h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Prof. A. Upegui

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Start-Up Tutorial to SoPC design tools : Quartus II , Qsys, and Eclipse

Introduction

This documents lists a sequence of operations to be carried out on Altera tools in order to build and program a minimalistic and complete System on Programmable Chip (SoPC).

For doing so you will have to go through three different tools, each one allowing you to specify the different parts of the system :

- Qsys will allow you to specify your SoPC, include a processor, some memories, and some peripheral interfaces.
- Quartus will allow you to add glue logic around your SoPC if required, map inputs and outputs to physical pins on the FPGA, and to launch the synthesis and place & route process in order to end up with a configuration bitstream for configuring your FPGA.
- Eclipse will allow you to program your processor with a C code. For doing so, it includes a compiler and a debugger.

1 Quartus-II

1.1 Create a Quartus-II project

- Launch the Quartus II software either from the menu Applications → Programming → Quartus or from a terminal by typing *quartus*.
- Click on File → New Project Wizard. This will launch the New Project Wizard. An "Introduction" dialogue box may appear. If so, Click *next* to move to the dialogue box for the Name, Directory and Top-Level Entity.
- Select a folder and a name for your project (up to you...). **Keep in mind to avoid spaces in the names of files and folders !**. Click *next*.
- In the "Project Type" window select "Empty project". Click *next*.
- Then you will be asked to add files to the new project. We will not add any file, we will create them later. Click *next*
- Aftherwards, you will be asked to select a device. First you will need to select Cyclone IV E from the Family pulldown. Then, select EP4CE30F23C7 from the "Available devices" list. Just in case, verify the reference on the FPGA on your board. Click *next*.
- In the "Select EDA Tool Settings" window leave the default values. Click *next*.
- At the end you have a summary. Click *finish*.
- You are done ! Your project has been created.

1.2 Create a Block file as top-level

Now we need to create a top-level for our system :

- Select File → New, and then in ”Design files” select ”Block Diagram/Schematic File”
- Save it with the same name of the project.
- Right click on the schematic empty page, and select insert → symbol. It will open a window called symbol that will allow you to insert components to the schematic.
- In the symbol window, expand ”.../libraries”, then expand ”primitives”, then expand ”pin”. Select ”input”. Click OK.
- Place it on the schematic sheet, and modify the input name to Clk_50M
- By following the same procedure, add another input called Reset and an output called LEDS[7..0]. Note that [7..0] refers to an 8 bits bus, with every bit numerated from 7 downto 0.
- Now on the Quartus II menu select again ”File” → New. This time select a ”Qsys System File”. This will launch the Qsys Tool.

2 Qsys

2.1 Setting your input frequency

Your board contains an oscillator at a frequency of 50 MHz. Double click on the ”clk_0” clock source on your ”System Content” tab in order to parameterize it to the desired value : 50000000.

2.2 Build your system

At the left of the window, you find a list of available components to be included in your system. They are grouped by functionality.

2.2.1 Nios processor

Lets start by including the core of our system, the processor. In the component library at the left, expand ”Processors and Peripherals”, then ”Embedded processors” and select ”Nios II Processor”. Double-click on it.

A configuration window allows you to select some options for your processor, through several tabs, have a look at them.

In the ”Main” tab, you can select the type of processor. Select the fast version (Nios II/f)

The ”Vectors” tab allows you to indicate in which address the processor will start after a reset or an exception. We will come back to this later.

The ”Cache and Memory Interfaces” permits you to configure the instruction and data cache size, and their avalon interface when acessing the system memory. You can keep the default values.

The remaining tabs can be kept by default. Anyway have a look at them. and click on ”finish”

Back to the Qsys main window, right click on the Name field on the Nios II processor and choose Rename from the pop up menu. Name it “CPU”.

You will obtain some error messages on the bottom tab. Ignore them for now.

2.2.2 Clock management

The board has a 50 MHz clock, but a complex system may require different clock frequencies and/or shifted clocks in order to ensure a correct synchronization between both devices. FP-

GAs contain PLLs which allow to modify the clock frequencies and adapt such clock phases.

We will setup a DLL that will generate 2 clocks @ 50 MHz : one for internally driving the Nios II processor and its internal components, and a second one for driving the external SDRAM memory.

For doing so, search the component ″ALTPLL Intel FPGA IP″.

In the first configuration window ″Parameter settings″, tab ″General/Modes″ select a speed grade of 7 for your device, select 50 MHz as the input frequency, and keep the remaining settings by default.

Then in the tab ″Inputs/Lock″ uncheck all the options.

Then in the tab ″output clocks″, select for the clk c0 a requested frequency of 50 MHz.

