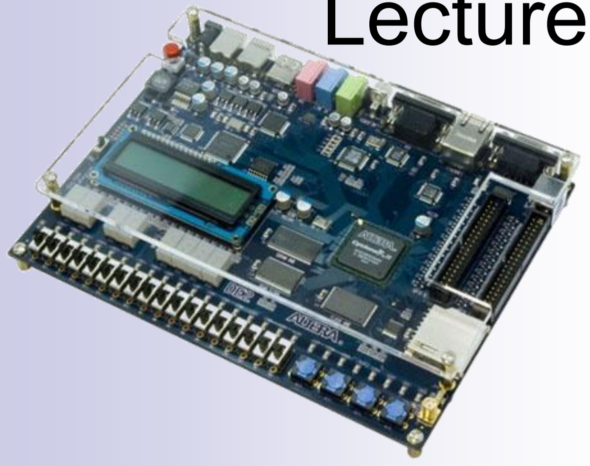
Logic Systems and Processors
cz:Logické systémy a procesory

Lecturers: Martin Hlinovský+Richard Šusta



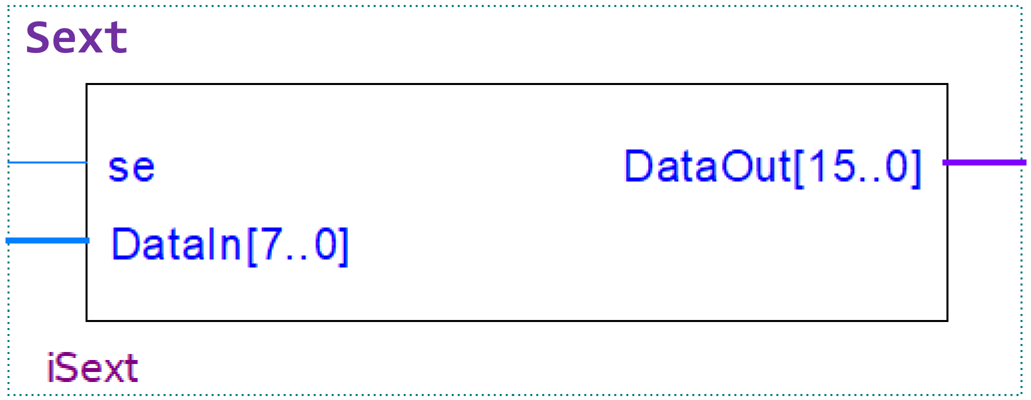
Version V1.1

CTU-FEE in Prague, CR – subject BE5B35LSP

Warm-up Example

The example focuses on the coding styles
that are suitable for describing circuits.

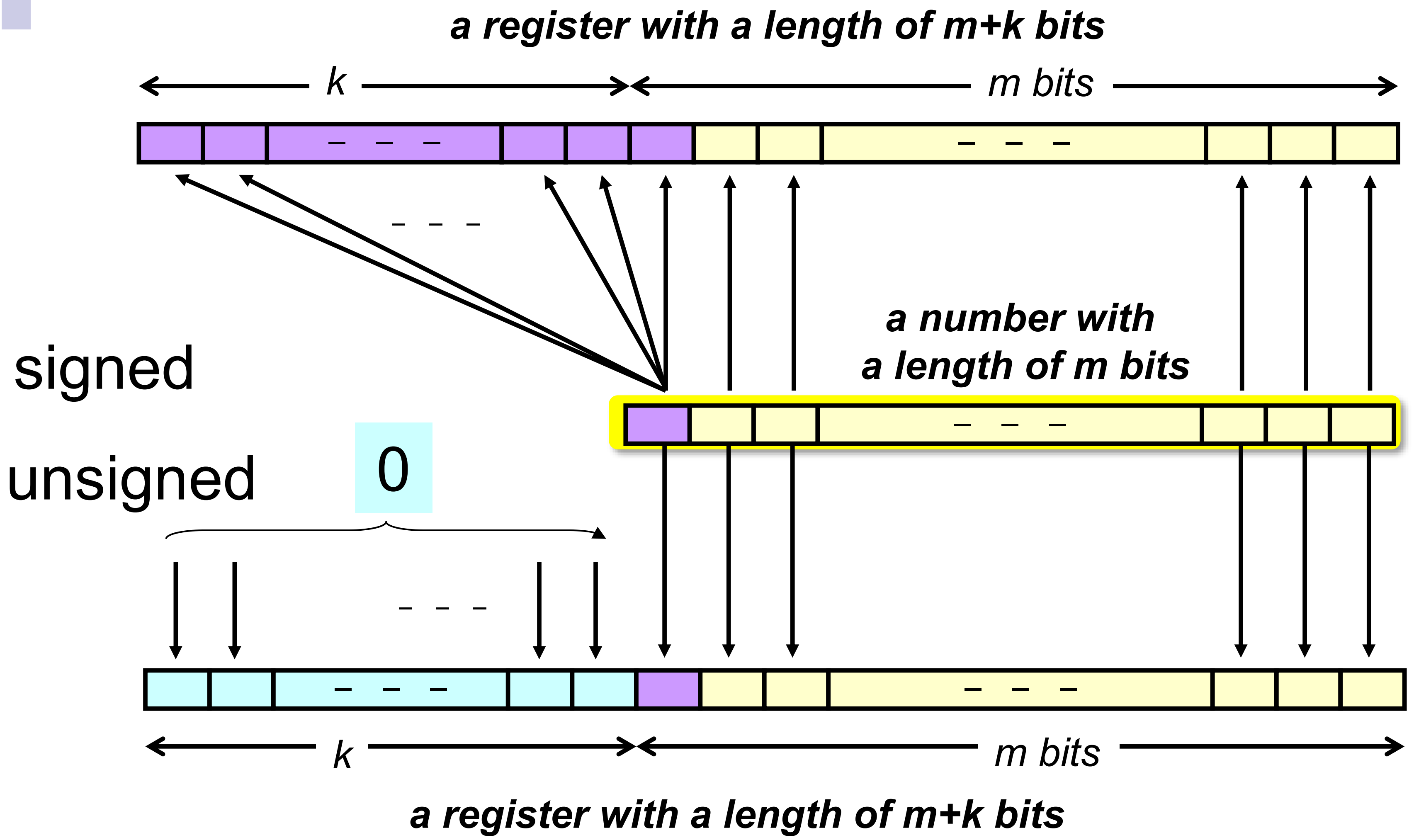
Image: http://dr-wo.de/



Create **Sext** = Sign Extension Circuit

*It has the established abbreviation **sext***

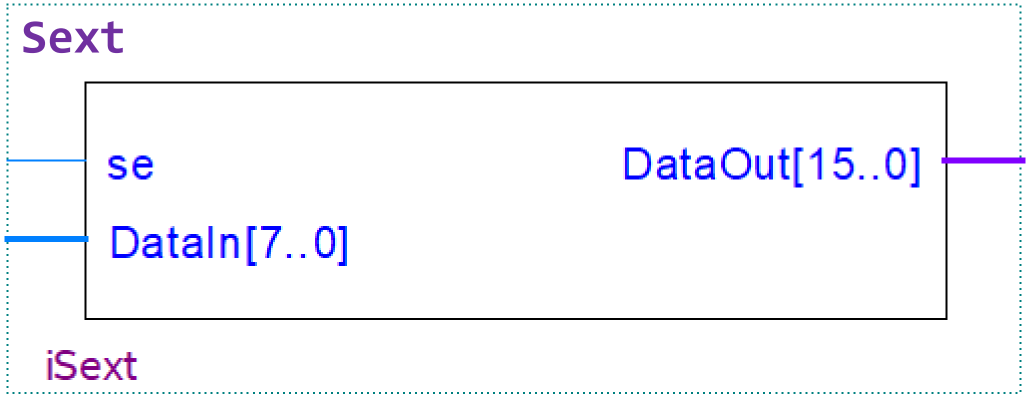
Unsigned / Signed Extensions



a register with a length of m+k bits
k
m bits
– – –
signed
unsigned
0
a number with a length of m bits
k
m bits
a register with a length of m+k bits

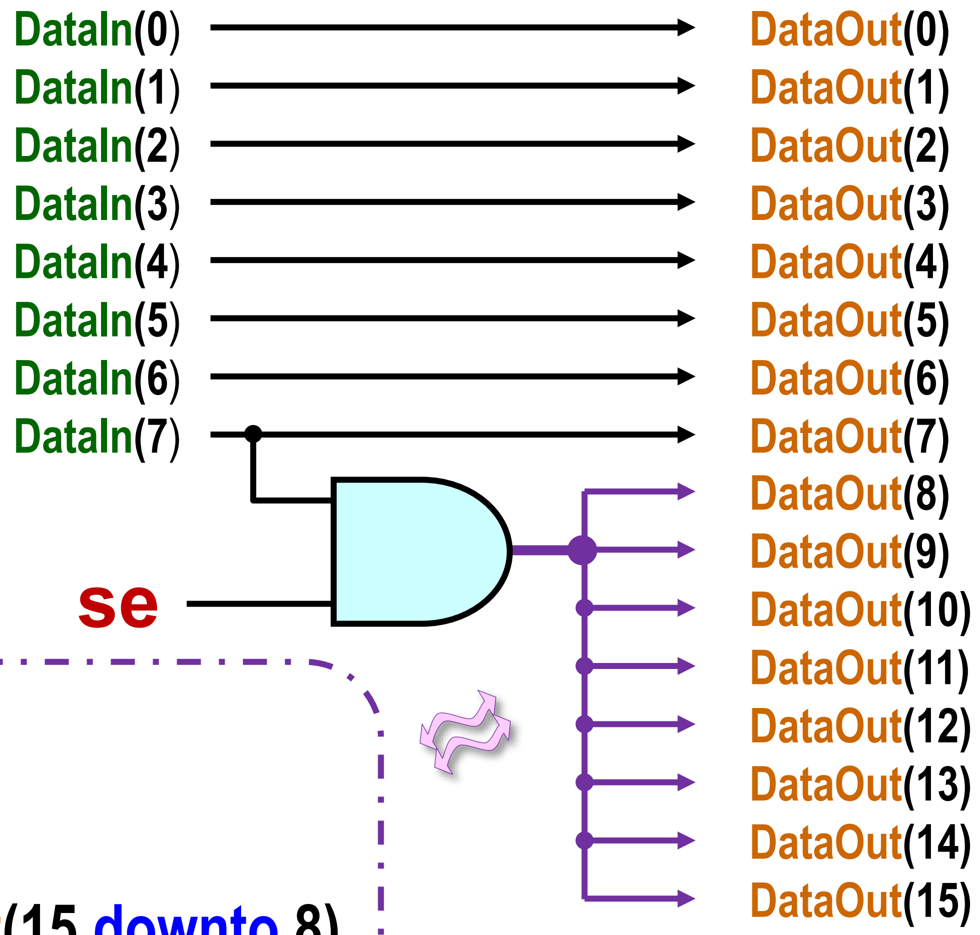
VHDL Entity - Step 1

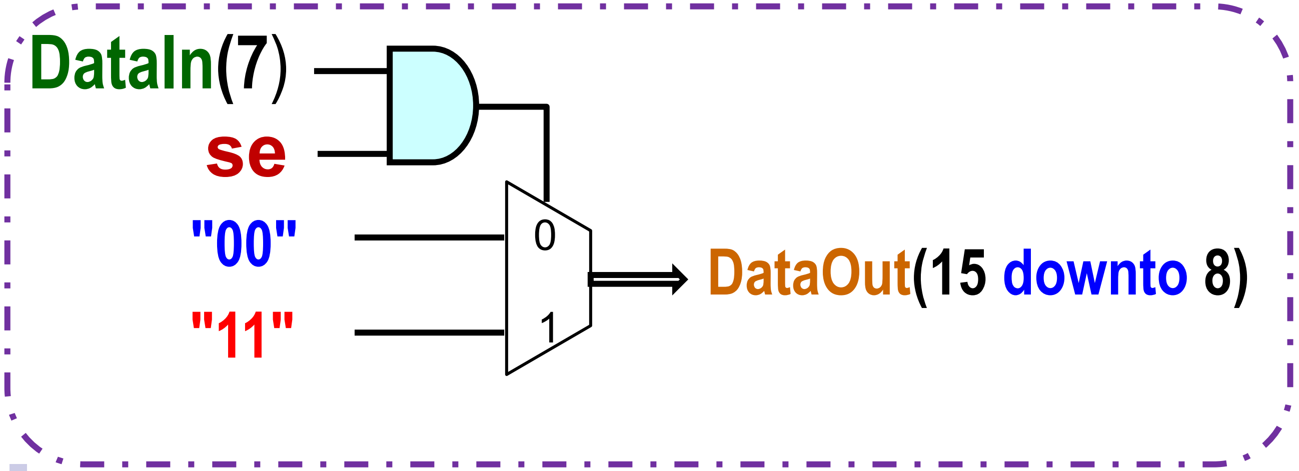
```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Sext is
port( se : std_logic; -- sign extension is on
        DataIn : in std_logic_vector(7 downto 0);
        DataOut : out std_logic_vector(15 downto 0));
end entity;
```



VHDL Architecture - Step 2

The VHDL describes circuits, so we will start with the sext's schema:



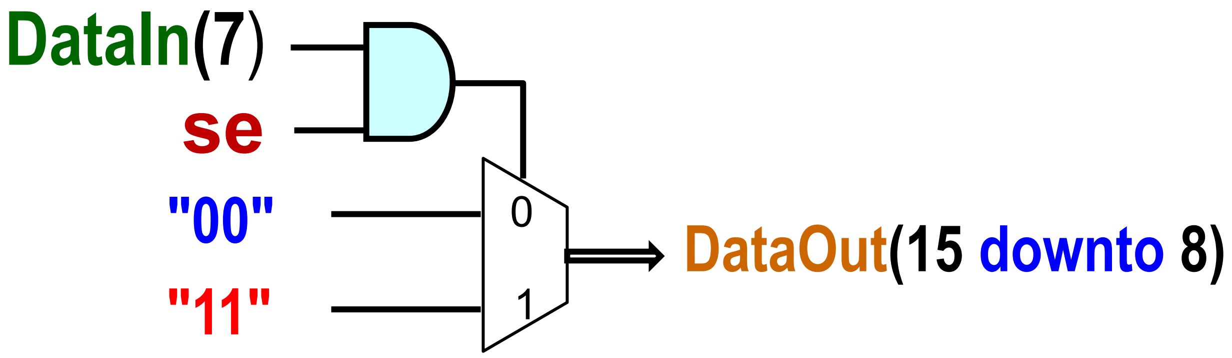


Step 2 explanation

It is certainly possible to assign 0 or 1 to DataOut from 8 to 15 according to the value of **(DataIn(7) and se)** condition, e.g., either

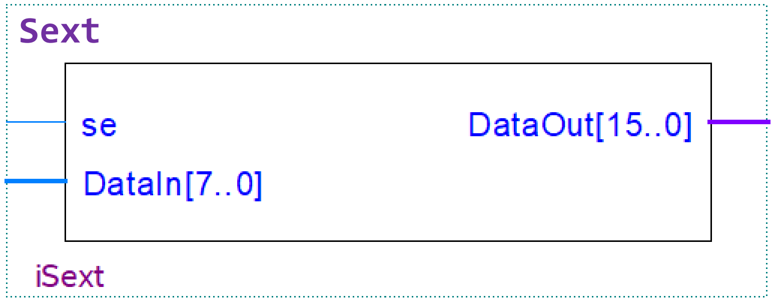
- ❑ using eight separate assignments, or
- one VHDL for-generate loop statement,

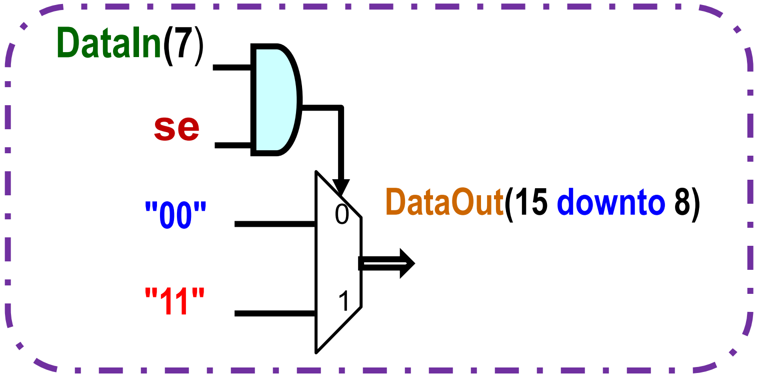
However, the result would be the same as for the multiplexer below, which can be described with a shorter VHDL code.



