

Synthesis Tutorial

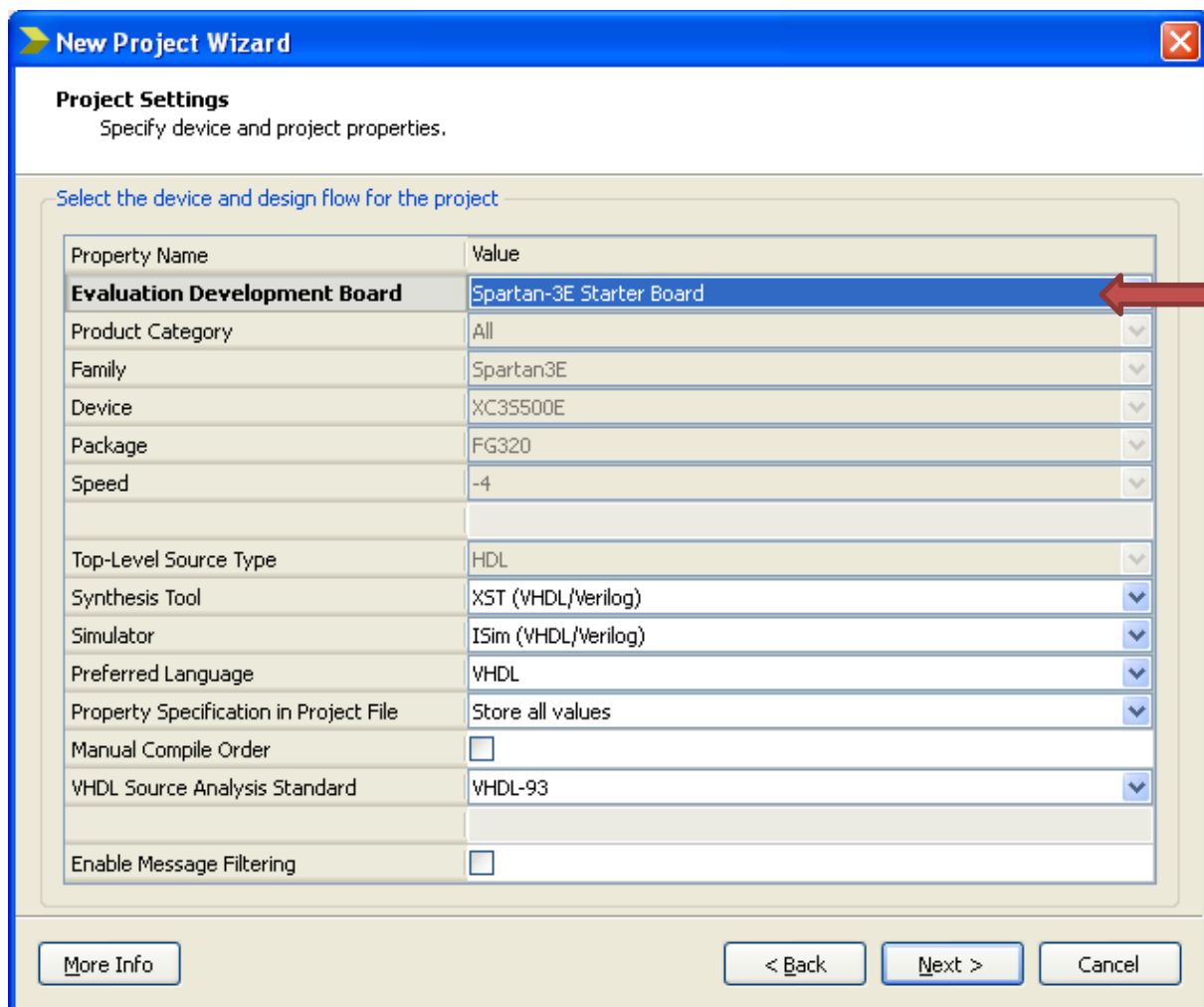
This tutorial describes the workflow of synthesizing a VHDL design to a target hardware platform (the FPGA), and the basics of interfacing with external components on the FPGA board.

For this example, we want to make one of the LEDs on the FPGA board blink. The files `led.vhd` and `led.ucf` used in this tutorial, as well as the Spartan-3E Starter Kit Board User Guide can be downloaded from the course website.

Create a Xilinx ISE project

To synthesize a design, you must import all of the VHDL sources into a new Xilinx ISE project. To do so, start Xilinx ISE and select "*File > New Project ...*" Choose a name and location for your project and click "*Next*" (Top-level source type is "HDL").

In the Device Properties dialog, make sure all options are set as shown here:



New Project Wizard

Project Settings
Specify device and project properties.

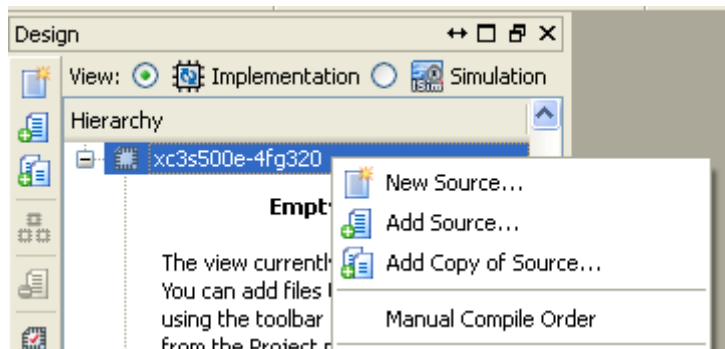
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	Spartan-3E Starter Board
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

Click "*Next*" and then "*Finish*" until the project is created.

To add files to the project, right-click on the top-level entry in the "Sources"-tab and choose either "Add Source..." or "Add Copy of Source..." to add all of your VHDL files to the project.



