FPGA4U Tutorial

This is an overview of the tools we will use during the labs.

Learning Goal: Review FPGA4U Hardware and Tools.

Requirements: FPGA4U Board. Quartus II Web Edition and ModelSim-Altera.

1 Introduction

This tutorial will introduce you to the FPGA4U and the tools needed to use it. The instructions are targeting Quartus II 7.0 and ModelSim 6.1g, but they should be compatible with any other version of Quartus II and ModelSim.

2 Preparing the workspace

2.a) Create a new folder in which you will save all your project files (e.g., Z:/Archord/). Be sure the pathname does not contain any space, this will save a lot of trouble.

2.b) For each lab, you will create a new subfolder. Create a new subfolder for this tutorial (e.g., Z:/Archord/tutorial). Please, remember to create a new subfolder for each new project to avoid conflicts.

2.c) We will organize the project and source files in 3 different folders:

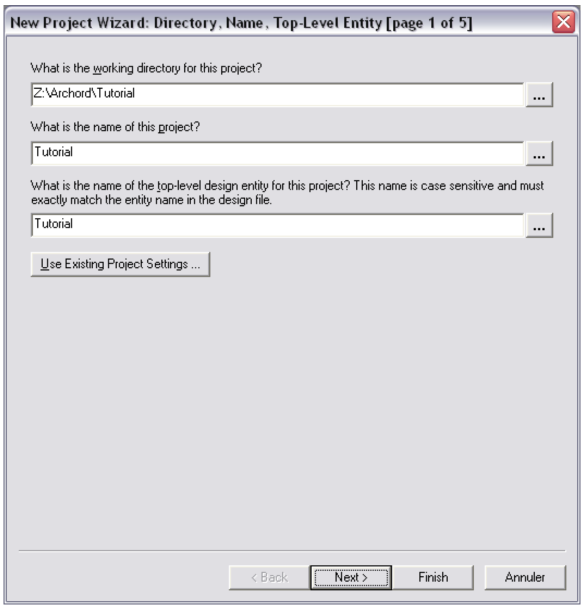
- **vhdl**: it will contain the VHDL sources of the lab.
- **quartus**: for the Quartus project files of the lab.
- **modelsim**: for the ModelSim project files of the lab.

2.d) Create these 3 folders in your tutorial subfolder.

3 Creating a Quartus project

3.a) Run **Quartus II**.

3.b) Start the **New Project Wizard** by selecting File > New Project Wizard...

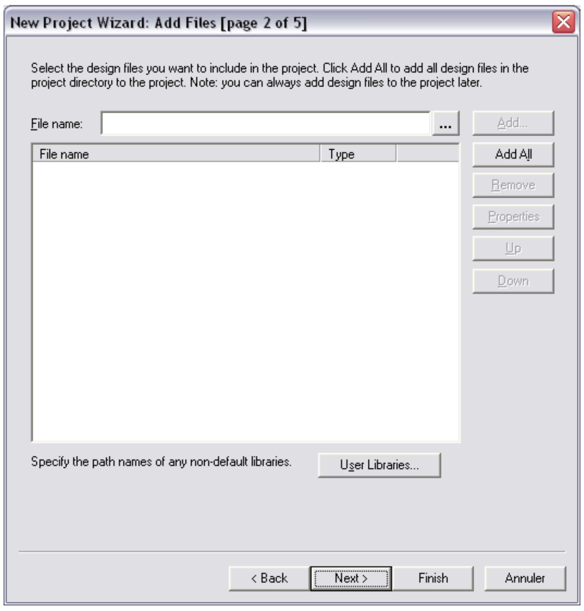

New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?
Z:\Archord\Tutorial

What is the name of this project?
Tutorial

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
Tutorial

3.c) Select the newly created **quartus** folder (e.g., Z:/Archord/tutorial/quartus) and name the project (e.g., tutorial). By default, the top-level entity name will be the same as the project. Click the Next button to continue to the next step.


