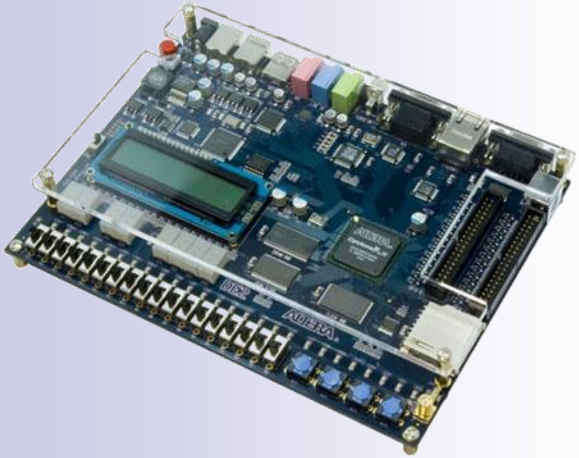
Logic Systems and Processors
cz:Logické systémy a procesory



Lecturer: Richard Šusta

richard@susta.cz, susta@fel.cvut.cz,
+420 2 2435 **7359**

Version V1.0

CTU-FEE in Prague, CR – subject BE5B35LSP

Example:

BINARY-REFLECTED GRAY CODE

Task: Create a circuit for
converting binary numbers to Gray code.

Note: It will be handy at the end of this lecture

Binary-reflected Gray Code

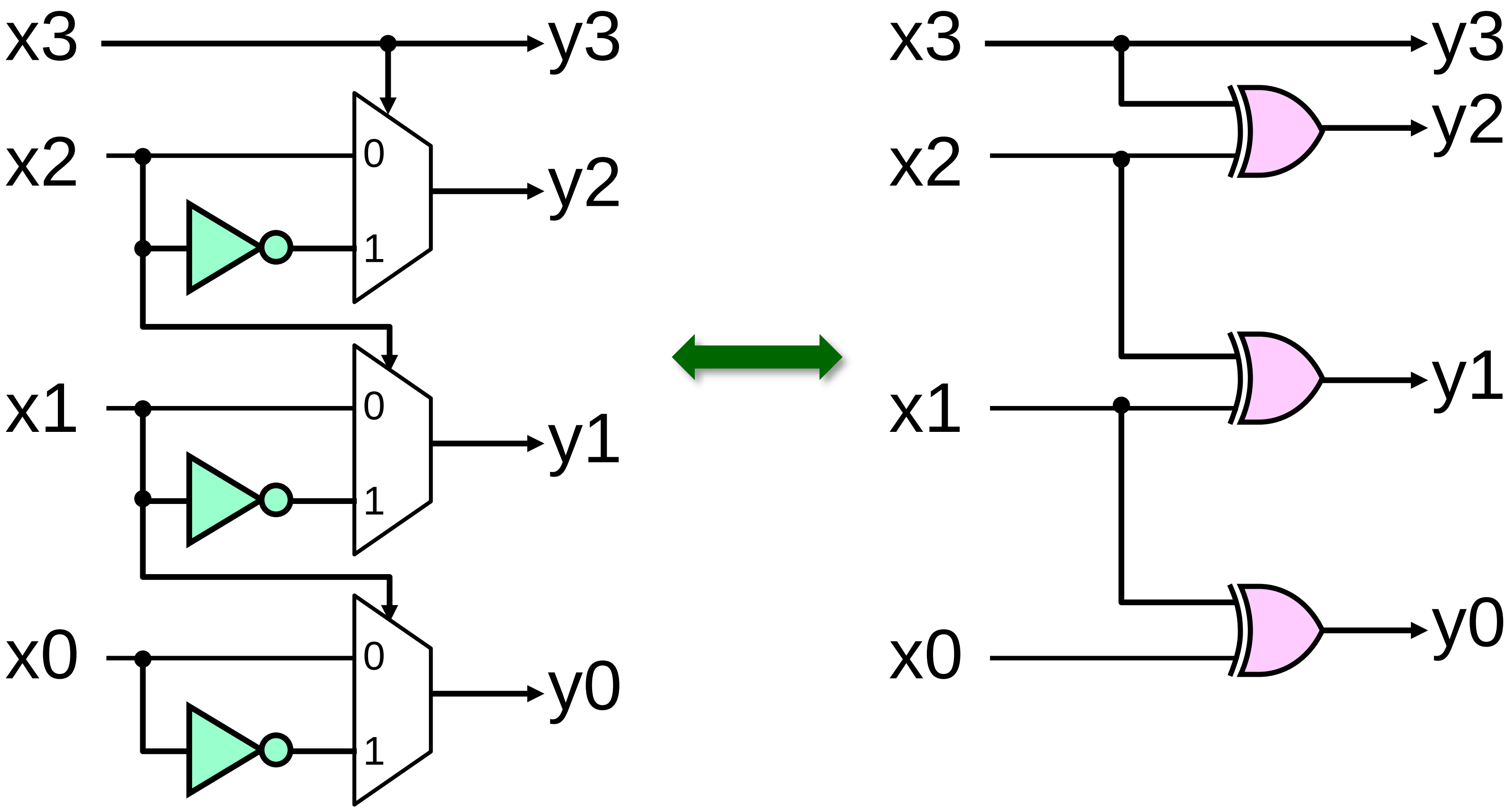
N
0
1
2
3
4
5
6
7

bin2	*bin1*	*bin0*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

gray2	*gray1*	*gray0*
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

In the Gray code table,
we copy corresponding binary bits.
Bits are negated
if they lays below the more significant binary bit in '1'.

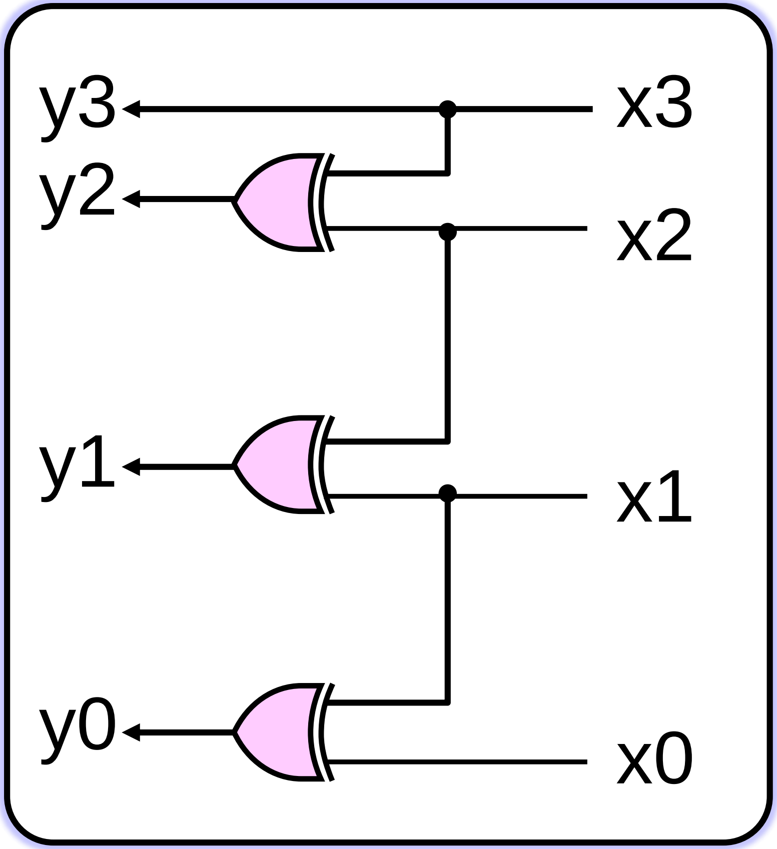
Binary-reflected Gray Code



4-bit Binary

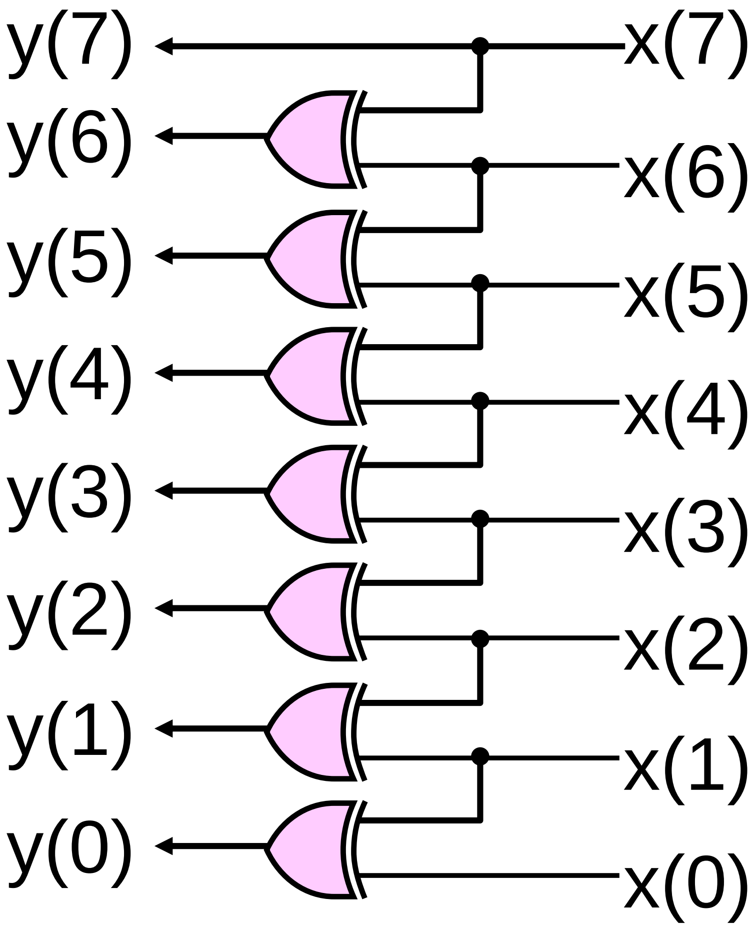
We begin with the simple 4-bit case to outline coding.

```
library ieee; use ieee.std_logic_1164.all;
entity ToGray4 is
port ( x0, x1, x2, x3 : in std_logic;
       y0, y1, y2, y3 : out std_logic);
end entity;
architecture rtl of ToGray4 is
begin -- architecture
  y3<= x3 ;
  y2 <= x2  xor  x3;
  y1 <= x1  xor  x2;
  y0 <= x0  xor  x1;
end architecture;
```



Idea demonstrated with 8-bit Binary

y(7)	<=	x(7)		
y(6)	<=	x(6)	xor	x(7)
y(5)	<=	x(5)	xor	x(6)
y(4)	<=	x(4)	xor	x(5)
y(3)	<=	x(3)	xor	x(4)
y(2)	<=	x(2)	xor	x(3)
y(1)	<=	x(1)	xor	x(2)
y(0)	<=	x(0)	xor	x(1)



ToGray8 - main parts of the code

Description of inputs with outputs

```
x : in std_logic_vector(7 downto 0);
y: out std_logic_vector(7 downto 0));
```

Custom wiring:

```
   y(7) <= x(7);

  y(6 downto 0) <= x(6 downto 0) xor x(7 downto 1);
```

y(7)	<=	x(7)		
y(6)	<=	x(6)	xor	x(7)
y(5)	<=	x(5)	xor	x(6)
y(4)	<=	x(4)	xor	x(5)
y(3)	<=	x(3)	xor	x(4)
y(2)	<=	x(2)	xor	x(3)
y(1)	<=	x(1)	xor	x(2)
y(0)	<=	x(0)	xor	x(1)

A more elegant solution

```
architecture dataflow1 of ToGray8 is
begin
   y <= x xor ('0' & x(7 downto 1)); -- x(7) xor '0' = x(7)
end architecture;
```

y(7)	<=	x(7)	xor	'0'
y(6)	<=	x(6)	xor	x(7)
y(5)	<=	x(5)	xor	x(6)
y(4)	<=	x(4)	xor	x(5)
y(3)	<=	x(3)	xor	x(4)
y(2)	<=	x(2)	xor	x(3)
y(1)	<=	x(1)	xor	x(2)
y(0)	<=	x(0)	xor	x(1)

```
byte toGray(byte x)
{ return x ^ (x >> 1);
}
  /* The code supposes x of
     an unsigned type
```

&

*The VHDL **&** operator concatenates two strings,*
*is analogous to the "concatenation" operator '**.**' in PHP*

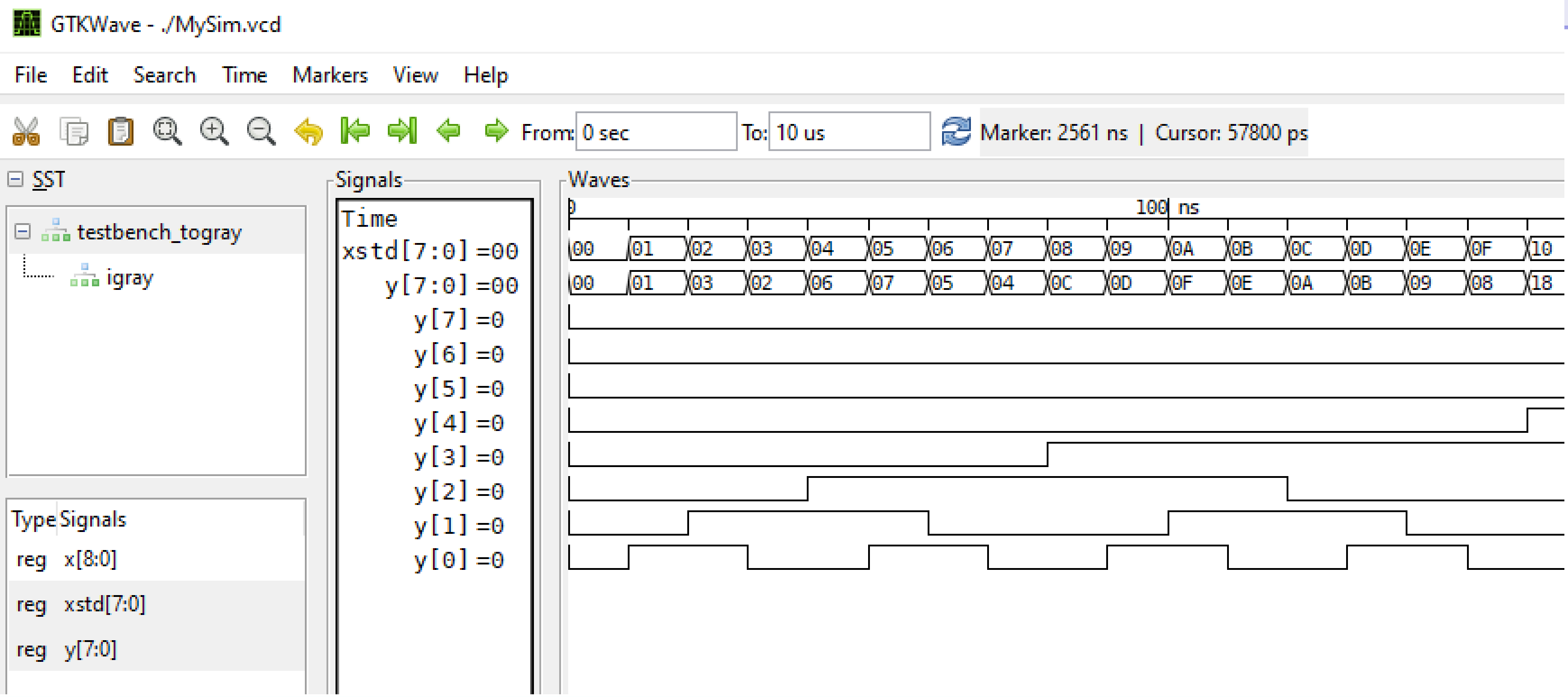
Universal VHDL code

```vhdl
/* ToGray converts Binary x input to Binary-reflected Gray number
N specifies bit lengths of x input and y output*/
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity ToGray is
   generic(N : integer:=8 ); -- bit length
   port( x : in std_logic_vector(N-1 downto 0);
         y : out std_logic_vector(N-1 downto 0));
   begin --The passive process of sequential domain checks parameters
      assert N>1 report "Required N bit length>1" severity failure;
end entity;
architecture dataflow of ToGray is
begin
   y <= x  xor ('0' & x(N-1 downto 1));
end architecture;
```

