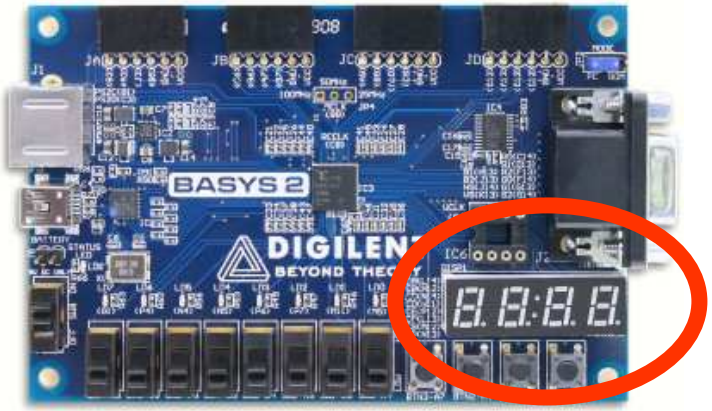
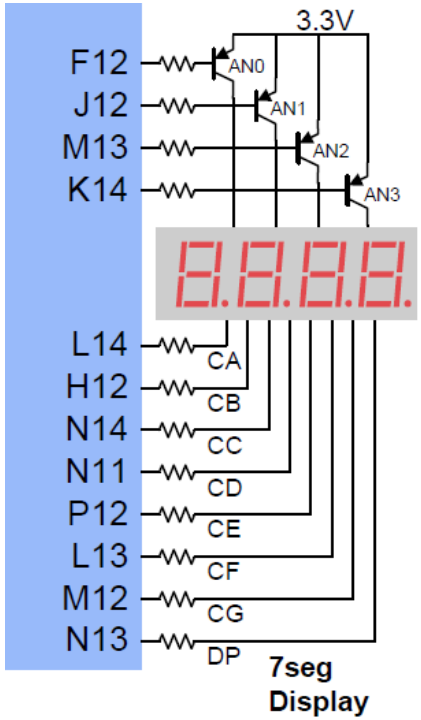
RD51 electronics school
FPGA beginner's course
Task 3 and 4

Task 3: Seven segment display

```
ROCESS(clk)
EGIN
  IF( clk'EVENT AND clk = '1' )THEN
     CASE s_mux IS
        when "00"=> an<= "1110";
                    s_char <= s_counter(3 downt
        when "01"=> an<= "1101";
```

- Increase your counter from exercise 2 to a depth of 16 bits
- Use the BASYS2 onboard seven segment display to show the value of your counter in hexadecimal reading

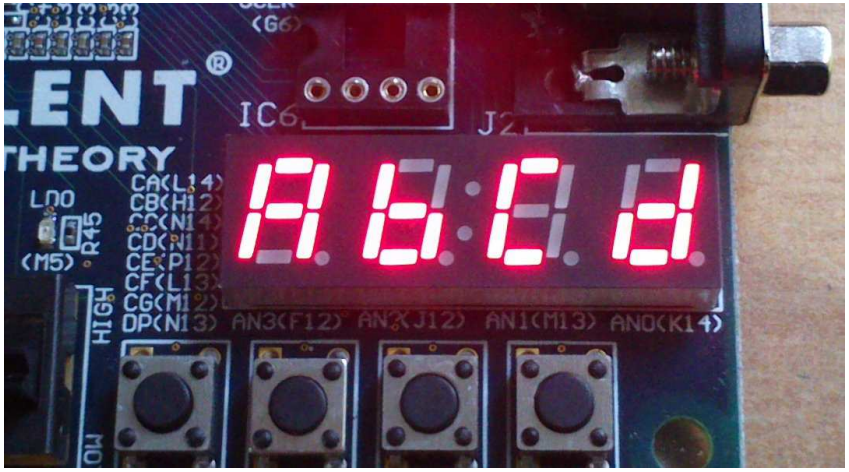




- Note that not each segment can be controlled individually at the same time
- You need to cycle through the four digits by driving only one of the anodes high at a given time (Note that a PNP transistor inverts the signal, so the FPGA needs to drive ist pins low to enable the digit)
- Drive a segment „low“ to enable the corresponding LED of the active digit