New Project Wizard: Add Files [page 2 of 5]

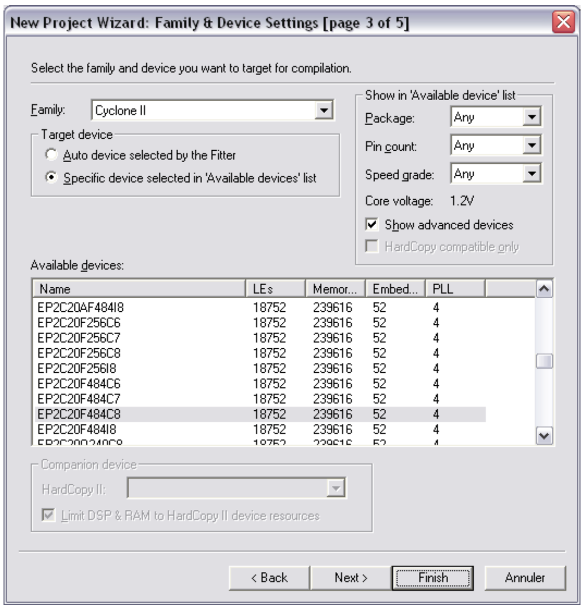
Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project. Note: you can always add design files to the project later.

File name:

File name	Type

Specify the path names of any non-default libraries.

3.d) This step allows you to add design files to the project. As we currently don't have any, we can skip this step. Click the Next button.



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Family: Cyclone II

Target device
- Auto device selected by the Fitter
- Specific device selected in 'Available devices' list (selected)

Show in 'Available device' list
- Package: Any
- Pin count: Any
- Speed grade: Any
- Core voltage: 1.2V
- Show advanced devices (checked)
- HardCopy compatible only

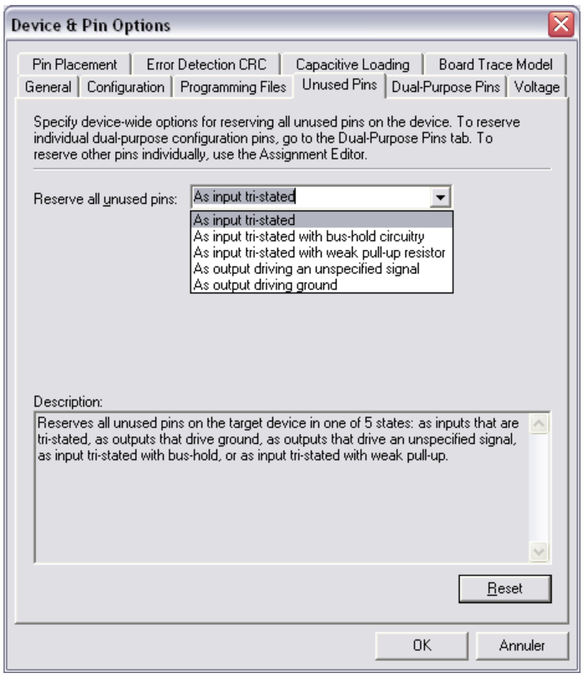
Available devices:

Name	LEs	Memor...	Embed...	PLL
EP2C20AF484I8	18752	239616	52	4
EP2C20F256C6	18752	239616	52	4
EP2C20F256C7	18752	239616	52	4
EP2C20F256C8	18752	239616	52	4
EP2C20F256I8	18752	239616	52	4
EP2C20F484C6	18752	239616	52	4
EP2C20F484C7	18752	239616	52	4
EP2C20F484C8	18752	239616	52	4
EP2C20F484I8	18752	239616	52	4

Companion device
- HardCopy II:
- Limit DSP & RAM to HardCopy II device resources

3.e) In this page, we have to specify which FPGA device we will use. In the **Family** field, select **Cyclone II**. In the **Available devices** list, select the **EP2C20F484C8** device. No changes are required in the next two pages. Click on the **Finish** button.

3.f) The project is now created but we must set some properties. Open the **Device settings** dialog (Assignments > Device...) and click on the **Device & Pin Options...** button.



Device & Pin Options

Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated
- As input tri-stated
- As input tri-stated with bus-hold circuitry
- As input tri-stated with weak pull-up resistor
- As output driving an unspecified signal
- As output driving ground

Description:

Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

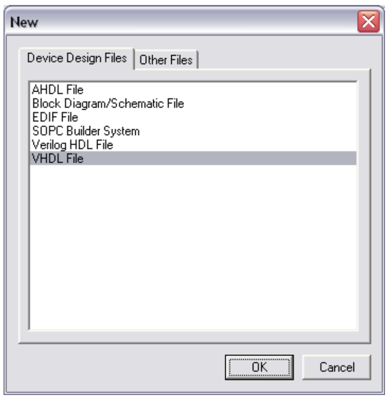
3.g) Select the **Unused Pins** tab and, in the **Reserve all unused pins** field, select **As input tri-stated**. This action specifies that by default all pins of the FPGA device will be in an electrically neutral state, so that no damage can arise to your FPGA4U board.

3.h) Click **OK** twice to save the modifications and to return to the workspace.

4 Adding a VHDL File

We are now going to add three new VHDL files to the project.

