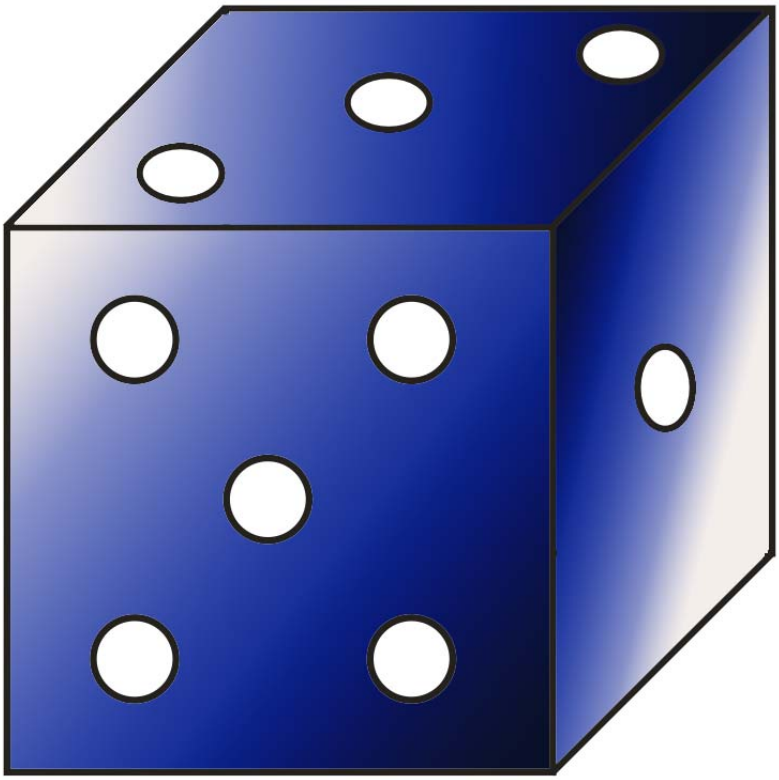
Design, simulation, synthesis and implementation in Active-HDL



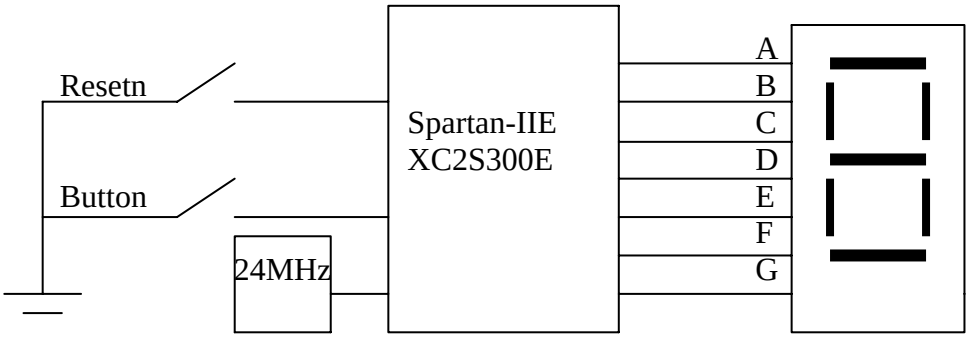
DICE

Purpose
The purpose of this tutorial is to show how to use Active-HDL for **design** and **simulation**.

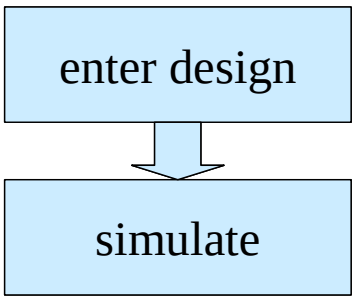
The task
A simple electronic die shall be implemented. It consists of a 7-segment LED display, a random button, and a reset button. These components are connected to a FPGA chip from Xilinx (Spartan2E XC2S300E-5FG456C). Everything is mounted on a development board. The LEDs and all buttons on the board are **active low**. The 7-segment displays are **active high**.

The function of the die

When the reset signal is low, the display shall show the value 0 and the random button should not have any effect. When the reset signal goes high, the display should still show 0, but if the random button now is pushed the display should show digits between 1 and 6 (in a high frequence). When the button is released, a number between 1 and 6 should be shown.