Digit multiplexing

• When the display redraw frequency is high enough (>100 Hz) no flicker is perceptible. The maximum display cycle time for a single digit is in this case 2.5 ms, but you can also try other values.




```
-------------------------------------------------
-- Encode 4-bit binary to 1-digit hex
-- note that to switch on a segment, the corresponding I/O line must be drawn to '0'
-------------------------------------------------
-- HEX-to-seven-segment decoder
-- segment encoding
--      0
--     ---
--  5 |   | 1
--     --- <------6
--  4 |   | 2
--     ---
--      3
PROCESS(s_char)
BEGIN
   CASE s_char IS
      when "0000"=>seg<= "1000000";     --'0'
      when "0001"=>seg<= "1111001";     --'1'
      when "0010"=>seg<= "0100100";     --'2'
```

Parallel processes

- Try to break down different tasks into different processes.
  - Generate a 4 bit switching signal, all ones with a sweeping zero through the different digits at ~1 kHz, using the 50 MHz onboard clock source and reasonable clock dividers. Drive the segment anodes with this signal
  - Use this signal also to decide, which digit of your number you want to display.
    Example: signal(7 downto 4) represents the second lowest hex digit of your signal
  - Encode this 4 bit value to be displayed into ist hex representation, switching on the corresponding segments. Feed this bitmask to your FPGA segment outputs
  - Use signals
    STD_LOGIC or STD_LOGIC_VECTOR(... DOWNTO ...)
    to transport data in between your different processes

Design simulation

• When you finished your design (or an intermediate step with only few processes), simulate its behaviour before flashing it into your FPGA.
Note that more complicated designs may take hours to be built, and simulation helps you find bugs much easier
=> Always simulate each of your design entities! Even the simple ones, just to be sure!

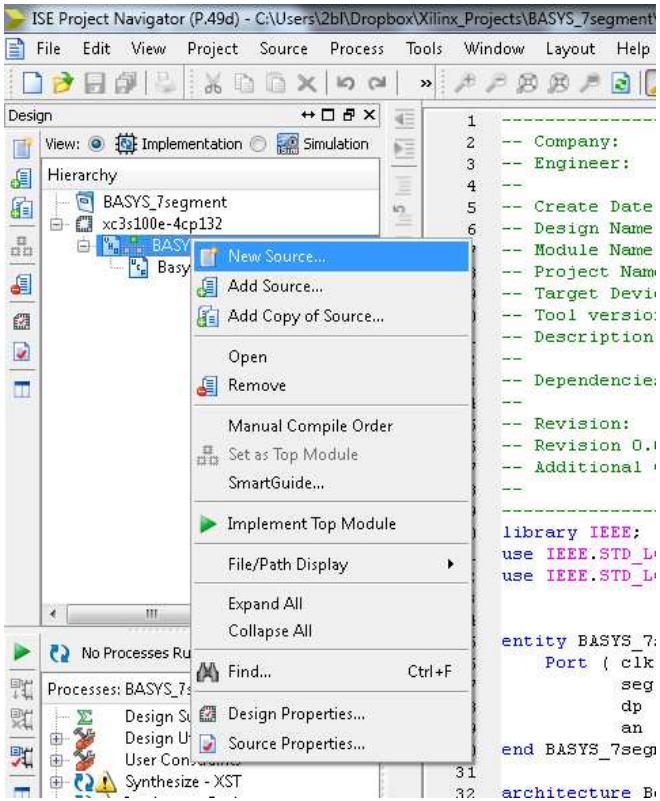
• In our case, no external stimuli apart from the clock source are required, the design runs alone without intervention from outside. In this case the testbench is simple – only the cock input needs to be simulated