4.a) Click on File > New…



AHDL File
Block Diagram/Schematic File
EDIF File
SOPC Builder System
Verilog HDL File
VHDL File

4.b) Select **VHDL File** and click **OK**. Fill the file with the **counter** VHDL code bellow, name this file `counter.vhd` and save it to your **vhdl** folder. Beware, the filename must match the entity name declared inside; the two names are not case sensitive.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(
    clk     : in std_logic;
    reset_n : in std_logic;
    clk_en  : in std_logic;
    value   : out std_logic_vector(31 downto 0)
    );
end counter;

architecture synth of counter is
  signal count : std_logic_vector(31 downto 0);
begin

value <= count;

process(clk, reset_n)
begin
  if (reset_n = '0') then
    count <= (others=>'0');
  elsif (rising_edge(clk)) then
    if (clk_en='1') then
      count <= count + 1;
    end if;
  end if;
end process;
end synth;
```

4.c) Create another file the same way, fill it with the following code, and save it to `alu.vhd` in the **vhdl** folder.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ALU is
  port(
    A : in std_logic_vector(7 downto 0);
    B : in std_logic_vector(7 downto 0);
    add_op : out std_logic_vector(7 downto 0);
    and_op : out std_logic_vector(7 downto 0);
    or_op  : out std_logic_vector(7 downto 0);
    xor_op : out std_logic_vector(7 downto 0)
  );
end ALU;

architecture synth of ALU is
begin

  add_op <= A + B;
  and_op <= A and B;
  or_op  <= A or B;
  xor_op <= A xor B;

end synth;
```

4.d) Repeat the same steps to create a `multiplexer.vhd` file in the **vhdl** folder.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multiplexer is
  port(
    I0   : in  std_logic_vector(7 downto 0);
    I1   : in  std_logic_vector(7 downto 0);
    I2   : in  std_logic_vector(7 downto 0);
    I3   : in  std_logic_vector(7 downto 0);
    sel  : in  std_logic_vector(1 downto 0);
    Z    : out std_logic_vector(7 downto 0)
    );
end multiplexer;

architecture synth of multiplexer is
begin

process(I0, I1, I2, I3, sel)
```

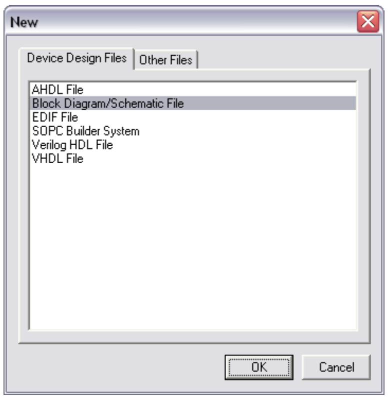
```
begin
  case sel is
    when "00" => Z <= I0;
    when "01" => Z <= I1;
    when "10" => Z <= I2;
    when "11" => Z <= I3;
    when others =>
  end case;
end process;