In this tutorial, we have only one VHDL file, `led.vhd`:

```
1  -- led.vhd
2  library ieee;
3  use ieee.std_logic_1164.ALL;
4  use ieee.numeric_std.ALL;
5
6  entity led is
7      port (
8          clk          : in std_logic;
9          reset        : in std_logic;
10         status_led   : out std_logic;
11         led0          : out std_logic;
12         led1          : out std_logic;
13         led2          : out std_logic;
14         led3          : out std_logic;
15         led4          : out std_logic;
16         led5          : out std_logic;
17         led6          : out std_logic );
18 end led;
19
20
21 architecture Behavioral of led is
22     signal count      : unsigned (25 downto 0) := (b"00" & x"000000");
23     signal led_int     : unsigned (3 downto 0) := x"0";
24 begin
25
26     process(clk)
27     begin
28         if rising_edge(clk) then
29             -- unsigned overflow -> reset to zero
30             count <= count + 1;
31             if reset = '0' then -- leds off while reset, but counter continues running
32                 led_int <= "0000";
33             else
34                 led_int <= count(25 downto 22);
35             end if;
36         end if;
37     end process;
38
39     status_led <= '1';
40     led0 <= led_int(3);
41     led1 <= led_int(3) and led_int(2);
42     led2 <= led_int(2);
43     led3 <= led_int(2) and led_int(1);
44     led4 <= led_int(1);
45     led5 <= led_int(1) and led_int(0);
46     led6 <= led_int(0);
47
48 end Behavioral;
```

User Constraints

To connect the inputs and outputs of our top-level entity ("clk" and "ledX") to pins of the FPGA, we need to create a so-called "user constraints file" (ucf). We must specify voltage levels and other properties for all pins we want to use. For clock inputs, we also need to specify the timing of the external clock signal.

Note: It is very important to always specify the correct voltage levels for all output pins; otherwise the components connected to these pins could be damaged!

To find out how the pins of the FPGA are connected on the Board, consult the "Spartan-3E Starter Kit Board User Guide".

Clock period constraint

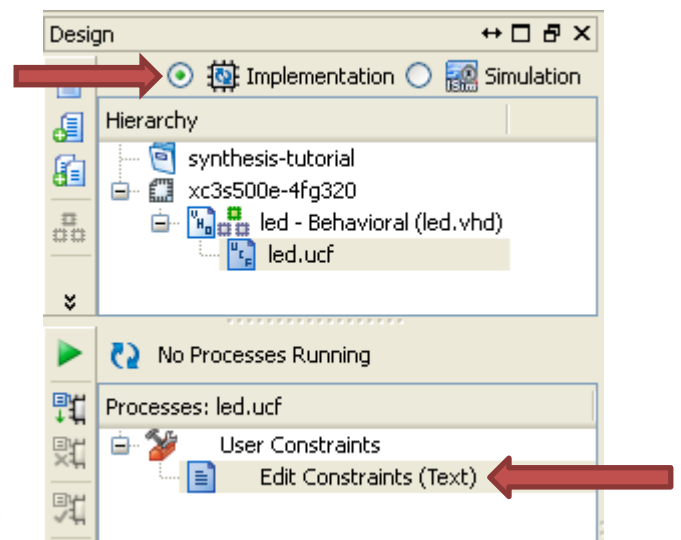
Our "clk" input is going to be connected to the external clock. The FPGA board has a 50 MHz oscillator which we will use as the clock input. To tell the Synthesis tool about this clock, we need to include the following entry in the constraints file (this line can be found in the User Guide):

NET "clk" PERIOD = 20.0ns HIGH 50% ;

Location Constraints

Here we connect the "clk" signal to the external clock input and the "ledX"-signals to the output pins connected to LED0 to LED7 (the required lines can be found in the User Guide)

Add the ucf-file that is provided with the vhd-file to your project by right-clicking on the top-level VHDL file and choosing "Add Source". The UCF file should appear under your VHDL file. You can edit it later by selecting the file in the "Sources"-tab and choosing "User Constraints > Edit Constraints (Text)" in the "Processes"-tab.



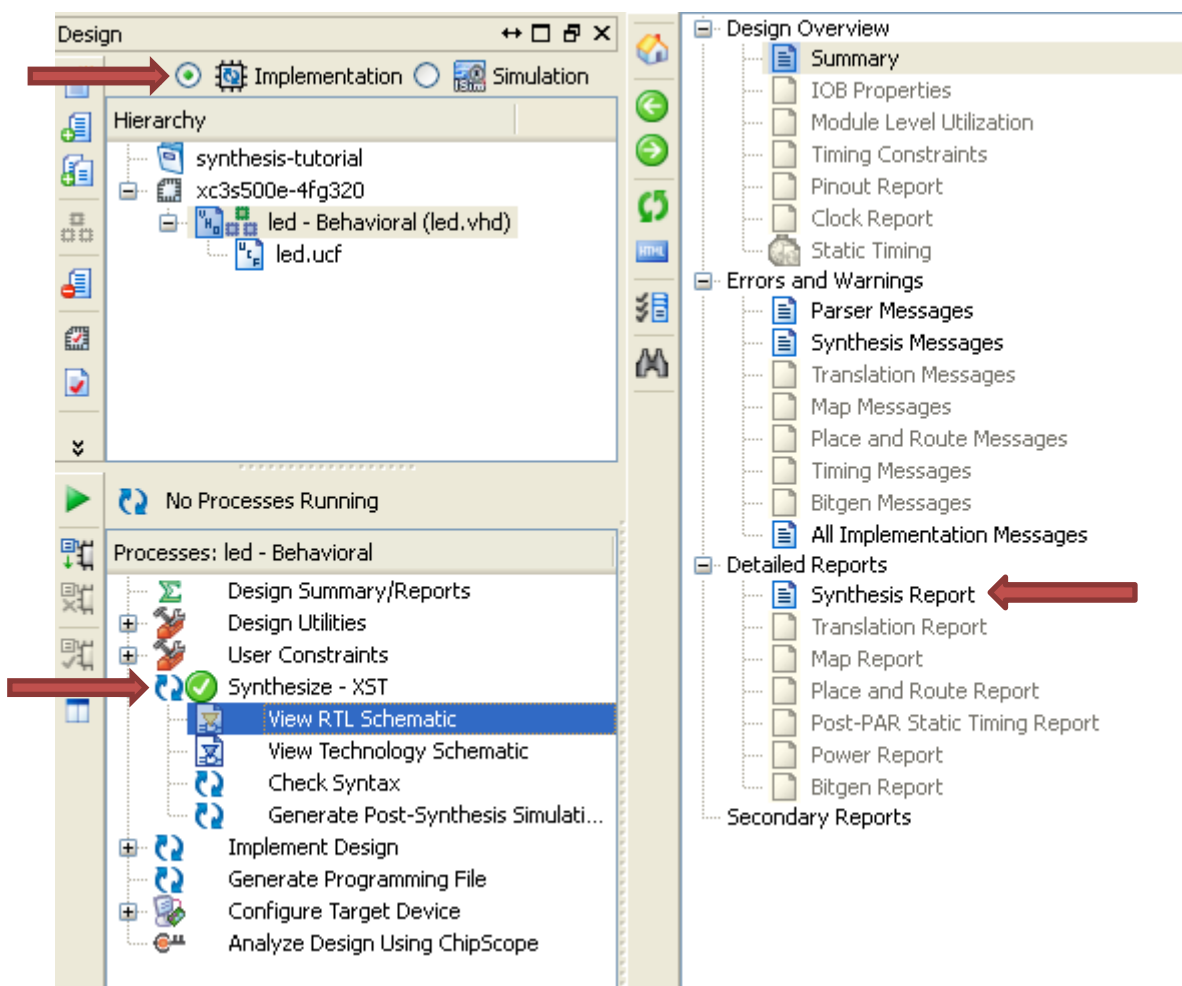
```
1 NET "clk"          PERIOD = 20 ns HIGH 50% ;
2 NET "clk"          LOC = "C9" | IOSTANDARD = LVCMOS33 ;
3
4 NET "led0"          LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
5 NET "led1"          LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
6 NET "led2"          LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
7 NET "led3"          LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
8 NET "led4"          LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
9 NET "led5"          LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
10 NET "led6"          LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
11 NET "status_led"   LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
12 NET "reset"        LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP;
13
```