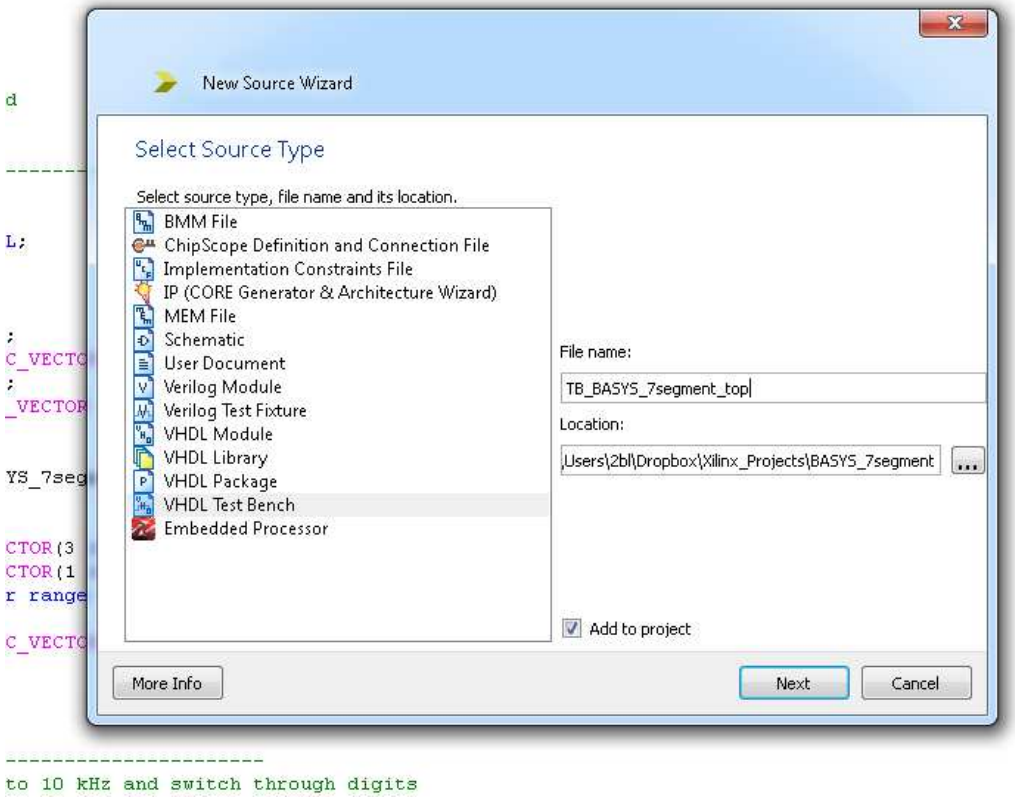
• The ISE software helps you with the generation of the testbench

• Later simulation steps during the design process also include timing simulation and signal propagation. We don‘t need this here.

Design simulation

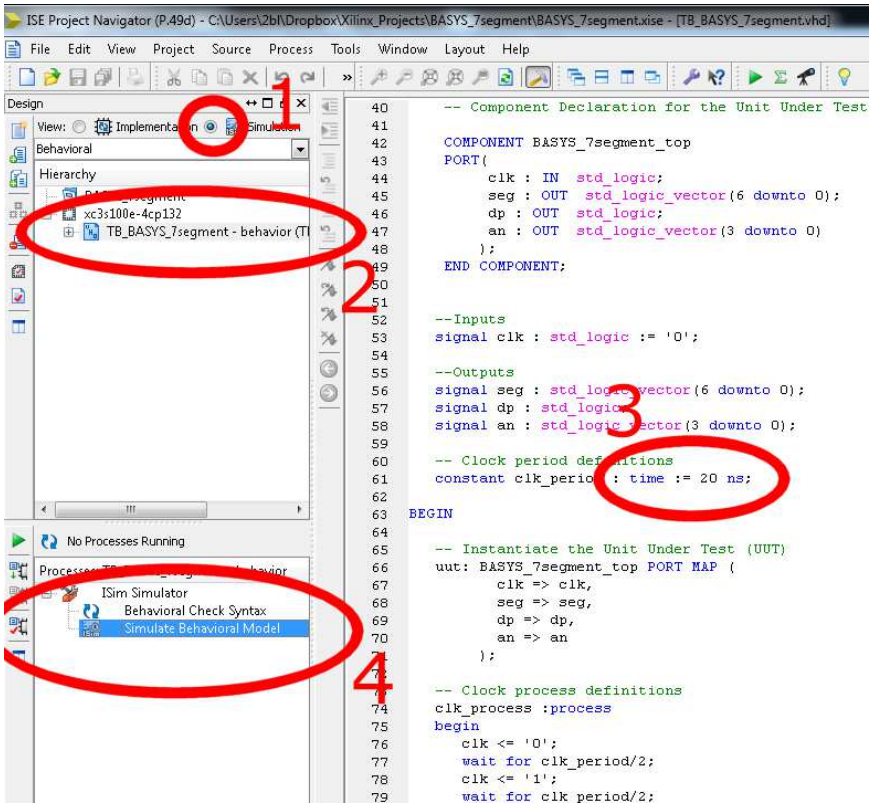
- Add a new source to your design and choose „VHDL Test bench“.
- Select your top module as target, and let the ISE software generate the test bench file for you