end synth;
```

5 Adding a Schematic File

For the top-level entity of this project, we will use a schematic file. Schematic files are only supported by Quartus, therfore there is no need to save them in the **vhdl** folder, you can save them in the **quartus** folder.

5.a) Click on File > New... to add a file.


New

Device Design Files | Other Files

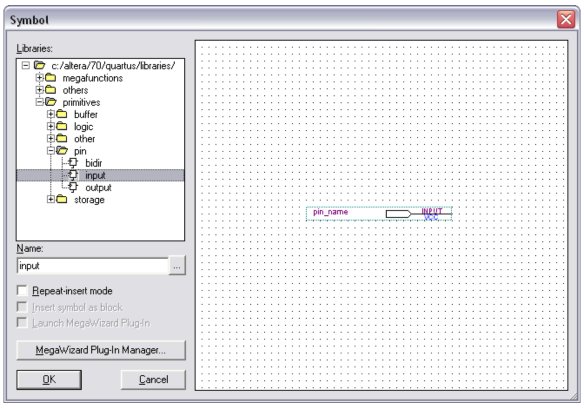
AHDL File
Block Diagram/Schematic File
EDIF File
SOPC Builder System
Verilog HDL File
VHDL File

OK | Cancel

5.b) Select **Block Diagram/Schematic File** and click **OK**.

5.c) Click on File > Save As.... Normally, the default filename that is proposed should be the same as your project's name. If it's not, rename it correspondingly. Click **OK**.

5.d) To add content inside the schematic file, open the **Symbol** dialog by clicking the **Symbol Tool** button or by double-clicking anywhere on the background grid.


Symbol

Libraries:
c:/altera/70/quartus/libraries/
megafunctions
others
primitives
buffer
logic
other
pin
bidir
input
output
storage

Name: input

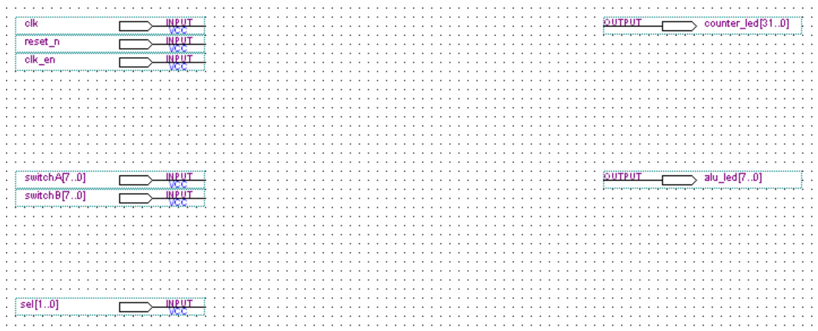
Repeat-insert mode
Insert symbol as block
Launch MegaWizard Plug-In

MegaWizard Plug-In Manager...

OK | Cancel

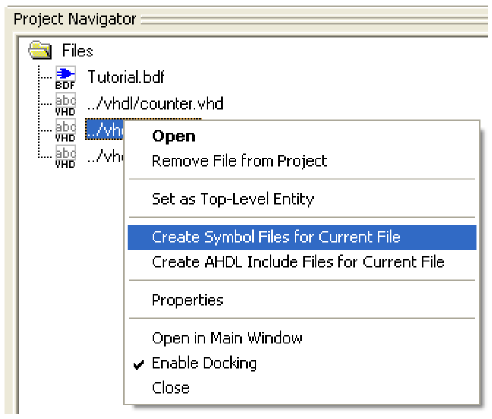
5.e) First, we must add some pins to the system. In the Symbol dialog box, select the **input** pin and click **OK**. Now you must click somewhere on the grid to add the input pin. You can rename it by double-clicking on it.

5.f) Add other input and output pins, and rename them to match the following schematic. To go faster, you can duplicate the pins you already added. To do so, copy and paste a pin with Ctrl+C and Ctrl+V, or select a pin and drag it while pressing the Ctrl key. A signal width greater than 1 bit can be defined using square brackets. For example, a 6-bit **toto** signal will be written **toto[5..0]**.



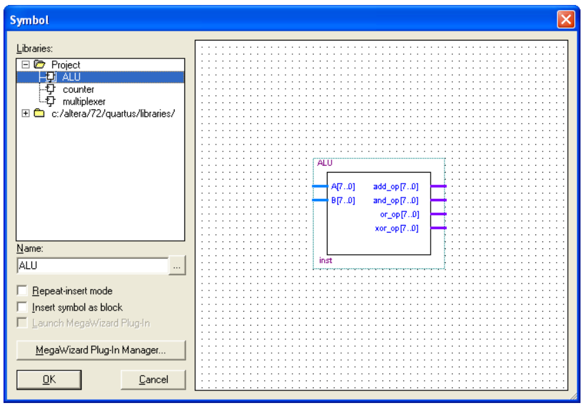
6 Adding a User-defined Module in the Schematic File

Now that we have a schematic file, we want to insert our VHDL modules inside. For that, we need to generate a graphical description of each entity.



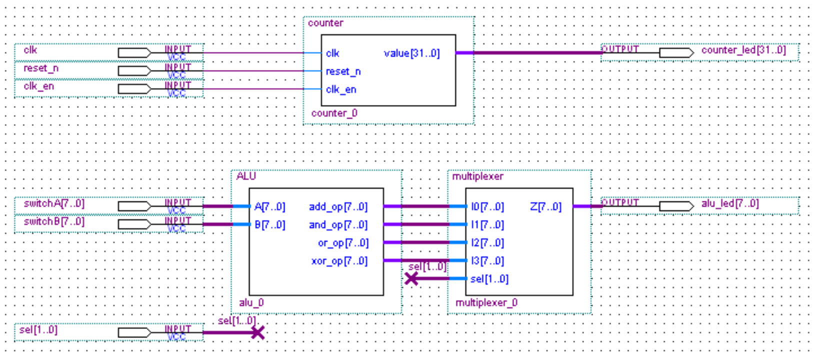
6.a) In the **Project Navigator**, select the **File** tab, open the **Device Design Files** folder and right-click on one of the VHDL files. In the pop-up menu, select **Create Symbol Files for Current File**. Wait until the end of the process and repeat the operation for the other VHDL files.