Generic

- *The generic parameter allows you to specify a different value each time you create an entity instance, and thus customize it. This is roughly analogous to the value that might be specified in normal programming when calling a class constructor.*
- *Generic parameter behaves like a readonly data member in C#, or as a read-only property (Java, C#) inside VHDL code.*
- *Several generic parameters can be used.*

Simulation



GTKWave - ./MySim.vcd

File Edit Search Time Markers View Help

From: 0 sec To: 10 us

Marker: 2561 ns | Cursor: 57800 ps

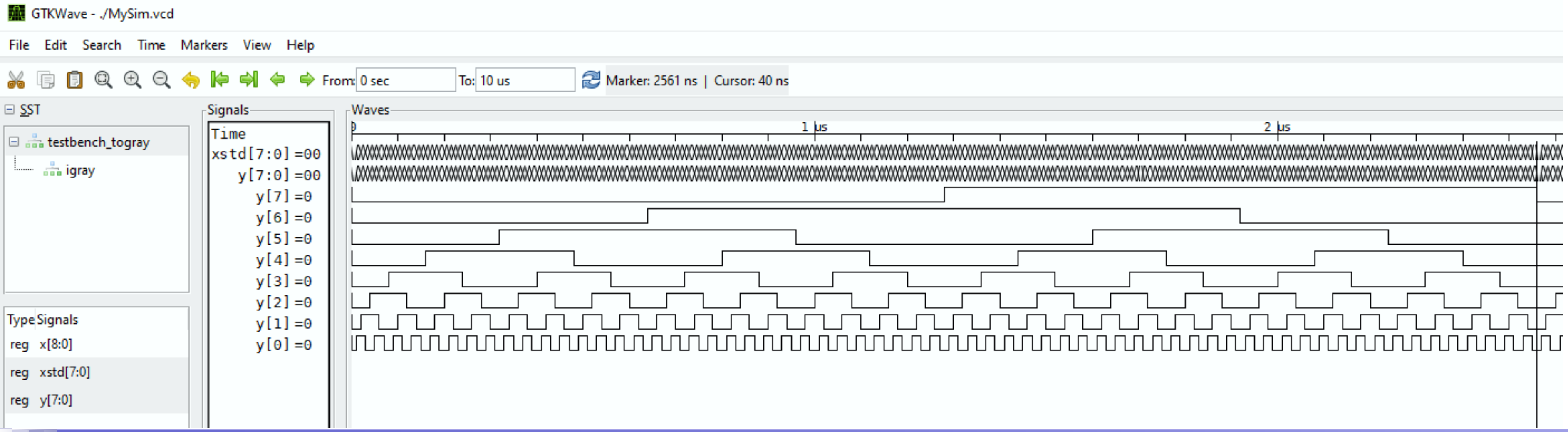
SST

- testbench_togray
  - igray

Type	Signals
reg	x[8:0]
reg	xstd[7:0]
reg	y[7:0]

```
Signals
Time
xstd[7:0] =00
    y[7:0] =00
      y[7] =0
      y[6] =0
      y[5] =0
      y[4] =0
      y[3] =0
      y[2] =0
      y[1] =0
      y[0] =0
```

Waves
0
100 ns
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
00 01 03 02 06 07 05 04 0C 0D 0F 0E 0A 0B 09 08 18



GTKWave - ./MySim.vcd

From: 0 sec To: 10 us

Marker: 2561 ns | Cursor: 40 ns

SST

- testbench_togray
  - igray

Type	Signals
reg	x[8:0]
reg	xstd[7:0]
reg	y[7:0]

```
Signals
Time
xstd[7:0] =00
  y[7:0] =00
    y[7] =0
    y[6] =0
    y[5] =0
    y[4] =0
    y[3] =0
    y[2] =0
    y[1] =0
    y[0] =0
```

Can we describe the whole MHL2 beacon with one multiplexer?

Yes, but we obtain awkward code...

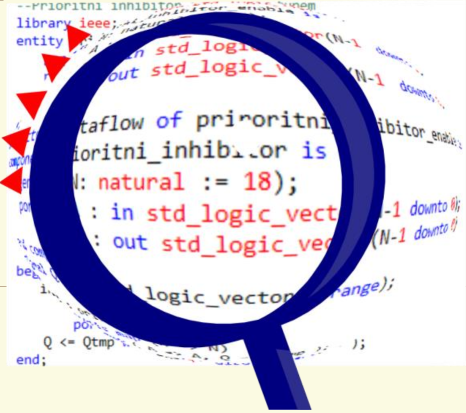
"MHL2" = "01110111000101010100010111010100010101110111011100"

MHL2 beacon in VHDL *very awkward style*

"MHL2" = "01110111000101010100010111010100010101110111011100"

```
library ieee; use ieee.std_logic_1164.all;
entity MuxMHL2 is port ( X : in std_logic_vector(5 downto 0);
                                             STOP, Y : out std_logic);
end entity;
architecture clumsy of MuxMHL2 is
begin
   with X select
Y <= '1' when "000001" | "000010" | "000011" | "000101" | "000110" | "000111" -- M
                       | "001011" | "001101" | "001111" | "010001" -- H
                       | "010101" | "010111" | "011001" | "011010" | "011011" | "011101" -- L
                       | "100001" | "100011" | "100101" | "100110" | "100111" --2
                       | "101001" | "101010" | "101011" | "101101" | "101110" | "101111",
      '0' when others; -- X has possible 9^6 values =531441
        STOP <= X(5) and X(4) and X(0);
end architecture;
```

Remember: VHDL comments can contain only ASCII characters!



Why is not MHL2 stored in an array?

- □ X is of std_logic_vector type, but the indexes to arrays are integer types.
- □ VHDL for synthesis is not compiled into assembler-like programming languages.
- □ VHDL describes circuits in which numbers have different lengths to minimize operation with them.
- □ Therefore, we should look at the standard numeric library ieee.numeric_std.

Types

signed, **unsigned** + **integer**

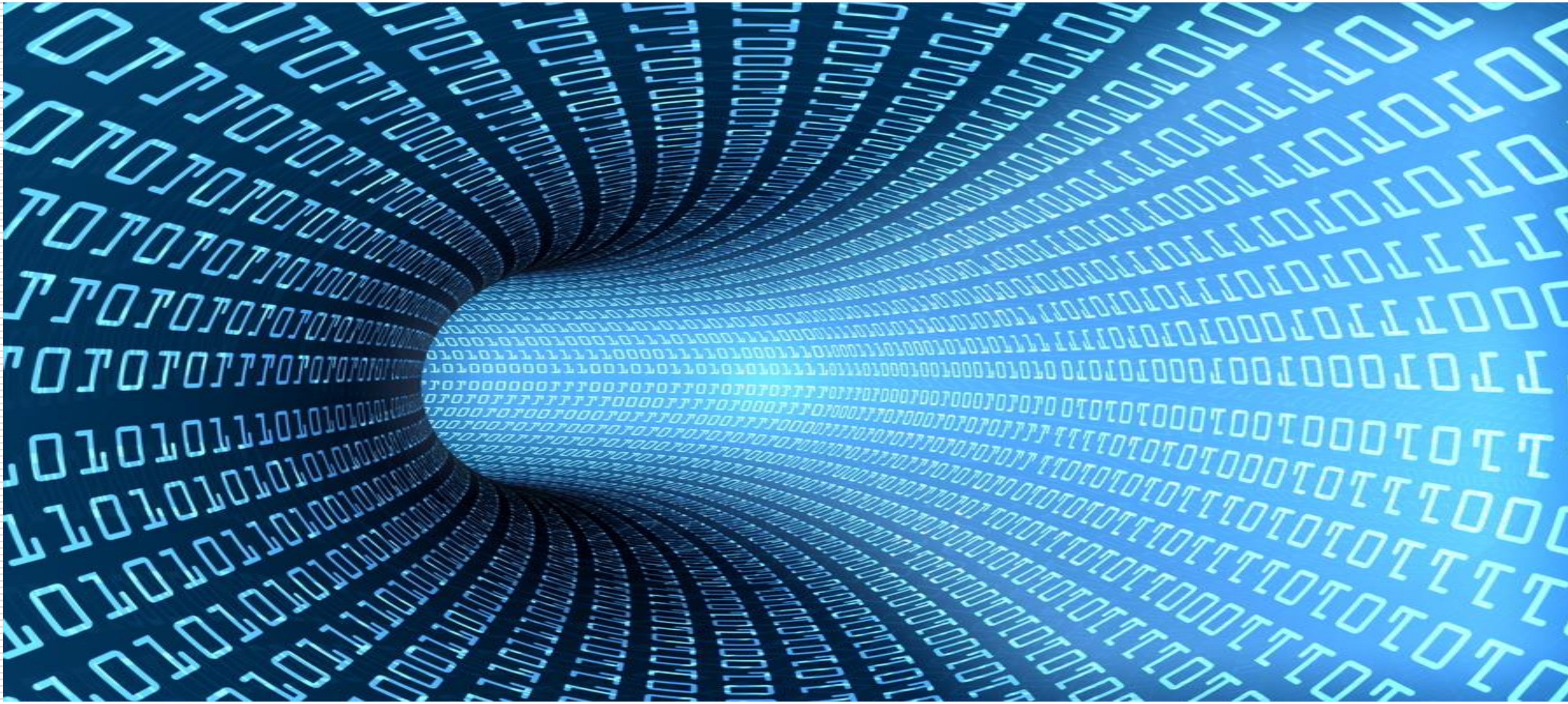


Image source: http://philontheprairie.files.wordpress.com

type / subtype

- **type** creates isolated data types;
- **subtype** places new types into
  a member group of an existing type.

```
type slvA is array (7 downto 0) of std_logic;
type slvB is array (7 downto 0) of std_logic;
signal slvA1, slvA2 : slvA;
signal slvB1, slvB2 : slvB;
```

```
slvA1 <= X "12"; slvA2 <= not slvA1; -- OK, it it allowed
```

```
slvB1 <= slvA1; -- Error - types do not match
```

```
subtype slvA is array (7 downto 0) of std_logic;
subtype slvB is array (7 downto 0) of std_logic;
signal slvA1, slvA2 : slvA;
signal slvB1, slvB2 : slvB;
```

```
slvA1 <= X "12"; slvA2 <= not slvA1; -- OK, can
```

```
slvB1 <= slvA1; slvB2 <= slvA2 xor slvB1; -- OK, can
```

Attributes

suitable also when a range is defined by generic

```
subtype slvA is array (7 downto 0) of std_logic;
subtype slvB is array (7 downto 0) of std_logic;
signal slvA1, slvA2 : slvA;
signal slvB1, slvB2 : slvB;
subtype slvC is std_logic_vector (5 to 27);
signal slvC : slvC; -- signal slvC : std_logic_vector (5 to 27);
```

Defined for all types

Attribute on	slvA*, slvB*	slvC	
LEFT	7	5	*left range value*
RIGHT	0	27	*true range value*
LOW	0	5	*lower numerical value of the range*
HIGH	7	27	*higher numerical value of the range*
LENGTH	8	23	*number of elements*
ASCENDING	*FALSE*	*TRUE*	*TRUE when defined with* ***to***
RANGE	7 **downto** 0	5 **to** 27	*definitional scope*
REVERSE_RANGE	0 **to** 7	27 **downto** 5	*inverted definitional range*

integer *in library* numeric_std

has the possibility of automatic management of its bit length

In the Quartus, VHDL integers are defined in the max ranges

-231** to **2**31-1**, i.e., -2147483648 to +2147483647, i.e.,

integer'LOW **to** integer'HIGH, *or* integer'LEFT **to** integer'RIGHT;

But in many cases, we should reduce their ranges, e.g., **variable**

variable inImg : integer **range** 0 **to** 2 := 0;

-- LCD image, 0-none, 1 upper or 2 lower

variable x, y : integer:=0; *-- integers derived from xcolumn and yrow*

-- They do not need ranges that are correctly derived from assignments.

Integer subtypes

- ❑ **natural** has limited range to 0 to 2**31-1
- ❑ **positive** covers range from 1 to 2**31-1

signed / unsigned

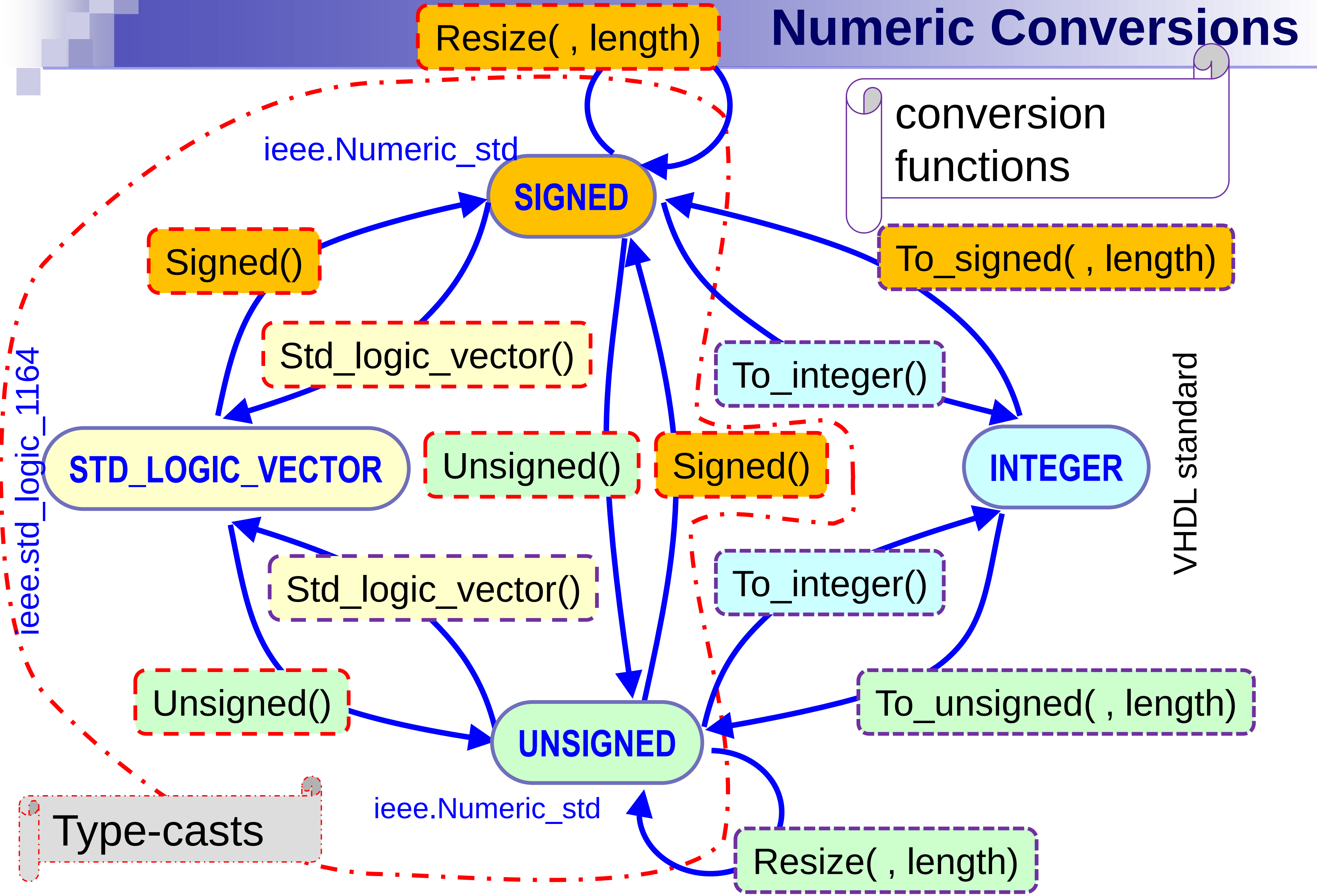
in library numeric_std

```
type std_logic_vector
        is array (natural range <>) of std_logic;
type unsigned
      is array (natural range <>) of std_logic;
type signed is array (natural range <>) of std_logic;
```