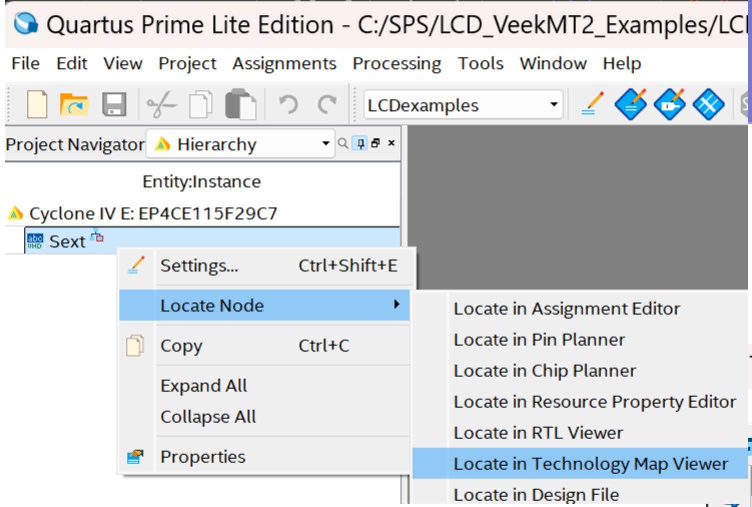
VHDL ← Circuit

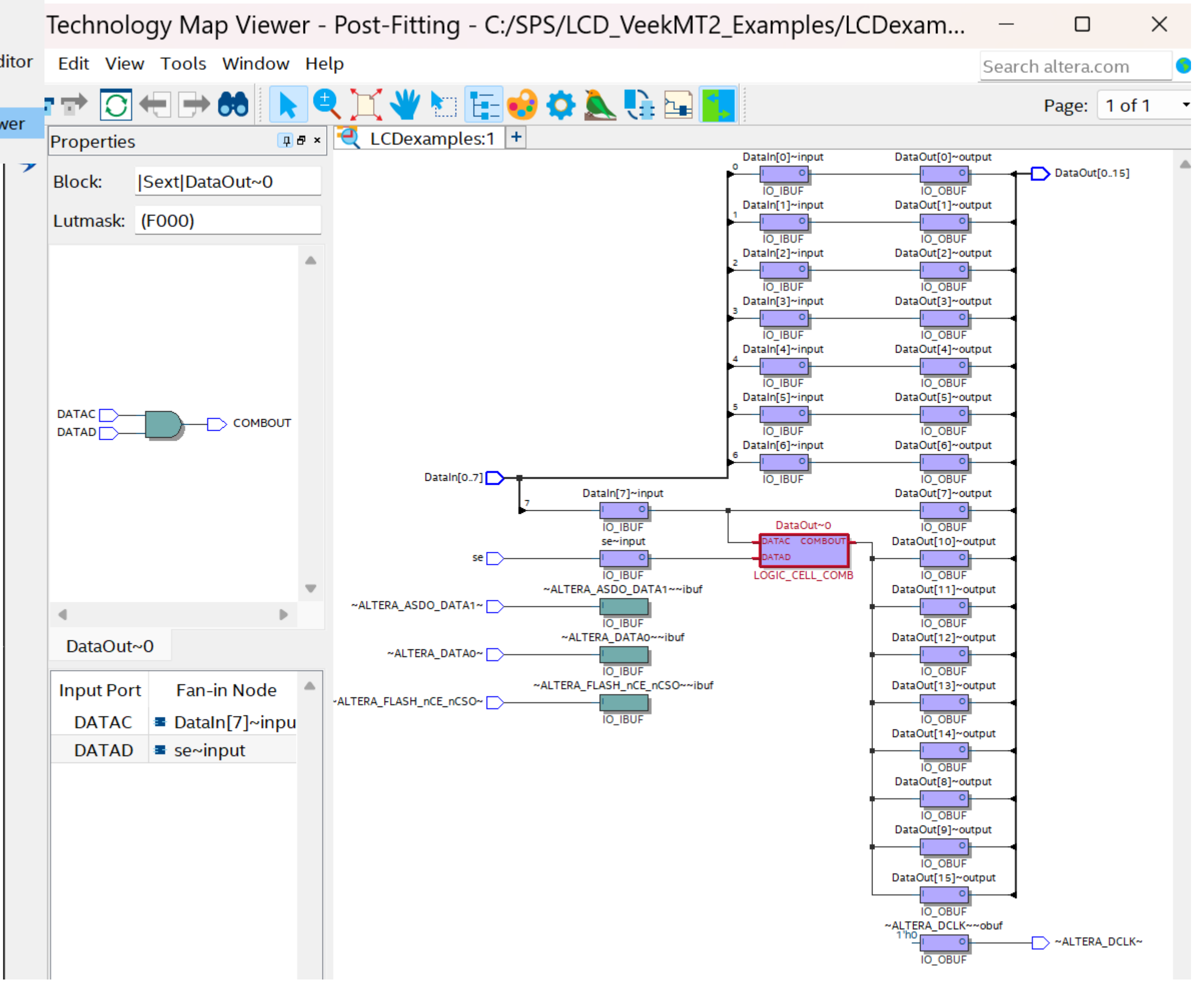
```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Sext is
port(   se : std_logic;
        DataIn : in std_logic_vector(7 downto 0);
        DataOut : out std_logic_vector(15 downto 0));
end entity;


architecture dataflow of Sext is
begin
  DataOut(7 downto 0) <= DataIn;
  DataOut(15 downto 8) <= X"FF" when DataIn(7) and se else X"00";
-- all by 1 line:  DataOut <= X"FF" & DataIn when DataIn(7) and se else X"00" & DataIn;
end architecture;
-- the code is for VHDL2008 (Quartus: Assignments->Settings->Compiler Settings->VHDL input)
```





Technology Map Viewer


Quartus Prime Lite Edition - C:/SPS/LCD_VeekMT2_Examples/LC
File Edit View Project Assignments Processing Tools Window Help
LCDexamples
Project Navigator
Hierarchy
Entity:Instance
Cyclone IV E: EP4CE115F29C7
Sext
Settings... Ctrl+Shift+E
Locate Node
Copy Ctrl+C
Expand All
Collapse All
Properties
Locate in Assignment Editor
Locate in Pin Planner
Locate in Chip Planner
Locate in Resource Property Editor
Locate in RTL Viewer
Locate in Technology Map Viewer
Locate in Design File



Block: |Sext|DataOut~0

Lutmask: (F000)

DataOut~0

Input Port	Fan-in Node
DATAC	DataIn[7]~inpu
DATAD	se~input

LCD - Liquid Crystal Display

precisely TN LCD (Twist Nematic Liquid Crystal Display)

LSP course also describes the principles of modern digital systems, so we briefly look inside the topic of the LCD background task

Image: Terasic

Liquid Crystal Display principle

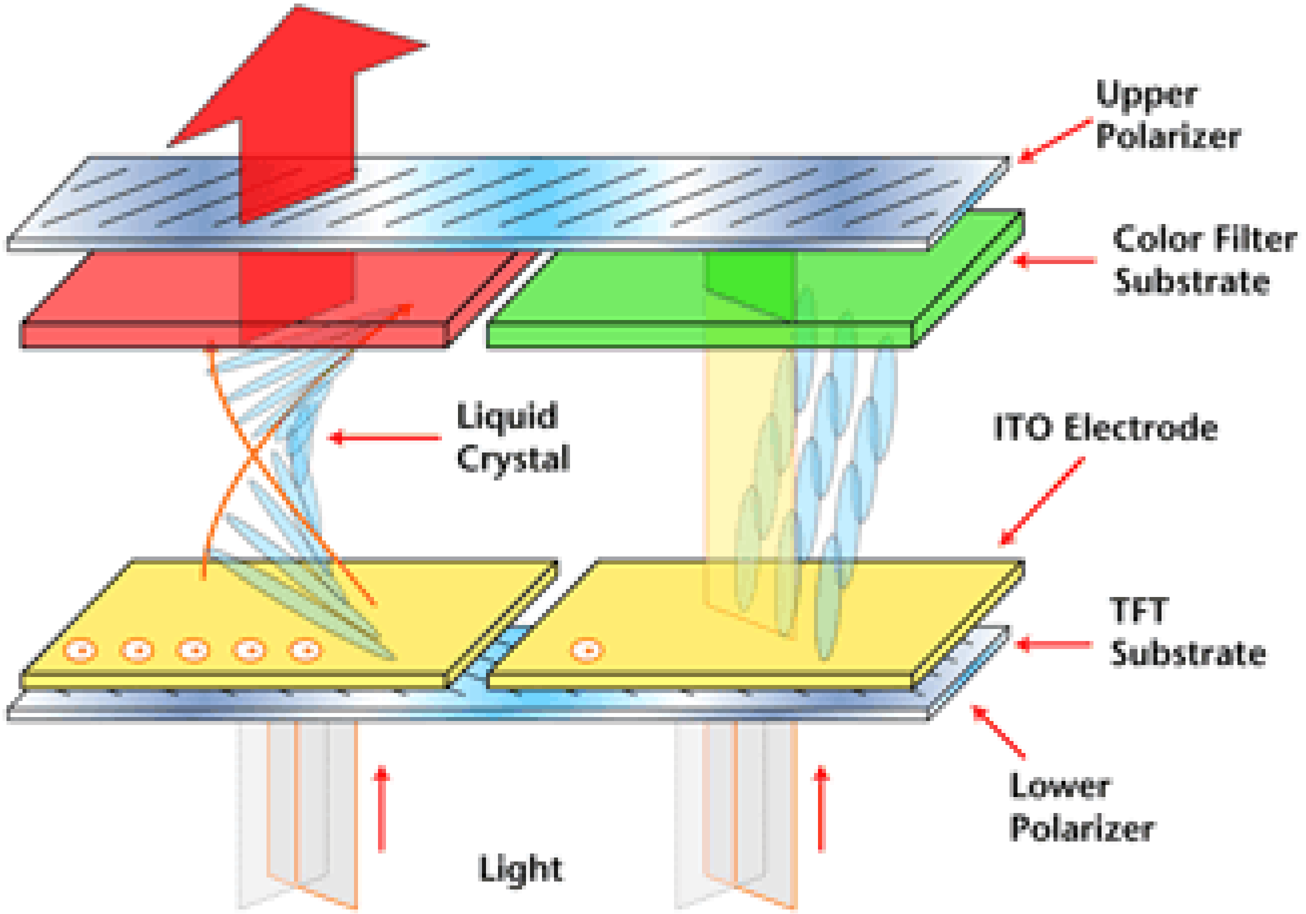


Image source: Matrix

<u>Nowadays, 3 arrangements are used:</u>

TFT TN – [*twisted LC -> vertical*] cheaper and with faster responses

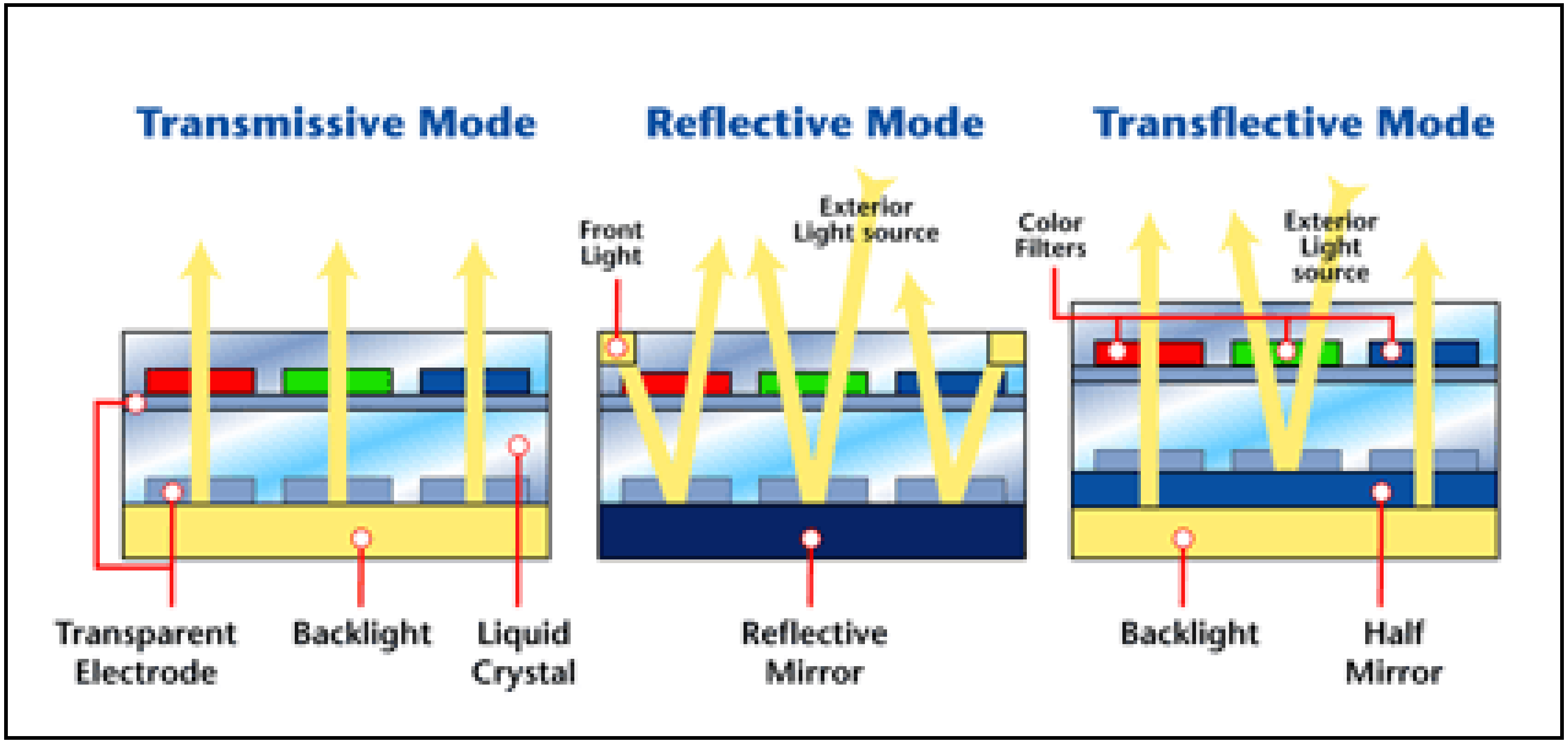
TFT **VA** (Vertical Alignment) – [*vertical LC -> twisted*] the best colors and contrast, good viewing angle

<u>slow response</u>

TFT **IPS** (*In-Plane-Switching electrodes*) – high price, better colors, wider viewing angle, usually slower.

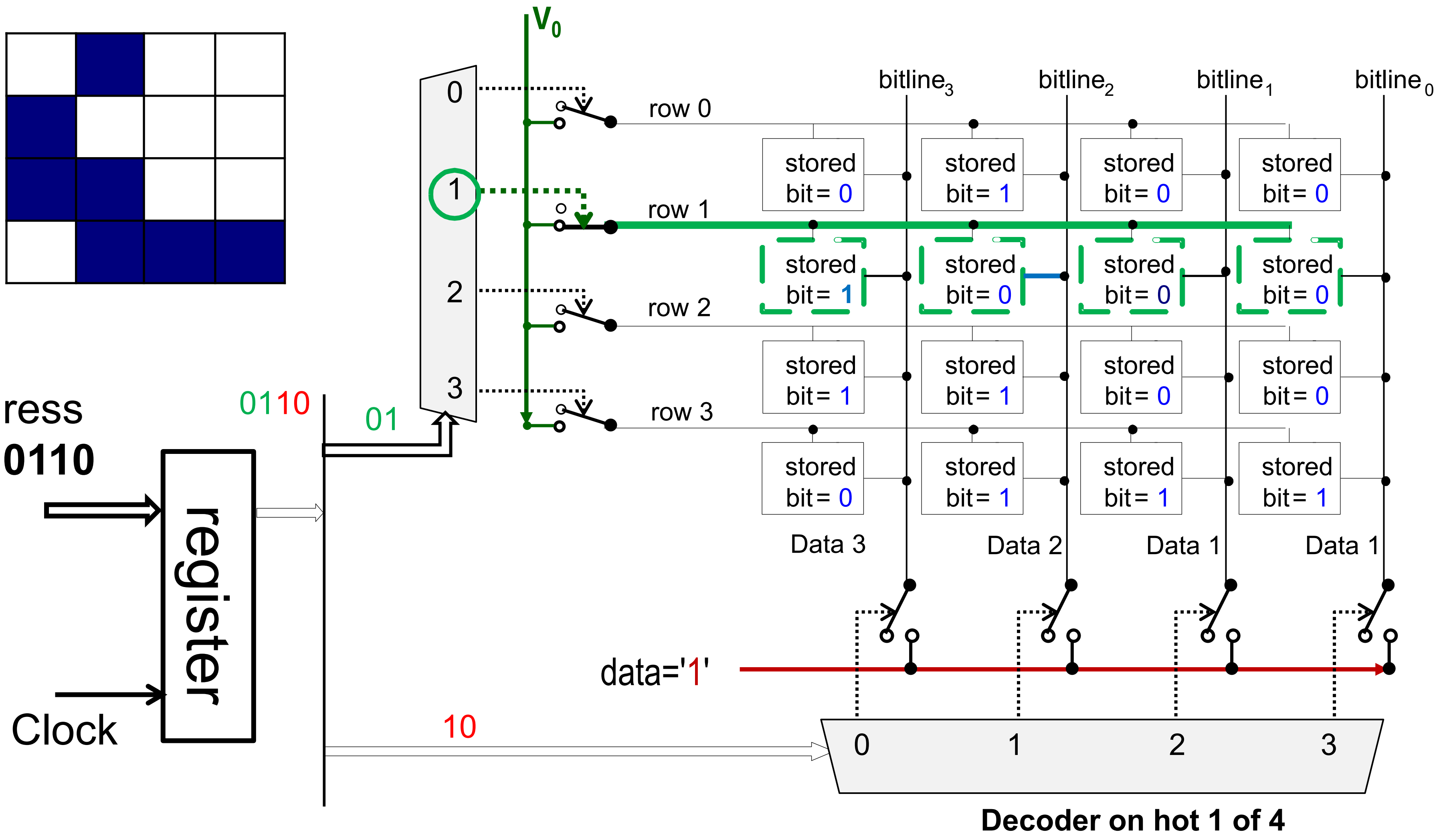
Approximately 15% higher energy consumption compared to TFT TN