Synthesis

Select the top-level VHDL file in the "Sources"-tab. You will now see that the "Process"-tab lists four processes called "Synthesize", "Implement Design", "Generate Programming File" and "Configure Target Device". All of these steps need to be run in order to synthesize the design and program the FPGA.

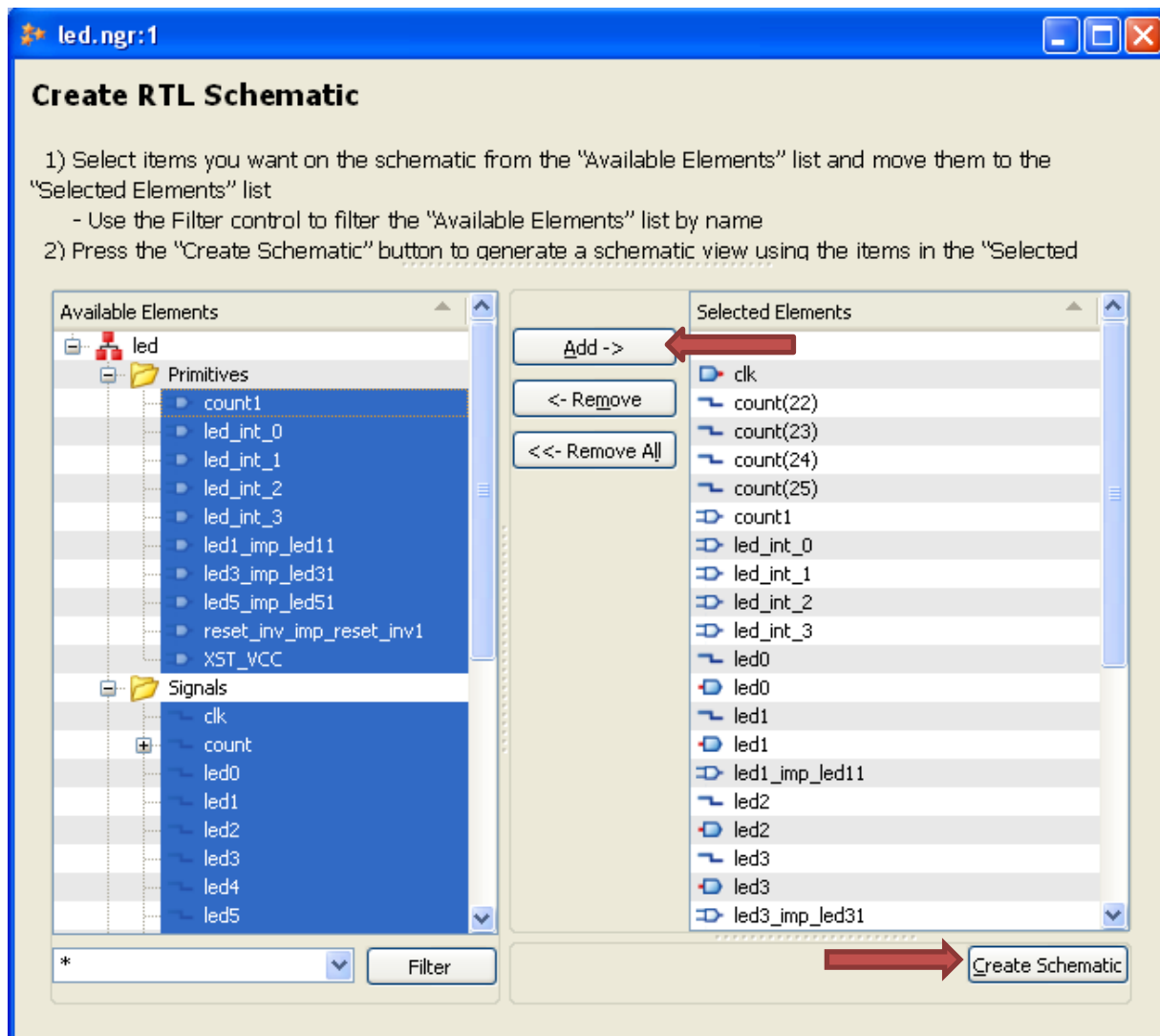
Double-click on "Synthesize". The VHDL file will now be compiled. If the compilation fails, the errors will be listed in the "Errors"-tab at the bottom of the screen.

The "Design Summary"-tab shows detailed information about the synthesized design, such as the number and types of devices used (such as flip-flops, lookup tables, etc.), and timing information such as the critical path and the maximum clock frequency. Even further detailed information can be found in the synthesis report, which is found under "Detailed Reports".