***range* <>**   *means not specified (unconstrained)*

→ *In such cases, range must be given in the definition.*

Numeric Conversions



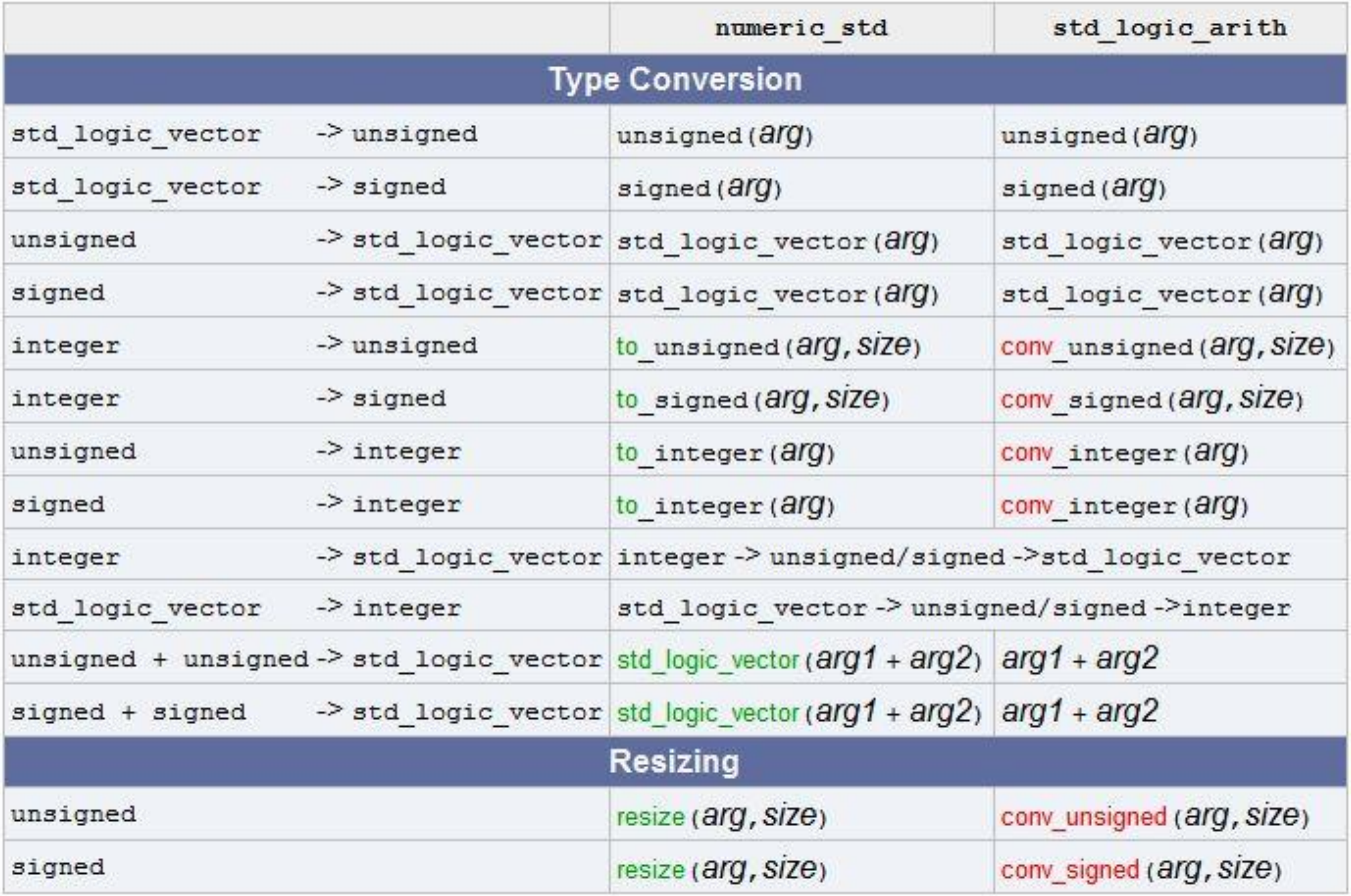
Standard numeric library

Warning:

- *On the web you can find codes with older package IEEE.STD_LOGIC_ARITH.ALL*
  - this one requires different conversion functions and rewrite methods.
- *The use of the IEEE.STD_LOGIC_ARITH library is prohibited in the LSP subject*
  in accordance with IEEE recommendations.

If you want, you can shorten the conversions by defining your own functions, like these (they can be used in both sequential and concurrent source code domains):

Numeric_std versus std_logic_arith



	numeric_std	std_logic_arith
Type Conversion		
std_logic_vector -> unsigned	unsigned(*arg*)	unsigned(*arg*)
std_logic_vector -> signed	signed(*arg*)	signed(*arg*)
unsigned -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
signed -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
integer -> unsigned	to_unsigned(*arg*,*size*)	conv_unsigned(*arg*,*size*)
integer -> signed	to_signed(*arg*,*size*)	conv_signed(*arg*,*size*)
unsigned -> integer	to_integer(*arg*)	conv_integer(*arg*)
signed -> integer	to_integer(*arg*)	conv_integer(*arg*)
integer -> std_logic_vector	integer -> unsigned/signed ->std_logic_vector	
std_logic_vector -> integer	std_logic_vector -> unsigned/signed ->integer	
unsigned + unsigned -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
signed + signed -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
Resizing		
unsigned	resize(*arg*,*size*)	conv_unsigned(*arg*,*size*)
signed	resize(*arg*,*size*)	conv_signed(*arg*,*size*)

From: http://dz.ee.ethz.ch/support/ic/vhdl/vhdlsources.en.html

Operators signed, unsigned

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

result	operand	operand
unsigned	unsigned	unsigned
unsigned	unsigned	natural
unsigned	natural	unsigned
signed	signed	signed
signed	signed	Integer
signed	Integer	signed

not and or nand nor
xor xnor

result	operand	operand
unsigned	unsigned	unsigned
signed	signed	signed

Summary:

- signed and unsigned types **cannot be** used together in one operation
- the types of signed and unsigned are **not changed** by the operation
- we cannot subtract from unsigned

C language equivalents in 64 bit compilers

C type	VHDL equivalents	
char	character or signed (7 downto 0)	integer range -2**7 to 2**7-1
byte	unsigned (7 downto 0)	integer range 0 to 2**8-1
short int	signed (15 downto 0)	integer range -2**15 to 2**15-1
short unsigned int	unsigned (15 downto 0)	integer range 0 to 2**16-1
int *or* long	signed (31 downto 0)	integer *without any range*
unsigned int *or* unsigned long int	unsigned (31 downto 0)	
long long int	signed (63 downto 0)	
unsigned long long int	unsigned (63 downto 0)	

Integer versus Signed/Unsigned

Advantages of integers

- easier calculations and
- their ranges specified by min/max values

Drawbacks of integers

- no direct bit operations or logical operations
- limited ranges, practically about to -2*28 to 2**28, higher integers begin overflow

Advantages of signed/unsigned

- defined logical operations
- direct access to their bits
- their sizes limited only by used hardware

Drawbacks of signed/unsigned

- not so user-friendly for numeric calculations
- we need to know their bit lengths in compile-times

Width in Bits versus Maximal Values

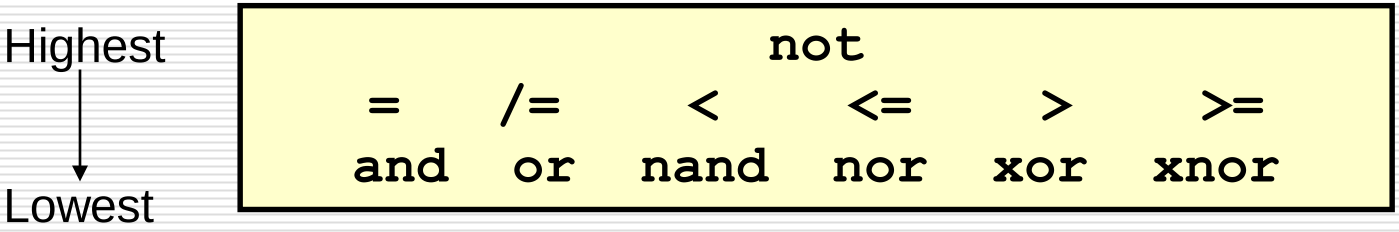
```
constant MAXVALUE : integer:=255; -- = 2^8-1

/* Float point numbers can be used
   if evaluated during compile times or simulations. */

use ieee.math_real.all;

constant BITWIDTH: positive :=

    positive ( ceil( log2( real(MAXVALUE+1) ) ) );

constant MAXINT : integer:= 2**BITWIDTH-1;
```

Precedence of logical operators

- □ Relational operators

```
=    /=    <    <=    >    >=
```

- □ Precedence



Highest

Lowest

```
                 not
=    /=    <    <=    >    >=
and  or  nand  nor  xor  xnor
```

There is no priority between operators.

Logical operators are defined for

- boolean and bit
- all types derived from the **std_logic** data type, as **unsigned** and **signed.**
- **but not** for integer types

If we have signal A, B, Z: unsigned(7 downto 0), then

Z <= A xor B;

Z(7)	<=	A(7)	xor	B(7)
Z(6)	<=	A(6)	xor	B(6)
Z(5)	<=	A(5)	xor	B(5)
Z(4)	<=	A(4)	xor	B(4)
Z(3)	<=	A(3)	xor	B(3)
Z(2)	<=	A(2)	xor	B(2)
Z(1)	<=	A(1)	xor	B(1)
Z(0)	<=	A(0)	xor	B(0)

Warning: Quartus Lite does not support VHDL2008 reduction operators of type **and**(Z)

Examples of signed conversions

```
architecture dataflow of conversions is
    signal slv, slv2 : std_logic_vector (3 downto 0);
    signal si: integer range -8 to 7;
    signal sv, sv2: signed (3 downto 0);
begin
    slv<="1111"; sv<=signed (slv); -- -1
-- between std_logic_vector and integer is converted via signed
    si <=to_integer(sv);
    sv2<=to_signed (si, 4);
    slv3<=std_logic_vector (sv2);
end architecture;
```

Examples of unsigned conversions

```
architecture dataflow2 of conversions is
    signal slv, slv2 : std_logic_vector (3 DOWNTO 0);
    signal ui: integer range 0 to 15;
    signal uv, uv2: unsigned (3 DOWNTO 0);
begin
    slv<="1111"; uv<=unsigned (slv); --uv=15
-- between std_logic_vector and integer is converted via unsigned
    ui <= to_integer (uv);
    uv2 <= to_unsigned(ui,4);
    slv2 <= std_logic_vector (uv2);
end architecture;
```

Sequential VHDL domain

the next lecture topic

is surrounded by keywords:

- **function ... end function;**
- **procedure ... end procedure;**
- **process ... end process;**

and by a passive entity process.

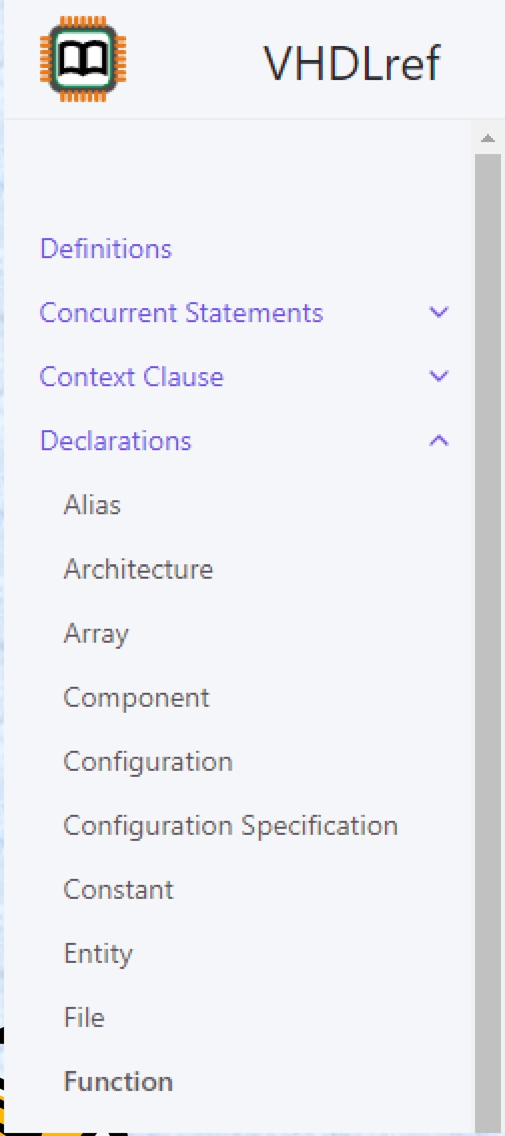
However, the sequential domain is compiled into concurrent statements.
Therefore, we can insert it into our
VHDL concurrent domain code.