LC LCD Types

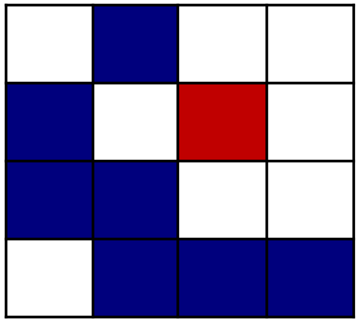


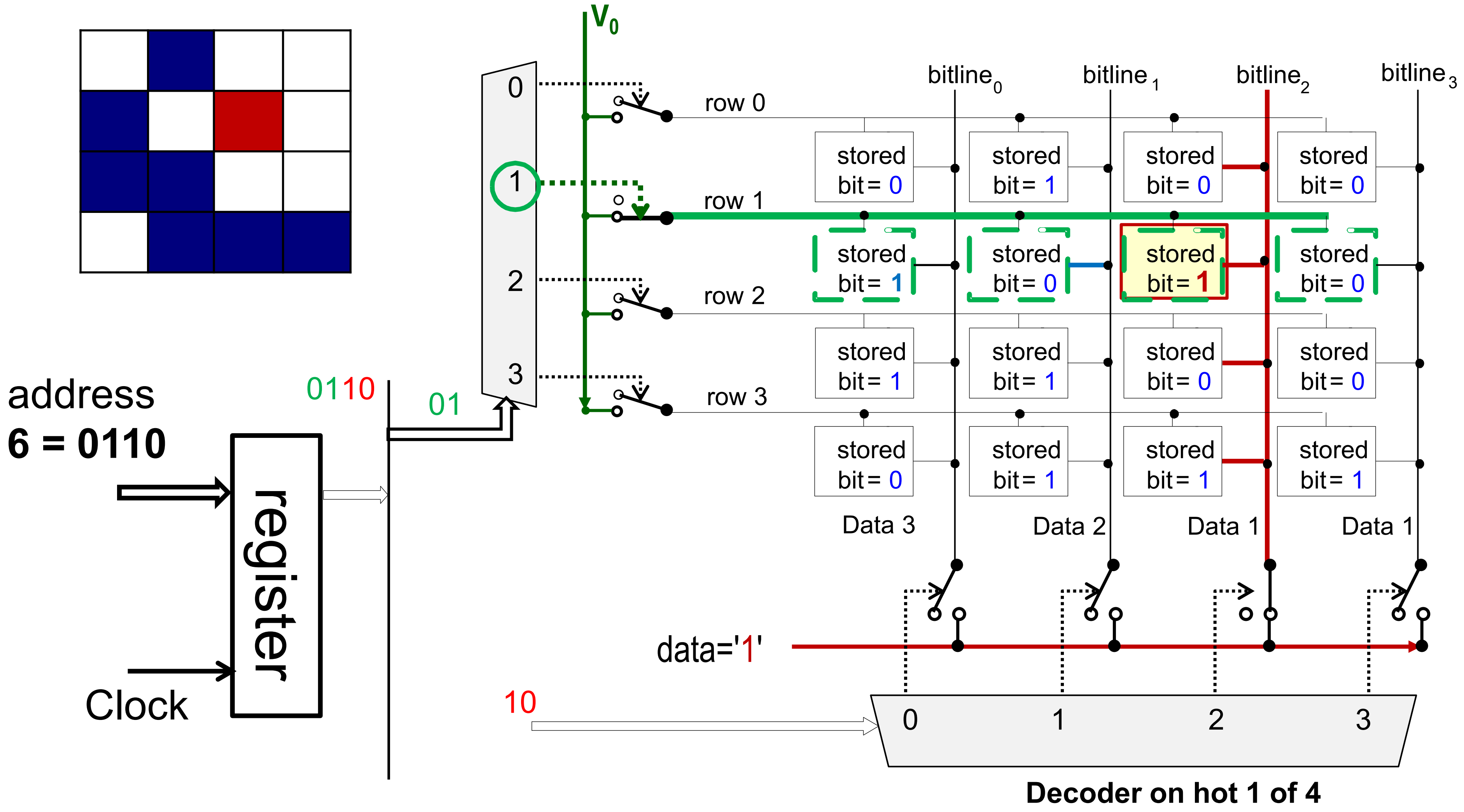
Source: Matrix

Address Matrix

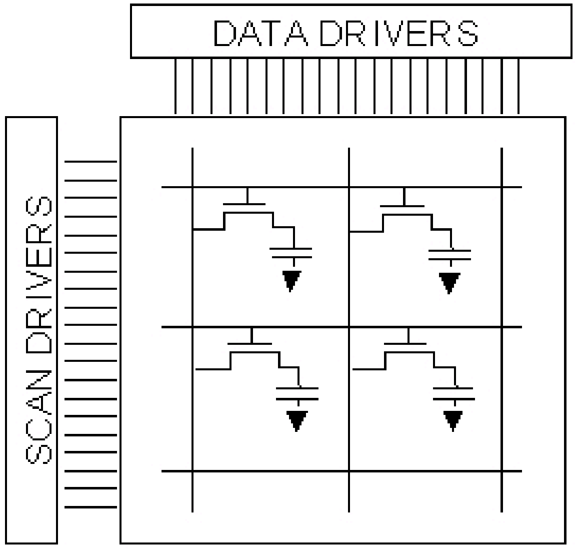


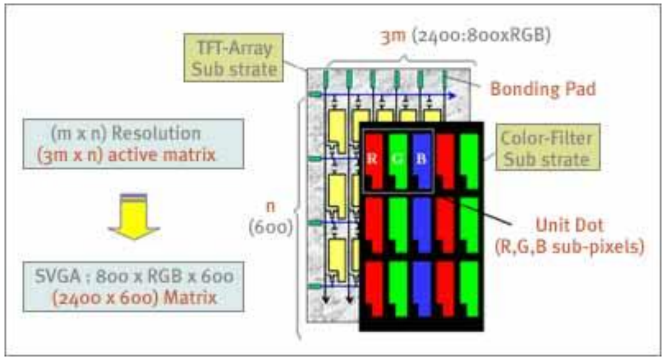
Address Matrix

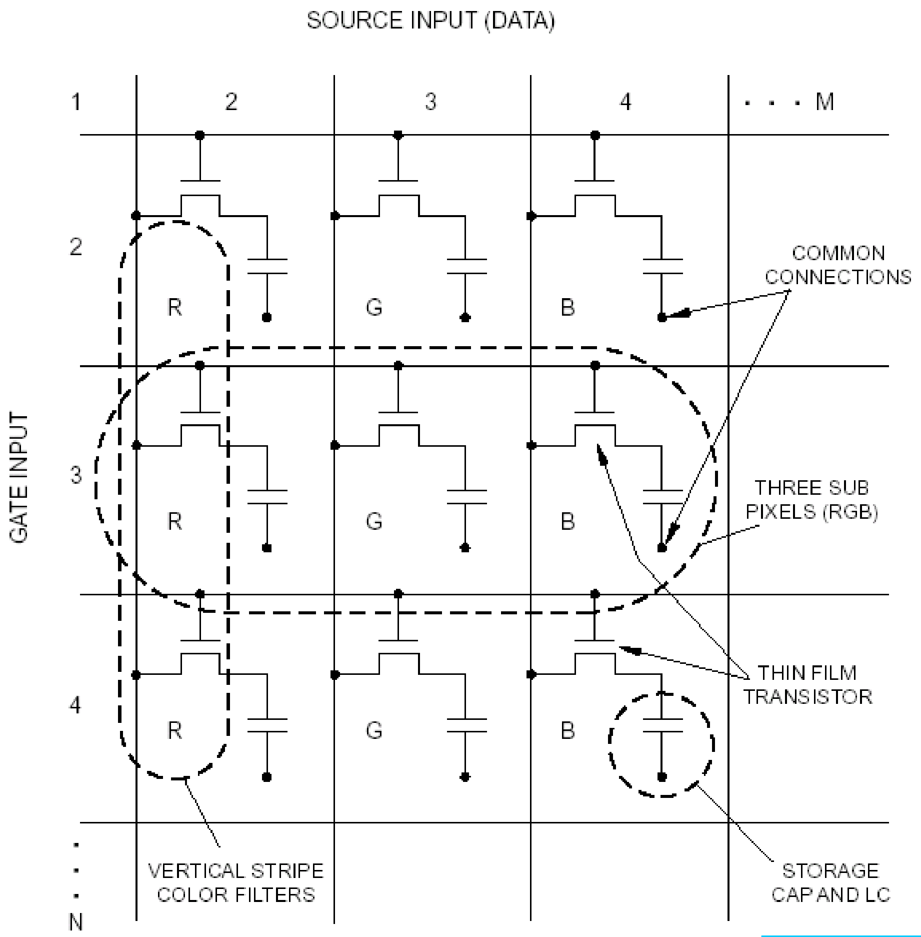




TFT (Thin-Film Transistors) Active Matrix



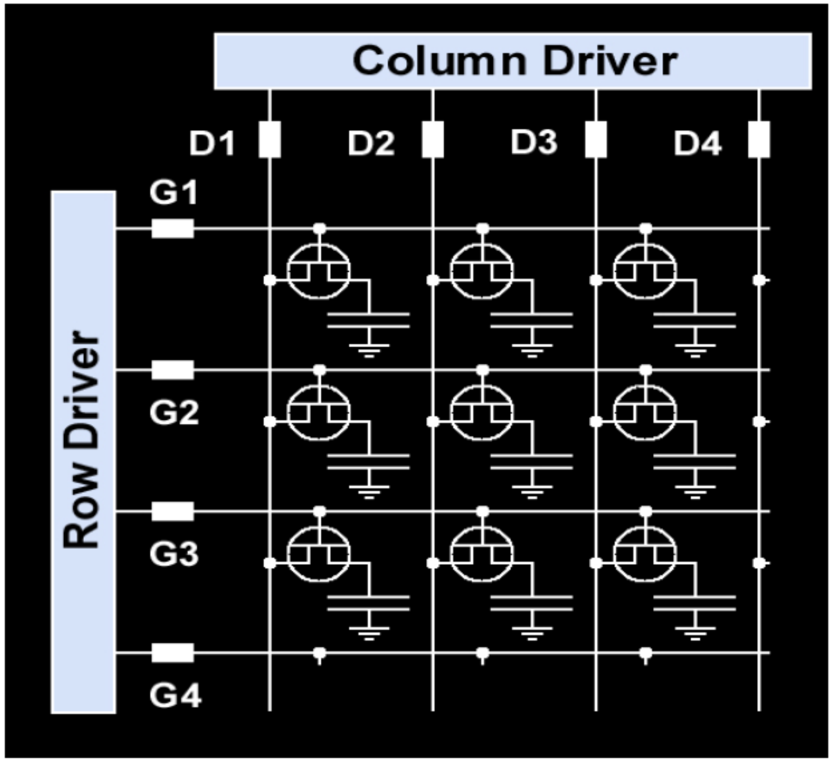

TFT-Array Sub strate
3m (2400:800xRGB)
Bonding Pad
(m x n) Resolution (3m x n) active matrix
Color-Filter Sub strate
n (600)
R G B
Unit Dot (R,G,B sub-pixels)
SVGA : 800 x RGB x 600 (2400 x 600) Matrix



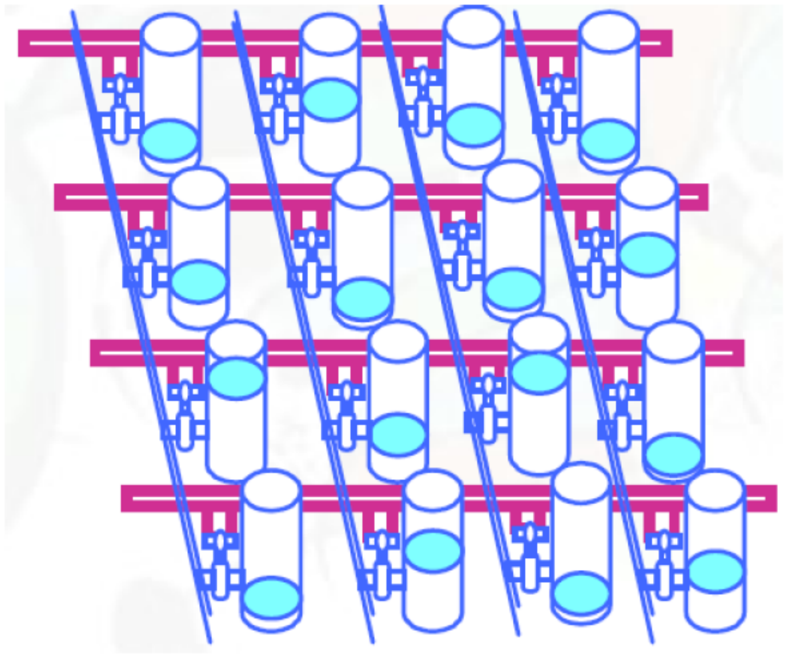
Source of images: Matrix

LCD capacitors store multiple levels!

TFT (switch) + LC cell (capacity)



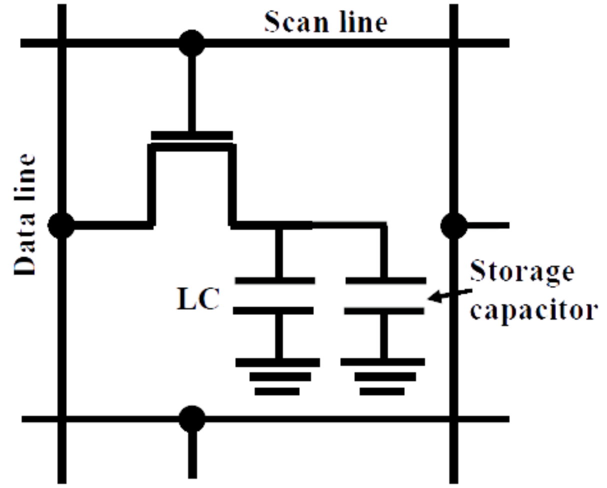
Hydrant (switch)+ Bucket (capacity)

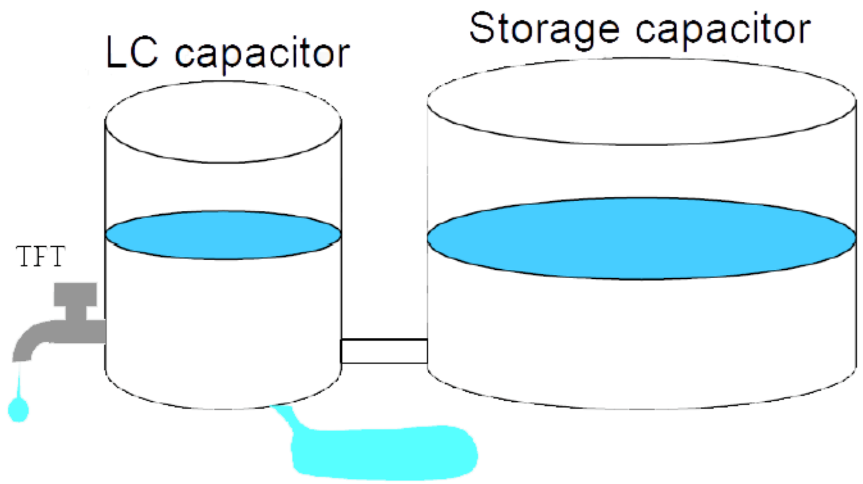


Source: Dr. Zhibing Ge, College of Optics and Photonics

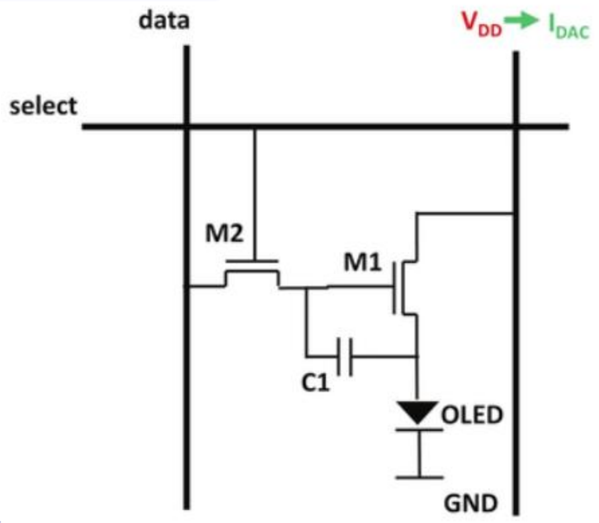
LC LCD versus AMOLED - Refreshing

LCD with Liquid Crystals





AMOLED

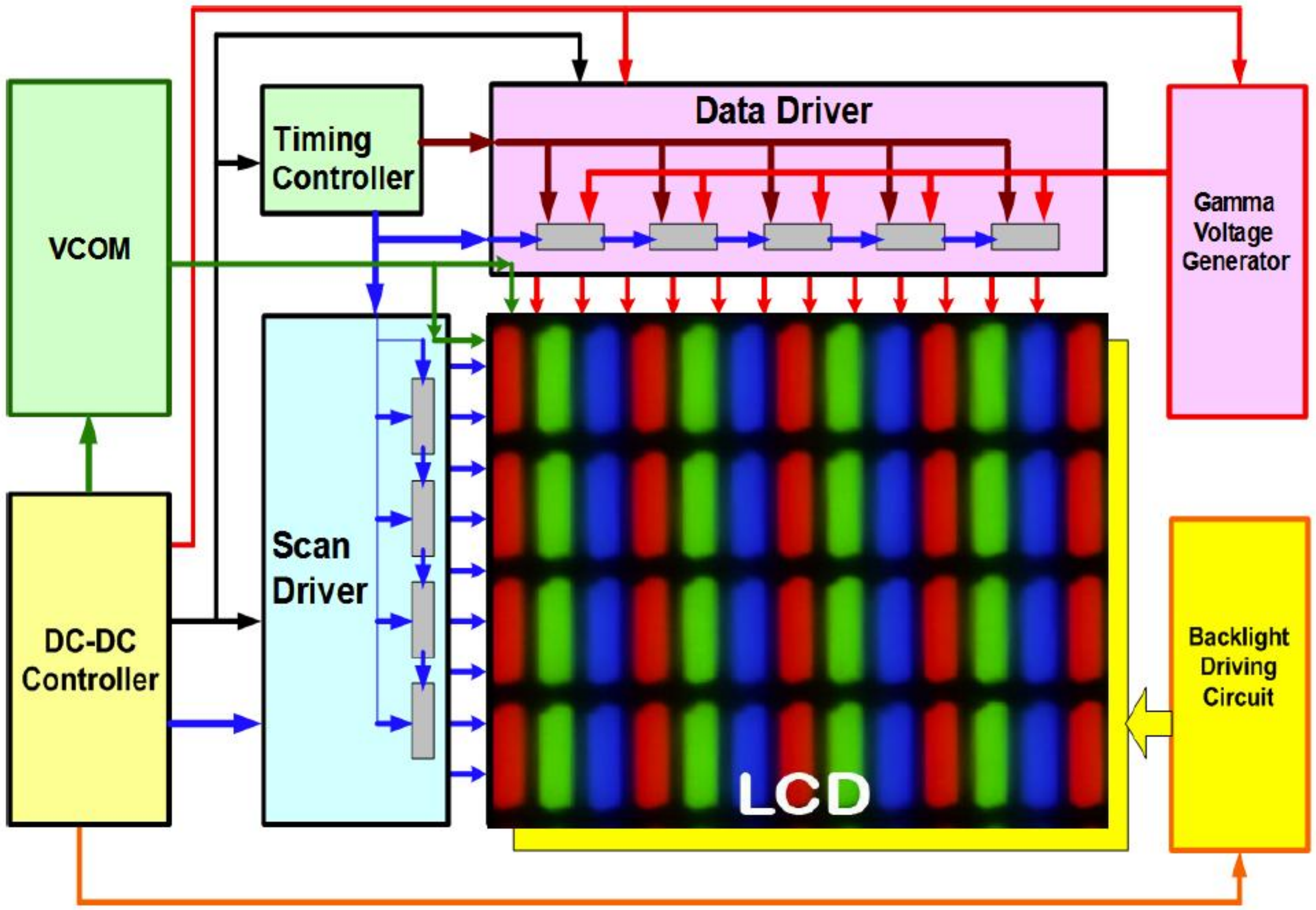


The capacitors in LC LCDs have a typical value range from 100 fF (femtoFarads) to 2000 fF = from 0.1 pF to 2 pF, and need **regular refreshes** as DRAMs.

Nowadays, 2 OLED LCD types are used:

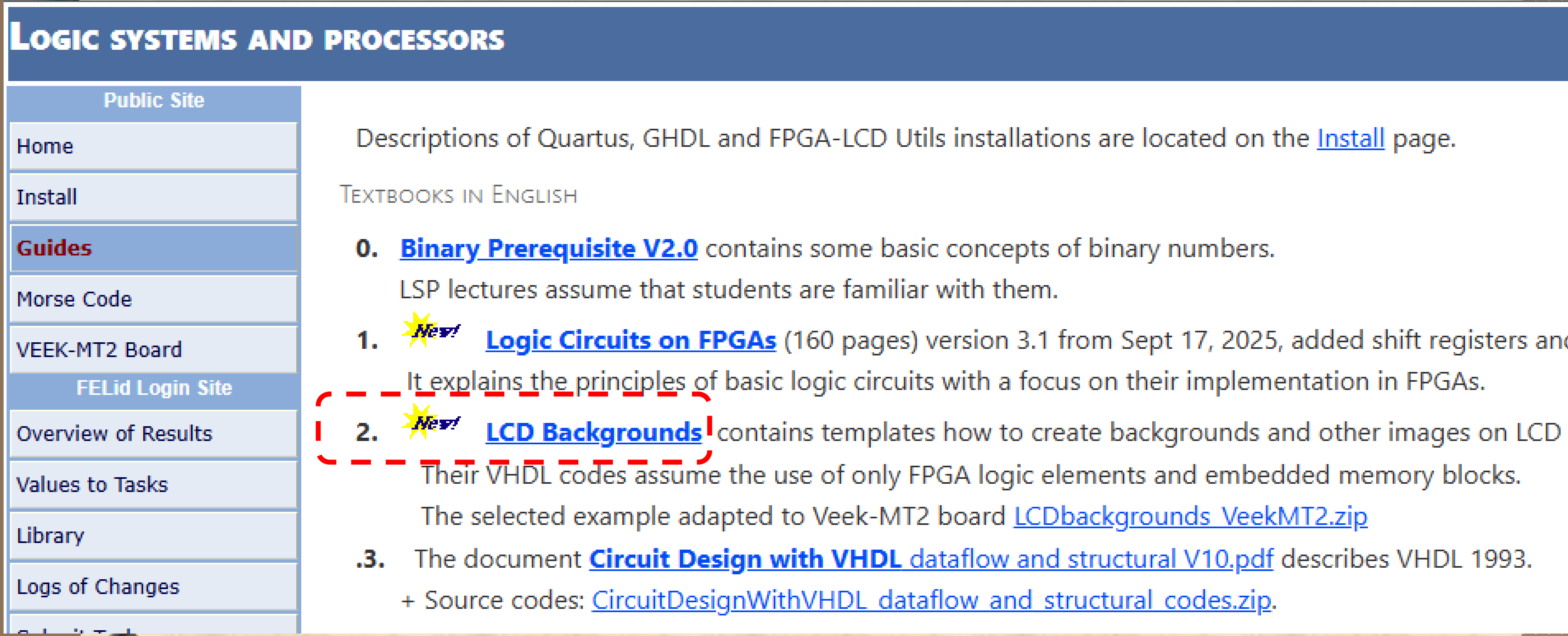
- **AMOLED** = Active Matrix OLED
- **OLED** with passive matrix – cheap LCD panels or very large LCDs, e.g, low-cost TVs. Slower responses than AMOLED.