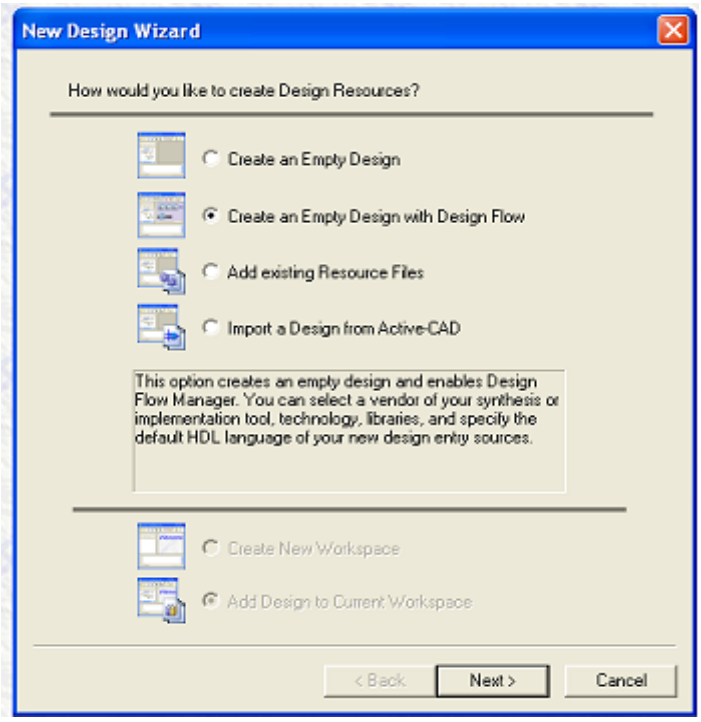
The design consists of two blocks. One block generates random numbers, and one block shows digits on the display. The display block is an ordinary BCD to 7-segment decoder (0-9, A-F), and the random block is a counter, counting 1, 2, 3, …
When the counter in the random block reaches 6, it restarts at 1 and continues. The counter is clocked at 24 MHz, so the number shown when the button is released could be considered a random number.



The flow.

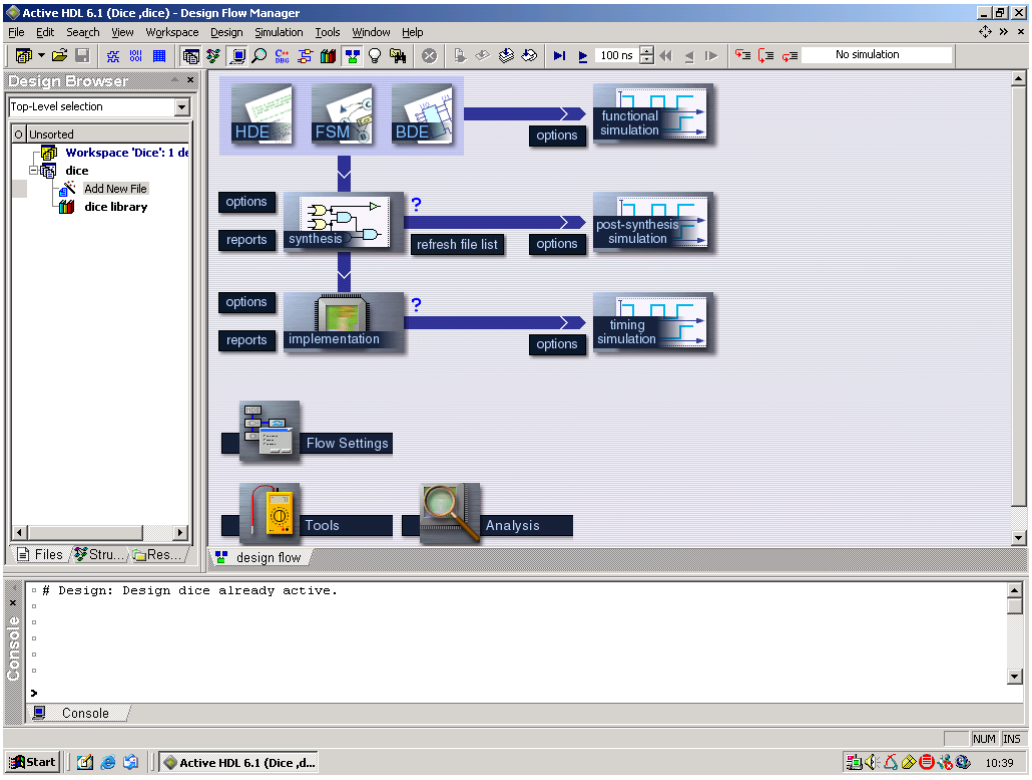
A. Design entry

Start Active-HDL. *"Create new workspace"* and type in "*workspace name*", e.g. "*dice*". Create thereafter a new design, *"Create an Empty Design with Design Flow"* [Next].

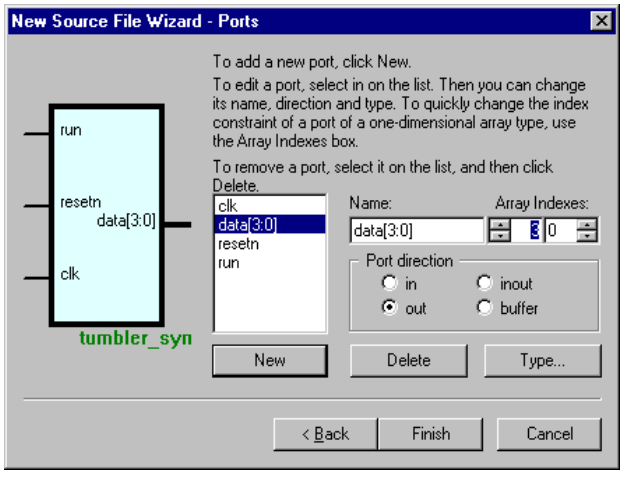


After this, the tools to be used are specified (Flow Settings), e.g.,
Synthesis tool:*"Synplicity Synplify Pro 7.x"*
Implementation tool:*"Xilinx ISE 6.1"*.
These programs will automatically be located and added.
The circuit "*Xilinx6x SPARTAN2E"* , which is used here, is chosen in "Default Family".
[Next]