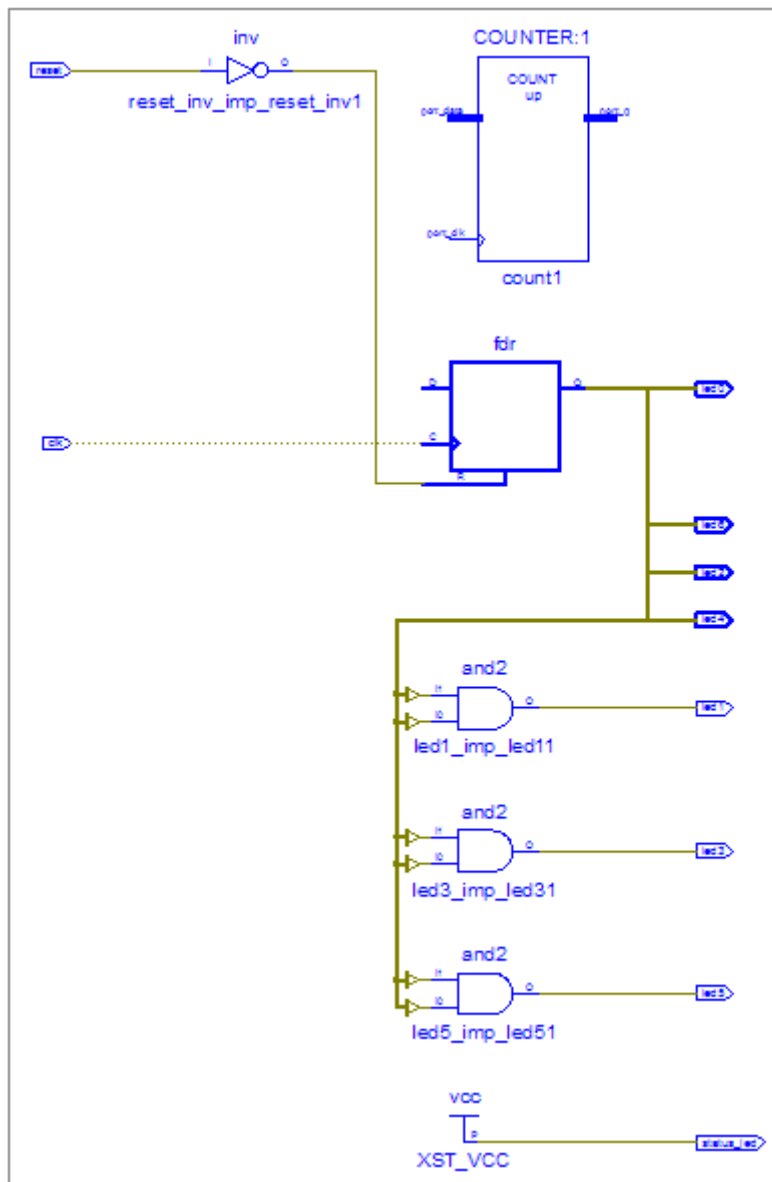
To view the RTL-level structure that was generated, double-click "View RTL Schematic" (after expanding the "Synthesize"-Process by clicking the "+" next to it).

Due to a bug of the current version you have to choose "Start with the Explorer Wizard" in the following dialog. Add all available elements except for the top level block "led".



The design is implemented as a 26 bit up-counter. Since the overflow of such an element results in the value 0, no reset logic is required. The 4 MSBs of the counter are stored in the register fde, the output of which is connected to the outputs of the entity itself, either directly or via combinatorial gates.

The element that is affected by the current bug is the counter. All its ports are shown to be not connected and the actually present reset input is not depicted. The missing connections are indicated by the dotted signal lines "clk" and "reset".

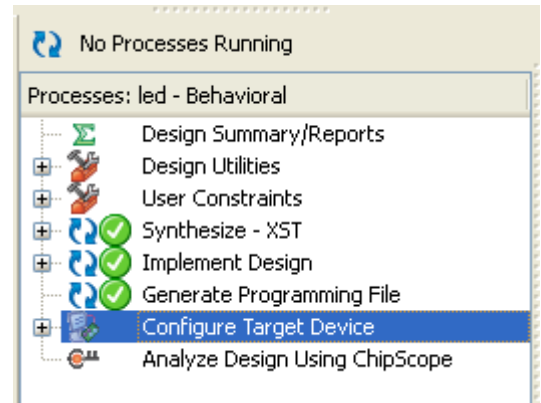


Implement Design

Next, the synthesized design must be placed on the FPGA. To do this, double-click on the "Implement Design" Process.

Generate Programming File

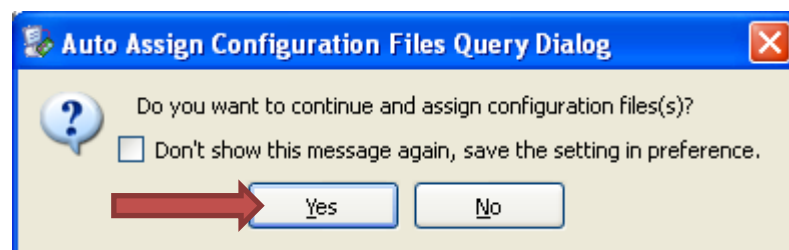
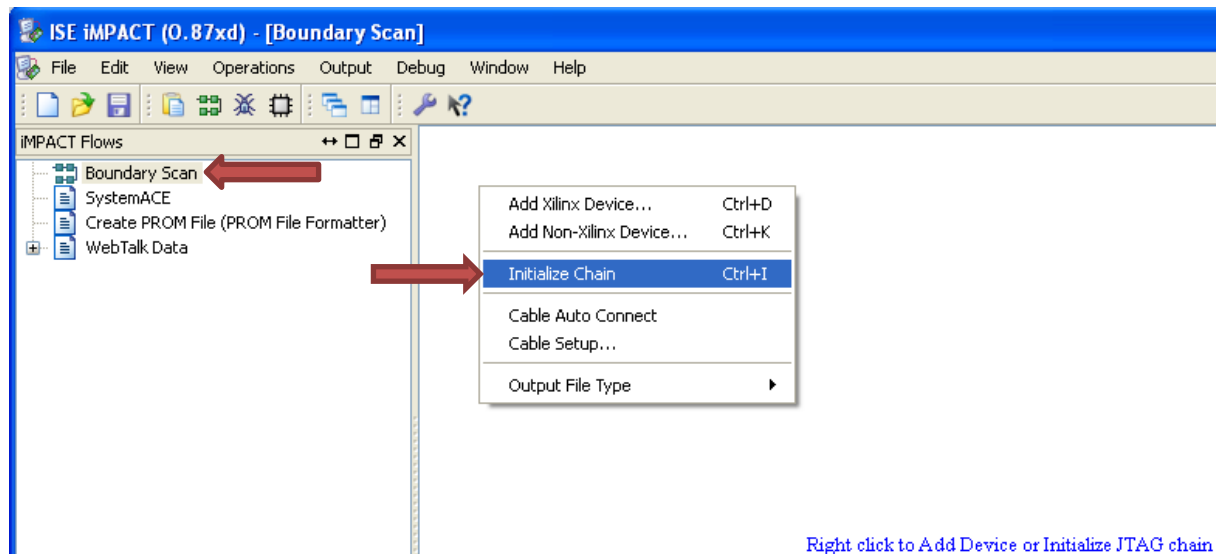
Next, double-click on "Generate Programming File". This process generates a file suitable for downloading to the FPGA. It will be named like your project with a ".bit" extension.