Example: Simpler unsigned conversions

```
architecture dataflow3 of conversions is
   signal slv, slv2: std_logic_vector (3 downto 0);
   signal ui: integer range 0 to 15;
   function SV2UInt(x:std_logic_vector) return integer is
   begin return to_integer(unsigned(x)); end function;
   function UInt2SV(x, size:integer) return std_logic_vector is
   begin return std_logic_vector(to_unsigned(x, size)); end function;

begin
    slv<="1111"; ui<=SV2UInt(slv);
    slv2<=UInt2SV(ui, slv2'LENGTH);
end architecture;
```

Examples of References with Examples:-)

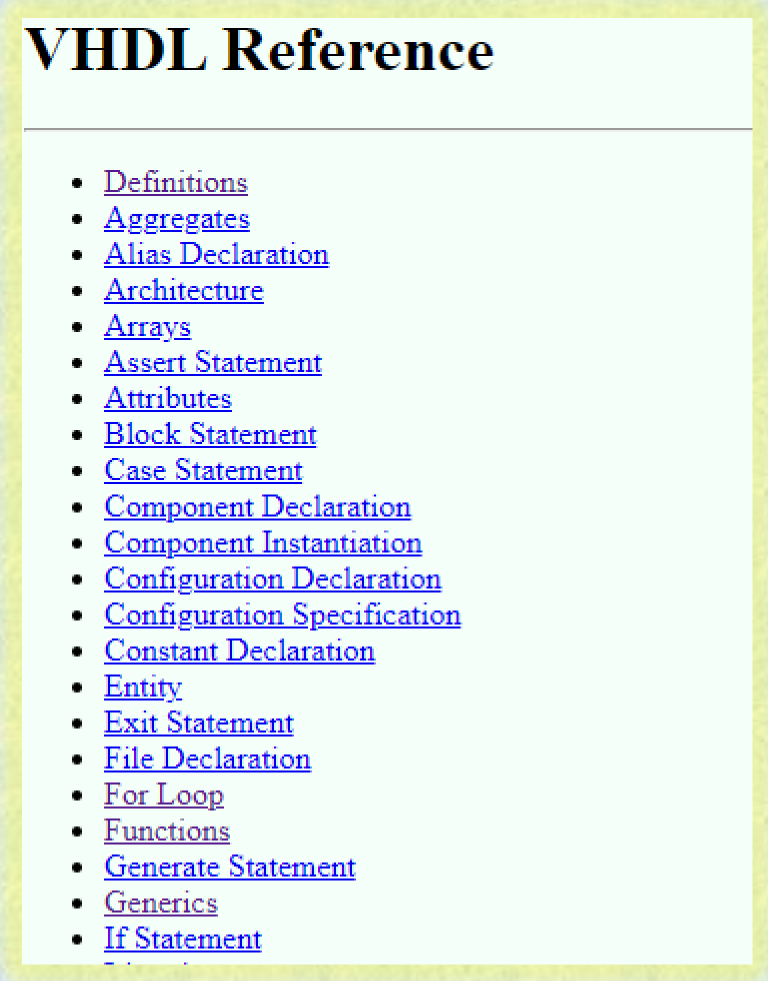
https://blog.eowyn.net/vhdlref/



VHDLref

- Definitions
- Concurrent Statements
- Context Clause
- Declarations
  - Alias
  - Architecture
  - Array
  - Component
  - Configuration
  - Configuration Specification
  - Constant
  - Entity
  - File
  - Function

https://ics.uci.edu/~jmoorkan/vhdlref/



VHDL Reference

- Definitions
- Aggregates
- Alias Declaration
- Architecture
- Arrays
- Assert Statement
- Attributes
- Block Statement
- Case Statement
- Component Declaration
- Component Instantiation
- Configuration Declaration
- Configuration Specification
- Constant Declaration
- Entity
- Exit Statement
- File Declaration
- For Loop
- Functions
- Generate Statement
- Generics
- If Statement

Inputs and outputs are the same

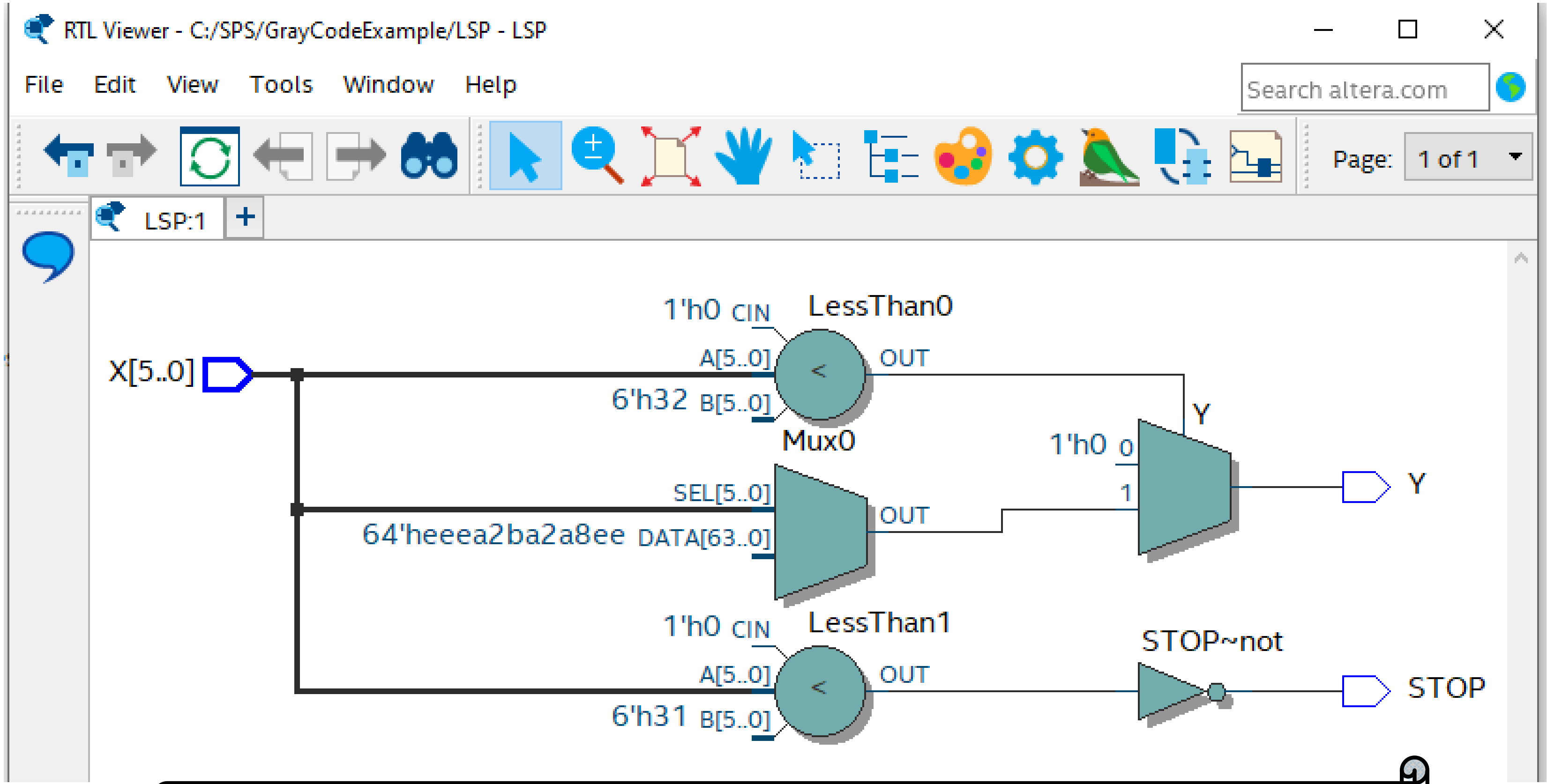
```
    port( X : in std_logic_vector(5 downto 0);
            m, stop : out std_logic); -- beacon output and stop signal indicating the end
```

Wiring

```
architecture witharray of MuxMHL2 is
signal index : integer range 0 to 2**X'LENGTH-1 :=0; -- X to integer index
constant data : std_logic_vector :=
      "01110111000101010100010111010100010101110111011100";
-- without the specified range, the range is always from 0 to data'LENGTH - 1
begin  index <= to_integer(unsigned(X)); -- X to integer index
            Y <= data(index) when index < data'LENGTH  else '0';
            STOP <= '0' when index < (data'LENGTH - 1) else '1';
end architecture;
```

⇨

MHL2 in RTL Viewer



We changed the VHDL description but it will be implemented again by multiplexer as the previous code.

And why should we solve first task so archaically?

Thou shouldst understanden logique parchments…

[note: The sentence above - Middle Age English]

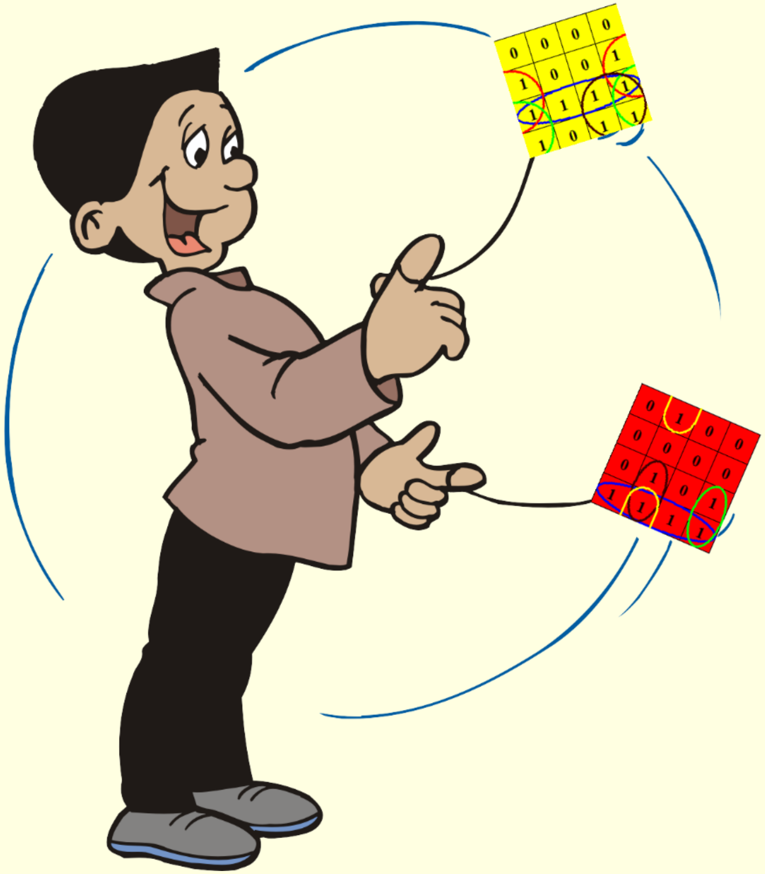
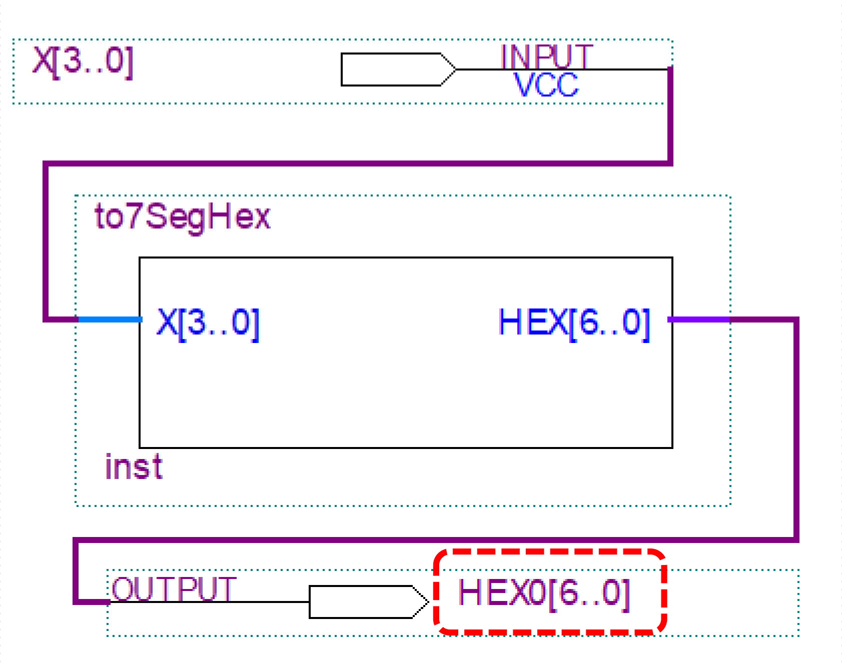
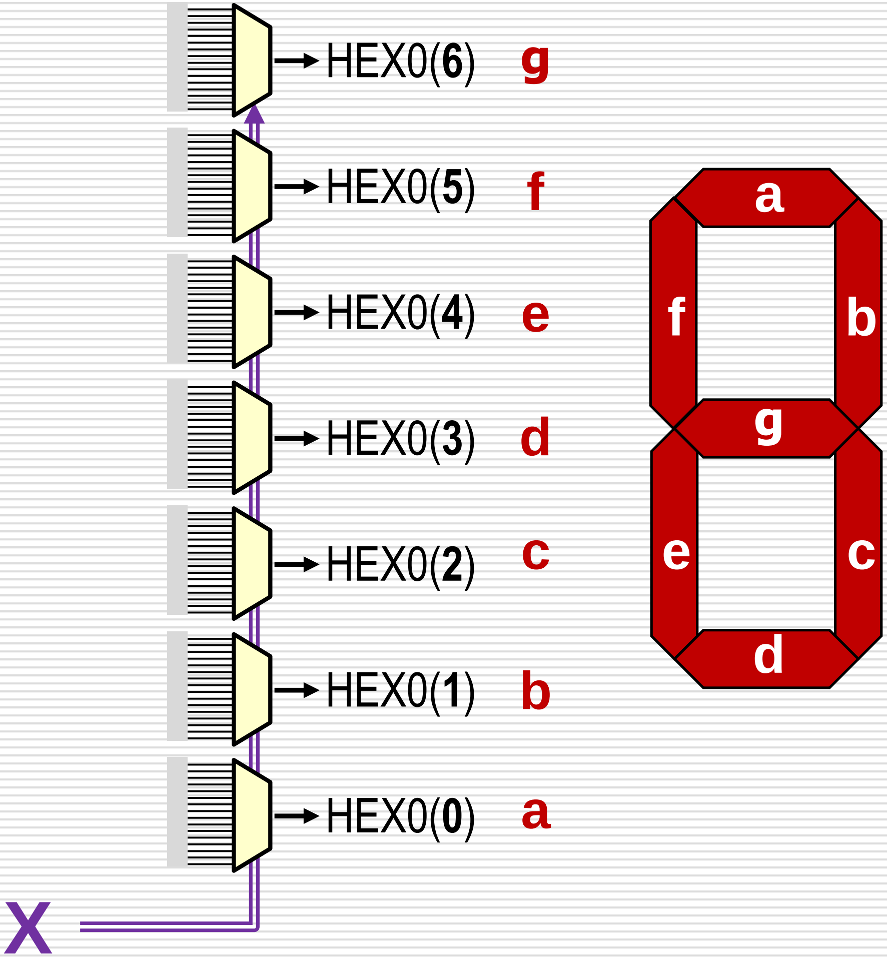