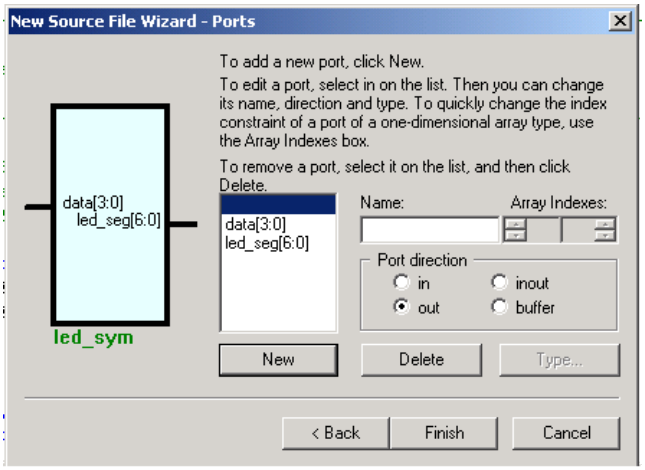
Type in the name and the location of your project. Here we call the design *"dice"*. It is an advantage to have the files on the local hard disk C:\, while working on the project (faster compilation etc.). Save in C:\my_designs\student_user\dice. Move to H:\ and delete the files from C:\ when you are ready.



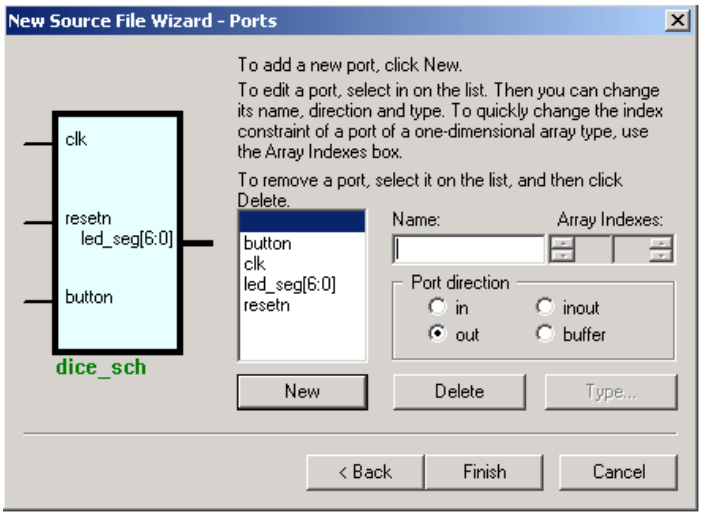
You will now get your working window with a file menu to the left, and a status window at the bottom.
We shall now create the first part of the project. Click on ⊕ and choose *"Add new file"* in the *Design Browser* to the left. Choose the tab "Wizards". Then click on the icon *"VHDL Source Code"*. [OK]. [Next]. Here we will name our first block. Call it *"tumbler_sym"* in the upper box. [Next].



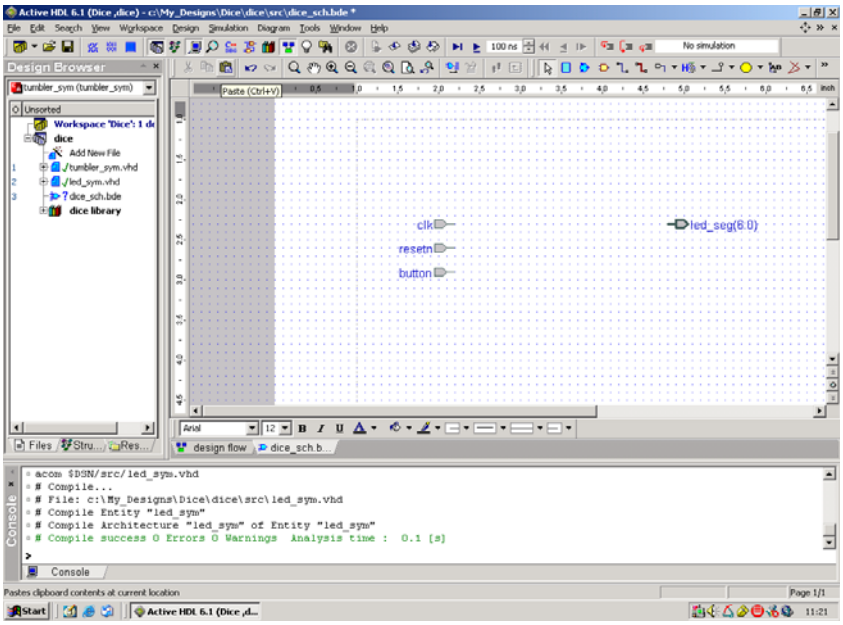
Symbols
In the next window the in and out ports of our block are defined. **Define as shown in the figure to the left.**
Click *"New"* and type in the names of the ports, directions, and number of bits. [Finish].
We have now created a skeleton and it is time to add some functionality.
Enter the code from *"tumbler_sym.txt"* directly after
"-- enter your statements here –" in the now created VHDL file. Add also *"use IEEE.STD_LOGIC_UNSIGNED.all;"* at the top.