6.b) Return to the schematic file and open the **Symbol** dialog.



6.c) Now, you should see your design entities in the **Project** folder. Add them to the schematic file and connect them to the input and output pins, as shown in the following figure. To connect a pin to an element, put your mouse pointer on the connector of a module (your cursor should look like ┼┐), then click and drag.

Notice that you can connect two lines together by giving them the same name, or label. There's an example in the next figure for the **sel[1..0]** signal. To add a label, right click on the desired line and select poperties.

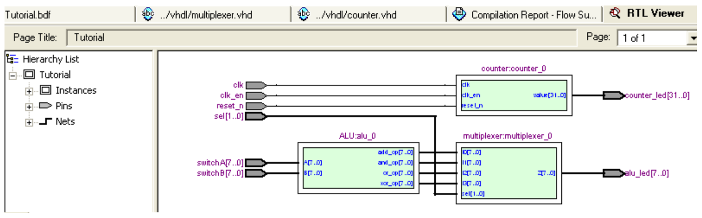


7 Synthesis

The system is now complete, and we can synthesize it.

7.a) Click on the **Start Analysis & Synthesis** button in the toolbar or in the menu: Processing > Start > Start Analysis & Synthesis.

7.b) Select Tools > Netlist Viewers > RTL Viewer to see the synthesis result. You should see something like this:



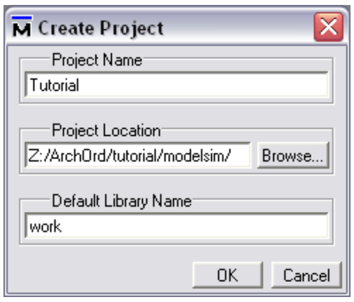
7.c) You can double-click on an instance in the schematic or on the left pane to look inside. Try to look inside the modules, and see how it has been synthesized.

8 Simulation

Now we are going to check the functional behavior of our system. For the simulation we will use ModelSim.

8.a) Run ModelSim.

8.b) Select File > New > Project. . . .



Create Project

Project Name: Tutorial

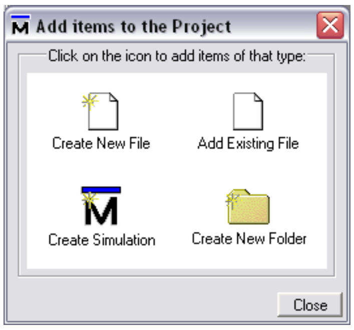
Project Location: Z:/ArchOrd/tutorial/modelsim/

Default Library Name: work

8.c) In the **Project Name** field insert a project name (e.g., tutorial).

8.d) Enter the path to your **modelsim** folder in the **Project Location** field (e.g., Z:/Archord/tutorial/modelsim).

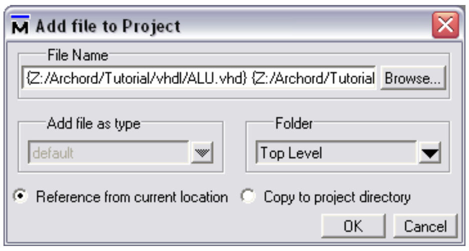
8.e) We will keep **work** as the **Default Library Name**. Click **OK**.



Add items to the Project

Click on the icon to add items of that type:

8.f) Now, we have to add our VHDL files to the project. Click on **Add Existing File** in the **Add items to the Project** dialog, or in File > Add to Project > Existing File. . . .



Add file to Project

File Name: {Z:/Archord/Tutorial/vhdl/ALU.vhd} {Z:/Archord/Tutorial

Add file as type: default

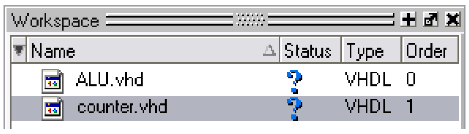
Folder: Top Level

(●) Reference from current location ( ) Copy to project directory

8.g) Select all the VHDL files present in your **vhdl** folder. Make sure **Reference from current location** is selected. Click **OK**, and close the **Add items to the Project** dialog.