Image created by editing a motif from Corel-Clipart library

DCE/library to7SegHex





to7SegHex written directly by the multiplexer

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity to7SegHex IS
  port( X :in std_logic_vector(3 downto 0); -- X(3)-MSB X(0)-LSB hex input
        HEX: out std_logic_vector(6 downto 0) );-- HEX[6..0] output to 7 segment
end entity;
architecture beh1 of to7SegHex IS
begin-- On VEEK-MT2 board, 7segment HEX LEDs are on when their inputs = '0'
   with X select
      HEX <= "1000000" when "0000", "1111001" when "0001", -- 0, 1
      "0100100" when "0010", "0110000" when "0011", -- 2, 3
      "0011001" when "0100", "0010010" when "0101", -- 4, 5
      "0000010" when "0110", "1111000" when "0111", -- 6, 7
      "0000000" when "1000", "0010000" when "1001", -- 8, 9
      "0001000" when "1010", "0000011" when "1011", -- A, b
      "1000110" when "1100", "0100001" when "1101", -- C, d
      "0000110" when "1110", "0001110" when others; -- E, F "1111"
end architecture;
```

to7SegHex written by arry

```
-- the code already uses ieee.numeric_std.all
 architecture beh2 of to7SegHex is
 type hexar_t is array(0 to 15) of std_logic_vector(HEX'RANGE);
 constant HEXARRAY : hexar_t :=
          ("1000000","1111001","0100100","0110000",
           "0011001","0010010","0000010","1111000",
           "0000000","0010000","0001000","0000011",
           "1000110","0100001","0000110","0001110");
 begin
     HEX<= HEXARRAY(to_integer(unsigned(X)));
 end architecture;
```

The resulting circuit is implemented in the same way as the previous one.

Logic element in FPGA

Cyclone II

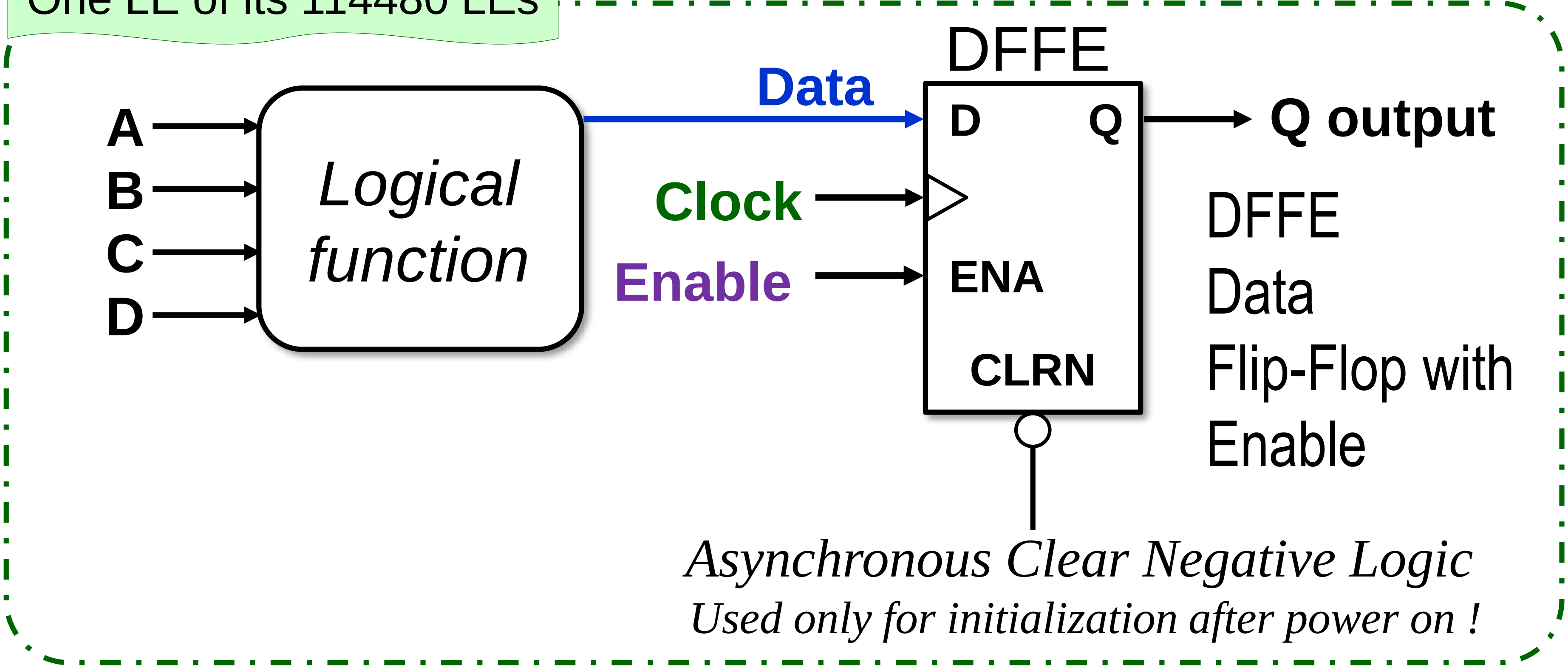
Cyclone IV

Outline of LE's structure in Cyclone IV family

Cyclone IV EP4CE115F29C7 in our Veek-MT boards contains **114480** LEs.

LE=Logical element

One LE of its 114480 LEs



DFFE
Data
Flip-Flop with
Enable

Asynchronous Clear Negative Logic
Used only for initialization after power on !

The vast majority of FPGAs use DFFE,
because it's easy to wire from CMOS transistors.

Register in VHDL

needs its exact structure suitable for LEs

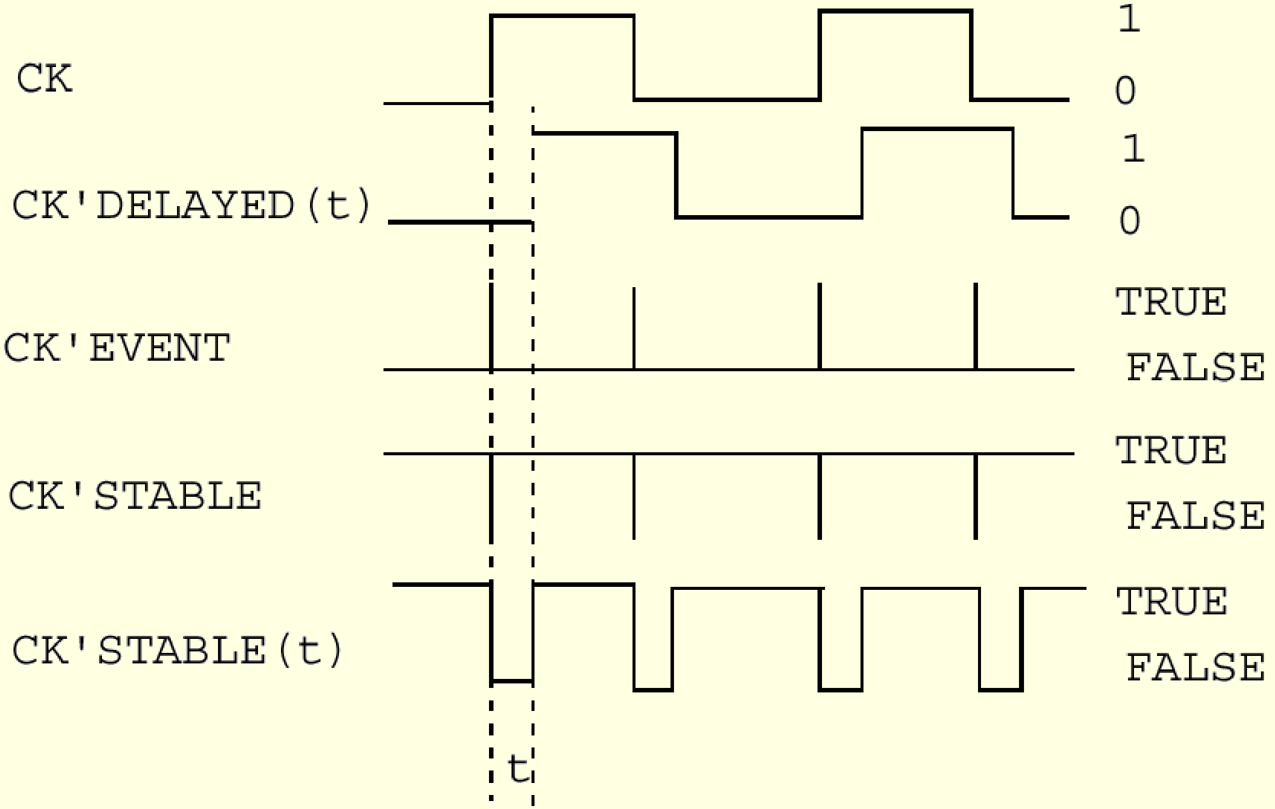
Signal-related attributes

VHDL signal attributes can be divided into two groups:

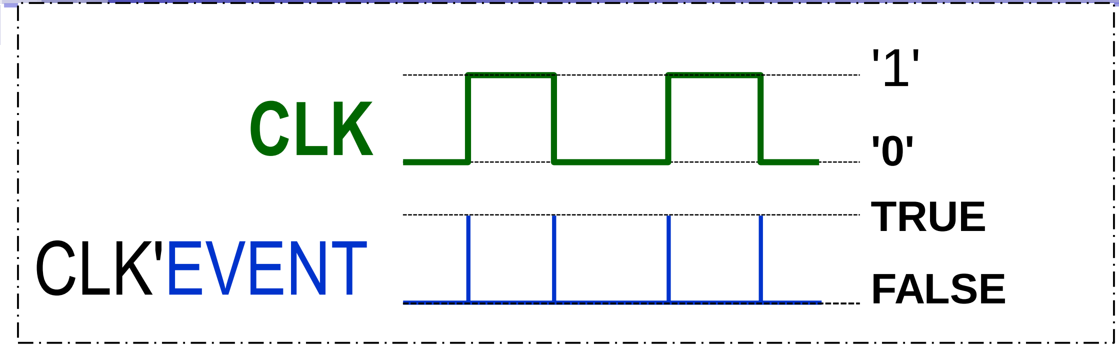
- attributes for signal definitions
- attributes informing about signals.

Only the EVENT attribute can be used in the synthesis!

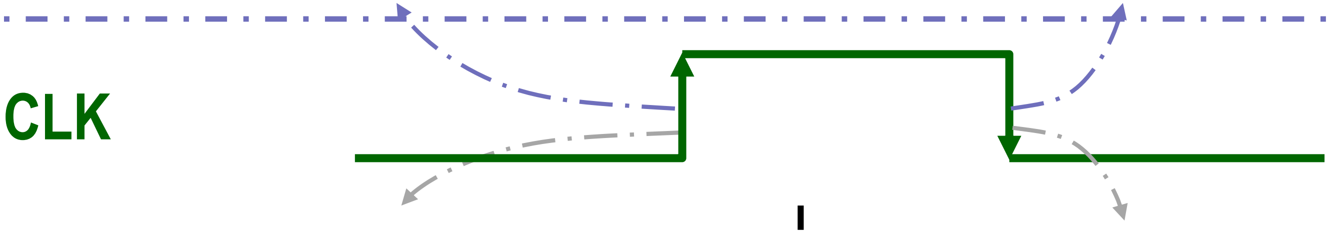
Attributes are also signals



Signal edge detection



Preferred **if rising_edge(CLK) then** **if falling_edge(CLK) then**



older version if CLK'EVENT and CLK='1' then | if CLK'EVENT and CLK='0' then

Edge detection
- beware of codes from the web

```
FUNCTION rising_edge (SIGNAL s : std_ulogic)
 RETURN BOOLEAN IS
   BEGIN RETURN ( s'EVENT AND (To_X01(s) = '1')
                                      AND (To_X01(s'LAST_VALUE) = '0') );
   END;
```

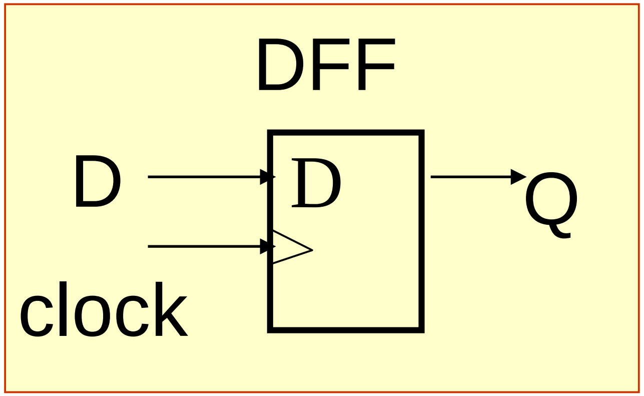
Construct if rising_edge(clk) then
is better than if (clk'event and clk='1') then
because rising_edge *only detects changes*
'0'->'1' *and ignores edges* 'U'->1, 'X'->1, *etc.*

The same applies to if falling_edge(clk) then

Note: The ***To_X01()*** *function belongs to the* std_logic_1164 *standard library and converts to 'X' all values in std_logic that are different from '0' and '1'*

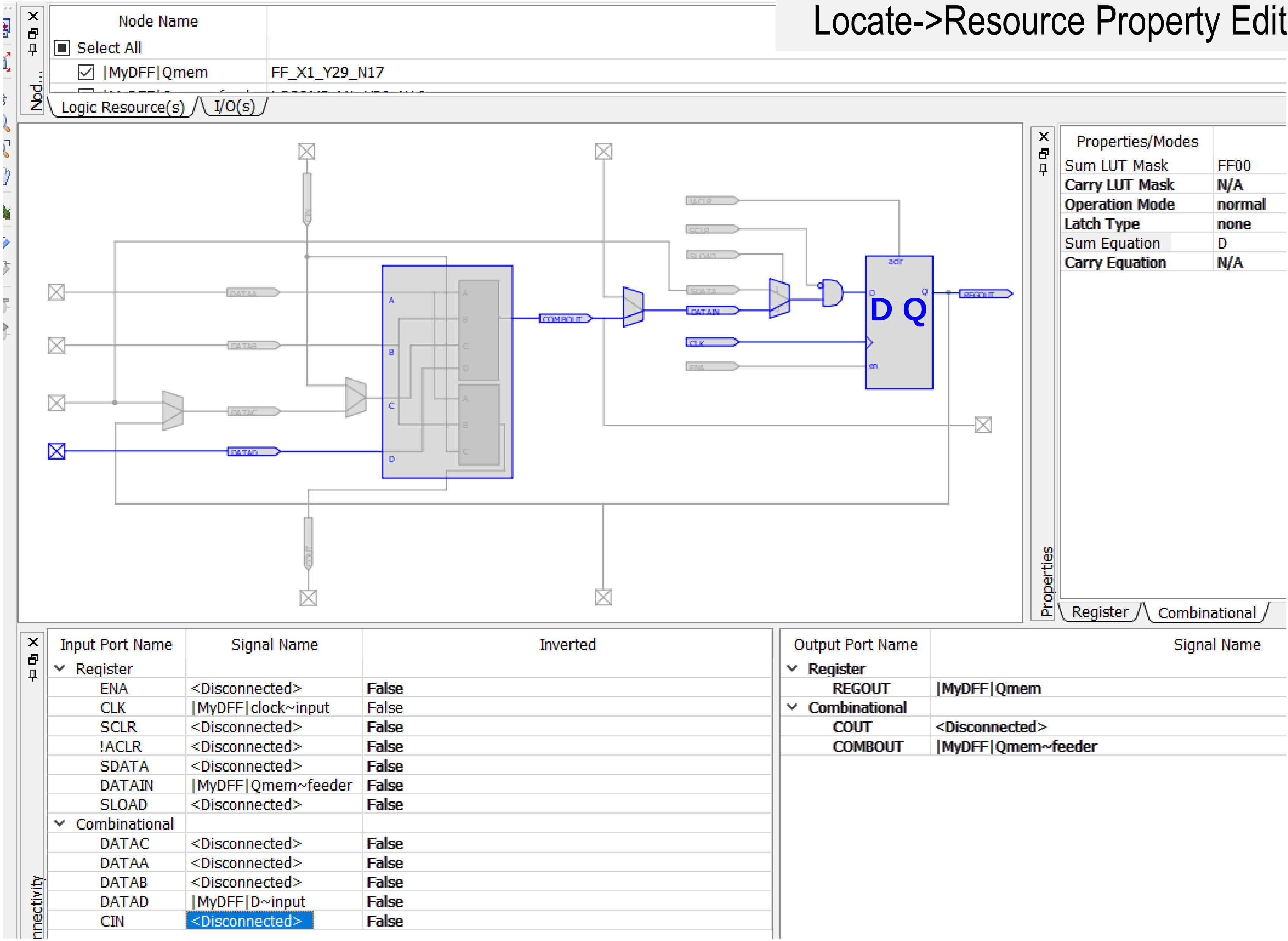
DFF without asynchronous input

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

entity MyDFF is
     port( D : in std_logic; -- data input
             clock : in std_logic;
             Q: out std_logic ); -- output
end entity;

architecture behl of MyDFF is
signal Qmem : std_logic :='0'; --the initialization is only for simulations
begin
  Qmem<= D when rising_edge(clock);
  Q <= Qmem;
end architecture;
```