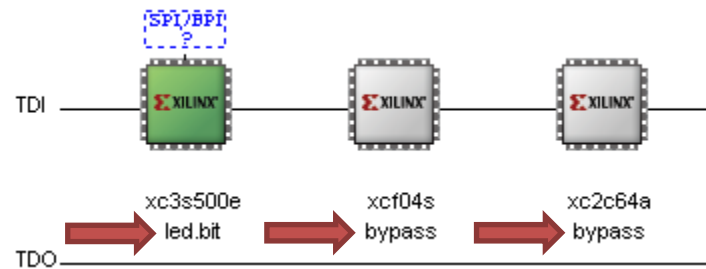
Configure Target Device

In this step, the connection to the FPGA via the USB programming cable is configured, and the file generated in the last step is downloaded into the FPGA.

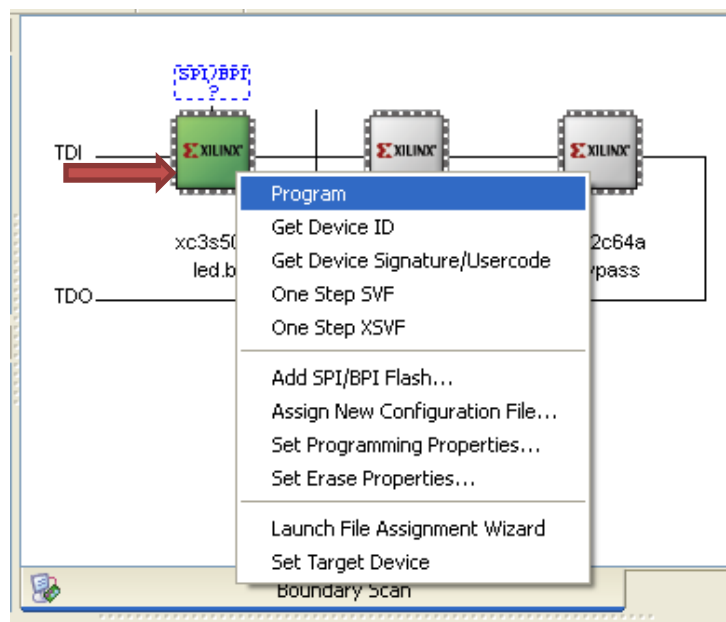
Make sure the USB cable is connected to the FPGA board, and then double-click "Configure Target Device". You can ignore the warning about a missing iMPACT file. In the following window, double-click "Boundary Scan". Choose "Initialize Chain" in the context menu to connect to evaluation board, which has to be switched on and recognized by your operating system. When asked if you want to assign configuration files, click "Yes".



After the connection to the programming cable has been established, you can configure the programming process, which is done via boundary scan. There are three devices on the FPGA board in the scan path. You will be asked to choose an action for every device. The first one, labeled "xc3s500e" is the FPGA. It is highlighted in green. In the dialog window, select the bit-file (led.bit) and click "Open". Now the second device will be highlighted. We don't want to program this device, so click "Bypass". Also select "Bypass" for the third device. In the "Device Programming Properties" dialog, keep all settings and select "OK".



When all three devices are configured, you can start the actual programming process by right-clicking on the device "xc3s500e" and selecting "Program".



When programming is finished, you should see LED07 glow on the FPGA board. SW0 can be used to activate the blinking pattern.

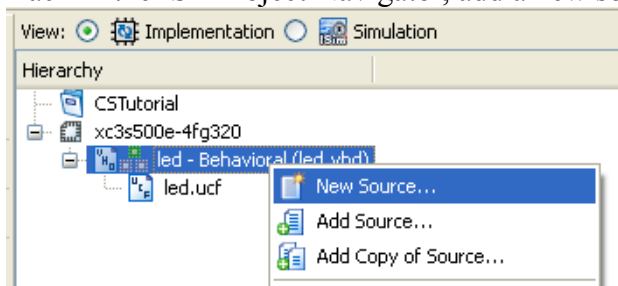
Congratulations, you have synthesized your first VHDL design! ☺

Using Chipscope to analyze your design

Xilinx ISE provides a powerful tool called "Chipscope". It can be seen as an on-chip logic-analyzer which allows you to look into your design while it is running on the FPGA. This can help a lot when you are trying to identify problems which did not occur during simulation, or which are not reproducible in simulation (e.g. when they involve non-ideal behavior of peripherals, like bouncing buttons or glitches).

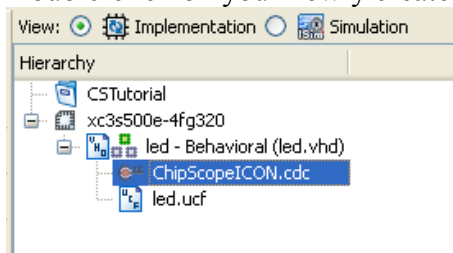
In this part of the tutorial, we are going to analyze the simple LED-example from above. We want to record the 7 LED lines, as well as the reset input.

Back in the ISE Project Navigator, add a new source to your project.

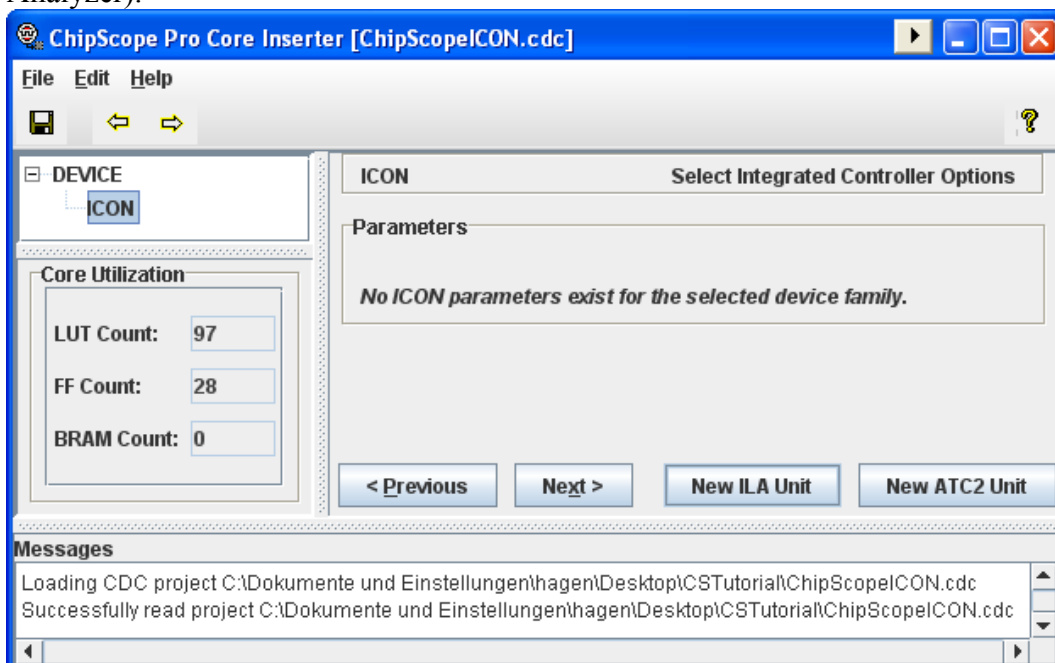


Select "ChipScope Definition and Connection File", give it a name, and click "Next" and then "Finish".