8.h) The **Workspace** panel should look like the following figure.

If you don't see the ***Workspace*** *panel, check* View > Workspace *in the menu. If it's not sufficient, selecting* Window > Layouts > Reset *should fix it.*



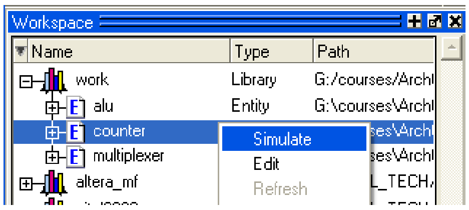
Workspace

Name	Status	Type	Order
ALU.vhd	?	VHDL	0
counter.vhd	?	VHDL	1

8.i) Notice the **?** in the **Status** fields. It means that the project files haven't been compiled since their last modifications. To compile all the files of the project, right-click in the **Workspace** panel and select Compile > Compile All. When the compilation is complete, a green mark should replace the **?**.

If compilation succeeded but you don't see any green mark, try to refresh the status by right-clicking in the ***Workspace*** *panel and selecting* Update.

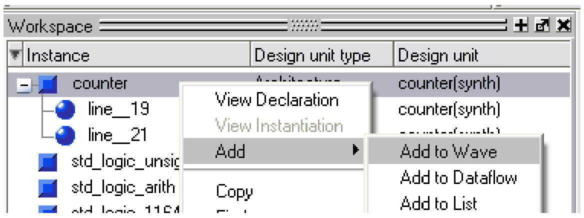
8.j) Now that the files are compiled, we can start a simulation. Select the **Library** tab in the **Workspace** panel and expand the work library. The work library contains all the design entities defined by our VHDL files. We will first simulate the **counter**. To start the simulation you can double-click on it's entity name or right-click on it and select Simulate.



Workspace

Name	Type	Path
work	Library	G:/courses/Archi
alu	Entity	G:\courses\Archi
counter		es\Archi
multiplexer		es\Archi
altera_mf		_TECH

8.k) To see the timing diagram of the simulation, we need to make the **Wave** appear, and add some signals to it. Right-click on the counter instance and select Add > Add to Wave (or Add > To Wave > All items in Region, depending on your version).

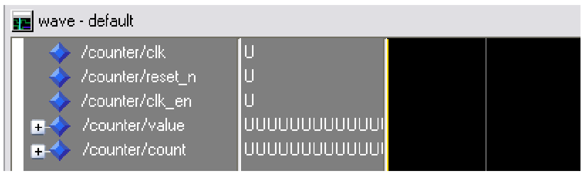


Workspace

Instance	Design unit type	Design unit
counter		counter(synth)
line__19		counter(synth)
line__21		
std_logic_unsig		
std_logic_arith		

8.l) Doing this should add all of the counter signals to the **Wave** and make the **Wave** appear.

If you can't see it, try to click on View > Debug Windows > Wave.



wave - default

/counter/clk	U
/counter/reset_n	U
/counter/clk_en	U
/counter/value	UUUUUUUUUUUU
/counter/count	UUUUUUUUUUUU

8.m) With ModelSim we can set some signal values with the `force` command. For convenience, we will write a little script to set all the input signal values. Create a new file: File > New > Source > Do. Fill it with the script bellow and save it to `counter.do` in your **modelsim** folder.

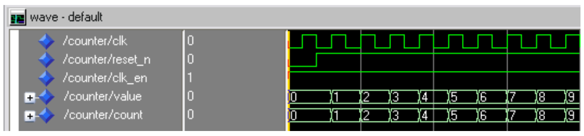
```
restart -f