MyDFF inside the Logical element

Locate->Resource Property Editor



Node Name	
■ Select All	
☑ |MyDFF|Qmem	FF_X1_Y29_N17

Properties/Modes	
Sum LUT Mask	FF00
Carry LUT Mask	N/A
Operation Mode	normal
Latch Type	none
Sum Equation	D
Carry Equation	N/A

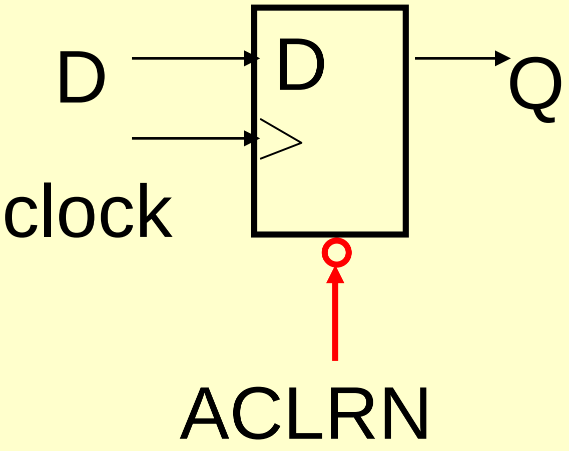
Input Port Name	Signal Name	Inverted
Register		
ENA	<Disconnected>	False
CLK	|MyDFF|clock~input	False
SCLR	<Disconnected>	False
!ACLR	<Disconnected>	False
SDATA	<Disconnected>	False
DATAIN	|MyDFF|Qmem~feeder	False
SLOAD	<Disconnected>	False
Combinational		
DATAC	<Disconnected>	False
DATAA	<Disconnected>	False
DATAB	<Disconnected>	False
DATAD	|MyDFF|D~input	False
CIN	<Disconnected>	False

Output Port Name	Signal Name
Register	
REGOUT	|MyDFF|Qmem
Combinational	
COUT	<Disconnected>
COMBOUT	|MyDFF|Qmem~feeder

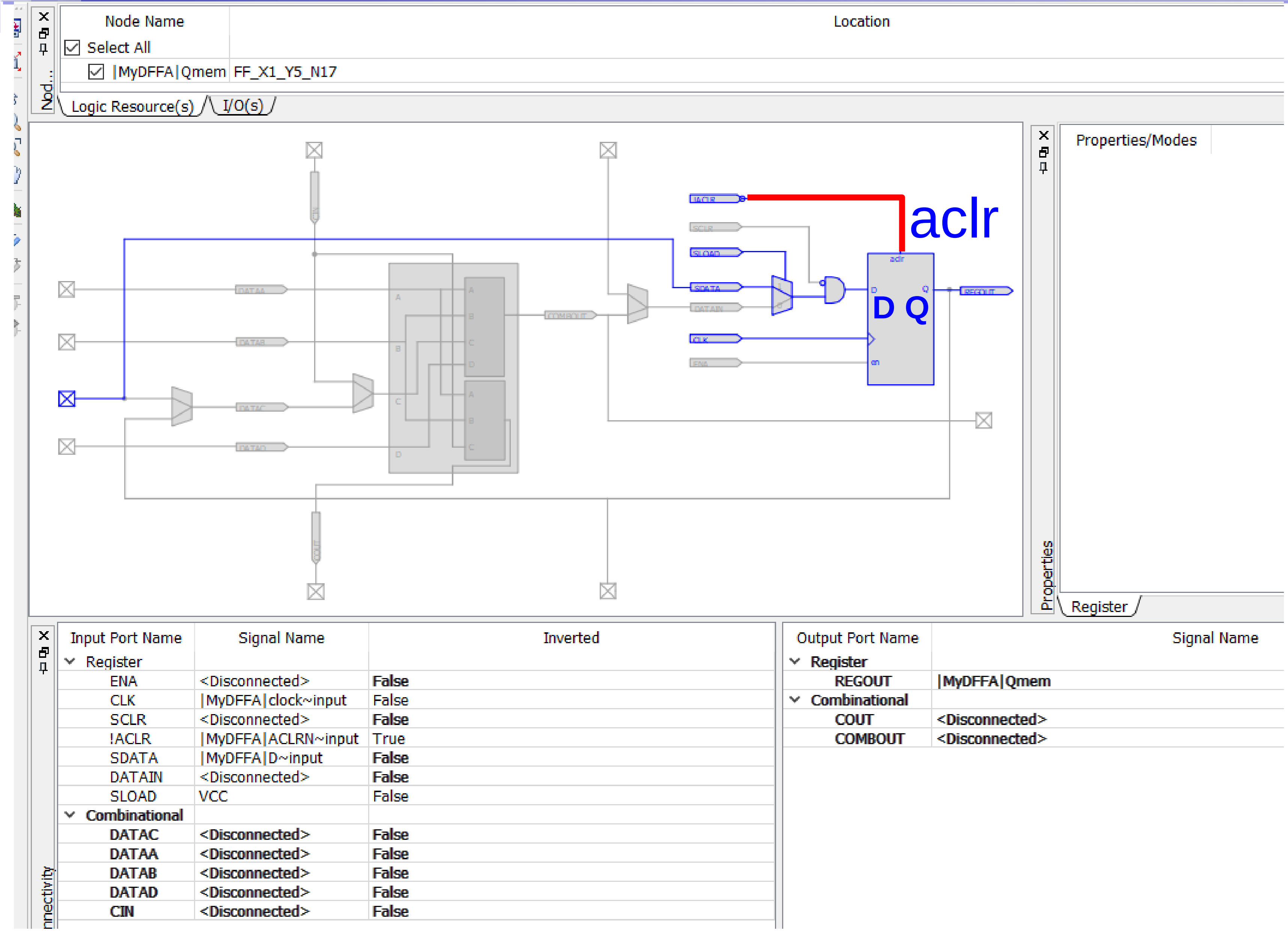
DFF with asynchronous input

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

entity MyDFFA is
     port( D : in std_logic; -- data input
        clock : in std_logic;
     ACLRN : in std_logic; -- clear
           Q : out std_logic ); -- output
end entity;

architecture behl of MyDFFA is
signal Qmem : std_logic :='0'; --the initialization is only for simulations
begin
  Qmem<= '0' when not ACLRN else
                D when rising_edge(clock);
  Q <= Qmem;
end architecture;
```

DFF with ACLRN



MyDFFA inside the Logical Element



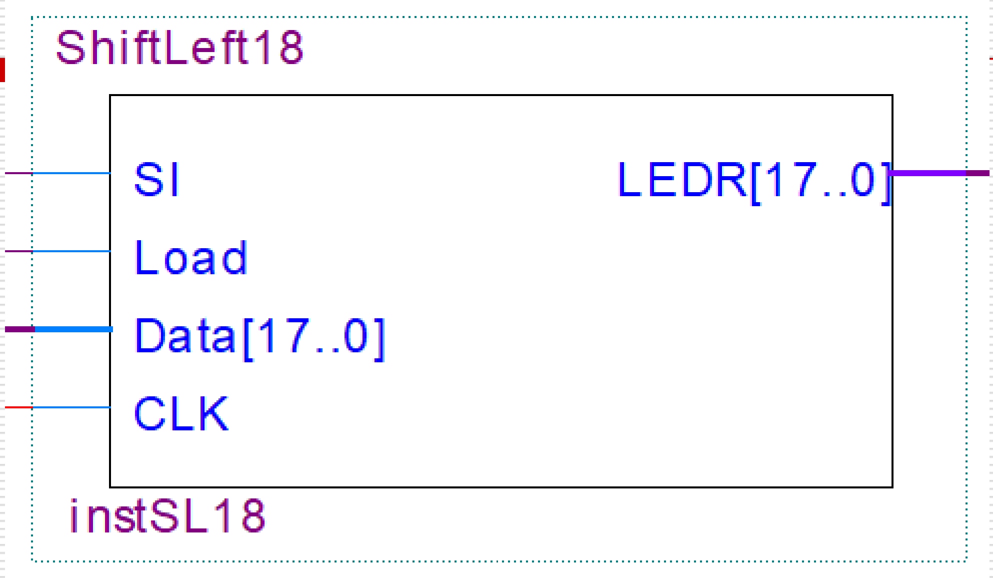
Node Name	Location
☑ Select All	
☑ |MyDFFA|Qmem	FF_X1_Y5_N17

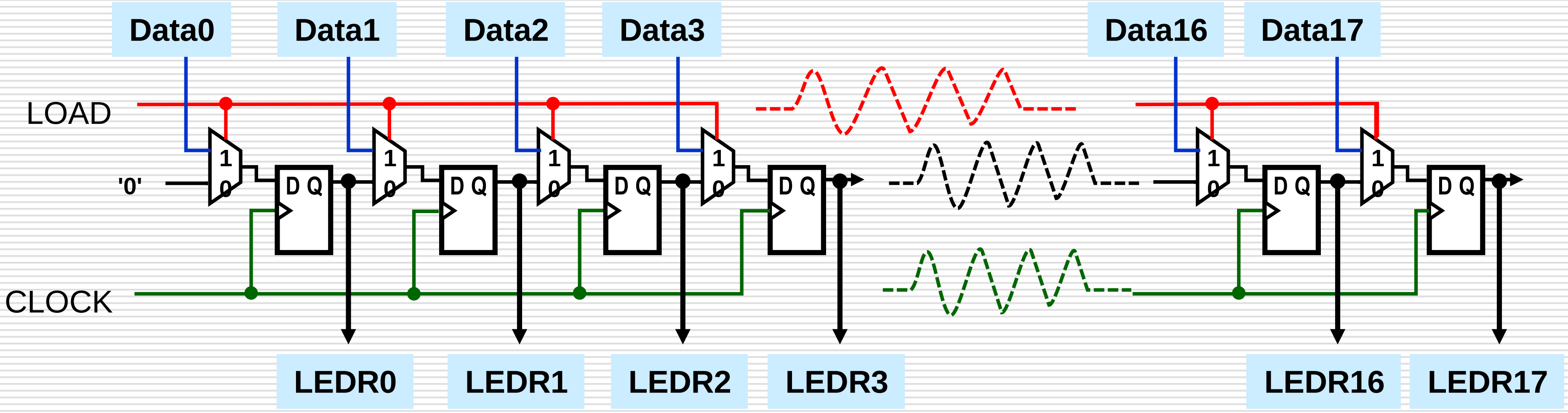
Logic Resource(s) / I/O(s)

Input Port Name	Signal Name	Inverted
Register		
ENA	<Disconnected>	False
CLK	|MyDFFA|clock~input	False
SCLR	<Disconnected>	False
!ACLR	|MyDFFA|ACLRN~input	True
SDATA	|MyDFFA|D~input	False
DATAIN	<Disconnected>	False
SLOAD	VCC	False
Combinational		
DATAC	<Disconnected>	False
DATAA	<Disconnected>	False
DATAB	<Disconnected>	False
DATAD	<Disconnected>	False
CIN	<Disconnected>	False

Output Port Name	Signal Name
Register	
REGOUT	|MyDFFA|Qmem
Combinational	
COUT	<Disconnected>
COMBOUT	<Disconnected>

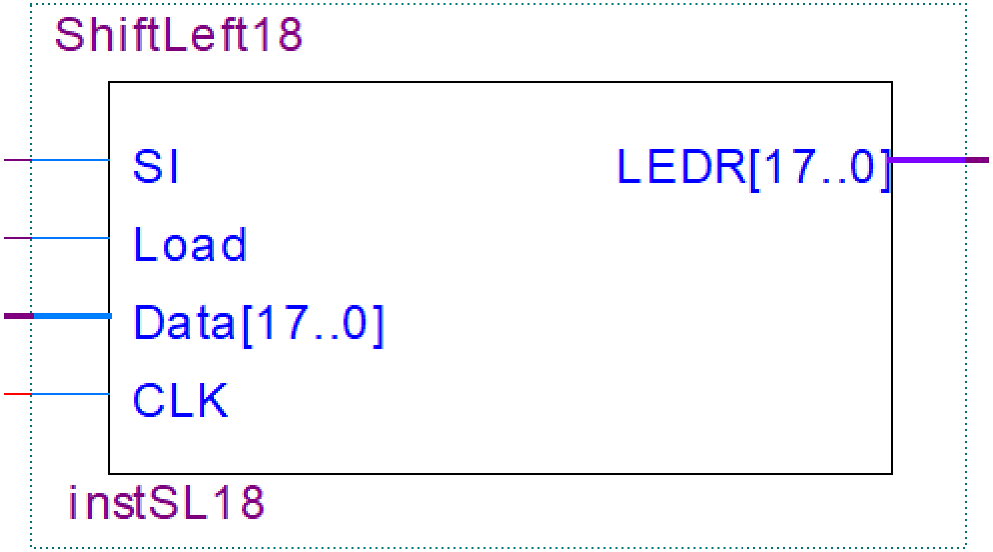
DCE/library Shifts





ShiftLeft18