Double-click on your newly created file to open the ChipScope Core Inserter.



Select the ICON (=Integrated Controller), and click "New ILA Unit" (=Integrated Logic Analyzer).



Select the new ILA-Unit on the left, and configure it as shown here:

The image displays two screenshots of the Xilinx IDE's 'Select Integrated Logic Analyzer Options' dialog, showing the configuration of a new ILA unit.

Left Panel (Project Hierarchy): The 'DEVICE' tree shows 'ICON' selected, with 'U0: ILA' highlighted.

Right Panel (Configuration):

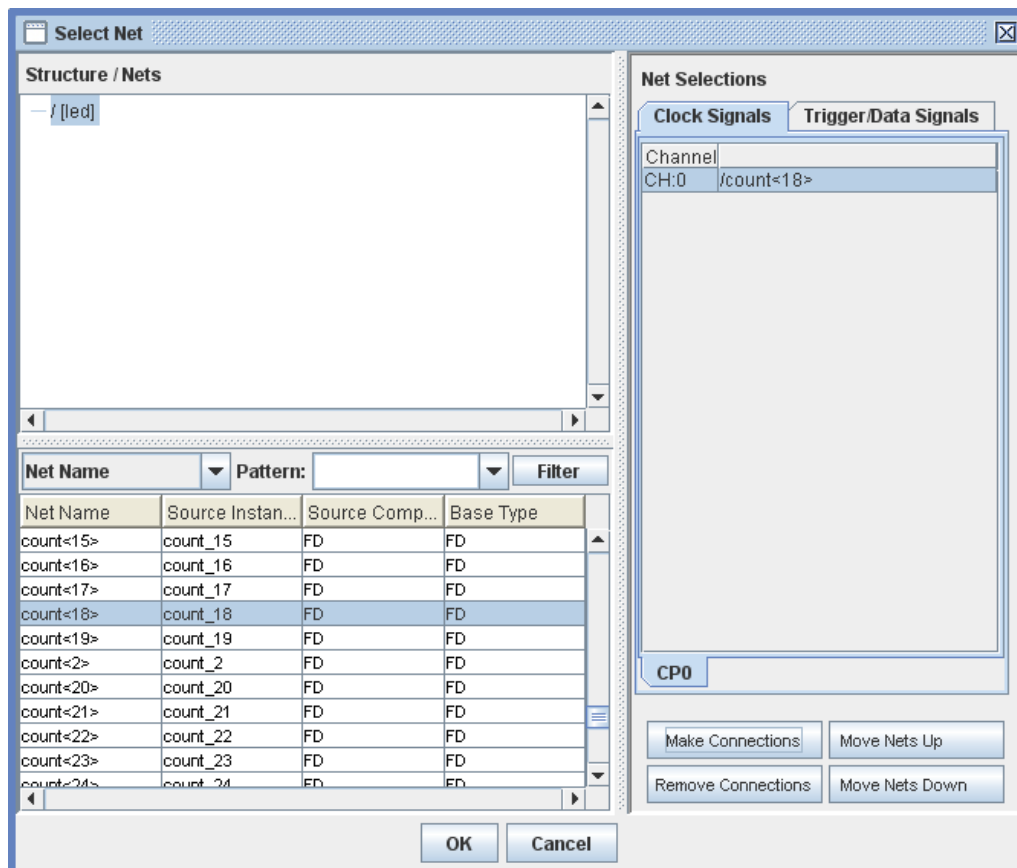
- Core Utilization:** LUT Count: 286, FF Count: 209, BRAM Count: 1.
- Trigger Parameters Tab:**
 - Trigger Input and Match Unit Settings:**
 - Number of Input Trigger Ports: 1
 - Number of Match Units Used: 1
 - TRIG0: Trigger Width: 8, Match Type: Basic w/edges, # Match Units: 1, Counter Width: Disabled, Bit Values: 0, 1, X, R, F, B, N, Functions: =, <>
 - Trigger Condition Settings:**
 - Enable Trigger Sequencer: ☐ (Max Number of Sequencer Levels: 16)
 - Storage Qualification Condition Settings:**
 - Enable Storage Qualification: ☐
- Capture Parameters Tab:**
 - Capture Settings:**
 - Sample On: Rising (Clock Edge)
 - Data Depth: 512 Samples
 - Data Same As Trigger: ☒
 - Trigger Ports Used As Data:
 - Include TRIG0 Port (width=8): ☒

Navigation buttons: < Previous, Next >, Remove Unit.

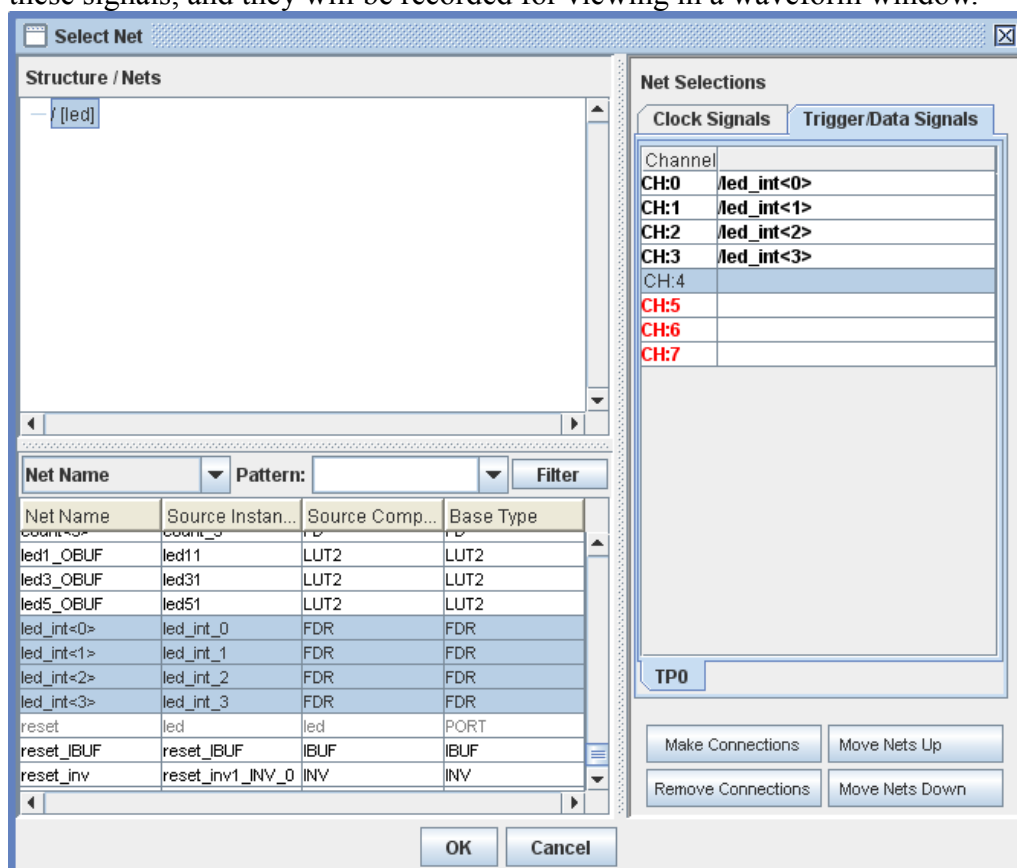
In the tab "Net Connections" click on "Modify Connections".