LCDs more often write in whole lines



Source: Dr. Zhibing Ge, College of Optics and Photonics

LCD Synchronization Generator



LOGIC SYSTEMS AND PROCESSORS

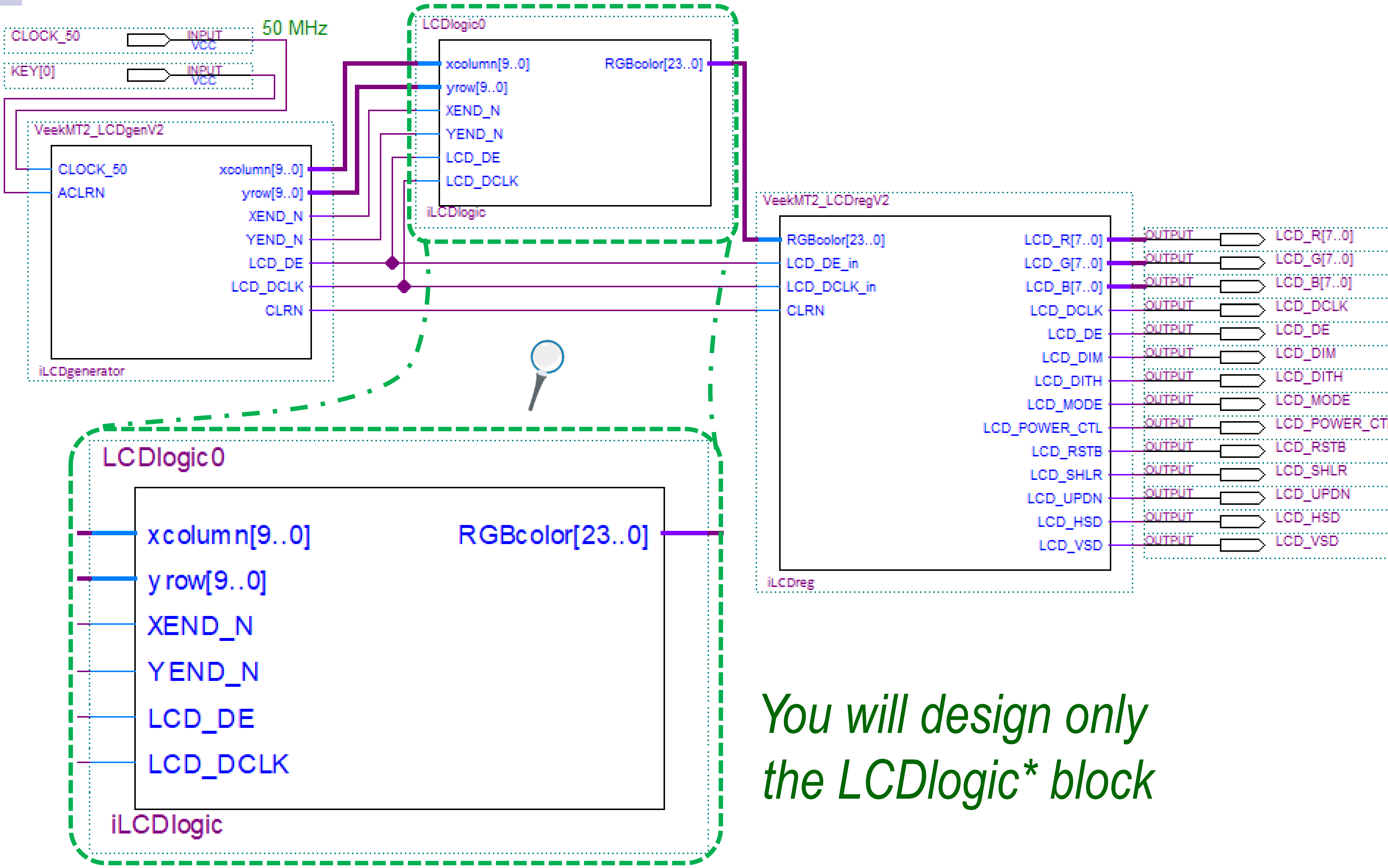
Descriptions of Quartus, GHDL and FPGA-LCD Utils installations are located on the Install page.

TEXTBOOKS IN ENGLISH

0. **Binary Prerequisite V2.0** contains some basic concepts of binary numbers.
   LSP lectures assume that students are familiar with them.
1. New! **Logic Circuits on FPGAs** (160 pages) version 3.1 from Sept 17, 2025, added shift registers an
   It explains the principles of basic logic circuits with a focus on their implementation in FPGAs.
2. New! **LCD Backgrounds** contains templates how to create backgrounds and other images on LCD
   Their VHDL codes assume the use of only FPGA logic elements and embedded memory blocks.
   The selected example adapted to Veek-MT2 board LCDbackgrounds_VeekMT2.zip

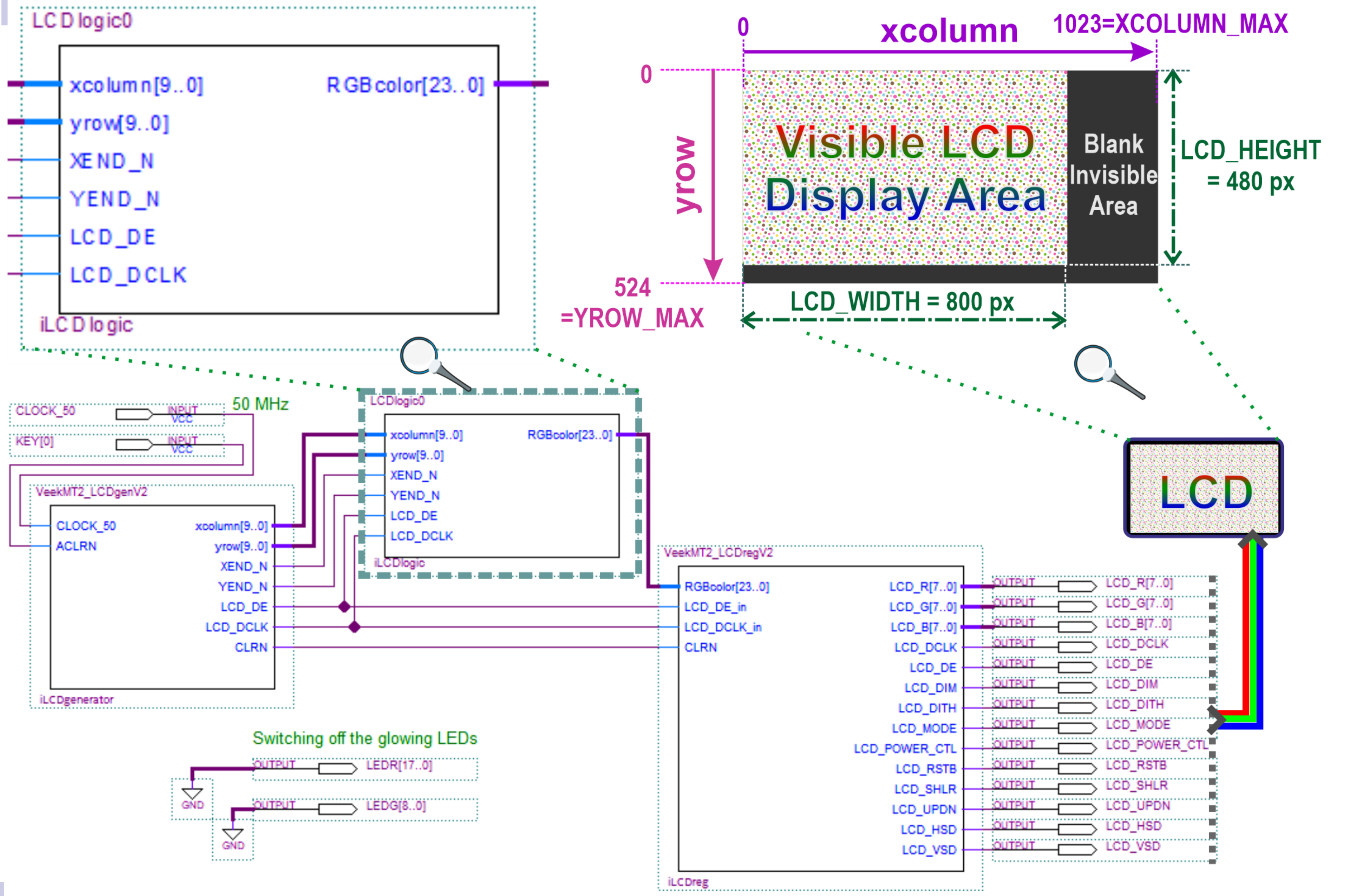
.3. The document **Circuit Design with VHDL** dataflow and structural V10.pdf describes VHDL 1993.
   + Source codes: CircuitDesignWithVHDL_dataflow_and_structural_codes.zip.

Recommended Diagram of LCD Task



You will design only the LCDlogic block*

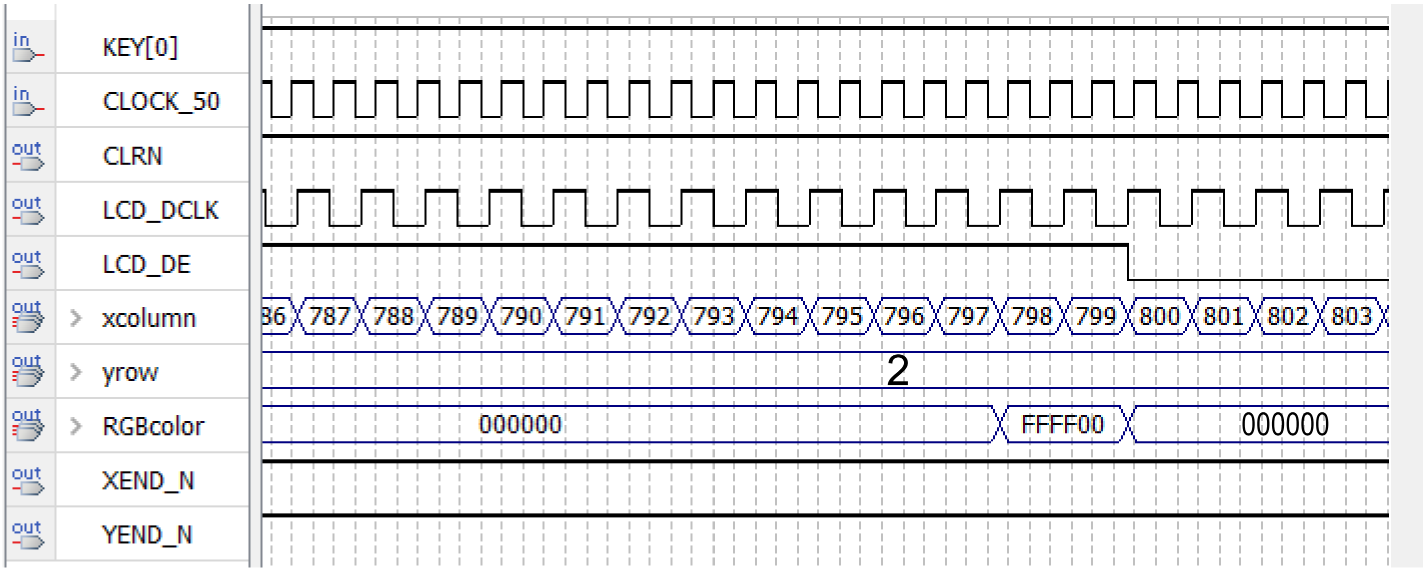
LCD Dimensions

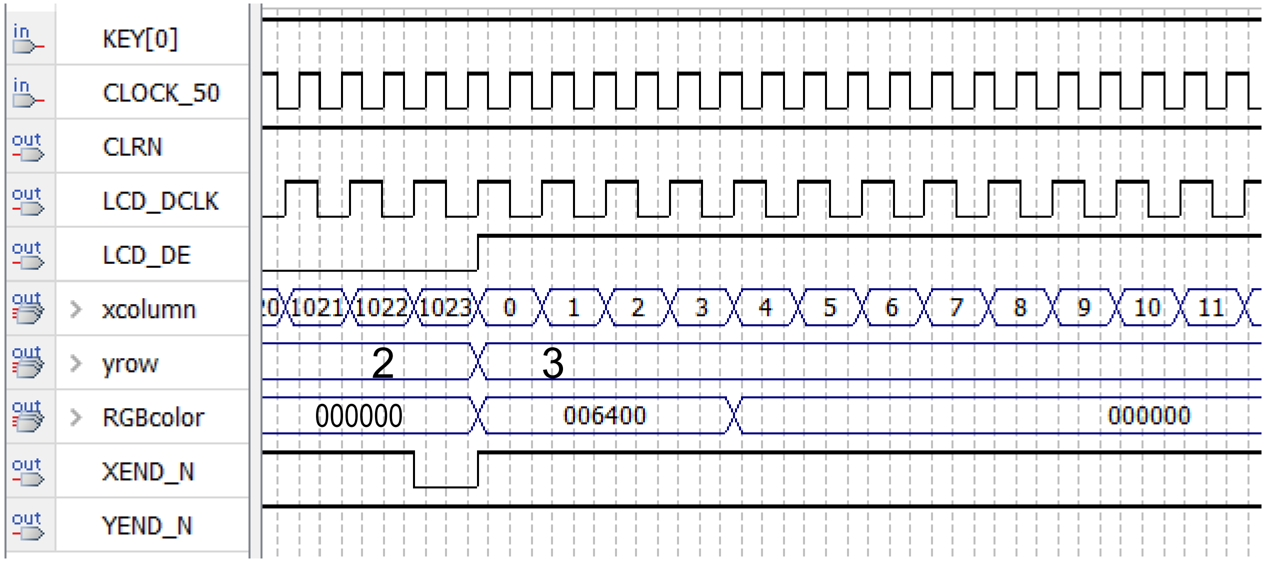


VeekMT2_LCDgenV2: Synchronization Generator

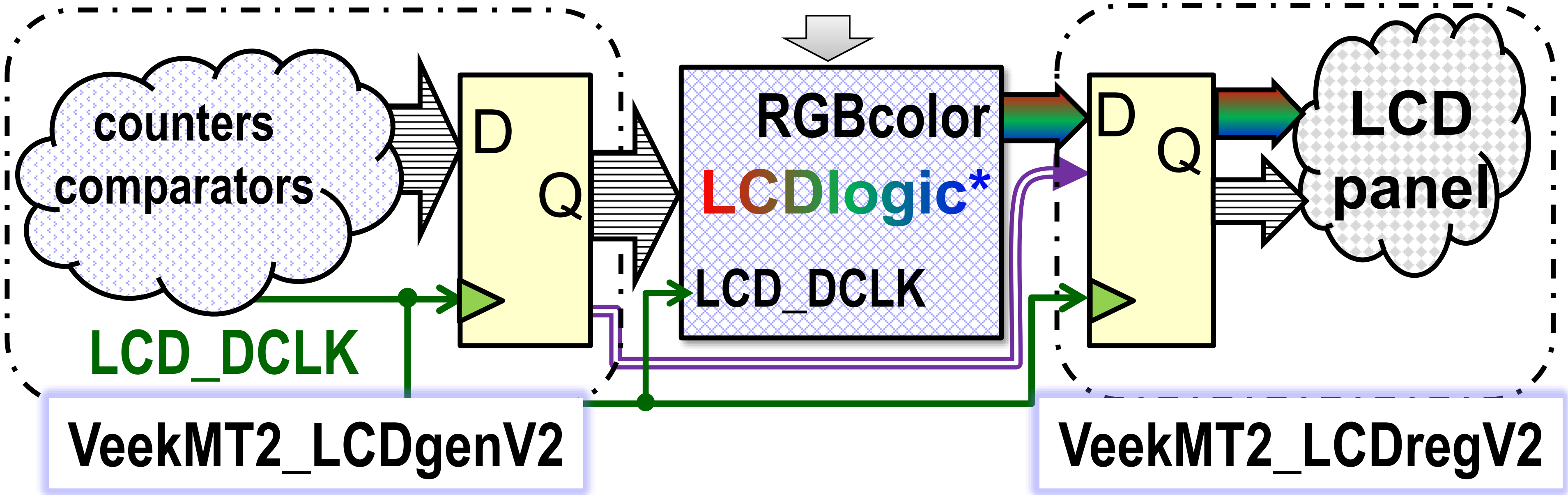
```
// its code implements the following C code loop by counters
unsigned short int xcolumn, yrow;
unsigned int RGBcolor;
bool LCD_DE, XEND_N, YEND_N;
for (yrow = 0; yrow < 525; yrow++)
{ for (xcolumn = 0; xcolumn <1024; xcolumn++)
   {  LCD_DE = xcolumn>=800 || yrow>=480 ? 0 : 1;
      XEND_N = xcolumn==1023 ? 0 : 1;
      YEND_N = yrow==524 ? 0 : 1;
      RGBcolor = LCDlogic( xcolumn, yrow );
   }             // Your circuit of LCD background
}
```

Sample of Outputs





RGBcolor is assigned by combinational logic!