```
library ieee; use ieee.std_logic_1164.all;
entity ShiftLeft18 is
   port( SI, Load, CLK :in std_logic;
         Data: in std_logic_vector(17 downto 0);
         LEDR: out std_logic_vector(17 downto 0));
end entity;
```



ShiftLeft18

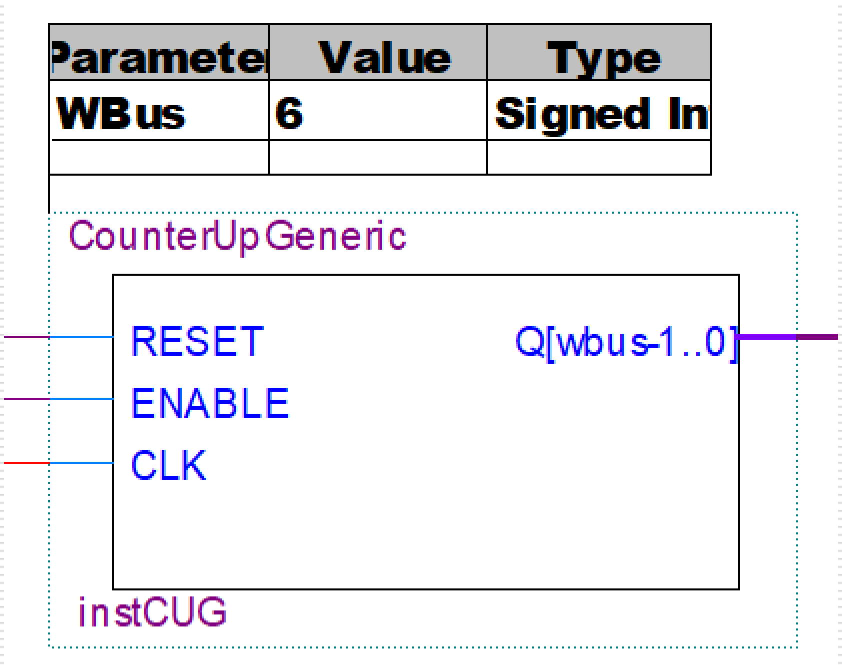
```
library ieee; use ieee.std_logic_1164.all;
entity ShiftLeft18 is
  port( SI, Load, CLK :in std_logic;
         Data: in std_logic_vector(17 downto 0);
         LEDR: out std_logic_vector(17 downto 0));
end entity;
architecture rtl of ShiftLeft18 is
signal rg, rnew : std_logic_vector(LEDR'RANGE):=(others=>'0');
begin
  rnew <= Data when Load else
             rg(LEDR'LEFT-1 downto 0) & SI;
  rg <= rnew when rising_edge(CLK);
  LEDR<=rg;
end architecture;
```

Comparison of ShiftLeftRight18 and ShiftLeft18

```
architecture rtlLR of ShiftLeftRight18 is
signal rg, rnew : std_logic_vector(LEDR'RANGE):=(others=>'0');
begin
rnew <= Data when Load else
        SI & rg(LEDR'LEFT downto 1) when RShift else
        rg(LEDR'LEFT-1 downto 0) & SI;
rg <= rnew when rising_edge(CLK);
LEDR<=rg;
end architecture;
```

```
architecture rtl of ShiftLeft18 is
signal rg, rnew : std_logic_vector(LEDR'RANGE):=(others=>'0');
begin
  rnew <= Data when Load else
          rg(LEDR'LEFT-1 downto 0) & SI;
  rg <= rnew when rising_edge(CLK);
  LEDR<=rg;
end architecture;
```

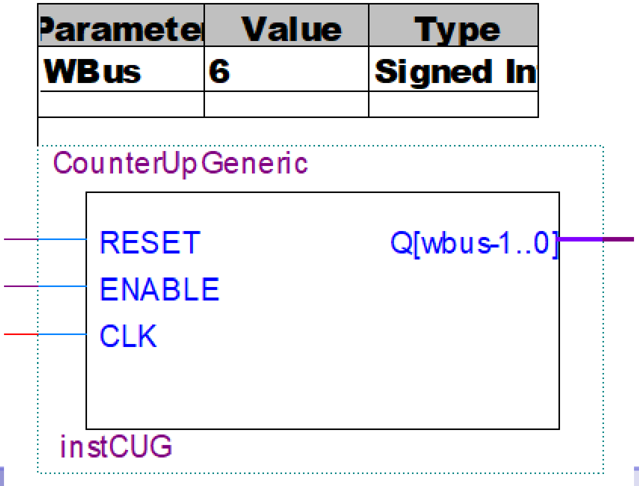
DCE/library CounterUpGeneric



Paramete	Value	Type
WBus	6	Signed In

CounterUpGeneric

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity CounterUpGeneric is
        generic(WBus:integer:=6 ); --Width (Length) of Q bus
        port(RESET, ENABLE, CLK : in std_logic;
              Q : out std_logic_vector(WBus-1 downto 0));
        begin assert WBus>0
                  report "Excepted the width of bus WBus>0"
                 severity ERROR;
end entity;
```



Paramete	Value	Type
WBus	6	Signed In

CounterUpGeneric

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity CounterUpGeneric is
      generic(WBus:integer:=6 ); --Width (Length) of Q bus
      port(RESET, ENABLE, CLK : in std_logic;
           Q : out std_logic_vector(WBus-1 downto 0));
      begin assert WBus>0 report "Excepted the width of bus WBus>0" severity ERROR;
end entity;
architecture rtl of CounterUpGeneric is
signal cnt, cntnew : unsigned(Q'RANGE):=(others=>'0');
begin
  cntnew <= (others=>'0') when RESET else cnt + 1;
  cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
  Q<=std_logic_vector(cnt);
end architecture;
```

ENABLE='1' ***with*** rising_edge is implemented by DFFE.

Condition Operator

The VHDL language provides a number of language constructs that require a condition to control the actions to be performed, where the condition is created, for example, by using relational and logical operators. In earlier versions of VHDL, one had to compare, e.g.

```
    ... '0' when RESET = '1' else...
```

VHDL-2008 provides two new language features that allow you to treat an std_logic value expression as a condition.

The first of these functions is the "??" condition operator, which converts a bitwise or std_logic value to a boolean value.

In std_logic "???" converts both '1' and 'H' values to TRUE and all other values to FALSE. The operator can be overloaded for other user-defined types. The above command condition allows VHDL 2008 to write:

```
... '0' when ?? RESET else...
```

The operator ?? is used implicitly in the condition when,

- if the expression cannot be interpreted with a different result
- and **there is a unique interpretation** using the condition operator that leads to a logical result.

So we rewrite the above condition

```
... '0' when RESET else...
```

However, there is no clear interpretation in conjunction with rising_edge(CLK), which returns a boolean, so either

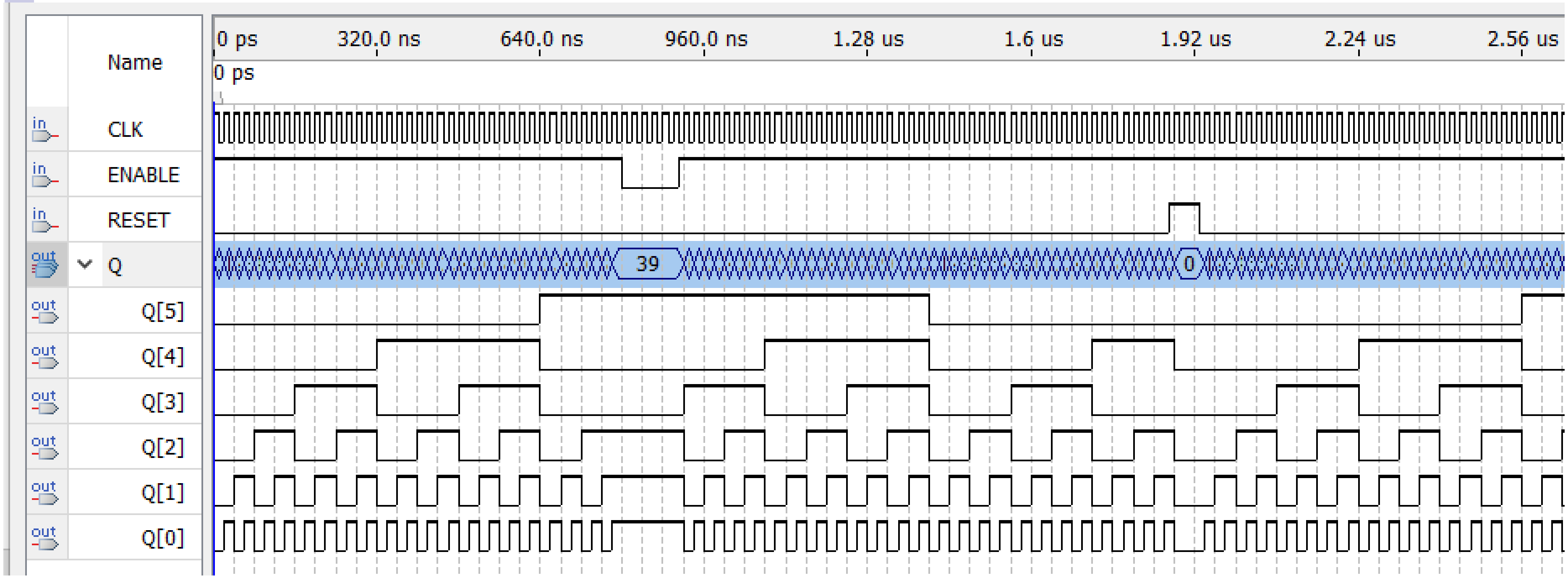
```
cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
```

or

```
cnt <= cntnew when rising_edge (CLK) and (?? ENABLE);
```

For more details see chapter 4.4: (Condition Operator) page 132, in pdf page 143
VHDL-2008_JustTheNewStuff.pdf#page=143

Simulation for WBus=6



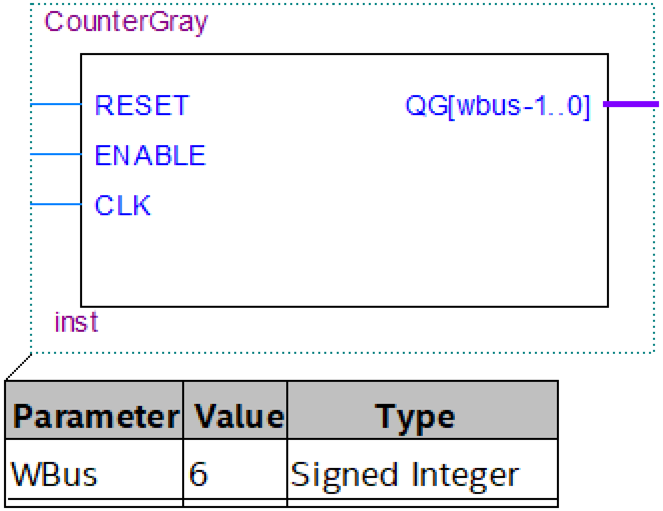
How to create versatile VHDL code

Gray Code Counter

In this simple example,
we demonstrate the possibilities of
VHDL coding styles.

CounterGray

```
library ieee; use ieee.std_logic_1164.all;  use ieee.numeric_std.all;
entity CounterGray is
   generic(WBus:integer:=6 ); --Width (Length) of Gray bus
      port(RESET, ENABLE, CLK : in std_logic:='0';
         QG : out std_logic_vector(WBus-1 downto 0):=(others=>'0'));
      begin assert WBus>0
      report "Excepted the width of bus WBus>0"
      severity ERROR;
end entity;
```



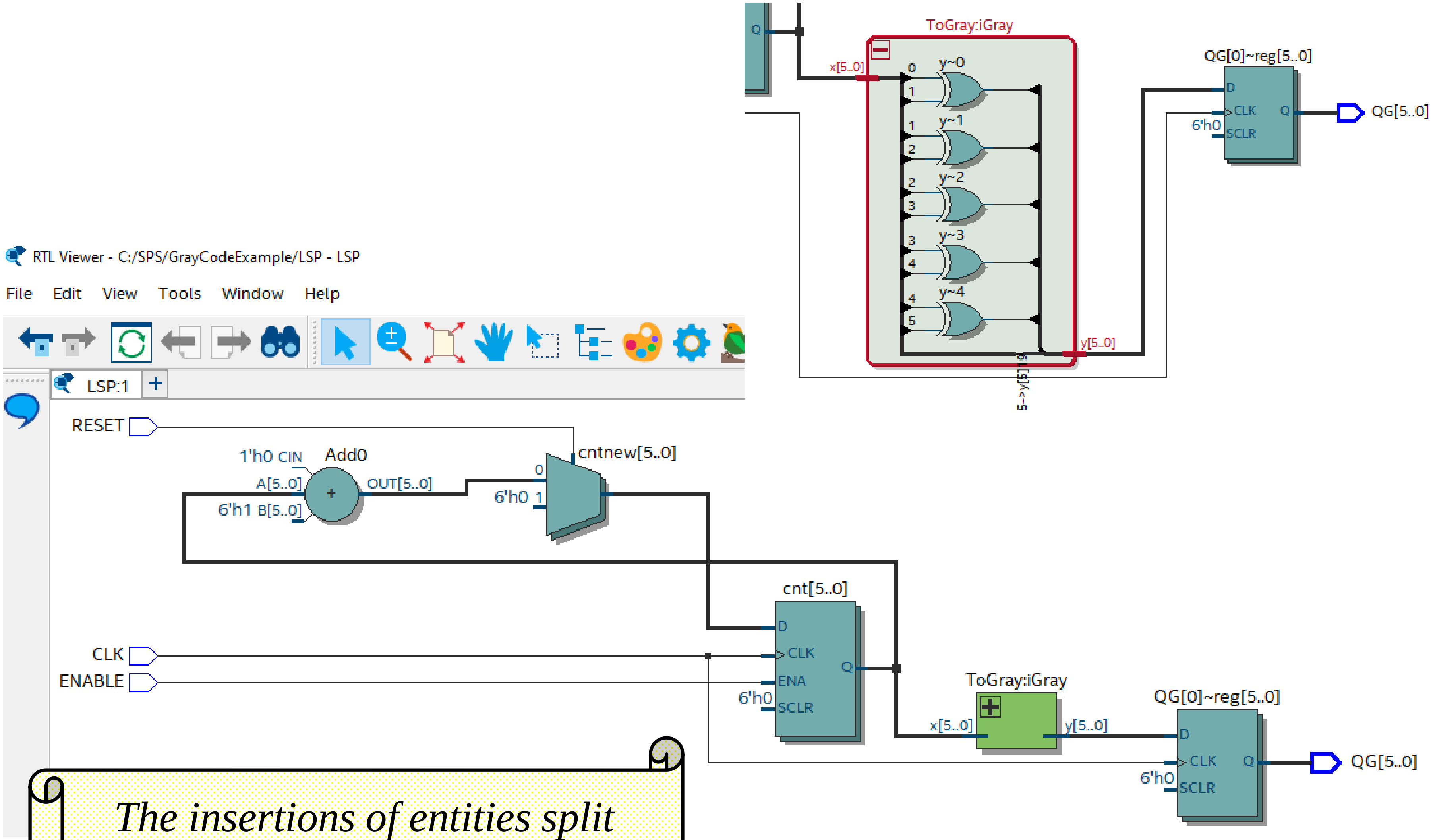
Parameter	Value	Type
WBus	6	Signed Integer

Inserting ToGray by port map