Now the file shall be compiled to verify the syntax. Right-click on *"tumbler_sym"* in *Design Browser* and choose *"Compile"*. Hopefully, *"Compile success 0 Errors 0 Warnings"* is shown in green text in the status window.
Now the second block, *"led_sym"*, shall be created. Do as for *"tumbler_sym"*, but with the ports defined as in the **figure to the left**.

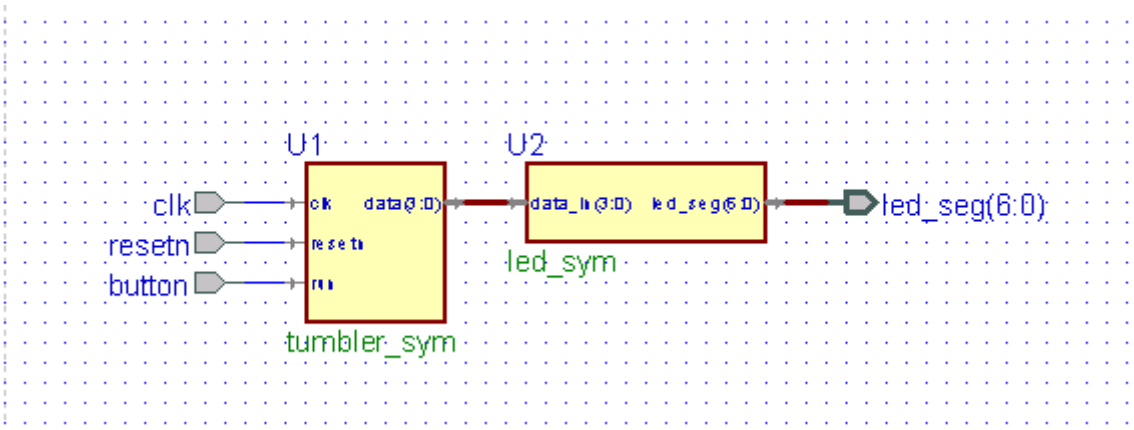


Schematic diagram
The blocks above shall be tied together into a single, big block, and the in/out ports of this big block shall be defined.
This is done with the schematic editor.
"Add new file".
Choose tab Wizards -> *"Block Diagram Wizards"* [OK] [Next][Next] and enter the name, *"dice_sch"*, of the new diagram here. [Next].
Set the ports as in the figure to the left. These are the ports of the schematic, that is, the external in and out signals.
[Finish].



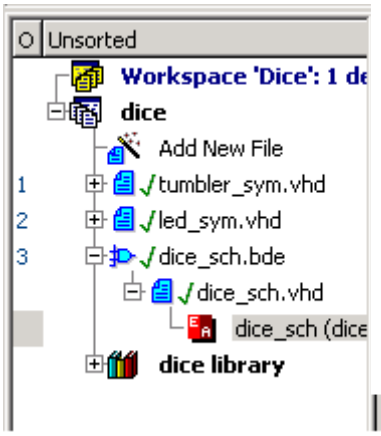
Now it should look like this. Here, the different symbols are interconnected with the help of *"Wire"* (blue thin line) and *"Bus"* (red thick line). *Wire* is used for individual signals, and *Bus* is used for bundles of signals from one place to another. Start with clicking on the yellow symbol at the top:

This makes all symbols in *"Symbols toolbox"* appear. We can choose from these, which are both the own that we have written code for, and those already included in the program. Click on *"Units Without"*. You will now see the two symbols you have constructed earlier. Drag in your two blocks and tie them together according to the figure below.



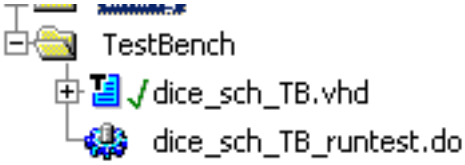
Now **compile** the file in the same way as before.

B. Simulation

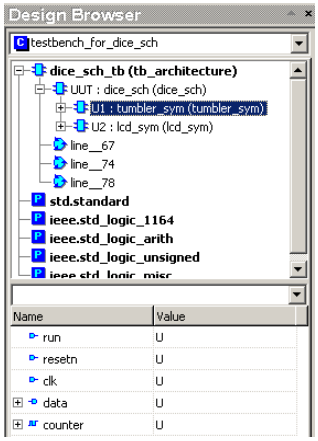


When the design is ready, it is time to simulate it in order to verify its functionality.

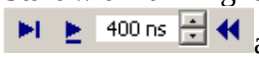
Click on the + sign before *"dice_sch.bde"*, then on + sign before "dice_sch.vhd". You will then get a red symbol, called dice_sch(dice_sch). Right-click on this symbol and choose *"Generate Test Bench"* and click on [Next] until you get no more questions, and finally [Finish.].