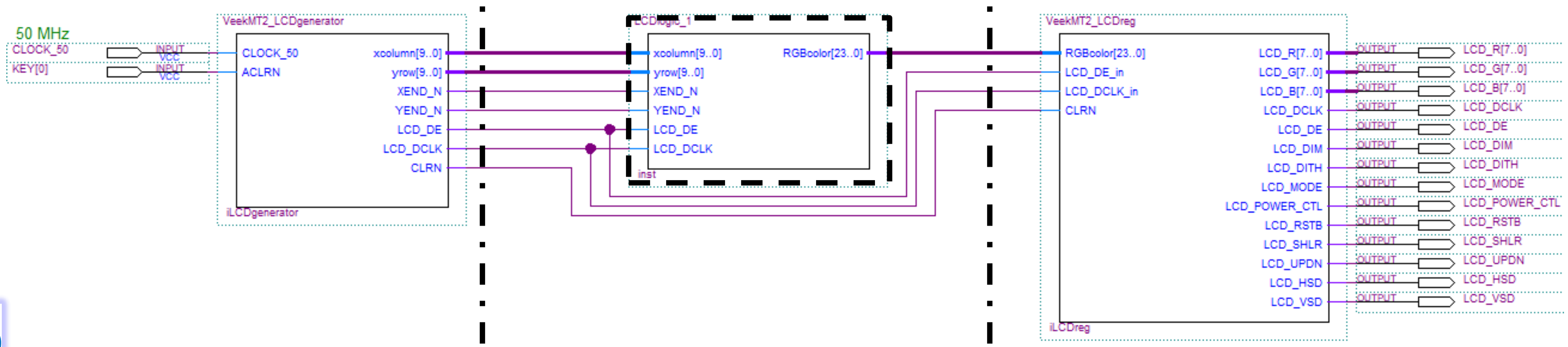
The LCDlogic* module:

- *It is inserted between a pair of registers.*
- *It **deliberately** does not include a register of its RGBcolor output.*
  Such a register would disrupt the synchronization of the RGBcolor output with other LCD synchronization signals, introducing metastability, which we do not want.
- ***RGBcolor must be created only with combinational logic!***
- *Its combinational logic can certainly depend also on signals created by additional processes sensitive to a clock, but the RGBcolor process itself should not use clocks.*

LCD Pipeline

data stream processing

LCD Pipeline and its Stages

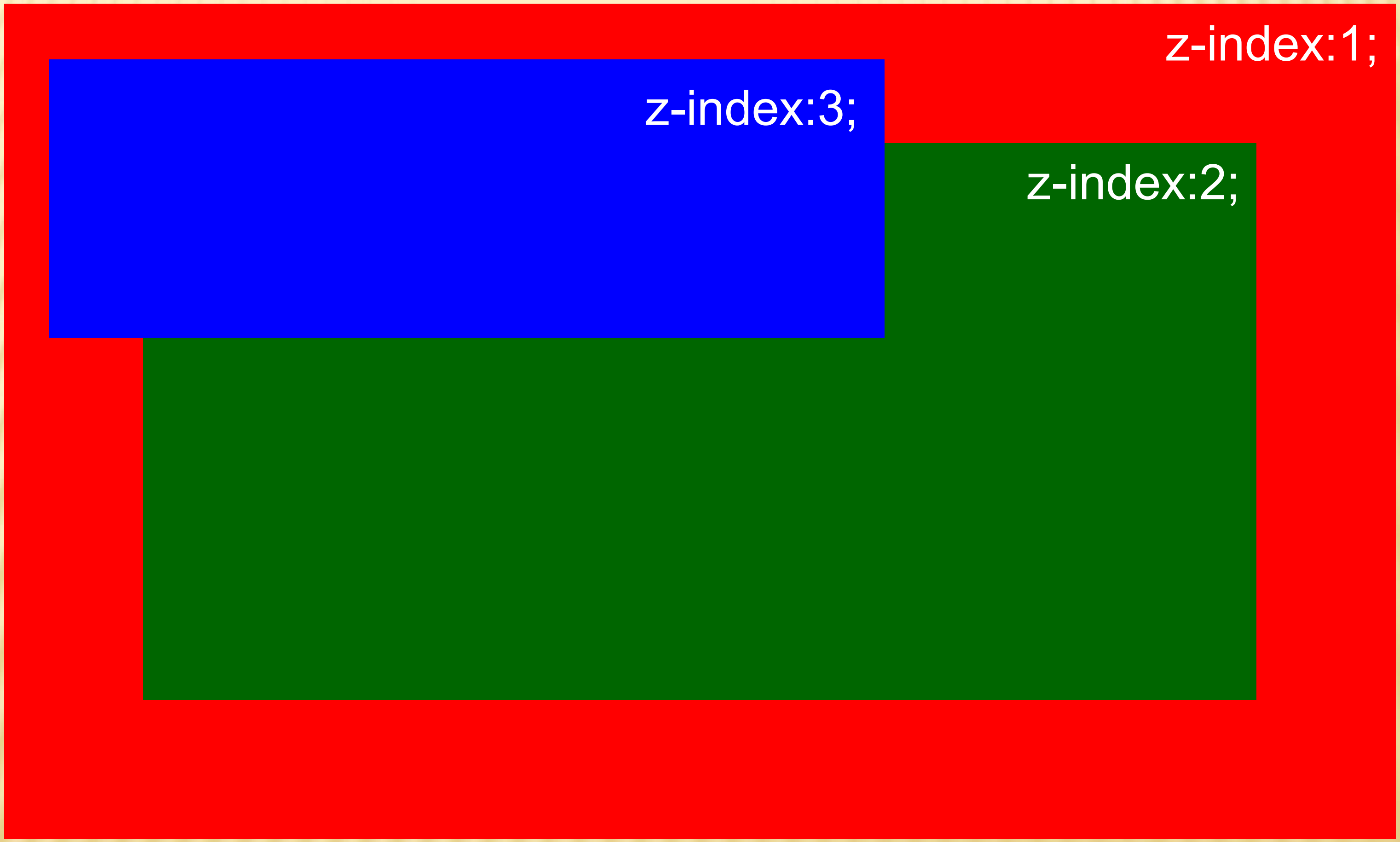


Time

1. stage - GENERATE | *2. stage -EXECUTE* | *3. stage - Write LCD*

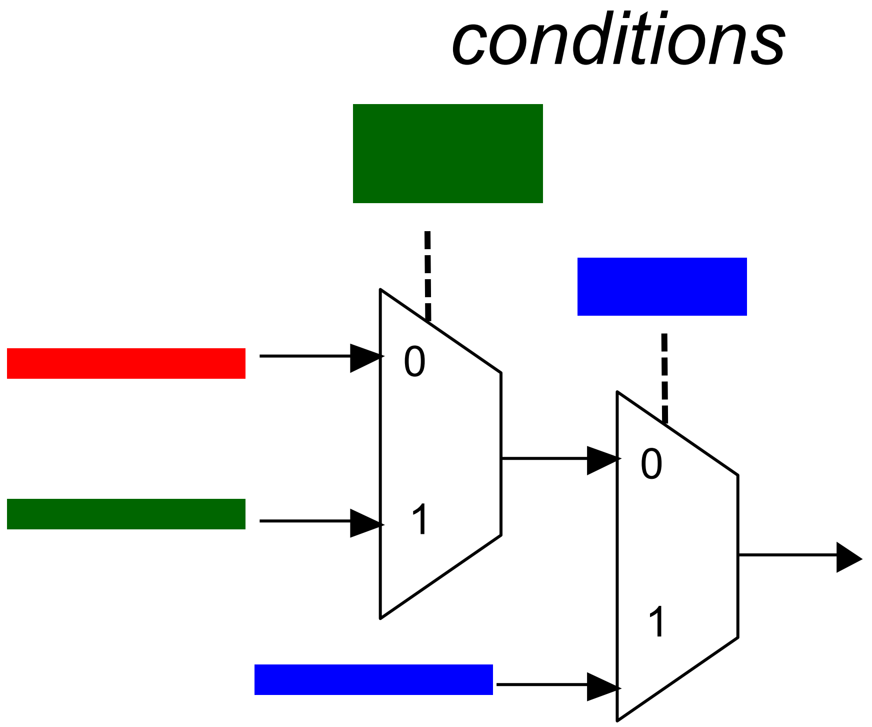
	LCD_generator	LCDlogic	LCDreg
t0	–	–	–
t1	generates x0,y0	–	–
t2	generates x1,y0	calculates x0,y0 color	–
t3	generates x2,y0	calculates x1,y0 color	writes x0,y0 color
t4	generates x3,y0	calculates x2,y0 color	writes x1,y0 color

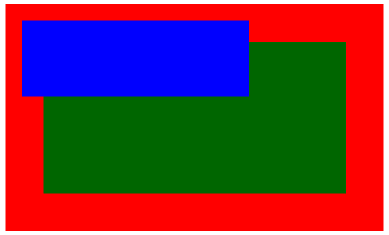
Priority task - visibility in a picture

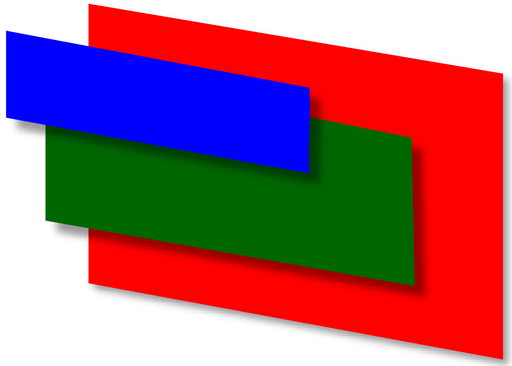


Z-index in CSS Styles

Image by Cascade of Multiplexers

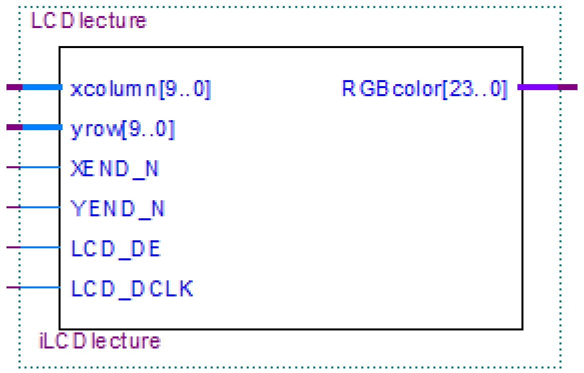




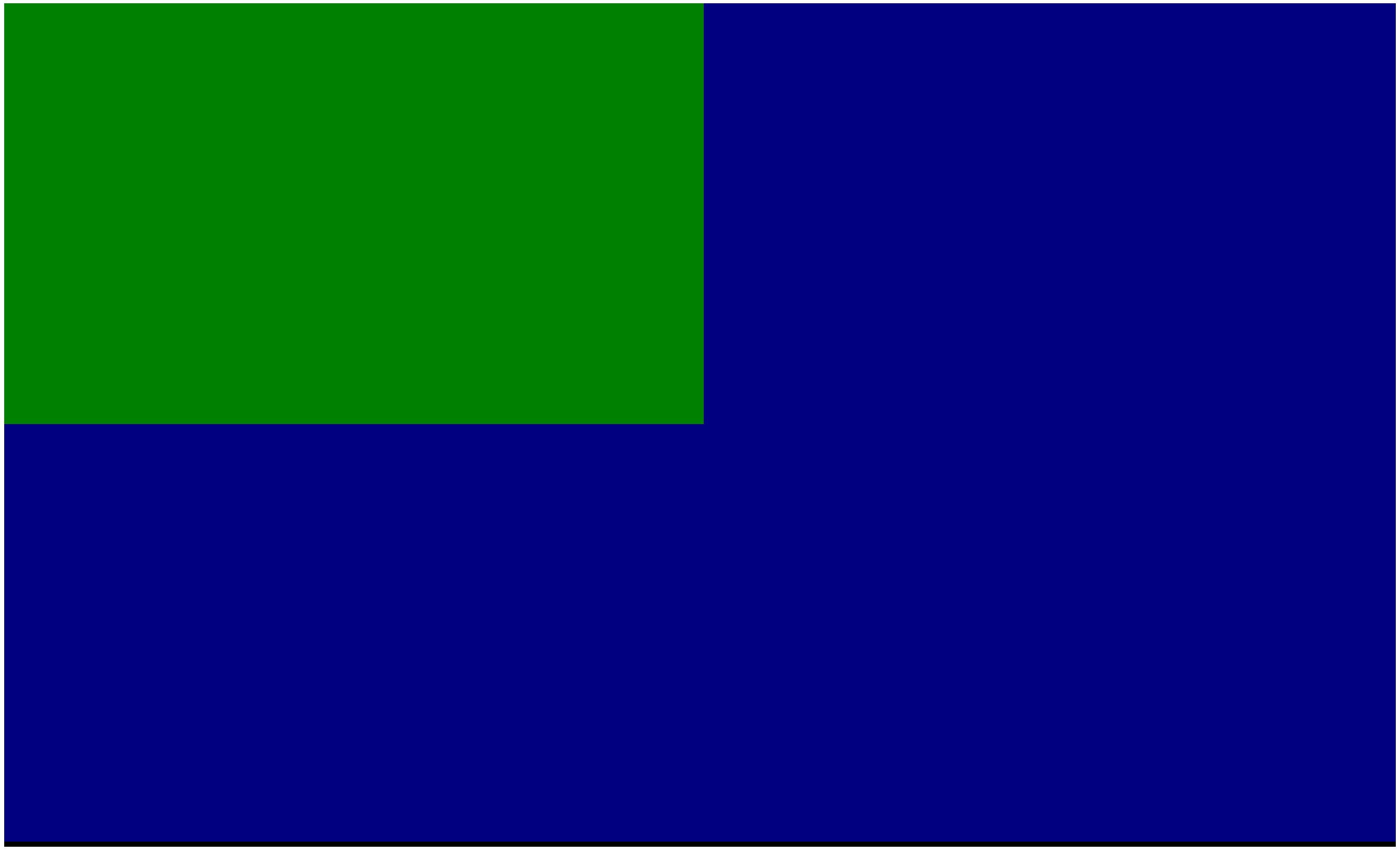


Entity

```
library ieee, work; use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- num
use work.LCDpackV2.all; -- V2.1 our definitions
entity LCDlecture is
 port( xcolumn, yrow  : in  xy_t := XY_ZERO; -- x,y-coordinates of pixel
       XEND_N, YEND_N : in  std_logic := '0'; -- the last xcolumn/yrow
       LCD_DE         : in  std_logic := '0'; -- DataEnable LCD
       LCD_DCLK       : in  std_logic := '0'; -- LCD data clock, 33 MHz exactly
       RGBcolor       : out RGB_t := BLACK);      --  colors assigned to pixels
end entity;
```



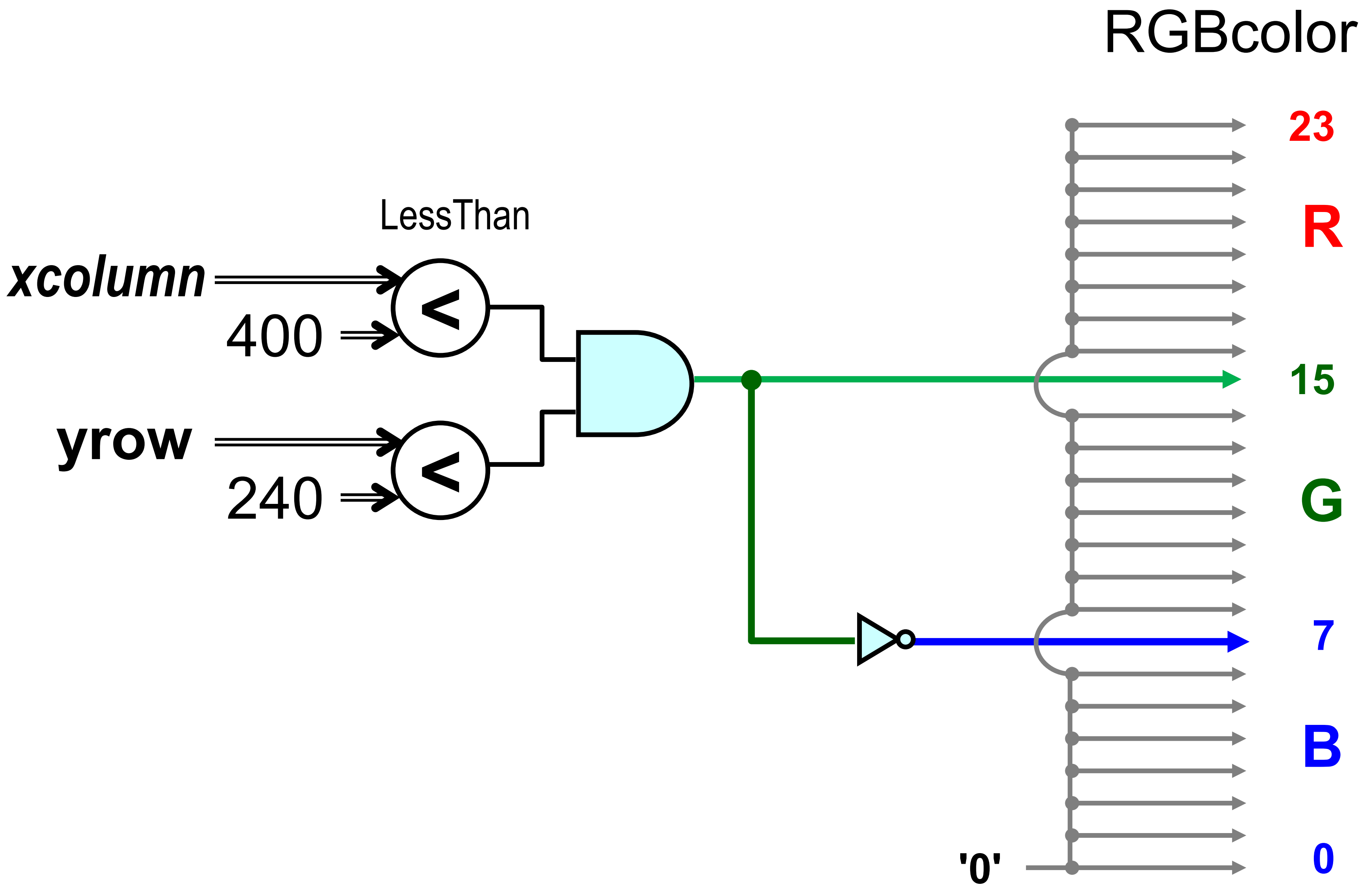
Square



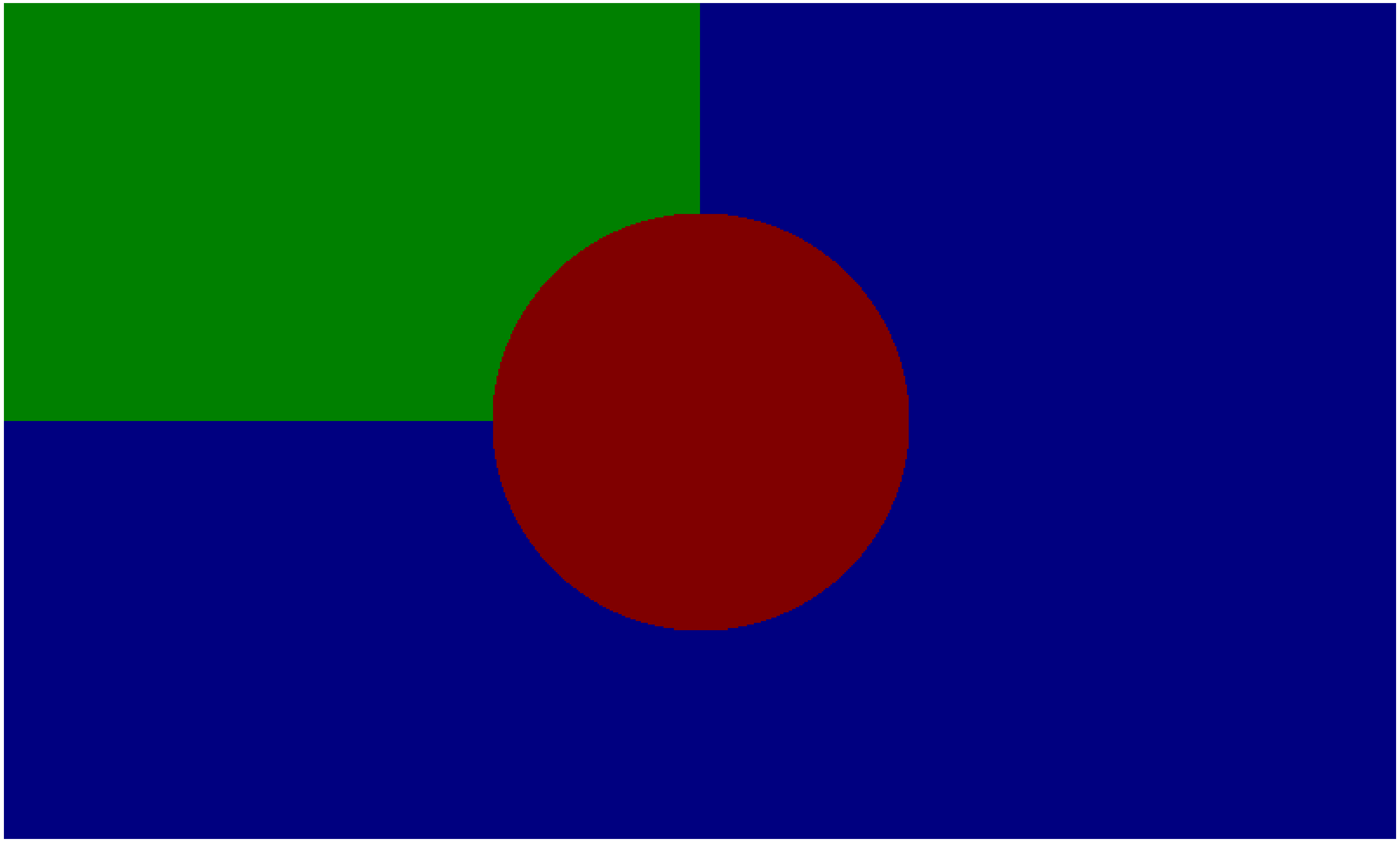
The intro code

```
architecture beh1 of Lecture is
signal x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
signal y : integer  range 0 to 524:=0; -- YROW_MAX-1
begin -- architecture
   x <= to_integer(xcolumn); y <= to_integer(yrow); -- unsigned -> integers

   ---------- our image ------------------------

                              --LCD_WIDTH/2   LCD_HEIGHT/2

RGBcolor <= GREEN when x<400 and y<240 else  NAVY;

--             X"008000"                          X"000080"

-------------------------------------------------------------

end architecture;
```