To select the clock signal for our logic analyzer, find the signal "count<18>" in the left signal list¹. Click on "Make Connections".

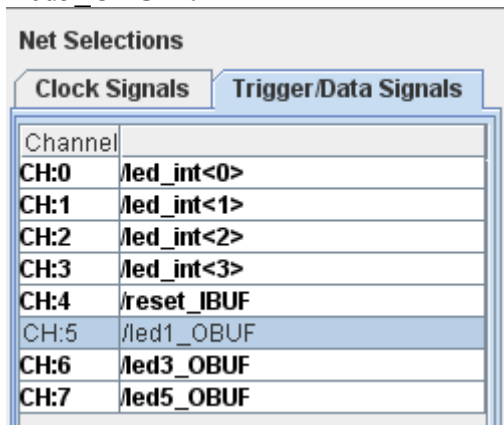
¹ We are using a very slow clock here, because our design is very slow (otherwise you wouldn't see the LEDs blinking). In most designs, you would use your system clock here. Use the buffered version of the clock in that case (i.e. "clk_BUFGP" instead of "clk").



Switch to the "Trigger/Data Signals"-tab in the right hand side of the window. Select signals "led_int<0>" to "led_int<3>" (using Shift or STRG to select multiple signals), and click on "Make Connections". Later on, we will be able to configure trigger conditions on these signals, and they will be recorded for viewing in a waveform window.



Following that procedure, connect "reset_IBUF", "led1_OBUF", "led3_OBUF" and "led5_OBUF":



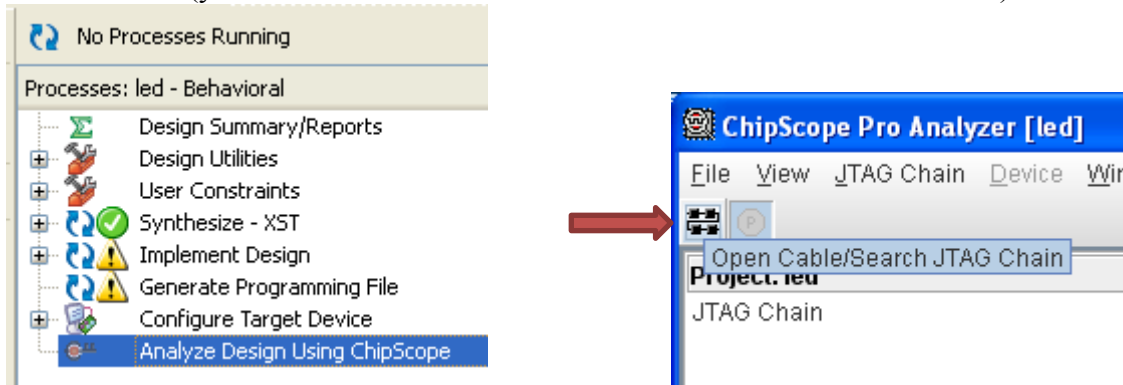
Hint: If you want to use the filter-capabilities in the signal-list, you need to use wildcards. I.e. trying to filter for "led" will not yield any results, while "led*" will.

Close the window by clicking "OK".

Back in the ChipScope Core Inserter, click on "Return to Project Navigator". Confirm to save the project.

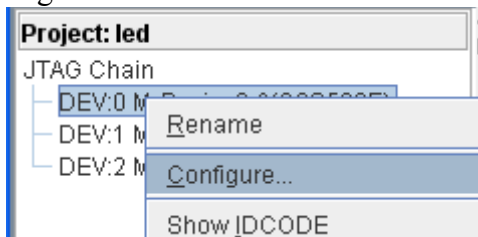
Now, you need to re-implement your design. This will now take longer because the ChipScope core is additional hardware which needs to be implemented. It will also generate some warnings which you can safely ignore.

Afterwards, click on "Analyze Design Using ChipScope", and then on "Open Cable/Search JTAG Chain" (your FPGA-board needs to be connected and switched on now):



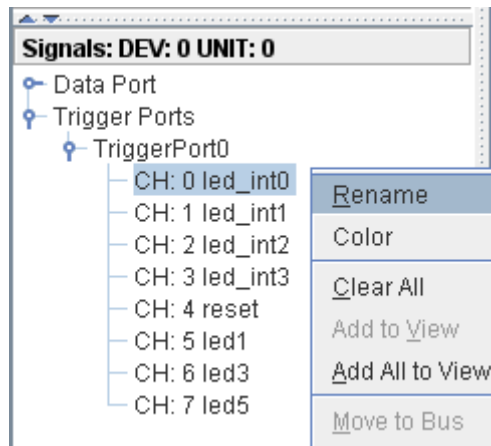
Confirm the appearing window about the JTAG chain device order with "OK". Note that the FPGA is the first device in this chain.

Right click on the FPGA-Device and select "Configure".