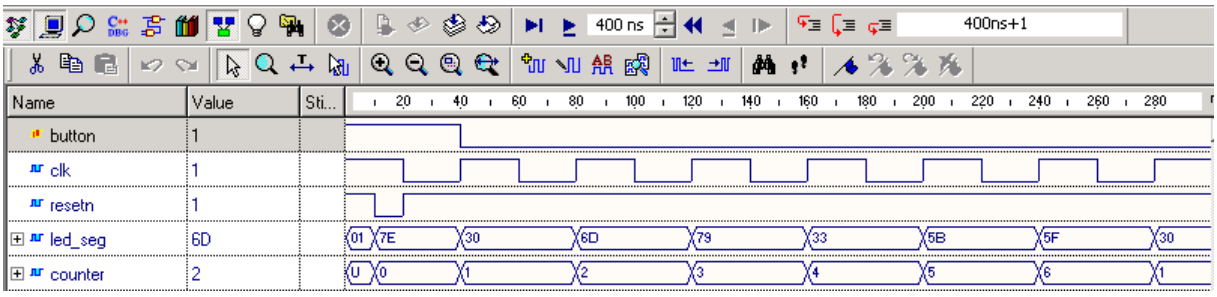
Now a file that will contain test values is created. The values are needed to verify the design. Now open the file, which has the name *"dice_sch_TB.vhd"*. Add the code in "testbank.txt" directly after the row *"-- Add your stimulus here …. "*.
This is the data the design is tested against. Compile the file. Right-click on *"dice_sch_TB_runtest.do"* and choose *"Execute"*. The simulator will now start.



Start with running it for 400 ns by pressing the underlined arrow



and type 400 ns. You can now see how the out signals are changed, depending on applied insignals. If these (external) signals not are enough, the internal signals can also be viewed. Add the signal *"counter"* by chosing *Waveforms-> Add signals* and look up *"counter"* (placed in *"tumbler_sym"* under *"dice_sch", see left*) and then press *"Add"*. *"Counter"* will now be added to the simulation window.
Restart the simulation by clicking on the back arrow, and run until 400ns again. Note that counter continuously changes between 1 and 6.



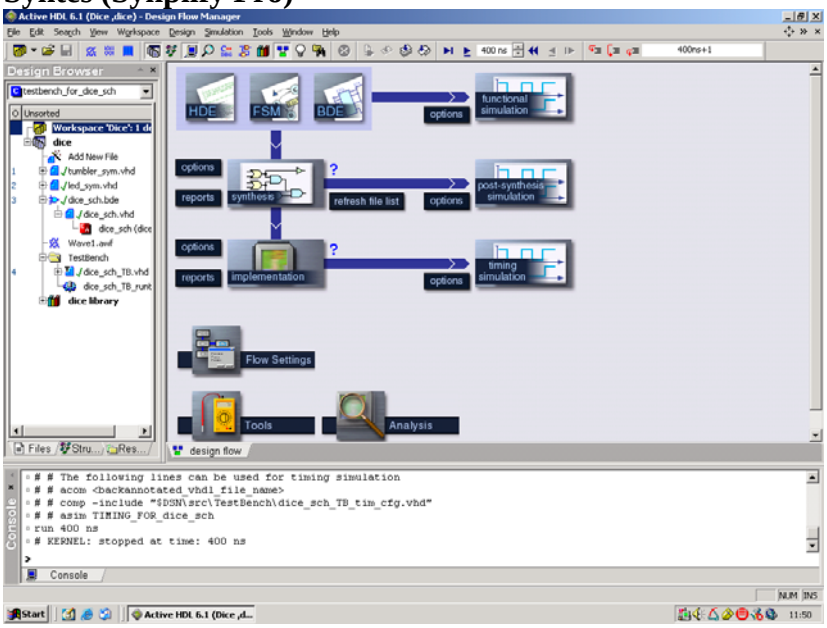
Note how the signals change depending in the insignals. When you are done with the simulation, select *"Simulation"-"End Simulation"* to terminate.

Now the functionality is verified.

B. Synthesis

We shall now proceed with the synthesis part. The use of synthesis software means that you yourself do not have to translate and minimize the VHDL code. Translation is made to gates and flip-flops (netlist). Differences between different synthesis tools in the area and timing may be significant. Here we have access to the Synplify Pro synthesis tool.

Syntes (Synplify Pro)



Now press the tab "*Design Flow*" so that the window looks like this.

a) Run *Synplify* manually

Select "Options" on the left of the picture "*Synthesis*". Enter *dice_sch* as *Top-Level Unit*, *Xilinx6x SPARTAN 2E* of *Family* and *2s300efg456* as *Device*. If you want to see the result of the synthesis, select "*GUI*" [OK] in "*Options*". Also go into the "*Settings*" and enter the frequency to 24 MHz.