Its circuit



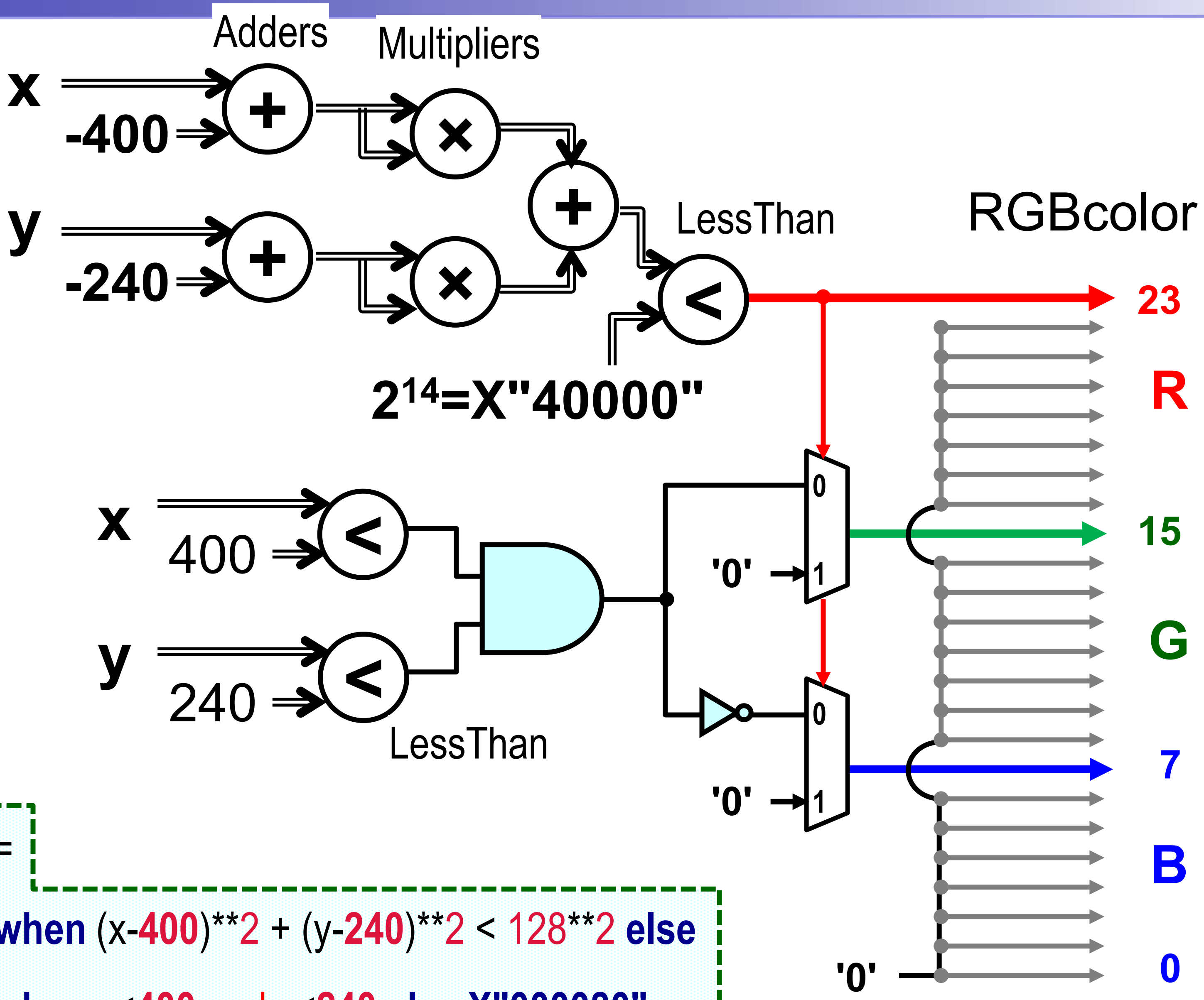
Adding Cicle



Adding Circle

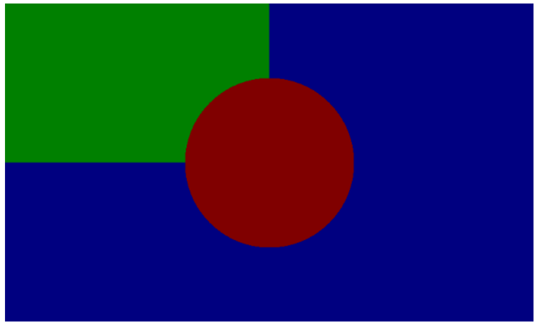
```
architecture beh1 of Lecture is
signal x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
signal y : integer  range 0 to 524:=0; -- YROW_MAX-1
begin -- architecture
  x <= to_integer(xcolumn); y <= to_integer(yrow); -- unsigned -> integers
 ---------- our image ------------------------

  --                X"800000"

RGBcolor <= MAROON when (x-400)**2 + (y-240)**2 < 128**2 else

                  GREEN when x<400 and y<240 else  NAVY;

--                X"008000"                                     X"000080"

--------------------------------------------------------------

end architecture;
```

Its circuit

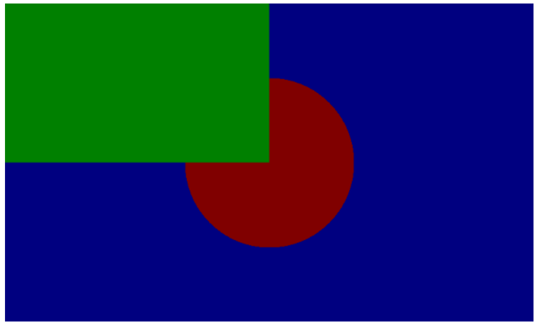


```
RGBcolor <=
X"800000" when (x-400)**2 + (y-240)**2 < 128**2 else
X"008000" when x<400 and y<240 else X"000080";
```

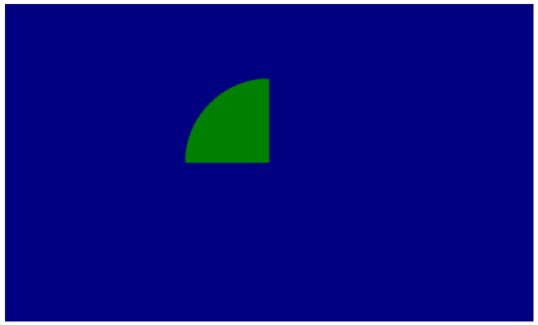
```
RGBcolor <=
MAROON when (x-400)**2 + (y-240)**2 < 128**2 else
GREEN when x<400 and y<240 else
NAVY;
```



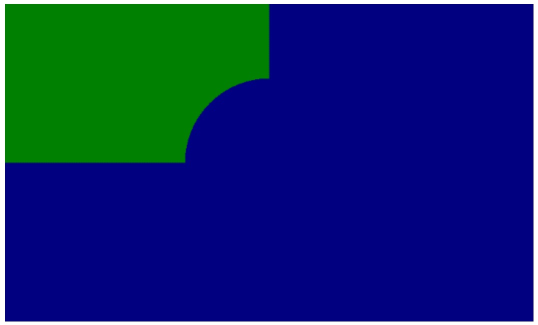
```
RGBcolor <=
GREEN when x<400 and y<240 else
MAROON when (x-400)**2 + (y-240)**2 < 128**2 else
NAVY;
```



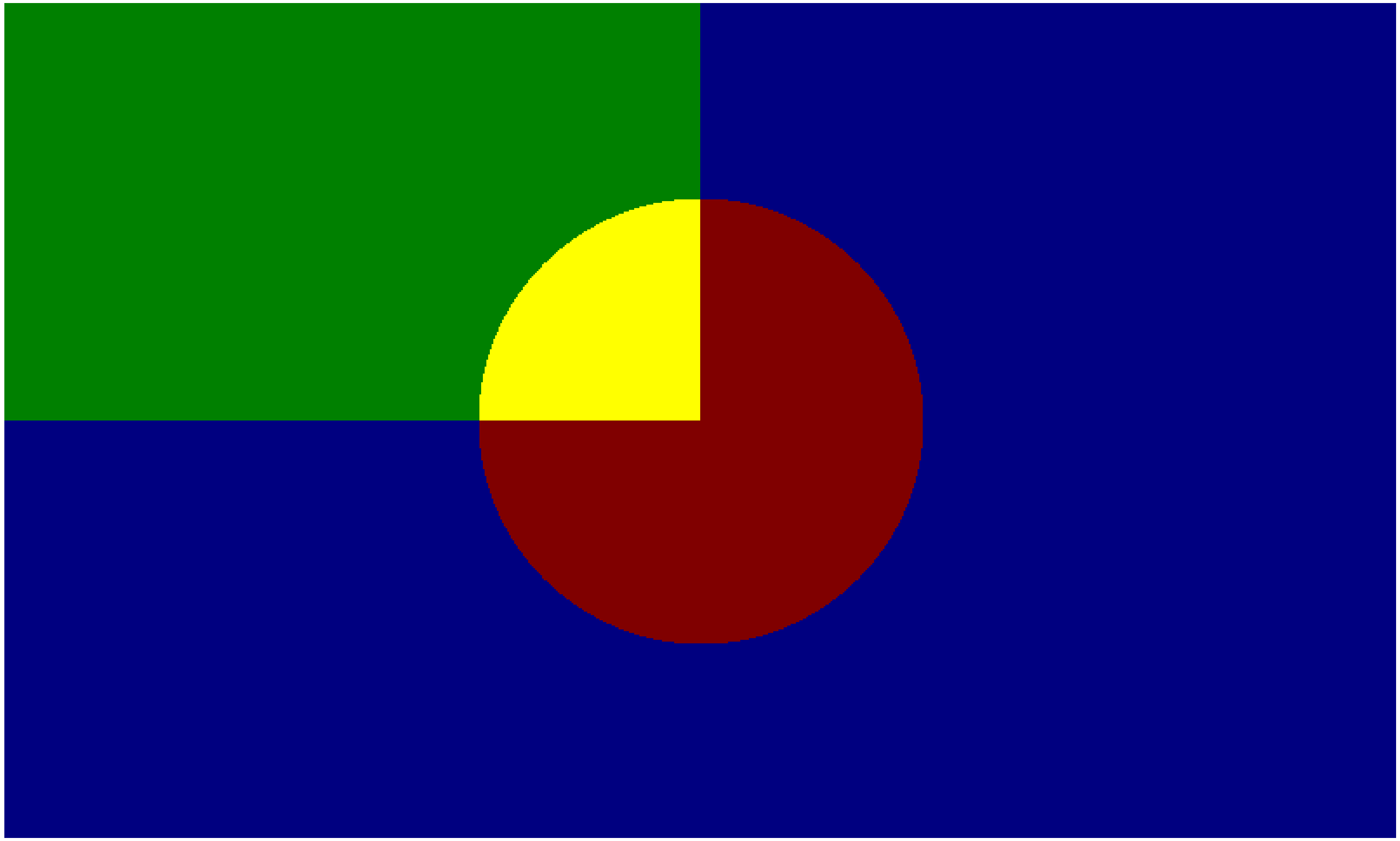
```
RGBcolor <=
GREEN when x<400 and y<240 and
        (x-400)**2 + (y-240)**2 < 128**2 else
NAVY;
```



```
RGBcolor <=
GREEN when x<400 and y<240 and
        (x-400)**2 + (y-240)**2 >= 128**2 else
NAVY;
```

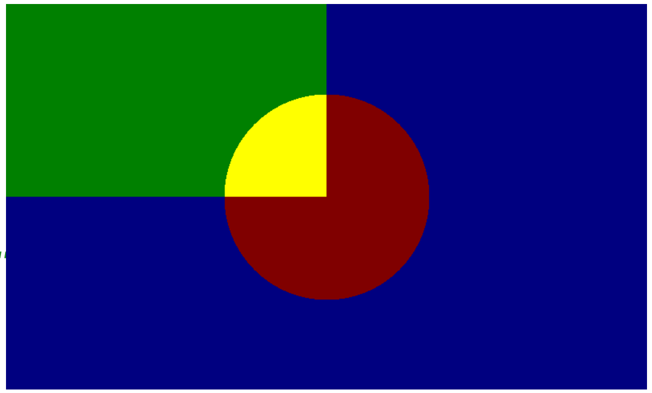


What about this picture?



We cannot nest "when else"

```
architecture beh2 of Lecture is
  signal x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
  signal y : integer  range 0 to 524:=0; -- YROW_MAX-1
  signal isR, isC : boolean:=FALSE;
  begin  x <= to_integer(xcolumn); y <= to_integer(yrow);
 ---------- our image ------------------------
    isR <= x<400 and y<240;
    isC <= (x-400)**2 + (y-240)**2 < 128**2;
    RGBcolor <= GREEN  when isR and not isC else
          YELLOW when isR and isC else
           MAROON when isC else
           NAVY;
   ------------------------------------------------
 end architecture;
```



Dataflow Limits

- The conditions are growing for nested rules
  — the concurrent select with or when else statements **cannot contain inserted statements, only expressions**. So, we must first assign values to auxiliary signals.
- In the sequential domain, it is possible to nest statements and also overwrite previously assigned values.
- The compiler transforms sequential domain codes into concurrent statements by introducing auxiliary signals.

VHDL Process

and its applications

☺

Three types of "concurrent" commands we are familiar with them from the concurrent domain.

<=

with-select-when

when-else

Visit For

Concurrent assignments

Selective assignments

Conditional assignments

implemented using

logic circuit

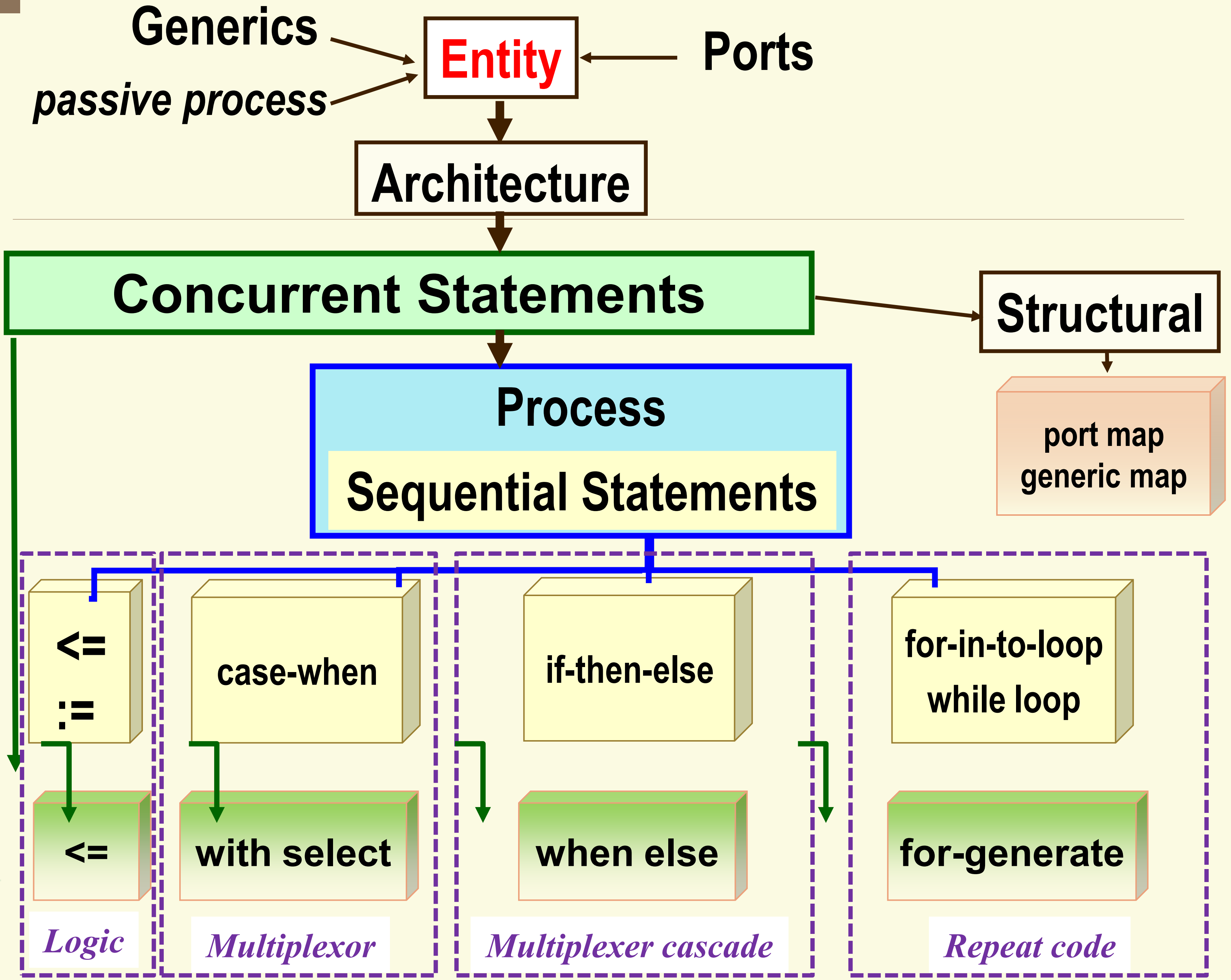
multiplexers

multiplexer cascades 2:1

Sequential VHDL domain

is surrounded by keywords:

- **function ... end function;**
- **procedure ... end procedure;**
- **process ... end process;**
- It also includes the passive process that starts with the optional begin keyword inside an entity declaration to its end.