Design simulation

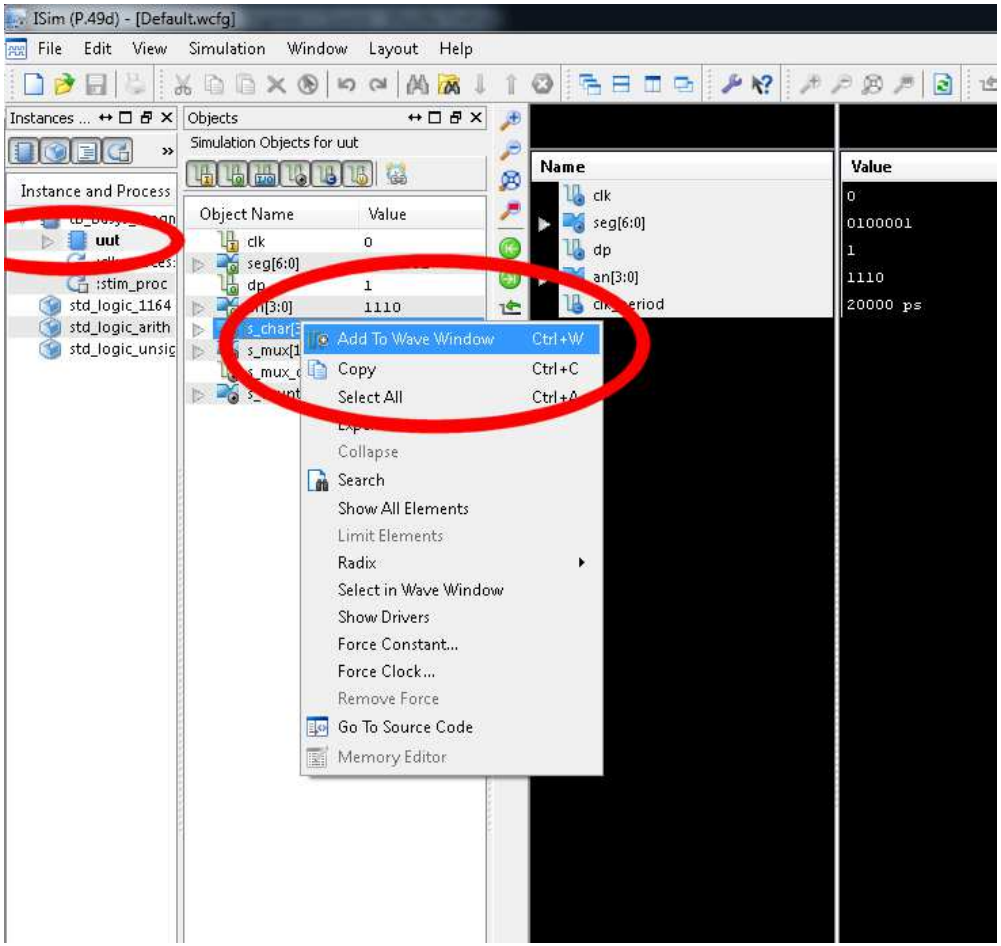
- Switch to the „Siimulation“ tab in your design overview, and edit your generated testbench file
- Change the clock period value to match the 50 MHz Oscillator on the BASYS2 Board
- This is the only modification, as no further inputs (stimuli) need to be simulated
- Start the simulation.