Do the same, with the same frequency for clk c2.

Note for Ubuntu and Debian users : The qsys interface may have problems with unity when inserting the altpll. Two workarounds are proposed :

- When the configuration window of altdll opens, immediately maximize the window. This will prevent the use of the scrollbars which seem to be at the origin of the problem.
- You can use another IP-core which also allows to generate both clocks. For this search the component called ″System and SDRAM Clocks for DE-series Boards″. Set the board selection to DE0-Nano and keep the other parameters unchanged.

2.3 Adding an SDRAM controller

We want to add an external memory in order to have enough space for storing our instructions and data. FPGAs have internal memory resources, but usually they are very limited, and they are intended for more specific uses, as internal FIFOs or caches.

Search for the component ″SDRAM Controller Intel FPGA IP″. The memory controller must fit the the actual memory present in the FPGA board. In the case of our board we have a data width of 16 bits, a single chip select with 4 banks, and the address is decomposed in 12 rows and 9 columns.

The default timing settings are ok for our SDRAM memory. Rename the interface "SDRAM_controller".

2.3.1 LEDs

From the component library, search for ″PIO (Parallel I/O) Intel FPGA IP″ Double-click on it.

Set the "Width" to 8 bits. Ensure that the "Direction" is set to "Output". Click Finish.

Rename the peripheral "LEDs".

2.3.2 Jtag UART

This will permit to use printf commands for debugging, input control commands, log status information, etc. The JTAG UART peripheral connects to the debugger console and is useful for these purposes.

Include a component ″JTAG UART Intel FPGA IP″. The default settings are acceptable. Click Finish. Rename as "jtag_uart".

2.3.3 System ID

From the System Contents include the ″System ID Peripheral Intel FPGA IP″. This interface allows to define a custom system ID for automatic identification of your system. It generates also a timestamp allowing to further verify if your processor is up to date.

2.4 Connecting components

Up to now we have a set of unconnected components (cpu and peripherals). For creating connections you must first point your mouse to the connections column. This will enable the view of a connections matrix where each possible interconnection is represented by an empty circle on every intersection.

By selecting these empty circles you will enable connections (they will become filled circles).

Lets start by connecting the input clk signal to the PLL input clock reference. For doing so, select the node present in the "inclk_interface" port of the DLL to the 50 MHz clock output called "clk" on the top component called "clk_0". Refer to figure 1. Afterwards, connect the PLL output c0 (pre-configured to 50 MHz) to all the components of the system.

Then connect the input reset signal (called clk_reset on the top clk_0 component) to all components. There are two reset sources, from the reset input, and from the jtag interface in the CPU. Each of them can generate a global reset, it can either be done by pushing an external button, or from the Jtag interface every time a new executable is loaded. Connect both resets to every component in the system.

Then connect the avalon buses. Keep in mind that there is an instructions bus and a data bus. The instruction bus must only be connected to the cpu and the SDRAM memory. The data bus must be connected to all the peripherals of the system.

Follow the same procedure for connecting the interruptions on the IRQ column at the right.

On the export column you have the option to export each one of the interfaces thanks to the option "click to export". Exporting a bus means that it will be accessible from the outside of the SoPC. You may realise that clk and reset are already exported.

There are three buses to export in this design. On the LEDs (PIO peripheral) you may export the "external connection" bus.

Export also the c2 port of the PLL. This will allow to set as output the clock generated for the external SDRAM.

Finally, export the SDRAM controller "wire" port. It contains the data and address buses and the control signals for accessing the external SDRAM.

At the end your window should look similar to the figure 1 :

(some differences may be due to the software version)

2.5 Assign Addresses and Interrupt Priorities to peripherals

Platform Designer provides two easy menu items that help clean up address map issues and interrupt priority issues.