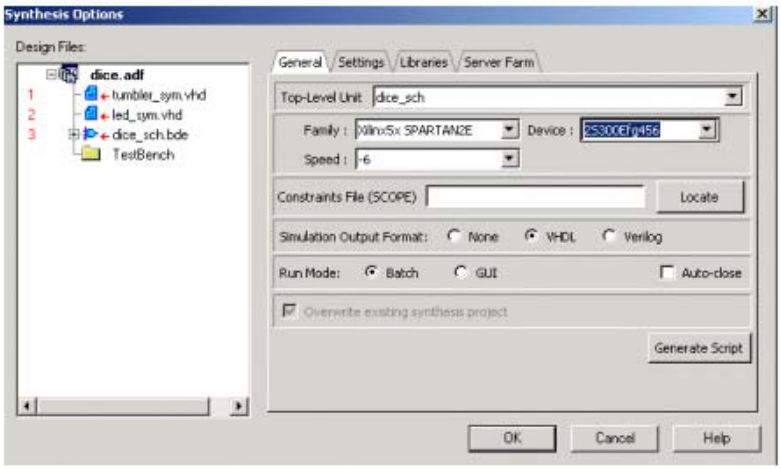
Press the "*synthesis*" to start the program "*Synplify Pro*" for manually driving the synthesis.

Press the big button "Run". Then click on the one of these  to look at the synthesized code in RTL or gate level.

You will then see something like this:

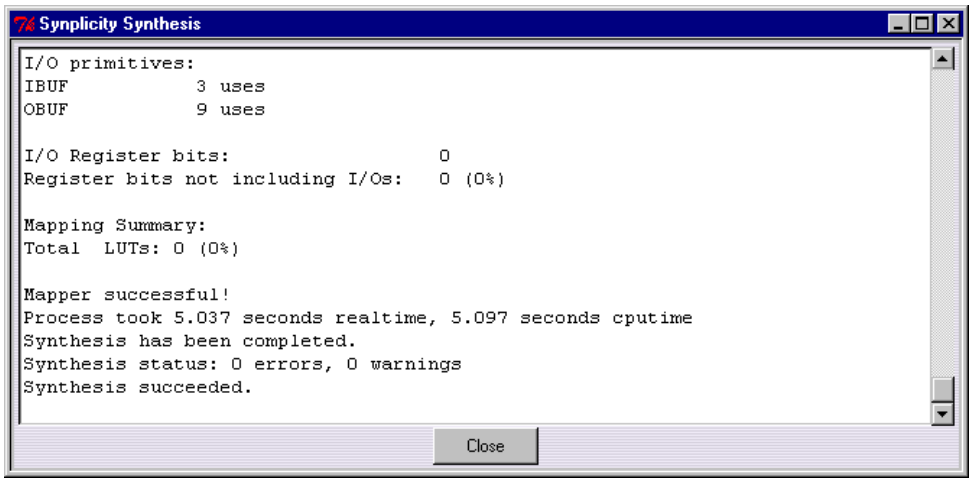
You can then move down and up into blocks by right-clicking on the blocks and select "*Push / Pop Hierarchy* ".

b) Run *Synplify* automatically



If you want automatic execution of *Synplify* select "*Batch*" instead of "*GUI*" in the Run Mode under "*Options*".

Synplify will now synthesize the files, ie. to translate the code into logic gates, this is done without having to switch applications, just click on the icon "*Synthesis*".



Area and Timing: Go into the synthesis report (under "*reports*") and locate the area consumption (the number of LUT:s) and the estimated time delay (the minimum period time).

Number of LUT:s: ... ...