Generics
passive process
Entity
Ports
Architecture
Concurrent Statements
Structural
port map
generic map
Process
Sequential Statements
<=
:=
case-when
if-then-else
for-in-to-loop
while loop
<=
with select
when else
for-generate
Logic
Multiplexor
Multiplexer cascade
Repeat code

Process Format

```
[label:] process (sensitivity list)
        declarations
      begin
        sequential statements
      end process [label];
```

-- Label is an optional parameter.
However, we can see it in the simulation,
in which it serves as a landmark.

Main Declarations in Sequantial Domain

- **type**, **subtype**, **constant** and **variable**
- **function**, **procedure** - sequential codes of circuit parts for local use. They are not called in the circuit, but inserted by inline expansion technique.
- **use** keyword allows adding additional libraries locally available only here, if they are needed.

❖ No **signal** or **component** declarations are allowed here.

Process and its Sensitivity List

- A ***process is a VHDL construct that contains a set of actions to be executed sequentially.*** These actions are known as ***sequential statements. The process itself is a concurrent statement.*** It can be interpreted as a circuit part enclosed inside a black box whose behavior is described by the sequential statements. We may or may not be able to construct physical hardware that exhibits the desired behavior.
- The **sensitivity-list is a list of signals to which the process responds (i.e., is "sensitive** to"). The **declaration part consists of various declarations that are local to the process.**
- The list of sensitivities in the process is not a formal parameter, but hints s what inputs will cause the requirement for new evaluations.
- Whereas the appearance of a VHDL process is like a function or procedure of a traditional programming language, the behavior of the process is very different. A VHDL process is not invoked (or called) by another routine. It acts like a circuit part, which is either active (known as ***activated) or inactive (known as suspended).***
- *In simulations*, ***a VHDL process is activated when*** a signal in the sensitivity list changes its value, like a circuit responding to an input signal. Once a process is activated, its statements will be executed sequentially until the process is complete. The process is then suspended until the next signal change.

[VHDL 1993 Reference, IEEE]

Variables

- A variable describes the abstract behavior of the system.
- We can declare variables in a process, function or procedure. We are not allowed to declare signals there, they belong only to the architecture.
- Variables are always local, i.e., they cannot be referenced from elsewhere than from the part that declared them.
- *Note: Shared variables were introduced in VHDL 93 for simulation purposes. Although they can be declared outside the process, they are difficult to implement in hardware. As a rule, they are circumvented by complex automatic conversions, which sometimes have unpredictable results.*
- Shared variables are strictly prohibited in the circuit synthesis portions of LSP projects. The rule is also applied by professional development companies.

Assignments: Variables vs. Signals

- In the sequential domain, we cannot define signals, only variables, but by their declarations are similar to signals:

  variable variable-name, variable-name , ... : **data-type**;

- New values are assigned to variables by **blocking** assignments:

  variable-name **:=** value-expression;

- Signals represent connections by wires and need **non-blocking** concurrent assignments:

  signal-name **<=** signal-expression;

  The reasons for this will be the topic of some the following lecture.

- **In VHDL 2008**, we can also use "concurrent statements" when-else and with-select in the sequential domain.
  But the Quartus Lite versions do not allow them here!

VHDL-2008 v GHDL

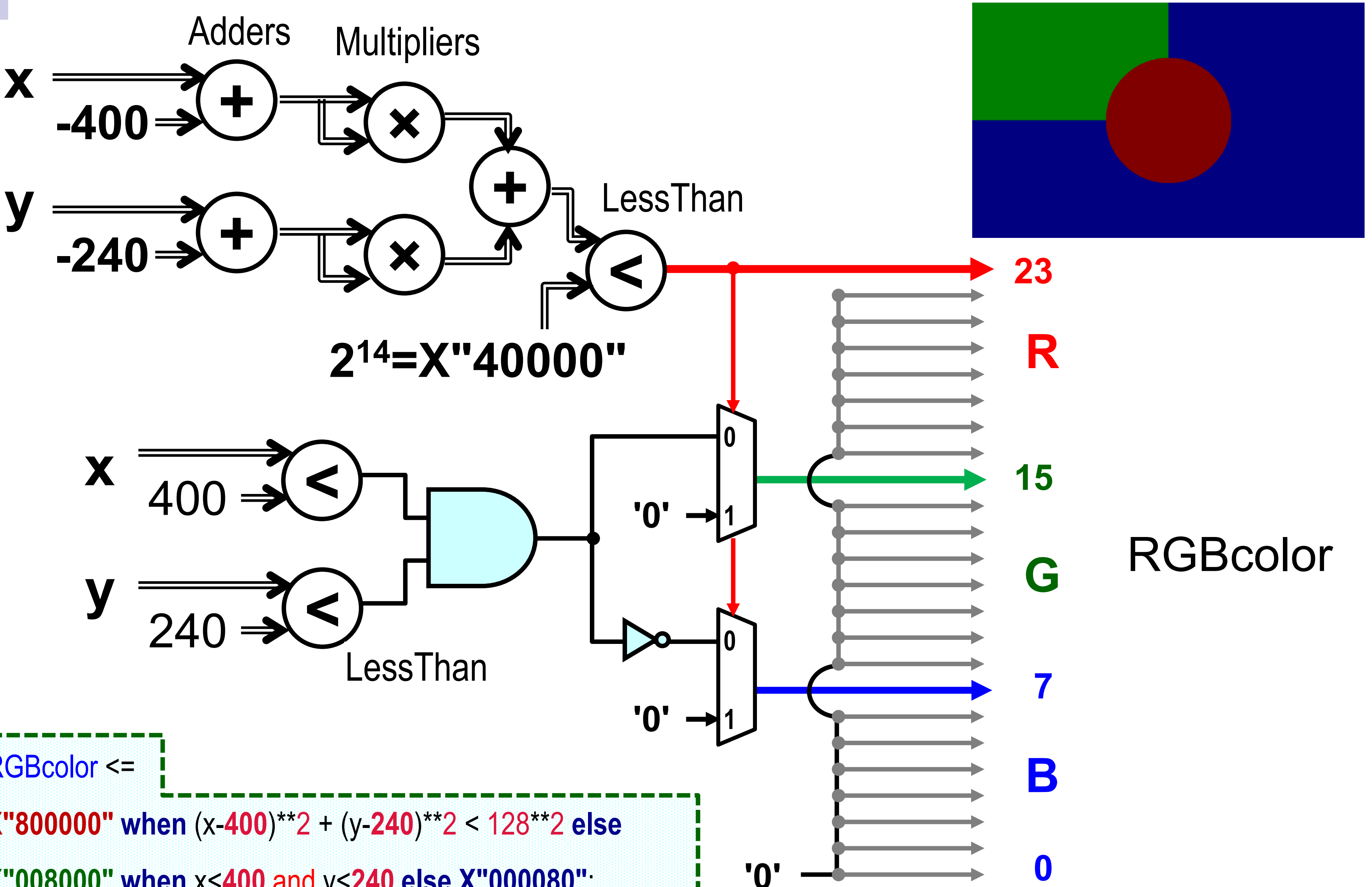
```
architecture rtl2008 of Lecture is
begin -- architecture
LSPimage : process( xcolumn, yrow)
-- In any process, we prefer variables. They must be initialized in the code!!!
-- The values after definitions are mainly for simulations.
 variable RGB :RGB_t :=BLACK; -- the color of pixel
 variable x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
 variable y : integer  range 0 to 524:=0; -- YROW_MAX-1
 begin  -- process
  x := to_integer(xcolumn); y := to_integer(yrow); -- unsigned to integers
  ---------- our image -----------------------
  RGB := MAROON when (x-400)**2 + (y-240)**2 < 128**2 else
           GREEN when x<400 and y<240 else  --LCD_WIDTH/2, LCD_HEIGHT/2
           NAVY;
---------------------------------------------------------
 RGBcolor <= RGB; -- assigning the output signal at the end
 end process;
end architecture;
```

However, Quartus Lite does not allow when else *in sequential parts*

VHDL in Quartus Lite

```
architecture rtl of Lecture is
begin -- architecture
LSPimage : process( xcolumn, yrow)
 variable RGB :RGB_t :=BLACK; -- the color of pixel
 variable x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
  variable y : integer  range 0 to 524:=0; -- YROW_MAX-1
 begin   x := to_integer(xcolumn); y := to_integer(yrow); -
---------- our image -----------------------
     if (x-400)**2 + (y-240)**2 < 128**2 then RGB := MAROON;
    elsif x<400 and y<240 then RGB := GREEN;
    else RGB :=  NAVY;
    end if;
------------------------------------------------------------

     RGBcolor <= RGB;
 end process; end architecture;
```

The result is the same



```
RGBcolor <=
X"800000" when (x-400)**2 + (y-240)**2 < 128**2 else
X"008000" when x<400 and y<240 else X"000080";
```

Two possible if-styles

more *if*-s

```
---------- our image ---------------------------------------
RGB :=  NAVY;
if x<400 and y<240 then RGB := GREEN; end if;
if (x-400)**2+(y-240)**2<128**2 then RGB := MAROON; end if;
------------------------------------------------------------
```

one long *if-elsif* command

```
---------- our image ---------------------------------------
 if (x-400)**2 + (y-240)**2 < 128**2 then RGB := MAROON;
 elsif x<400 and y<240 then RGB := GREEN;
 else RGB :=  NAVY;
 end if;
------------------------------------------------------------
```

Note: if then elsif vs multiple if then

- Both methods lead to optimal circuits, and it depends only on the designers which code style they prefer.
- In the first step of the compilation, multiple if-them statements start with the **highest** priority condition which leads to a meta-scheme consisting of cascading two-input multiplexers.
- The multiple if-then statements describe the same cascade. They only begin from the **lowest**-priority condition.

If statements can be nested

```
LSPimage : process( xcolumn, yrow, LCD_DE )
   variable RGB :RGB_t :=BLACK; -- the color of pixel
  variable x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
  variable y : integer  range 0 to 524:=0; -- YROW_MAX-1
  variable isR: boolean:=false; -- tis Rectangle
  begin   x := to_integer(xcolumn); y := to_integer(yrow);
    ---------- our image ------------------------
    RGB :=  NAVY;
    isR:=x<400 and y<240;
    if isR then RGB := GREEN; end if;
    if (x-400)**2 + (y-240)**2 < 128**2 then
      if isR then RGB:=YELLOW; else RGB := MAROON; end if;
    end if;
    ------------------------------------------------------------
  RGBcolor <= RGB; -- assigning the output signal at the end
 end process;
```

Duplicate conditions? - Quartus will merge them

```
---------- our image -------------------------
    RGB :=  NAVY;
    if x<400 and y<240   then RGB := GREEN; end if;
    if (x-400)**2 + (y-240)**2 < 128**2 then
    if x<400 and y<240  then RGB:=YELLOW;
                         else RGB := MAROON; end if;
    end if;    -------------------------------------------------
```

But better coding styles define variables for shared conditions to emphasize their repetition...

```
variable isR: boolean:=false;
--
isR:= x<400 and y<240;  -- We must assign isR in our process code
if  isR then RGB := GREEN; end if;
if  isR  then RGB:=YELLOW;  else RGB := MAROON; end if;
```

Syntax of Functions

```
[pure | impure ]
function function_name (parameter_list)
            return type_name is
   [sequential_declarations] -- identical to processes
  begin
           sequential statements…
   end [function] [function_name];
```

[ ] *indicates the possibility of omitting the element,*

| *denotes a selection of one option*

pure*, the default option, specifies that the function has no side effects.*

assignIf

```
function assignIf(cond:boolean;
                   colorTrue, colorFalse:RGB_t)
return RGB_t is
begin
    if cond then return colorTrue;
            else return colorFalse;
     end if;
end function;
```

```
RGB := assignIf(x<400 and y<240,
                        YELLOW, MAROON);
```

LCDpackV2 version V2.1 and higher includes assignIf also overloaded for integer results

Code with assignIf

```
LSPimage : process( xcolumn, yrow)
 variable RGB :RGB_t :=BLACK; -- the color of pixel
 variable x : integer  range 0 to 1023:=0; -- XCOLUMN_MAX-1
 variable y : integer  range 0 to 524:=0; -- YROW_MAX-1
 variable isR: boolean:=false;
  begin   x := to_integer(xcolumn); y := to_integer(yrow);
  ---------- our image ------------------------
   RGB :=  NAVY; isR:=x<400 and y<240;
   if isR then RGB := GREEN; end if;
   if (x-400)**2 + (y-240)**2 < 128**2 then
        RGB:=assignIf(isR,YELLOW, MAROON);
   end if;
   ----------------------------------------------------
 RGBcolor <= RGB; -- assigning the output signal at the end
 end process;
```

PROCESS

and its "sensitivity list"

[Obrázek: 9gag.com]

VHDL wait command for simulations

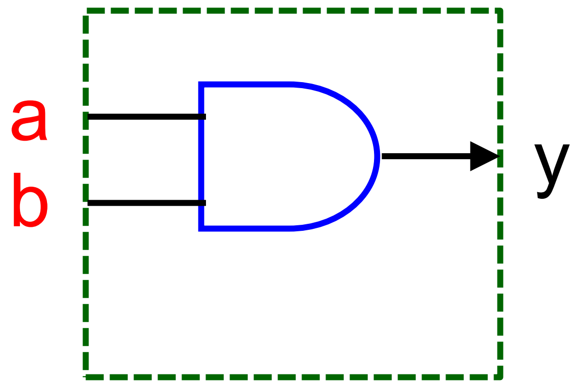
- **wait on** *signal list***;**
  - wait until the signal changes;
    e.g.: **wait on** a;
- **wait until** *condition***;**
  - wait until the condition is met;
    e.g.: **wait until** c='1';
- **wait for** *duration***;**
  - wait for a specified duration;
    e.g.: **wait for** 10 ns;
- **wait;**
  - wait indefinitely → end of the process in which the command was used.

"Sensitivity list" ≡ "wait on!

```
process(a,b)
begin

-- some statements, e.g.
y<=a and b;

end process;
```

⇔

```
process
begin

-- some statements, e.g.
y<=a and b;

wait on a, b;
end process;
```



If a or b input values do not change,
the value of y remains unchanged
← the property of a combinational circuit

Quartus

- *To speed up simulations, VHDL requires specifying sensitivity lists, i.e., the lists of all signals whose change requires a new simulation run of the process. Quartus discovers incomplete lists and shows warnings. Please correct it.*
- *The sensitivity lists control the simulation. The incomplete lists create incorrect simulations.*
- ***A sensitivity list is not analogous to function parameters**! It is the alternative syntax forms of the process with the "wait on" command. Instead of writing "wait on" signals at the end of a process, we see them at the beginning.*

```
process(a,b)
begin

-- some statements, e.g.
y<=a and b;

end process;
```

```
process
begin

-- some statements, e.g.
y<=a and b;

wait on a, b;
end process;
```

- *Assembling a circuit from code is not an easy task, so a subset of VHDL commands is only synthesizable. We must also adhere to many strict rules. In part intended only for simulations, we can use anything because they are not compiled as circuits, but executed as programs.*

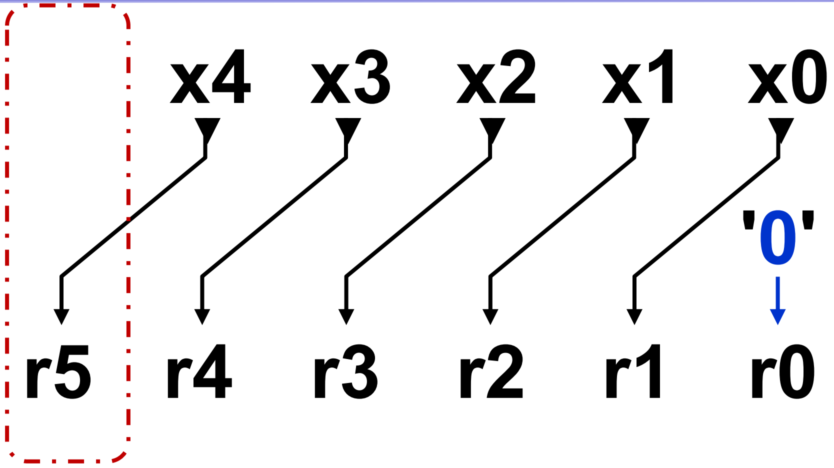
Arithmetic in Circuits

How is it done?

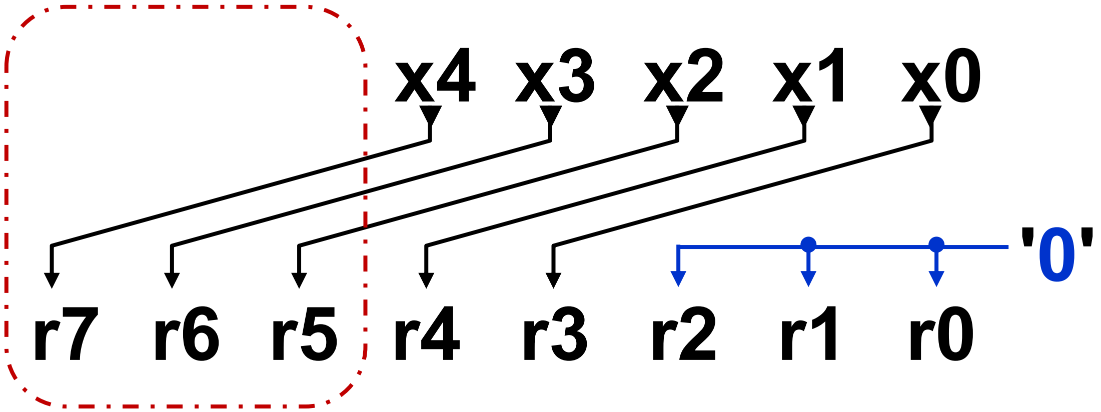
Logic Circuits textbook, chapter 6.