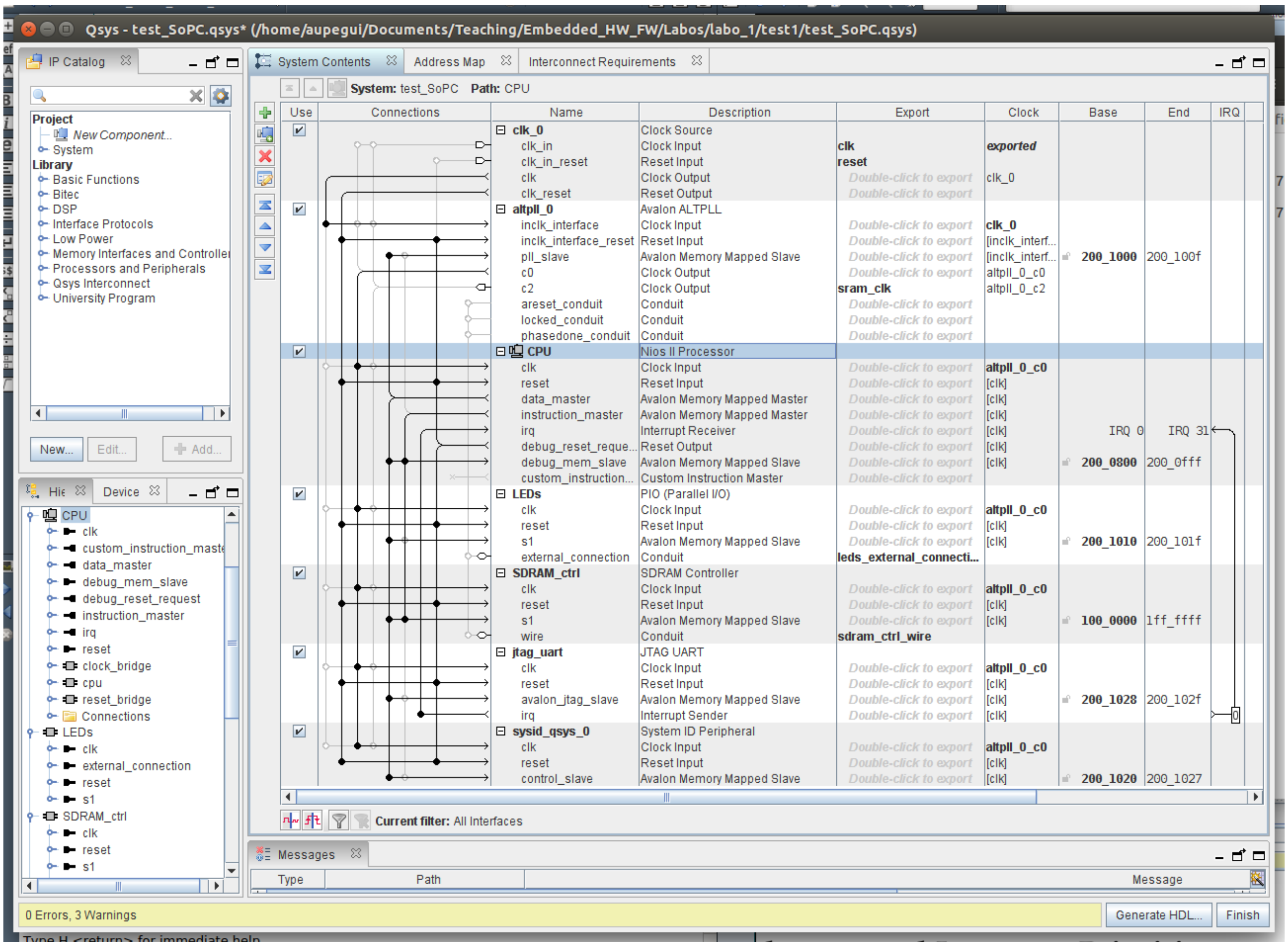
From the System menu, choose Assign Base Addresses. The tool will assign appropriate base addresses for the components by taking their widths into consideration.

Have a look at the "Address Map" tab, in order to observe the addresses assigned to each of your peripherals and memories.

From the System menu, choose Assign Interrupt Numbers. The tool will map IRQs mapping.

2.6 Nios II Boot Configuration

In the event of a reset, the software must begin executing from a predefined memory location. This is set by setting the reset vector. Similarly when a software exception event occurs the software must jump to a pre-defined location where the exception handling software resides. This location is set by setting the exception vector.


Qsys - test_SoPC.qsys* (/home/aupegui/Documents/Teaching/Embedded_HW_FW/Labos/labo_1/test1/test_SoPC.qsys)