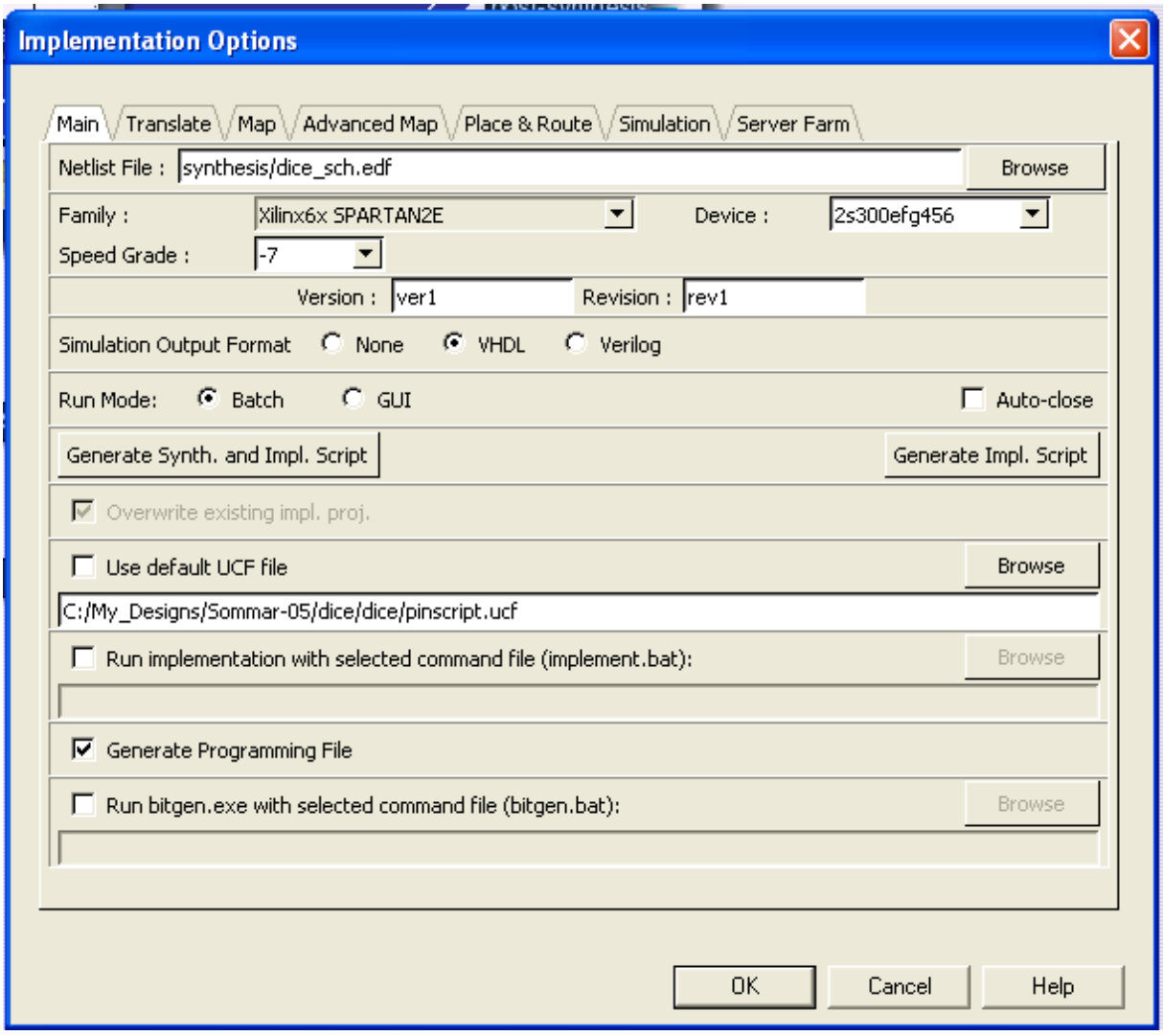
Minimum period time: ... ...

Decide also "slacktime" (margin) of 24 MHz: ... ...

Quit Synplify Pro.

C. Implementation

This translates the netlist, which was produced by the synthesis, to a bit-file, which can then be downloaded to the card.
First, however the pinscriptfile must be entered. It is in this file the port names are bound to the physical pins on the FPGA. Go into the "*File-New-Text Document*" and enter the pinscriptfile. Save as "pinscript.ucf" in C*: / My_Designs/ dice*. Switch to the "*Design Flow*" and click on "*Options*" right along with "*Implementation*".
Set as the picture below show.