Design simulation

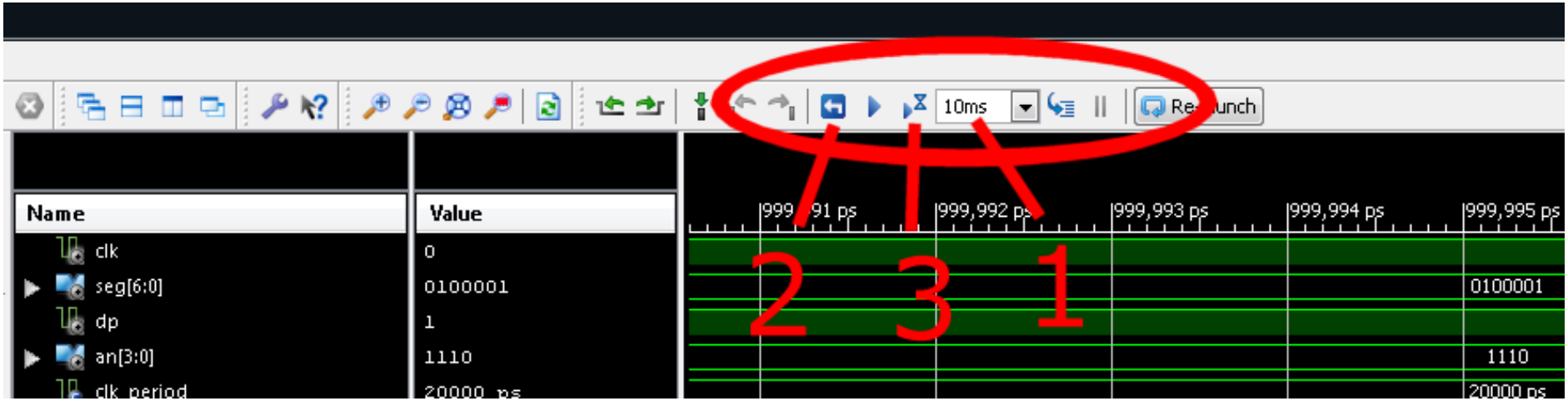
• Add additional signals to the waveform inspection list, by selecting „uut“ (unit under test) and pick the signals you want to inspect

• In our case choose the signals related to digit multiplexing and segment display



Design simulation

- Restart the simulation now including your added internal signals.
- You can then inspect each signal and port value for every single clock cycle!
- Digit multiplexing is „slow“ compared to the internal 50 MHz clock, so you need to increase the simulation time to several ms
- Now check, whether your signals behave with time as you expected



Task 4: VGA monitor output

• Use the BASYS2 onboard VGA connector to display something on a computer monitor

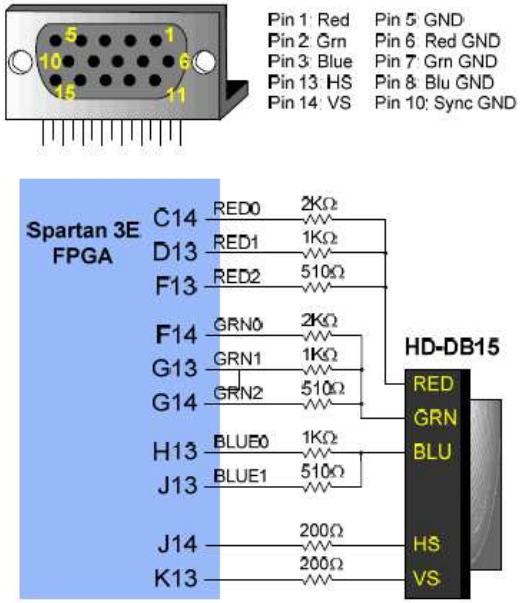


Figure 13. VGA pin definitions and Basys2 circuit

• VGA displays only need Horizontal Sync (HS), Vertical Sync (VS), and analog color information per pixel, in a certain timing sequence.