System: test_SoPC   Path: CPU

Use	Name	Description	Export	Clock	Base	End	IRQ
✓	clk_0	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double-click to export	clk_0			
	clk_reset	Reset Output	Double-click to export				
✓	altpll_0	Avalon ALTPLL					
	inclk_interface	Clock Input	Double-click to export	clk_0			
	inclk_interface_reset	Reset Input	Double-click to export	[inclk_interf...			
	pll_slave	Avalon Memory Mapped Slave	Double-click to export	[inclk_interf...	200_1000	200_100f	
	c0	Clock Output	Double-click to export	altpll_0_c0			
	c2	Clock Output	sram_clk	altpll_0_c2			
	areset_conduit	Conduit	Double-click to export				
	locked_conduit	Conduit	Double-click to export				
	phasedone_conduit	Conduit	Double-click to export				
✓	CPU	Nios II Processor					
	clk	Clock Input	Double-click to export	altpll_0_c0			
	reset	Reset Input	Double-click to export	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
	debug_reset_reque...	Reset Output	Double-click to export	[clk]			
	debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	200_0800	200_0fff	
	custom_instruction...	Custom Instruction Master	Double-click to export				
✓	LEDs	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	altpll_0_c0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	200_1010	200_101f	
	external_connection	Conduit	leds_external_connecti...				
✓	SDRAM_ctrl	SDRAM Controller					
	clk	Clock Input	Double-click to export	altpll_0_c0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	100_0000	1ff_ffff	
	wire	Conduit	sdram_ctrl_wire				
✓	jtag_uart	JTAG UART					
	clk	Clock Input	Double-click to export	altpll_0_c0			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	200_1028	200_102f	
	irq	Interrupt Sender	Double-click to export	[clk]			0
✓	sysid_qsys_0	System ID Peripheral					
	clk	Clock Input	Double-click to export	altpll_0_c0			
	reset	Reset Input	Double-click to export	[clk]			
	control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	200_1020	200_1027	

Current filter: All Interfaces

0 Errors, 3 Warnings

FIGURE 1 – Qsys screenshot

Right click on the "CPU" and click on "edit" to launch the "Nios II Processor" parameter settings GUI.

On the "Vectors" tab, set the "Reset Vector" to point to the SDRAM with an offset of 0x0. When the Nios II processor comes out of reset, it will begin executing software at this memory location.

Set the "Exception Vector" to point to the SDRAM_controller memory with an offset of 0x20. Under a software exception or interruption, the Nios II will jump to this location in memory.

2.7 Generate the System

You should not have error or warning messages anymore.

Save your Qsys project.

Click on "Generate HDL", select VHDL instead of Verilog and click on the "Generate" button. Platform Designer will now create :

- The HDL for the various components in your system
- System interconnect to connect the components together
- System description file used by the software development tools (the Nios II SBT) to build the software project

Once your system has been successfully generated you will see the info message "Info : System generation was successful". Exit Platform Designer by clicking the Finish button.

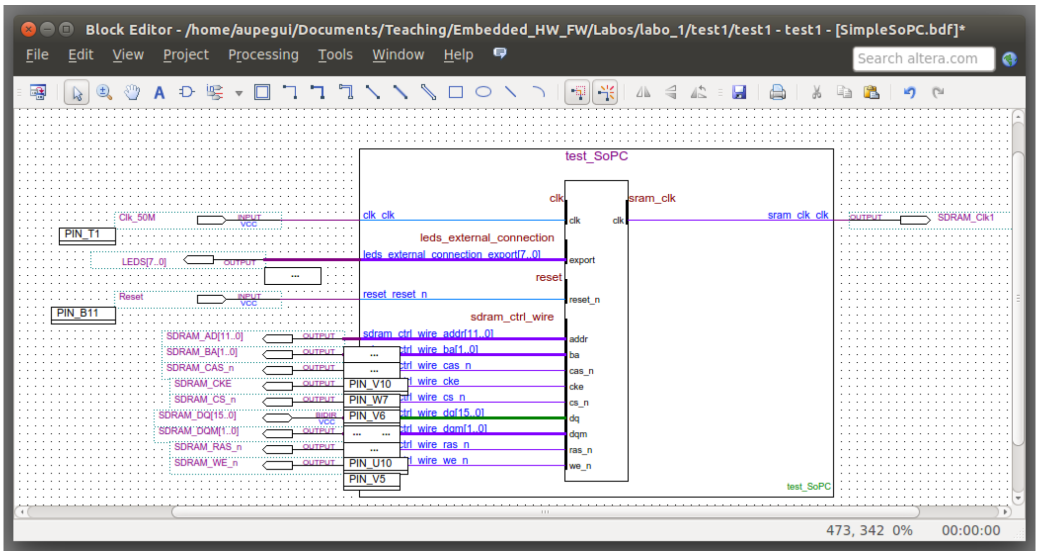


FIGURE 2 – Top level schematic

3 Back to Quartus

3.1 Including your SoPC on your top level

Once the SoPC has been successfully generated we can come back to Quartus in order to include it on our system.

In the "Files" tab, make sure that both, the bdf and the qsys design files are included on the project. If one of them is missing add it.

Go back to the schematic sheet were you had included earlier your inputs and outputs, right-click on the sheet, select insert, select symbol.