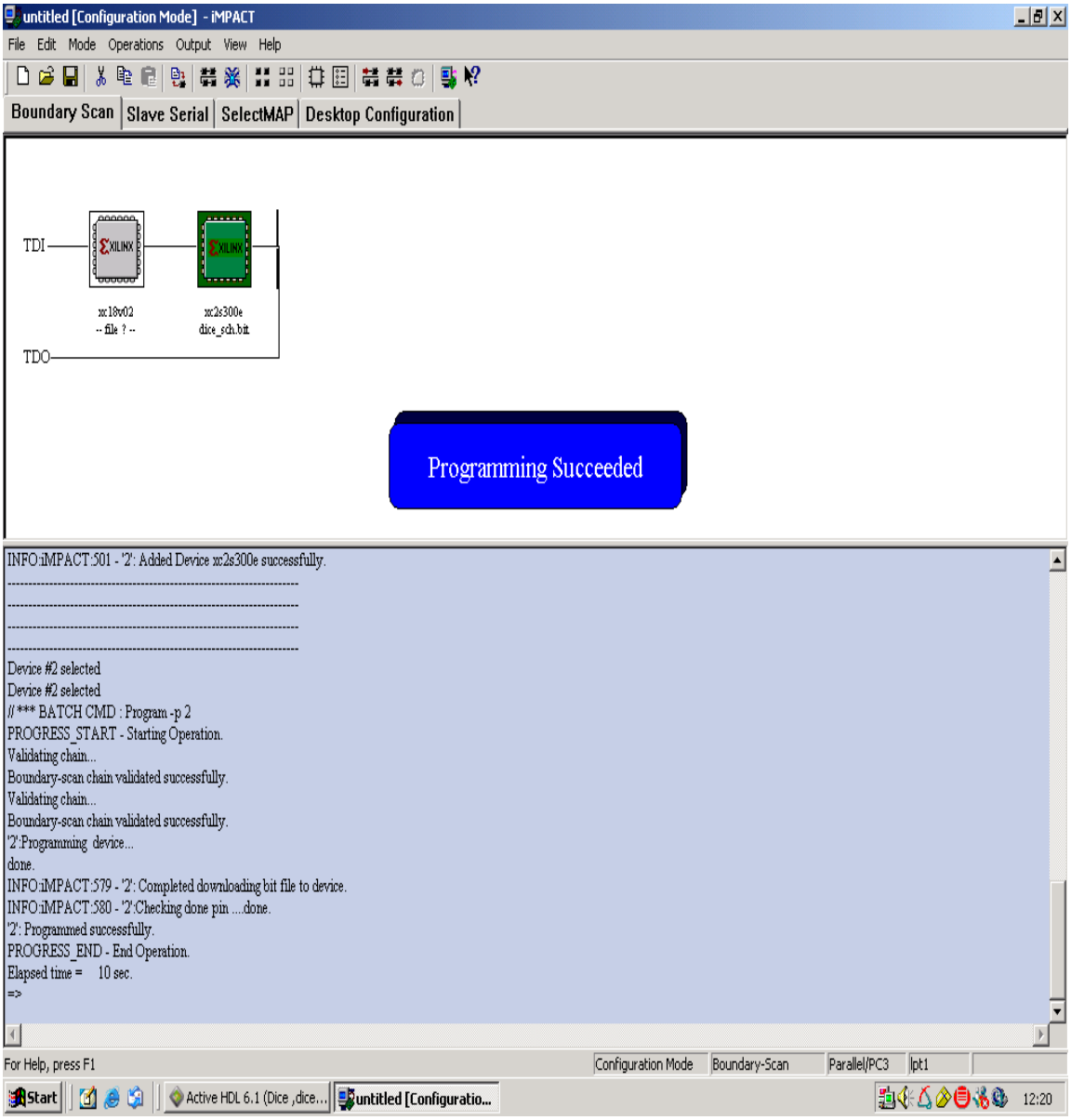


Enter the top levelname of your design in the *Netlist File* and *Device* (2s300efg456). Point out where you have "*pinscript.ucf*" under *Use the default UCF file* and mark the box for *Generate Programming File*.

Close the window and start the implementation by clicking on the icon "*Implementation*". The program will now perform a Place and Route. The warnings can usually be ignored.

D. Device programming

Connect the adapter to the parallel port. Click "*Start-Programs-Xilinx ISE 6-Accessories-iMPACT*", [Next] [Next] [Next].
Select the right FPGA, right click and select "*Assign New Configuration File*". Open "*dice_sch.bit*", which is located in C: \ My_designs \ dice \ implement \ ver1 \ rev1. Right-click on the right chip and select "*Program*". Neglect ev. warning. Hopefully, it now looks like it does below. You are now ready to test your dice.



Appendix 1.

tumbler_sym.txt

```
signal counter:         std_logic_vector(3 downto 0);
begin
        process(clk, resetn)
        begin
                if resetn = '0' then
                        counter <= "0000";
                elsif clk'event and clk = '1' then
                        if run = '0' then
                                if counter < "0110" then
                                        counter <= counter + 1;
                                else
                                        counter <= "0001";
                                end if;
                        end if;
                end if;
        end process;

data <= counter;
```

---

led_sym.txt

```
process(data_in)
begin
        case data_in is
                                                --abcdefg
                when "0000" =>          led_seg <=  "1111110";  -- 0
                when "0001" =>          led_seg <=  "0110000";  -- 1
                when "0010" =>          led_seg <=  "1101101";  -- 2
                when "0011" =>          led_seg <=  "1111001";  -- 3
                when "0100" =>          led_seg <=  "0110011";  -- 4
                when "0101" =>          led_seg <=  "1011011";  -- 5
                when "0110" =>          led_seg <=  "1011111";  -- 6
                when "0111" =>          led_seg <=  "1110000";  -- 7
                when "1000" =>          led_seg <=  "1111111";  -- 8
                when "1001" =>          led_seg <=  "1110011";  -- 9
                when "1010" =>          led_seg <=  "1110111";  -- A
                when "1011" =>          led_seg <=  "0011111";  -- b
                when "1100" =>          led_seg <=  "1001110";  -- C
                when "1101" =>          led_seg <=  "0111101";  -- d
                when "1110" =>          led_seg <=  "1001111";  -- E
                when "1111" =>          led_seg <=  "1000111";  -- F
                when others  =>         led_seg <=  "0000001";  -- -
                end case;
end process;
```

Appendix 2.

testbank.txt

```
process
begin
        clk <= '1';
        wait for 20 ns;
        clk <= '0';
        wait for 20 ns;
end process;

resetn <=   '1',
            '0' after 10 ns,
            ’1' after 20 ns;

button <=   '1',
            '0' after 40 ns,
            '1' after 350 ns;
```

---

pinscript.ucf

```
#Pinscript-fil

NET clk PERIOD = 40ns;

NET "clk"           LOC="C11";
NET "resetn"        LOC="A6";
NET "button"        LOC="D8";

NET "led_seg(0)"    LOC="E9";
NET "led_seg(1)"    LOC="E10";
NET "led_seg(2)"    LOC="E8";
NET "led_seg(3)"    LOC="E7";
NET "led_seg(4)"    LOC="B8";
NET "led_seg(5)"    LOC="A8";
NET "led_seg(6)"    LOC="B9";
```