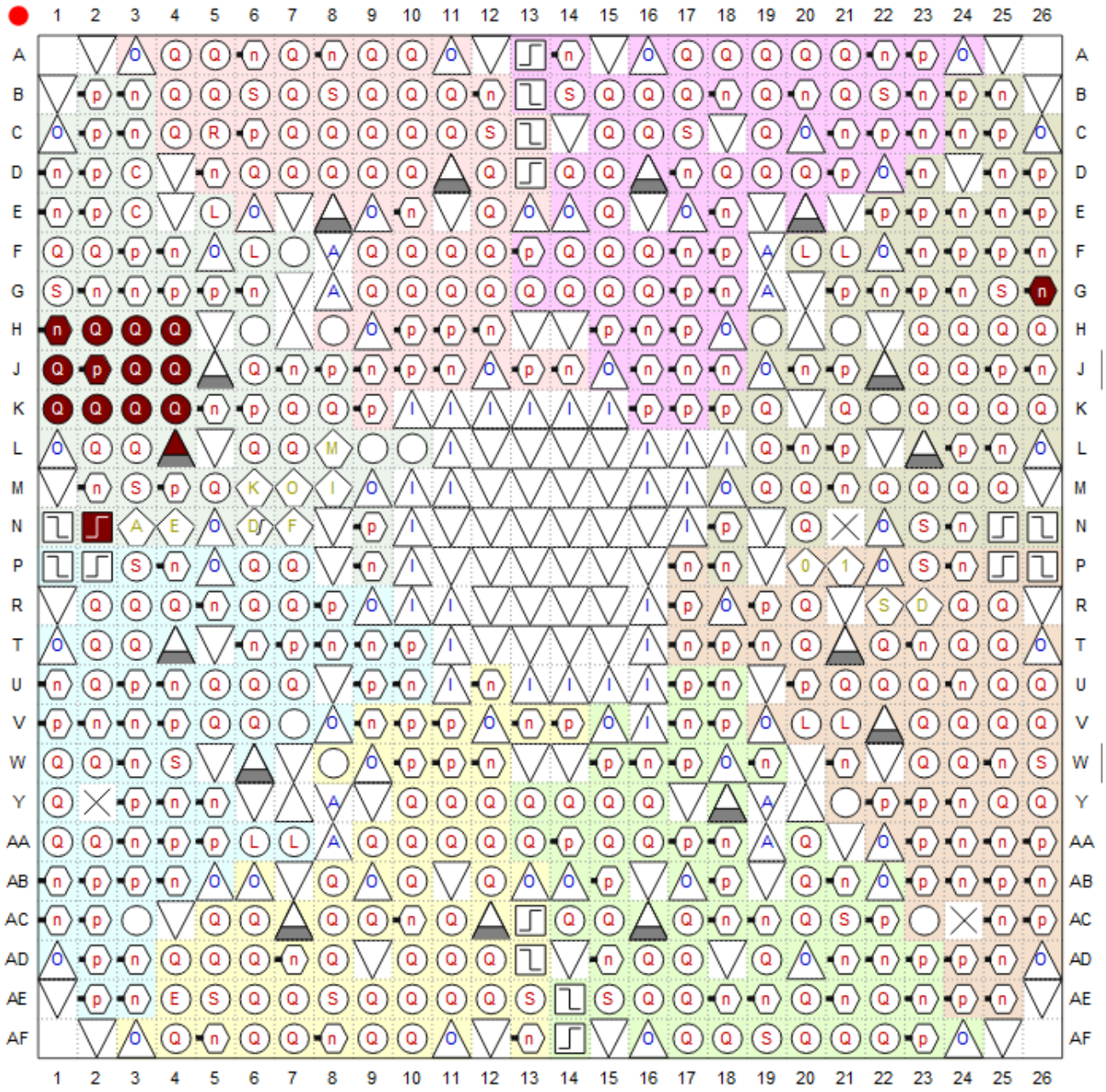
Cyclone II - EP2C35F672C7



Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

Top-level VHDL entity example

```
--define connections to outside
entity lcd_io_demo is
port
(
   clk                          : in std_logic;
   rst                          : in std_logic;

   lcd_data                     : out std_logic_vector(7 downto 0);
   lcd_rw                       : out std_logic;
   lcd_en                       : out std_logic;
   lcd_rs                       : out std_logic;
   lcd_on                       : out std_logic;
   lcd_blon                     : out std_logic
);
end lcd_io_demo;
```



Quartus example – assign pins for l02_quartus_pins project

Start Analysis and Synthesis

Assignments – Pin Planner

.qsf file with pins

```
set_location_assignment PIN_H3 -to lcd_data[7]
set_location_assignment PIN_H4 -to lcd_data[6]
set_location_assignment PIN_J3 -to lcd_data[5]
set_location_assignment PIN_J4 -to lcd_data[4]
set_location_assignment PIN_H2 -to lcd_data[3]
set_location_assignment PIN_H1 -to lcd_data[2]
set_location_assignment PIN_J2 -to lcd_data[1]
set_location_assignment PIN_J1 -to lcd_data[0]
set_location_assignment PIN_K3 -to lcd_en
set_location_assignment PIN_L4 -to lcd_on
set_location_assignment PIN_K1 -to lcd_rs
set_location_assignment PIN_K4 -to lcd_rw
set_location_assignment PIN_N2 -to clk
set_location_assignment PIN_K2 -to lcd_blon
set_location_assignment PIN_G26 -to rst
```

Lecture Summary

- Remember from this lecture:
  - Signal data types and operations – where to find how-to
  - Signal assignment options **(at least names of them!)** and where to find code examples
  - How to use entity as component in other entity
  - How to join top-level entity io with FPGA pins

To do till next lecture:

- Problem set 02 (submit by 2017-03-08)
- Ideas for final project