force clk 0 0ns, 1 10ns -repeat 20ns
force clk_en 1 0ns
force reset_n 0 0ns, 1 20ns

run 200ns
```

The `restart` command reinitializes the simulation: the time pointer is reset to 0ns and compiled files are reloaded, if necessary. The `force` command takes as parameters a signal name and a list of value and time pairs. Notice the `-repeat` statement, which allows the waveform to repeat the specified period. The `run` command starts the simulation for the specified time interval.

8.n) To execute the script, you can directly write `do counter.do` in the console, or go in Tools > Execute Macro... and select `counter.do`.



The format of a signal can be modified by right-clicking on the signal name and going into the Radix menu. Feel free to modify the script to try to change the clock period or disabling the clock signal during some time interval, for instance.

8.o) When you're done, repeat the same operations for the **ALU**. Start the simulation from the **Library** tab of the **Workspace** panel, add the signals to the wave and save the following script to `alu.do`.

```
restart -f

force a 01010101 0ns, 16#FF# 20ns, 10#85# 40ns
force b 00000011 0ns, 16#AA# 20ns

run 200ns
```

Notice that there are different ways to set an 8-bit value to the **a** and **b** signals. The default format is binary, but by surrounding a value with # character, we can specify another base.

8.p) Now run the script, look at the wave, and verify that the results are correct. You may need to change the radix to ease the comparisons.

8.q) Create yourself a `multiplexer.do` scripts, that gives a different value for each **I**$_n$ input, and sets the **sel** input to switch between them in time.

8.r) When you're done with the simulation, exit ModelSim and return to your Quartus project.

9 Downloading the design onto the FPGA

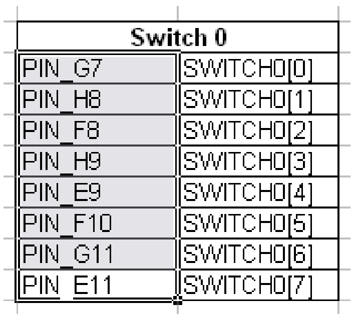
To download the design onto the FPGA, we must specify on which pins of the FPGA we would like to connect the input and the output of our design.

9.a) Open the **Pin Planner** by pressing the button or by selecting Assignments > Pins. All the input/output signals of our system are listed, and for each of them we have to specify a pin of the FPGA.

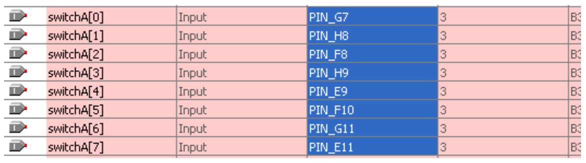
9.b) There are many pins to choose from, but we can accelerate the process. Download the `PinAssignment.xls` file from the FPGA4U website (`http://fpga4u.epfl.ch/`). You can directly copy/paste the cells from the spreadsheet to Quartus.

For example, to connect the **switchA** input to the **Switch 0** of the FPGA4U: in the spreadsheet, select the pins' column of **Switch 0** and copy the cells. Return to Quartus, select the pins of **switchA** and paste.

*By default, the **Pin Planner** sorts the signal bits from most significant to least significant, as opposed to the ordering in the spreadsheet. To reverse this ordering in the **Pin Planner**, sort the signals by name. To do so, click on the **Node Name** header.*



Switch 0	
PIN_G7	SWITCH0[0]
PIN_H8	SWITCH0[1]
PIN_F8	SWITCH0[2]
PIN_H9	SWITCH0[3]
PIN_E9	SWITCH0[4]
PIN_F10	SWITCH0[5]
PIN_G11	SWITCH0[6]
PIN_E11	SWITCH0[7]



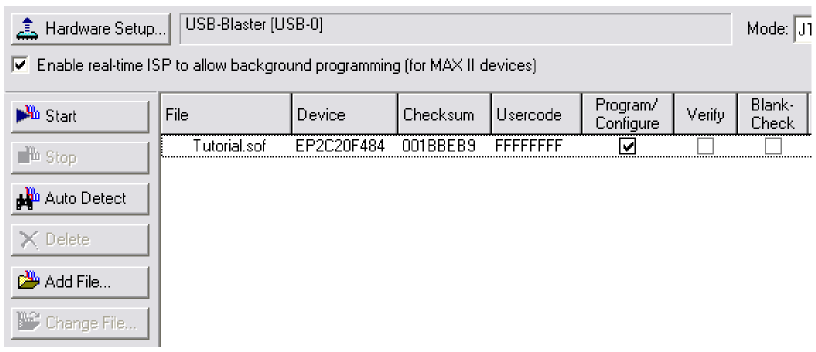
switchA[0]	Input	PIN_G7	3	B3
switchA[1]	Input	PIN_H8	3	B3
switchA[2]	Input	PIN_F8	3	B3
switchA[3]	Input	PIN_H9	3	B3
switchA[4]	Input	PIN_E9	3	B3
switchA[5]	Input	PIN_F10	3	B3
switchA[6]	Input	PIN_G11	3	B3
switchA[7]	Input	PIN_E11	3	B3

9.c) Connect the signals according to the following table. There are different orderings available for the LEDs in the spreadsheet. Take the **0:top-right** one.

Design I/O	FPGA4U Component Name
clk	Clk0_24MHz (PIN_L1)
clk_en	Button_n[0] (PIN_U11)
reset_n	Reset_n (PIN_B3)
counter_led	LED[95..64]
switchA	Switch 0
switchB	Switch 1
alu_led	LED[7..0]
sel	Button_n[3..2]

9.d) When you are done with the assignment, press the button, or click on Processing > Start Compilation to compile the design.

9.e) Connect the FPGA4U board on your computer and run the **Programmer** by clicking on the button, or on Tools > Programmer.



USB-Blaster [USB-0]   Mode: J

☑ Enable real-time ISP to allow background programming (for MAX II devices)

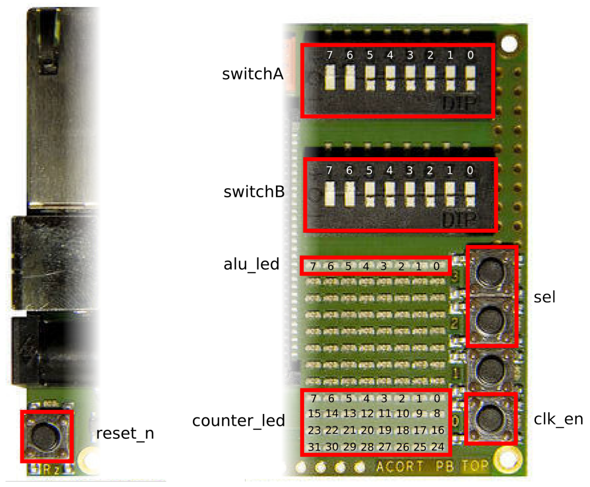
File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check
Tutorial.sof	EP2C20F484	001BBEB9	FFFFFFFF	☑	☐	☐

If the text next to the **Hardware Setup...** button is 'No Hardware', make sure you have connected the board to the computer with the USB cable, and click on the **Hardware Setup...** button. A new window will list the available hardware items connected on your computer. If **USB-Blaster [USB-0]** does not appear on the list, then you may have a problem with the installation of the driver. Please refer to `http://fpga4u.epfl.ch/wiki/Install_Quartus_II`.

9.f) To continue you must have the **USB-Blaster [USB-0]** selected and have checked **Program/Configure**.

9.g) Click on the **Start** button to program the FPGA.

The following figure describes the position of the input and output signals.



- **switchA** and **switchB** are controlled by the switches of the board. The most significant bit is on the left.
- The selection of the ALU operation is done with the **sel** input that is mapped on the push buttons 2 and 3. When the buttons are released, the value of **sel** is `"11"`.
- The selected output of the ALU is displayed on the 8 upper LEDs, the most significant bit being on the left.
- The counter value is displayed on the 32 bottom LEDs. Each 8-bit row represents a byte of the signal. The upper row is the first byte, and the most significant bit is on the left.
- **clk_en** is the button next to the counter value. Pressing on it disables the **clk** signal and pauses the counter.
- **reset_n** is the reset button. Pressing on it reinitializes the counter.

10 Exercises

This section contains some simple exercises of logic systems and VHDL to refresh your memory.

10.a) Give the 8-bit two's complement representation of -18.

10.b) What is the set of signed values that are representable on 8 bits using the two's complement representation?

10.c) Assume 8-bit signed numbers. Which of the following additions generates an overflow? (0x means that we are using hexadecimal numbers)

a) 0x10 + 0x20
b) 0xFF + 0x01
c) 0x7A + 0x20

10.d) What is a sequential system?

a) It's a system that depends only on the input values.
b) It's a combinatorial system with a clock signal.
c) It's a system, whose outputs depends on the history of the system.

10.e) Look at the following VHDL code. What component is it? Describe its behavior. Draw a schema of what should be the output of the synthesis.

- *Note that "*`(rising_edge(clk))`*" is equivalent to "*`(clk'event and clk='1')`*".*