```
architecture portmap  of CounterGray is
signal cnt, cntnew : unsigned(QG'RANGE):=(others=>'0');
signal G : std_logic_vector(QG'RANGE):=(others=>'0');
begin
   cntnew <= (others=>'0') when RESET else cnt + 1;
   cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
 iGray : entity work.ToGray generic map (N=>WBus)
              port map(X=>std_logic_vector(cnt), Y=>G);
    QG<=std_logic_vector(G) when rising_edge(CLK);
end architecture;
```

If the entity ToGray were more complex using it as an instance could be beneficial, but its algorithm is simple!

Independent Blocks



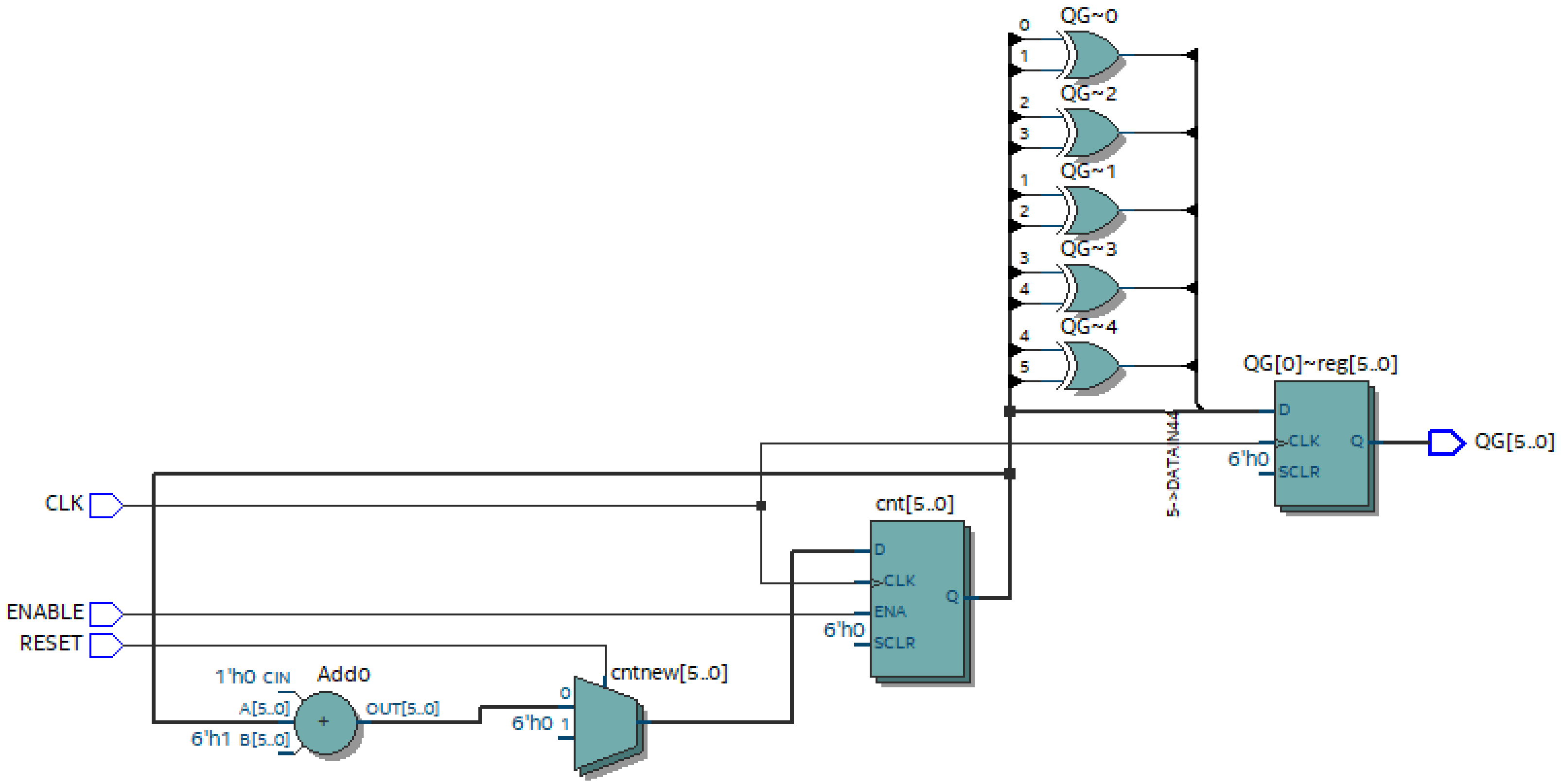
The insertions of entities split circuits into more manageable independent blocks.

Algorithm of ToGray by Copy-Paste

```
architecture copypaste of CounterGray is
signal cnt, cntnew : unsigned(QG'RANGE):=(others=>'0');
signal x : std_logic_vector(QG'RANGE):=(others=>'0');
begin
   cntnew <= (others=>'0') when RESET else cnt + 1;
   cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
   x<=std_logic_vector(cnt);

   QG<= X xor ('0' & X(X'HIGH downto 1)) when rising_edge(CLK);

end architecture;
```

The straightforward application of the popular copy-paste method is quick but not too versatile.

The result Copy-Paste



Algorithm of ToGray by Function

```
architecture func of CounterGray is


function bin2gray(X:std_logic_vector) return std_logic_vector is
begin
    return X xor ('0' & X(X'HIGH downto 1));
end function;


begin -- architecture
```

Algorithm of ToGray by Function

```
architecture func of CounterGray is
   signal cnt, cntnew : unsigned(QG'RANGE):=(others=>'0');
   function bin2gray(X:std_logic_vector) return std_logic_vector is
   begin
      return X xor ('0' & X(X'HIGH downto 1));
   end function;
begin
    cntnew <= (others=>'0') when RESET else cnt + 1;
    cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
    QG<= bin2gray(std_logic_vector(cnt))
                when rising_edge(CLK);
end architecture;
```

We have laboriously increased versatility but for one VHDL file only :-(

Moving bin2gray function into package

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

use ieee.numeric_std.all;

package MyFunctions is  -- headers

  function bin2gray(X:std_logic_vector) return std_logic_vector;

end package;
```

Moving bin2gray function into package

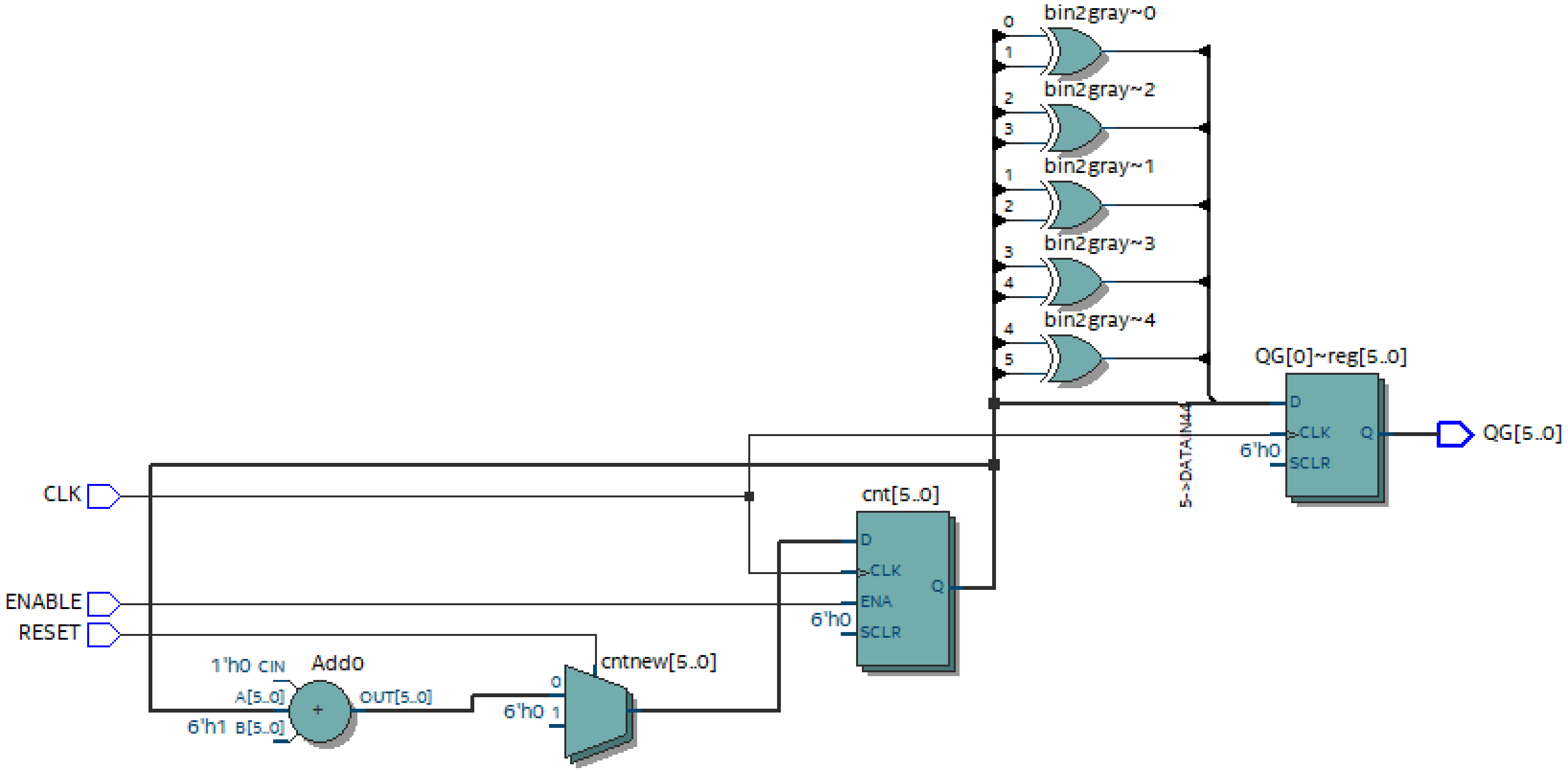
```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package MyFunctions is  -- headers
 function bin2gray(X:std_logic_vector) return std_logic_vector;
end package;
package body MyFunctions is -- full codes
  function bin2gray(X:std_logic_vector) return std_logic_vector is
  begin
    return X xor ('0' & X(X'HIGH downto 1));
  end function;
end package body;
```

The simplest code

```
architecture mypack of CounterGray is
signal cnt, cntnew : unsigned(QG'RANGE):=(others=>'0');
use work.MyFunctions.all;
begin
   cntnew <= (others=>'0') when RESET else cnt + 1;
   cnt <= cntnew when rising_edge(CLK) and ENABLE='1';
   QG<= bin2gray( std_logic_vector(cnt) )
           when rising_edge(CLK);

end architecture;
```

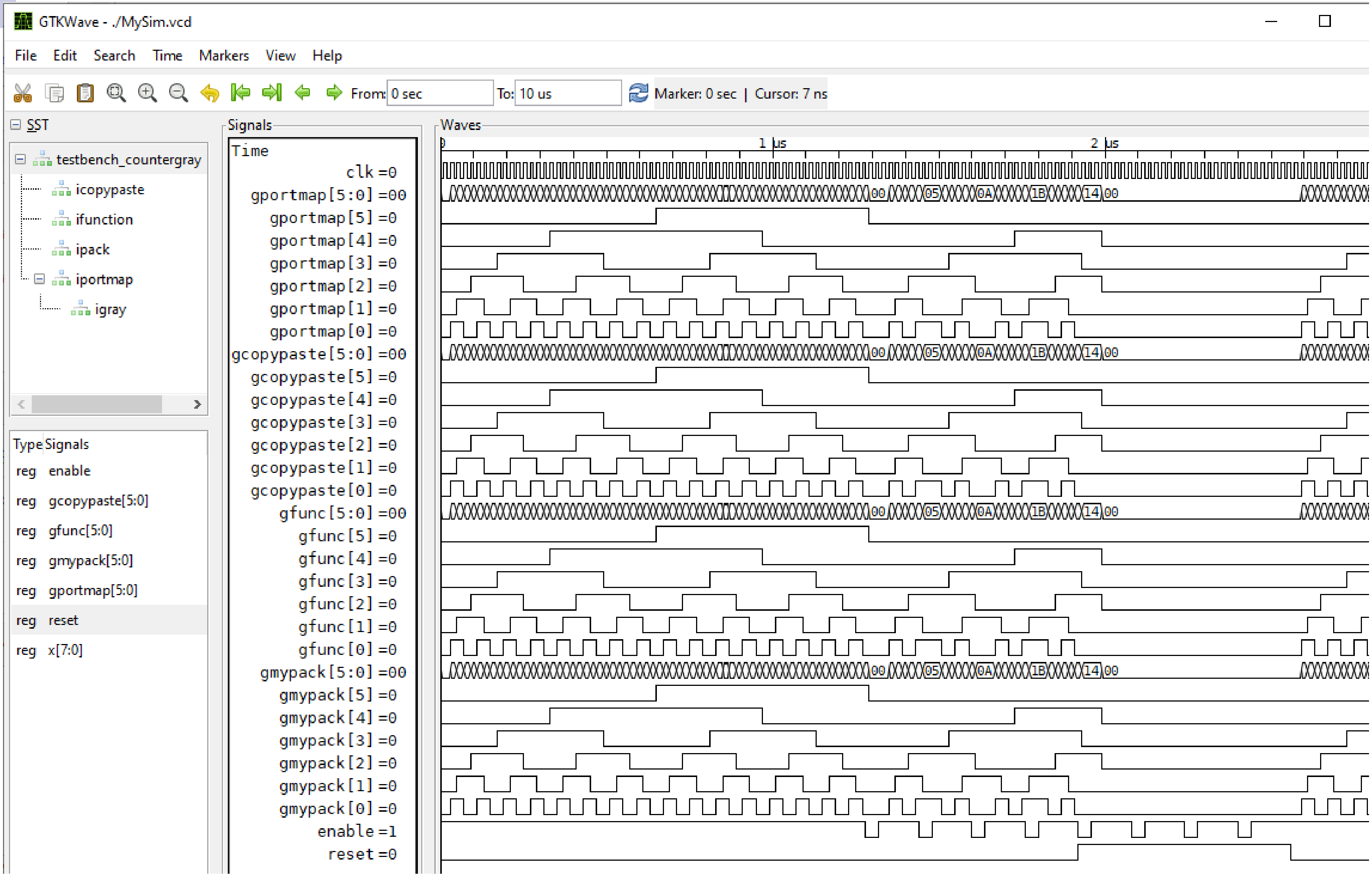
Now, our code is not only the simplest one but also the most versatile.

Functions are improved copy-paste



VHDL functions and procedures are **never** called !!!
They are always compiled by inserting their code, i.e., by inline expansions.

Simulation



- testbench_countergray
  - icopypaste
  - ifunction
  - ipack
  - iportmap
    - igray

Type	Signals
reg	enable
reg	gcopypaste[5:0]
reg	gfunc[5:0]
reg	gmypack[5:0]
reg	gportmap[5:0]
reg	reset
reg	x[7:0]

```
Time
clk =0
gportmap[5:0] =00
gportmap[5] =0
gportmap[4] =0
gportmap[3] =0
gportmap[2] =0
gportmap[1] =0
gportmap[0] =0
gcopypaste[5:0] =00
gcopypaste[5] =0
gcopypaste[4] =0
gcopypaste[3] =0
gcopypaste[2] =0
gcopypaste[1] =0
gcopypaste[0] =0
gfunc[5:0] =00
gfunc[5] =0
gfunc[4] =0
gfunc[3] =0
gfunc[2] =0
gfunc[1] =0
gfunc[0] =0
gmypack[5:0] =00
gmypack[5] =0
gmypack[4] =0
gmypack[3] =0
gmypack[2] =0
gmypack[1] =0
gmypack[0] =0
enable =1
reset =0
```