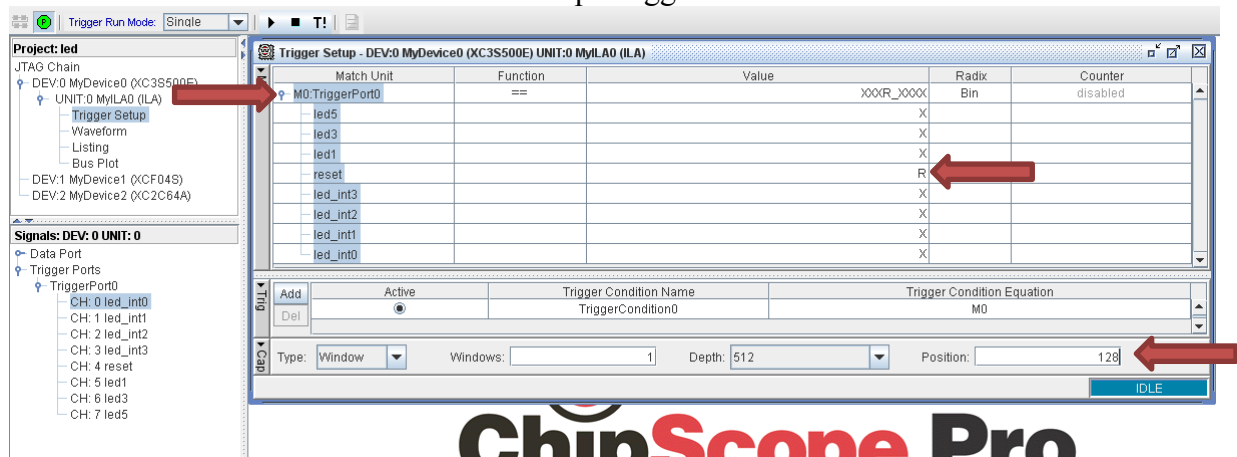


Make sure the right bit-file is selected (led.bit) and confirm the dialog with "OK".

In the "signals"-area, you can rename your signals. It's highly recommended to choose descriptive names in order to simplify the analysis of your design. The signal association is the same you chose earlier, while creating the chipscope core. The screenshot shows the association used in this tutorial:




Double-click on "Trigger Setup". Expand TriggerPort0 by clicking on the circle symbol. Set the "Value" of "reset" to "R" (=rising edge)². Set "Position" to 128. This is the size of the pretrigger.




Click on the Play-Symbol  in the Toolbar. This arms the logic analyzer.

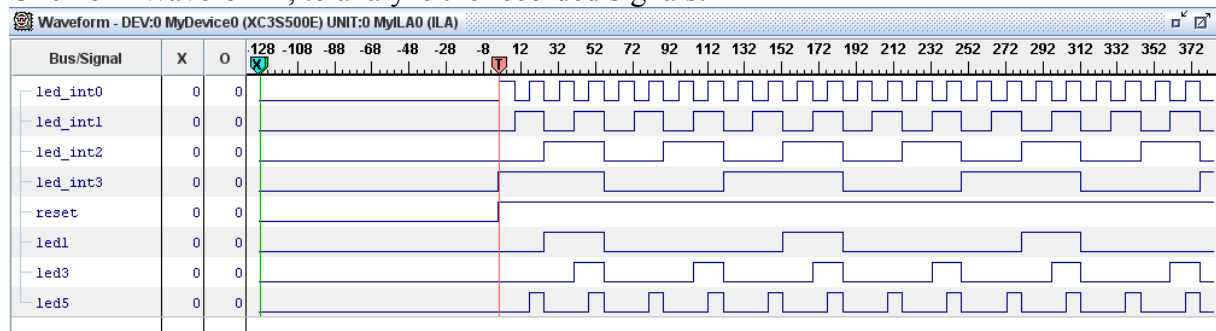
You are informed that the pretrigger is full, and the system is waiting for a trigger condition:

 Waiting for trigger. Sample Buffer has 128 samples (25%)., slow or stopped clock

When you generate a trigger-condition by setting reset=1 (SW0), you can see the following message:

 Capture started. Sample Buffer has 208 samples (40%)., slow or stopped clock

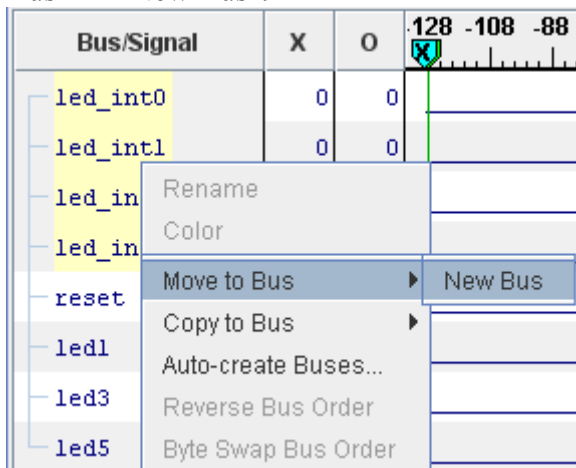
Click on "Waveform", to analyze the recorded signals:



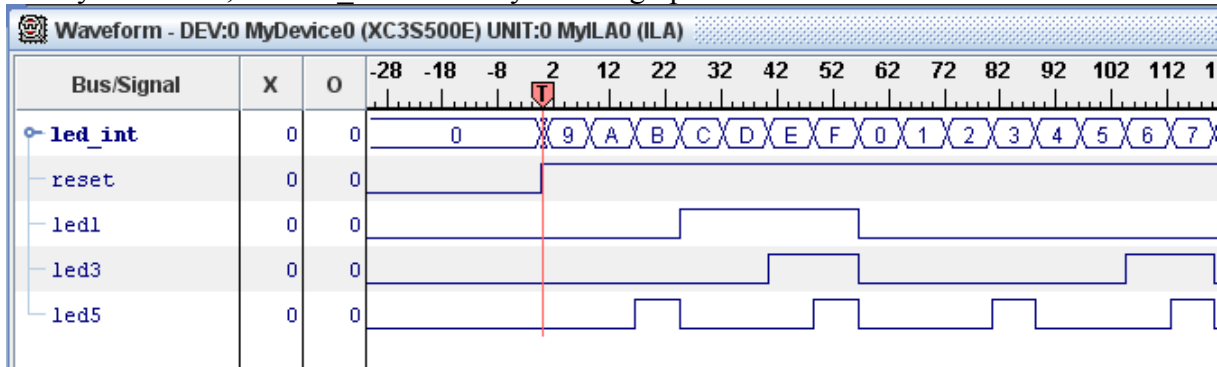
If you want, you can combine signals led_int0 to led_int3 as a bus. This is helpful in many

² Possible trigger values are 0, 1, X (don't care), R (rising edge), F (falling edge), B (both edges), N (no edge).

situations. To do so, select the signals (using Shift or STRG), right click and select "Move to Bus" → "New Bus".



Now you can see, that led_int is actually counting up:



ChipScope has a lot more features than those covered here. Try to experiment a bit with it using this simple design, it might help you a lot with the more complex design you are going to implement during this course.