```
process(clk, reset_n)
begin
  if (reset_n = '0') then
    q <= '0';
  elsif (rising_edge(clk)) then
    if (en = '1') then
      q <= d;
    end if;
  end if;
end process;
```

10.f) What is the difference between the previous code and the following one? Explain, and draw a timing diagram that illustrates a different behavior.

```
process(clk)
begin
  if (rising_edge(clk)) then
    if (reset_n = '0') then
      q <= '0';
    elsif (en = '1') then
      q <= d;
    end if;
  end if;
end process;
```

10.g) What component is described by the following code?

```
process(reset_n, en, d)
begin
  if (reset_n = '0') then
    q <= '0';
  elsif (en = '1') then
    q <= d;
  end if;
end process;
```

10.h) The following code generates a latch, why? Draw the circuit described by this code. Propose a solution to remove the latch.

```
process(sel, a, b, c)
begin
  if (sel = "00") then
    z <= a;
  elsif (sel = "01") then
    z <= b;
  elsif (sel = "10") then
    z <= c;
  end if;
end process;
```

10.i) Only one of the following 3 codes will compile. Which one and why?

```
-- code 1
x <= a;
x <= b;

-- code 2
process(a, b)
begin
  y <= a;
  y <= b;
end process;

-- code 3
process(a)
begin
  z <= a;
end process;

process(b)
begin
  z <= b;
end process;
```