Multiplication by a Power of 2

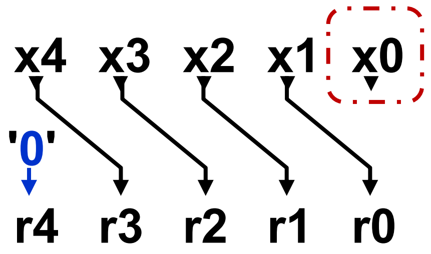
$x * 2^1$



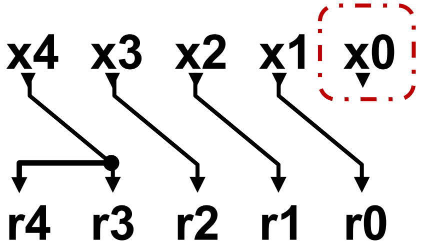
$x * 2^3$



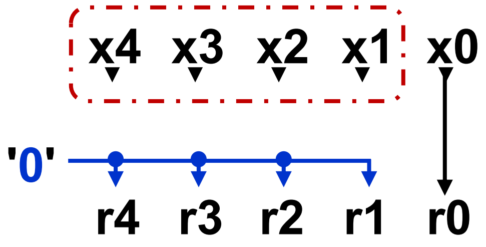
Division by Powers of 2 - Wires



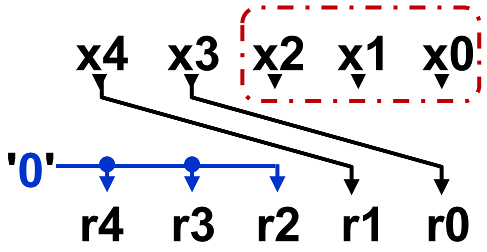
unsigned (x) / 2^1



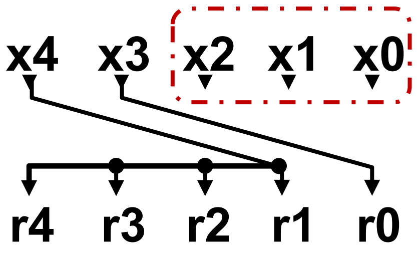
signed (x) / 2^1



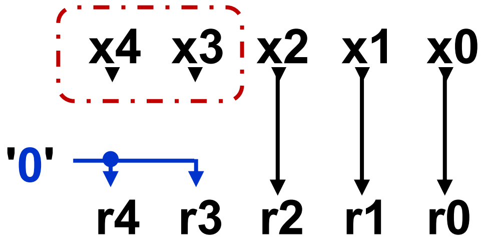
x mod 2^1



unsigned (x) / 2^3

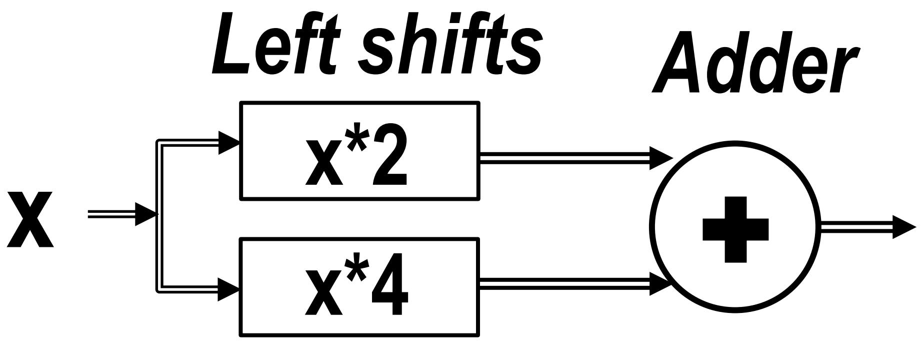


signed (x) / 2^3

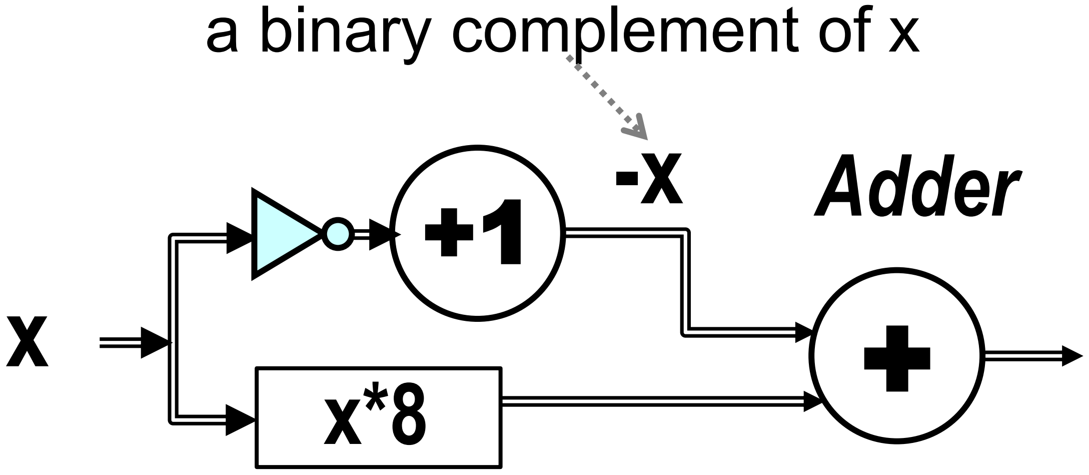


x mod 2^3

Multiplication by the Sum of Powers of 2



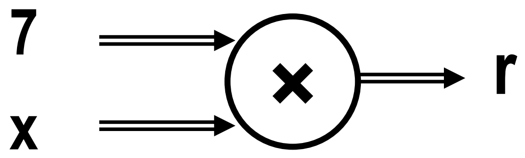
$6*x=(2^1+2^2)*x$



$7*x=(2^3-2^0)*x$

Quartus multiplies by 2^N -1 sometimes

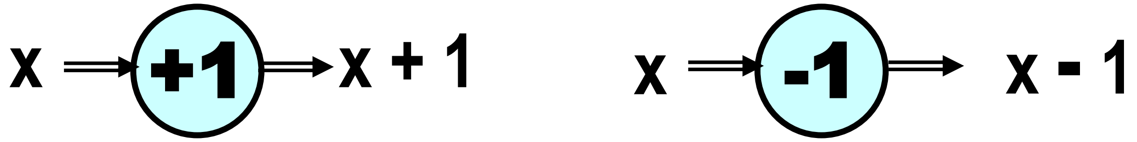
- *by using an adder...*
- *by a multiplier*



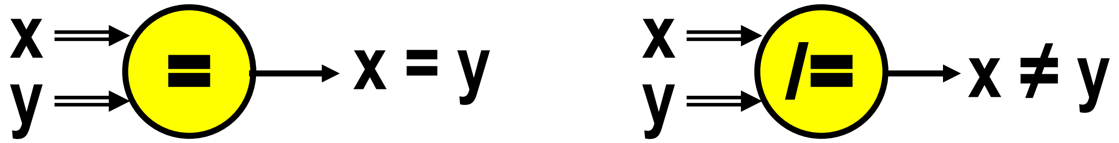
Combinational Arithmetic Circuits

By minimizing the SoP or PoS, it is appropriate to design:

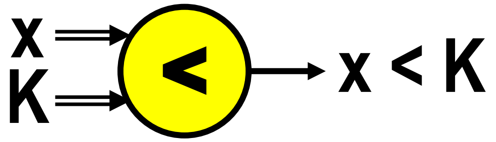
Adders +/-1



Comparators
equal and not equal



Comparators
with K constant

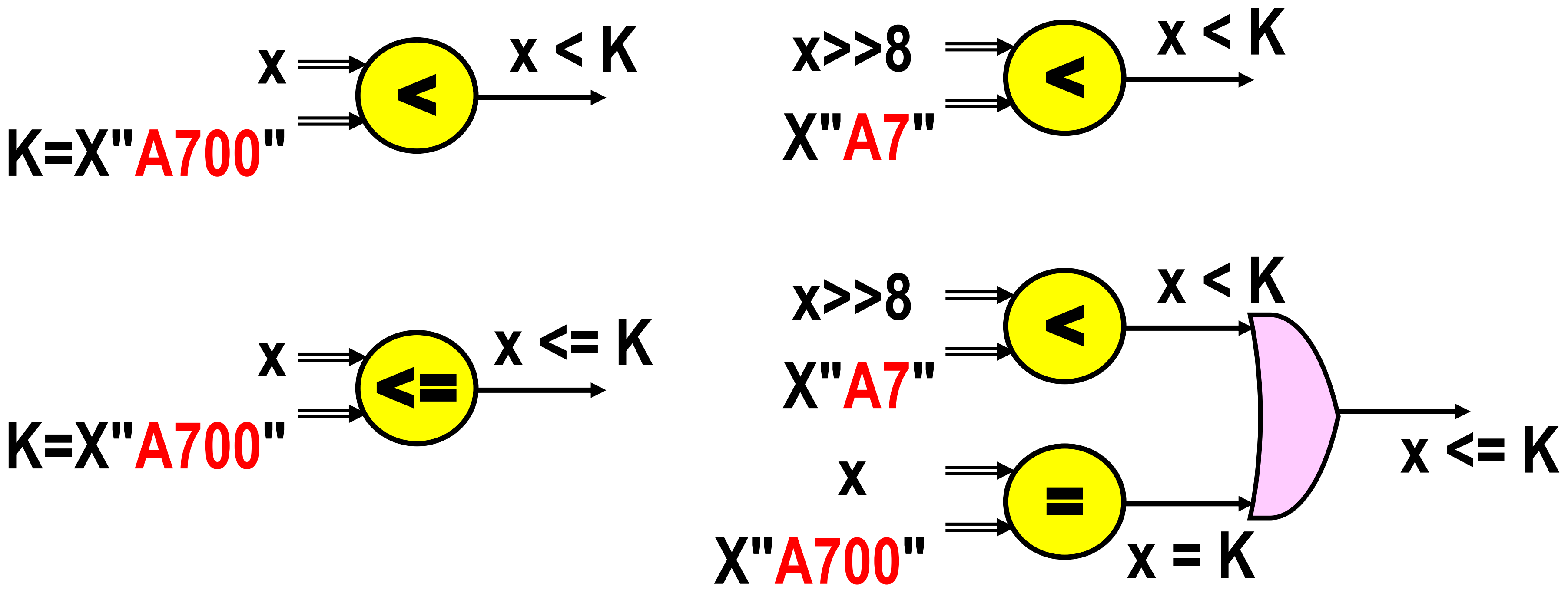


and other types <=   =   /=   >=

≠ is **!=** in C, **~=** in Matlab, <> in Basic or Pascal, but **/=** in VHDL

Constants *K* | 2^N

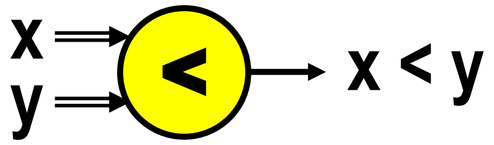
*If a constant **K | 2^N** , i.e., K is divisible by a N power of 2, where integer N>0, then K has **N lower bits = '0'**. In that case, **x<K** and **x>=K** comparators are smaller because they can test only the upper non-zero bits.*



Operation with two general numbers x,y

SoP and PoS minimization give too complex results for the following circuits. Their decompositions are necessary:

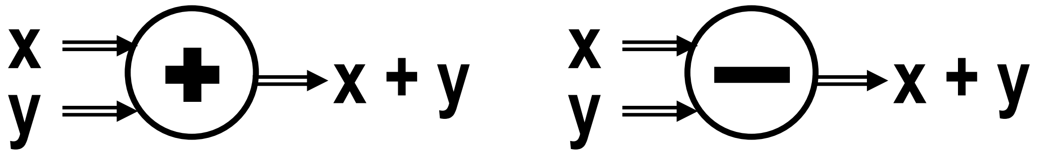
- Comparators



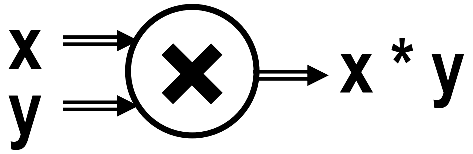
and comparisons

>= > <=

- Adders and subtractors

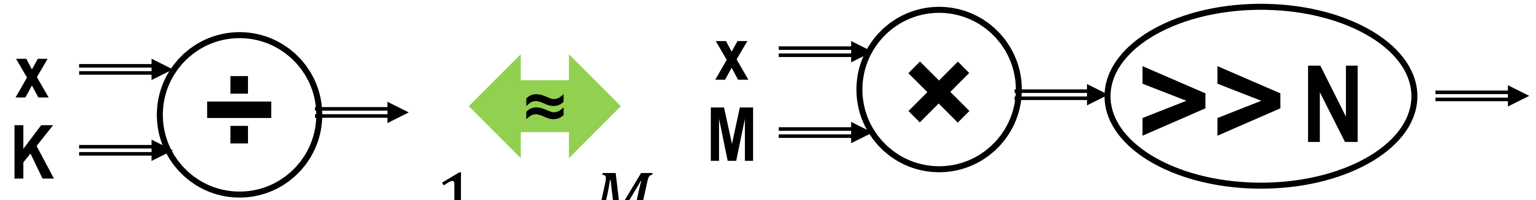


- Multipliers



Approximations

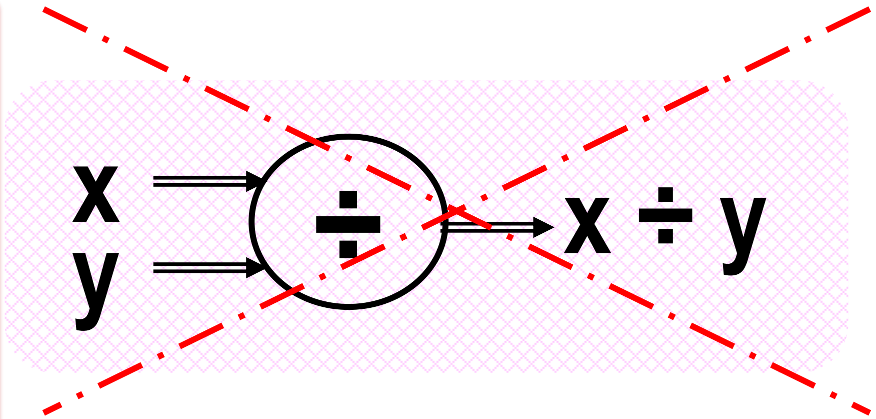
Divisions by the constants K≠2 ^{N} can be approximated



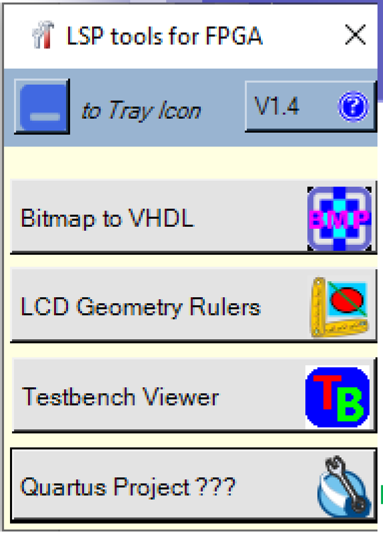
$$\frac{1}{K} \cong \frac{M}{2^N}$$

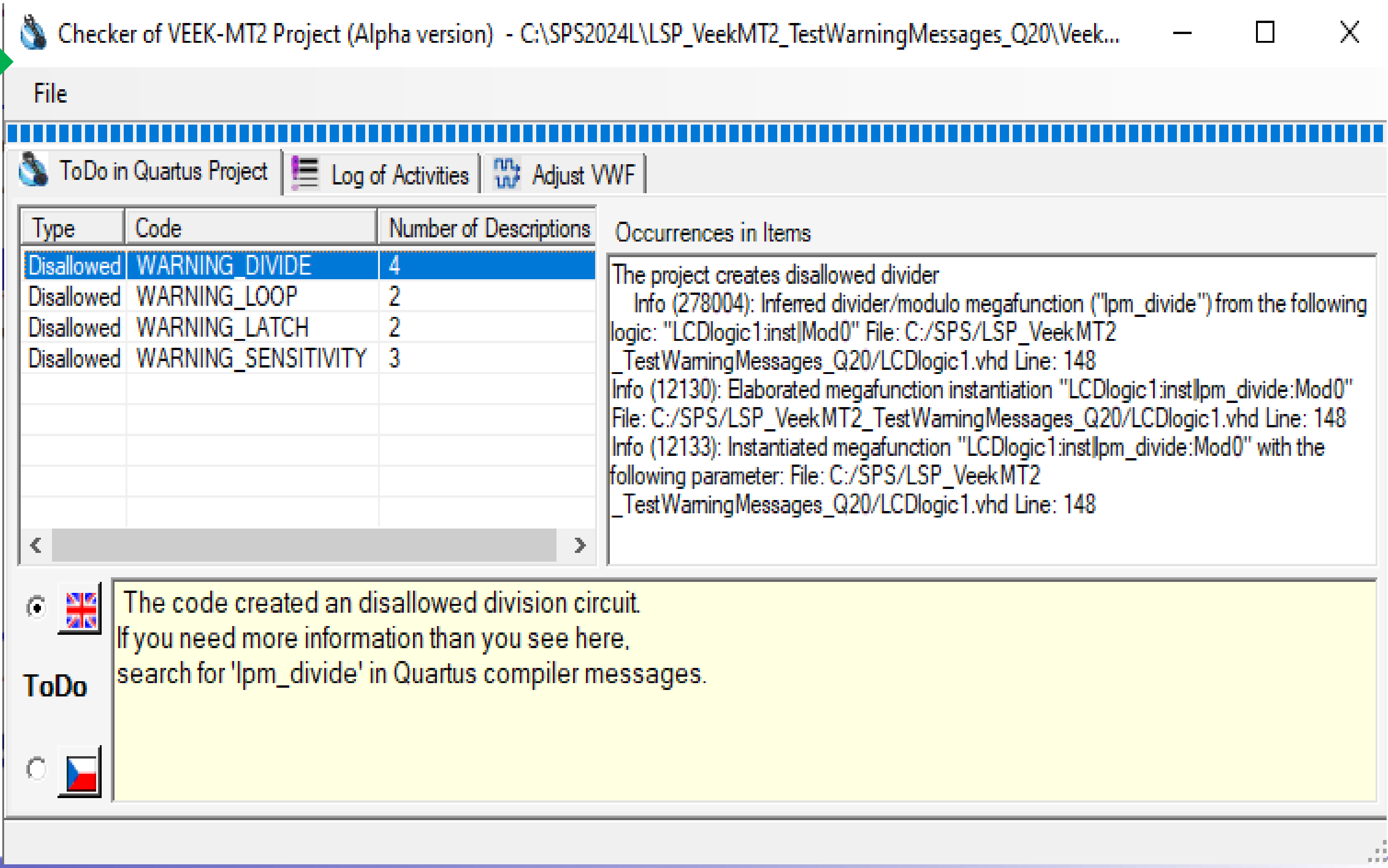
Note: In LCD, xcolumn and yrow coordinates can be divided by any number with the aid of counters, see LCD Backgrounds page 21.

Do not use
the direct divisions by
a general number or by constant ≠ 2^N
because they have extremely
complex circuit implementations
!!!



Use LSP tools Checker of Projects




Checker of VEEK-MT2 Project (Alpha version) - C:\SPS2024L\LSP_VeekMT2_TestWarningMessages_Q20\Veek...
File
ToDo in Quartus Project
Log of Activities
Adjust VWF
Occurrences in Items
ToDo

Type	Code	Number of Descriptions
Disallowed	WARNING_DIVIDE	4
Disallowed	WARNING_LOOP	2
Disallowed	WARNING_LATCH	2
Disallowed	WARNING_SENSITIVITY	3

```
The project creates disallowed divider
    Info (278004): Inferred divider/modulo megafunction ("lpm_divide") from the following
logic: "LCDlogic1:inst|Mod0" File: C:/SPS/LSP_VeekMT2
_TestWarningMessages_Q20/LCDlogic1.vhd Line: 148
Info (12130): Elaborated megafunction instantiation "LCDlogic1:inst|lpm_divide:Mod0"
File: C:/SPS/LSP_VeekMT2_TestWarningMessages_Q20/LCDlogic1.vhd Line: 148
Info (12133): Instantiated megafunction "LCDlogic1:inst|lpm_divide:Mod0" with the
following parameter: File: C:/SPS/LSP_VeekMT2
_TestWarningMessages_Q20/LCDlogic1.vhd Line: 148
```

The code created an disallowed division circuit.
If you need more information than you see here,
search for 'lpm_divide' in Quartus compiler messages.