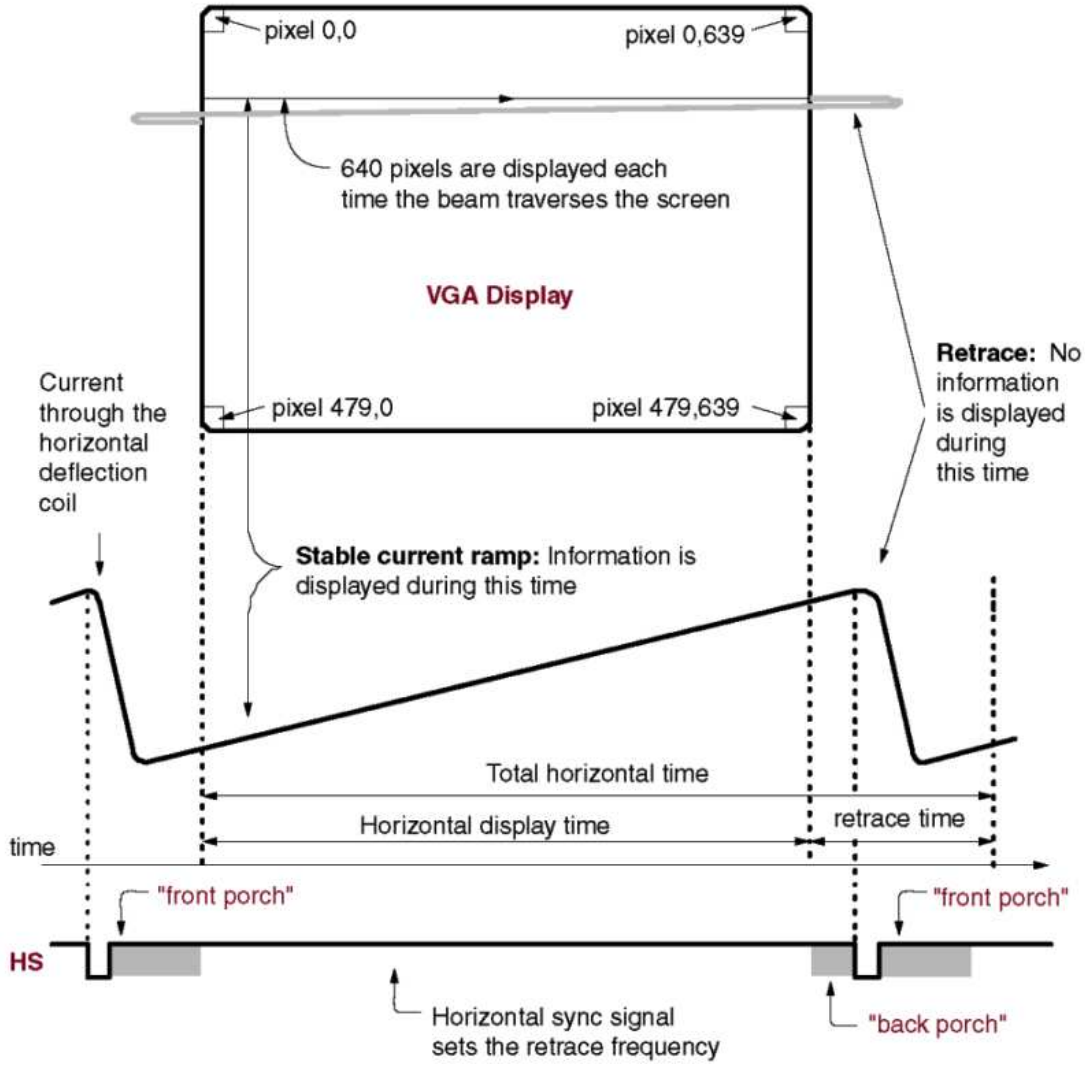
• The BASYS board uses a resistor ladder Digital-to-analog converter with 3 bit resolution for the red and green, and 2 bit for the blue channel to generate the 0V-0.7V analog signal

• For simplicity: only use the most significant bit of each color (e.g. GRN2), and tie the others (GRN1, GRN0) to zero

VGA timing

Image from http://www.cs.ucr.edu/~jtarango/cs122a_lab4.html

- VGA „Industy standard“ resolution of 640x480 pixels at 60 Hz refresh rate needs a 25.175 MHz pixel clock (close enough to 25 MHz, which we can derive from the 50 MHz BASYS2 onboard clock)
- Old Cathode Ray Tube (CRT) monitors do not only scan over the visible are of the screen. Additional lines and pixels are drawn with „invisible“ sync information within the area „outside“ the visible screen.
- Modern LCD screen adopted this scheme for compatibility
- Drawing of a frame starts on the upper left corner, when looking onto the screen