Expand the "project" folder and select your SoPC built earlier on Qsys. If not found browse the recently generated symbol. Click OK.

On the schematic sheet, place your SoPC block, and connect the clk, reset, and LEDs to the previously created pins as illustrated in figure 2.

Add the inputs and outputs required to connect the external memory. It is important to keep the same port names and directions as the next figure :

Common error : pay special attention to the bidirectional data bus!!!

3.2 Routing input and output pins

Download the file pins_all.tcl from the moodle webpage. Save it on the folder containing your Quartus project. Have a look at this file.

This tcl script allows to map the pins you defined in your system to the physical pins wired on the board.

For instance the line :

set_location_assignment PIN_T1 -to Clk_50M

means that the input called Clk_50M is directly connected to the physical pin T1 of the FPGA device.

Because of this, you must verify that the names used on your design fit exactly with the names defined in this tcl script.

Now run the tcl script by selecting "Tools" → "Tcl scripts...". On the window "TCL scripts", open the ”Project” folder, select ”pins_all.tcl” and click run.

Open the menu Assignments → Settings... → EDA Tool Settings, and set the Tool Name for Simulation to ¡None¿.

Then you are ready for generating the FPGA configuration file. From ”Processing” select ”start compilation”. This may take some minutes...

If you get an error ”Error : Can’t generate netlist output files. License problem”, verify that you have correctly set the simulator to None as specified above.

If you have a different error, try to fix it...

3.3 Configure the FPGA with your new processor

Connect your FPGA board to your computer through the USB port.

On Quartus run Tools → Programmer.

On the programmer window click on ”Hardware Setup”. On ”currently selected hardware” select USB-Blaster[***]. Then close.

If there is not reference to the freshly generated configuration file .sof on the ”Programmer” window, click on ”Add File”. Select the .sof file.

Back again on the programmer window, click ”Start”.

The ”progress” indicator on the top right of the screen should indicate ”100% (Successful)”.

A pop-up window can appear (if you don’t have a valid licence). **Do not click on CANCEL! Do not close the window!**

Now your are done, the SoPC is on the FPGA, now we have to write some code to be run on it.

4 Eclipse

Launch Eclipse, either from the Quartus menu ”Tools” → ”Nios II software build tools for Eclipse”, or from the menu Applications → Programming → Eclipse.

Select a workspace, if you are asked to. You can keep the default folder.

4.1 Create a BSP and a project

Select File → New → ”Nios II Application and BSP from Template”.

This will open a wizard that will help you to create a new BSP (Base support package) specific to the SoPC that you have created before. It will also help you to create the first project.

On the ”SOPC Information file name” menu, browse for the .sopcinfo file of your SoPC.

Give a name to the application project, for instance call it ”led_counter”.

As template select blank project.

Click on *next*

Give a name to the bsp, for instance led_controller_bsp.

Click on *finish*

Have a look at the files on the BSP. More precisely, at system.h and the files present on the drivers folder.

The functions in there will allow you to interface with your peripherals.

4.2 Adding sources

Click on the application project folder on eclipse. Right-click on it, select "New" → "Source File", name it "counter.c".

Edit the file counter.c with the following program :

```
#include "io.h"
#include <stdio.h>
#include "system.h"

int main()
{
    printf("Lets start counting \n");
    IOWR_8DIRECT(LEDS_BASE,0 ,0);
    int counter = 0;
    while(1)
    { counter ++;
      printf("counter = %d  \n",counter);
      IOWR_8DIRECT(LEDS_BASE,0 ,counter);
    }
}
```

Note : the native bus access functions for accessing a memory address are :

- IOWR_8DIRECT(BASE, REGNUM, DATA) : writes 8 bits of DATA on the memory address (BASE + REGNUM).
- IORD_8DIRECT(BASE, REGNUM) : returns the 8 bits word stored on the memory address (BASE + REGNUM).

4.3 Compile and debug

Now build the BSP project by selecting the BSP, right-click, and select "Build project".

Then, build the application project by selecting the application folder, right-click, and select "Build project".

Once compilation has been performed successfully, you are ready to run or debug the program. For debugging, right-click on the application, select "debug as" → "nios II hardware".

Sometimes the board is not recognised immediately. In this case, open the "Target Connection" tab on the "Debug Configuration" window, and click on "refresh connections". Sometimes you'll have to click several times... Afterwards click on "debug"

Now you can add some breakpoints on your code and have fun debugging it step by step !

You may also run the program without debugging. For doing so right-click on the software project directory and choose Run As and Nios II Hardware.

In order to get a smooth controlled counting, you can add a timer to your qsys and make it increase every $10ms$.