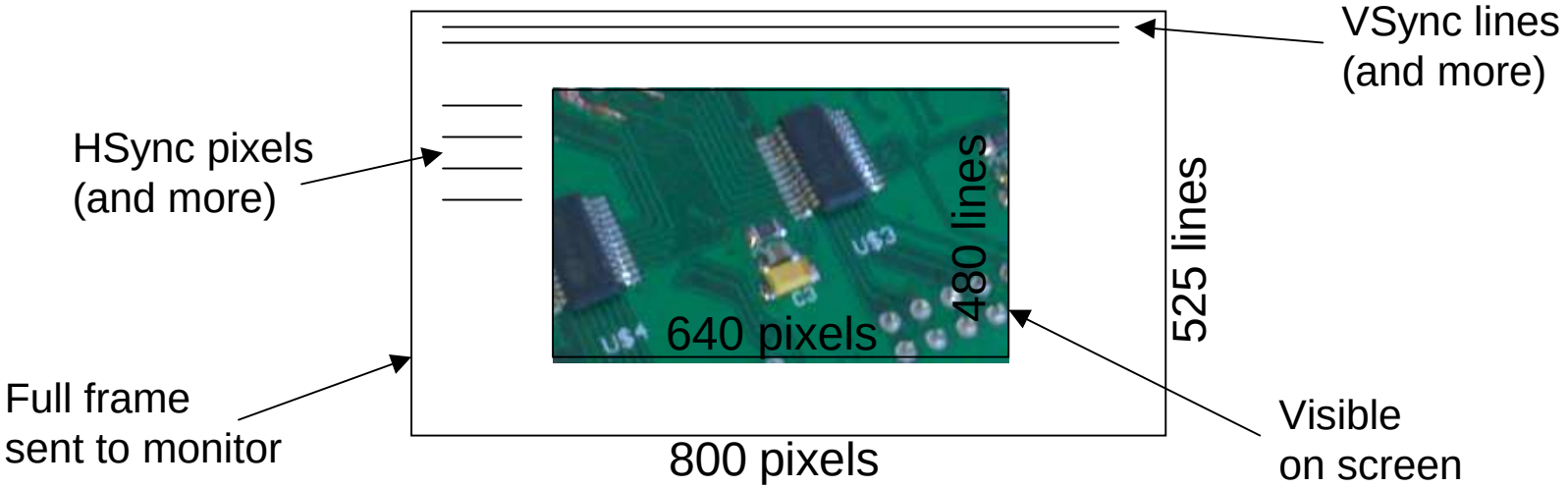
VGA timing

• You will need two counters (horizontal and vertical) to know at every time where on the screen and in which part of the image drawing cycle. These counters determine what your „graphics card“ needs to to (display a pixel, display nothing, generate a HS or VS signal, etc)

• Both sync signals (HS, VS) are inverted (active low), so they are kept „high“ normally, and pulled „low“ during sync!

• One line is composed of 800 pixels, with HSync pulse (low) during pixels 8 to 103 and image information only during pixels 152 to 791 (counting from 0)

• One image is composed of 525 such lines, with VSync pulse (low) during lines 2 and 3, and image information only during lines 37 to 516 (counting from 0)



Implementation

• Write a process to toggle a 25 MHz STD_LOGIC signal at each 50 MHz clock. This is now the pixel clock that drives your other processes.

• Increase the horizontal counter with each pixel clock cycle

• When your horizontal counter reaches its maximum value (e.g. 0 to 799), reset it to zero and increase the vertical counter by one

• When the vertical counter reaches its maximum value (e.g. 0 to 524) reset it to zero. Your image cycle is now complete.

• Use „if [...] AND [...]“– statements on your counters to check, whether you must assign HS within a line or VS during lines

• Use similar statements to check, whether you are allowed to assign signals to the Red, Green and Blue outputs

• The Red, Green and Blue outputs must be zero outside the „visible“ area of 640x480 pixels

• You can use certain bits of the horizontal and/or vertical counters (or their logical combinations) to drive the color outputs to generate different patterns on the screen. For a simple start – turn the complete screen red for example.

Hints

```
-- generate a 25Mhz clock by to
process (clk50)
begin
   if clk50'event and clk50='1'
      if (clk25 = '0') then
         clk25 <= '1';
      else
```

- 25MHz clock generation

```
if (horizontal_counter >= "0000001000" ) -- 8
   and (horizontal_counter < "0001101000" ) --
then
   HSYNC <= '0';
else
```

- Hsync timing

```
then
   OutRed <= horizontal_counter(3)
      and vertical_counter(3);
   OutGreen <= horizontal_counter(4)
      and vertical_counter(4);
```

- Content generation from counters
  -> „colored boxes“