Bern University
of Applied Sciences

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

MSE - TSM_EmbHardw

Start-Up Tutorial to SoPC Design Tools

TABLE OF CONTENTS

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).

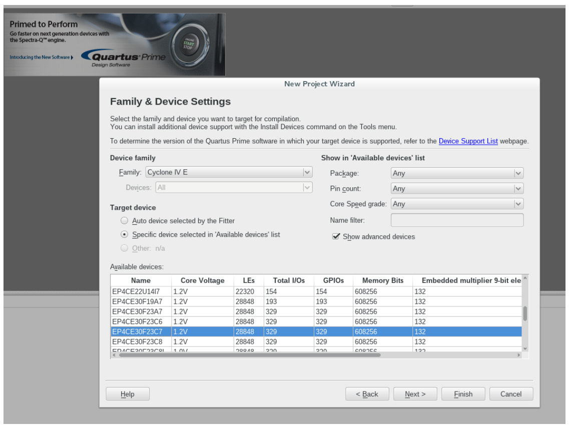
For doing so you will have to go though three different tools, each one allowing you to specify the different parts of the system:

- 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 firmware written in C. For doing so, it includes a compiler and a debugger (the cross-toolchain).

1 Quartus-II

1.1 Create a Quartus-II project

- Launch the Quartus II software either from the icon on the desktop or from a terminal by typing *quartus*[1].
- 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 the project (i.e. lab1GetStarted). **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*
- Afterwards, you will be asked to select a device. First you will need to select Cyclone IV E from the Family pulldown menu. 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! The project has been created.


New Project Wizard

Family & Device Settings

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family

Family: Cyclone IV E

Devices: All

Target device

( ) Auto device selected by the Fitter

(●) Specific device selected in 'Available devices' list

( ) Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core Speed grade: Any

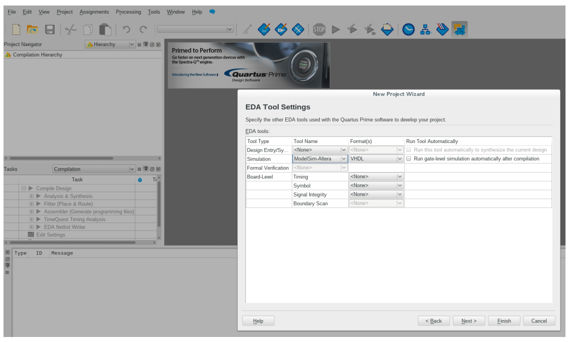
Name filter:

[✓] Show advanced devices

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit ele
EP4CE22U14I7	1.2V	22320	154	154	608256	132
EP4CE30F19A7	1.2V	28848	193	193	608256	132
EP4CE30F23A7	1.2V	28848	329	329	608256	132
EP4CE30F23C6	1.2V	28848	329	329	608256	132
EP4CE30F23C7	1.2V	28848	329	329	608256	132
EP4CE30F23C8	1.2V	28848	329	329	608256	132

Figure 1: Select correct target device


New Project Wizard

EDA Tool Settings

Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Sy...	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	Run gate-level simulation automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figure 2: If no simulator is set you will not get any warning at the p&p process (the error/warning is due to missing license)

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*

[1]Depending on the setup quartus is located in "/opt/altera/VER/quartus". Maybe there is a settings file in "/opt/altera" if so source it.

- By following the same procedure[2], add another input called *nReset* 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”. The three ports will look like the ones shown in figure 3
- Now on the Quartus II menu select again "File" → New. This time select a "Qsys System File". This will launch the Qsys Tool.

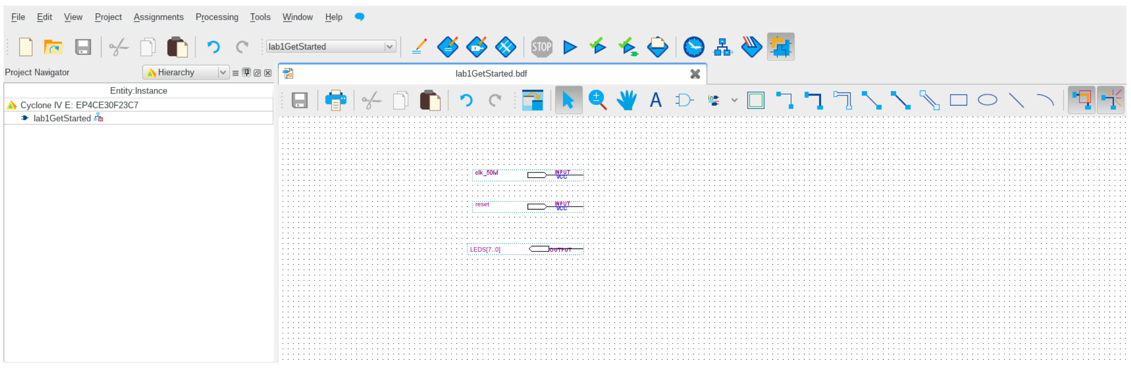


Figure 3: The three added IO ports

2 Qsys

After launching the Qsys tool you are ready for building up your particular SoPC system. To do so follow the steps below.

2.1 Setting the system 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 the 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 to them.

In the "Main" tab, one 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 accessing the system memory. You can keep the default values.

All the remaining tabs can also kept by default. Anyway have a look to them. and then click on "finish"

[2]There is a shortcut for placing IOs in the toolbar of block diagram drawing tool

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

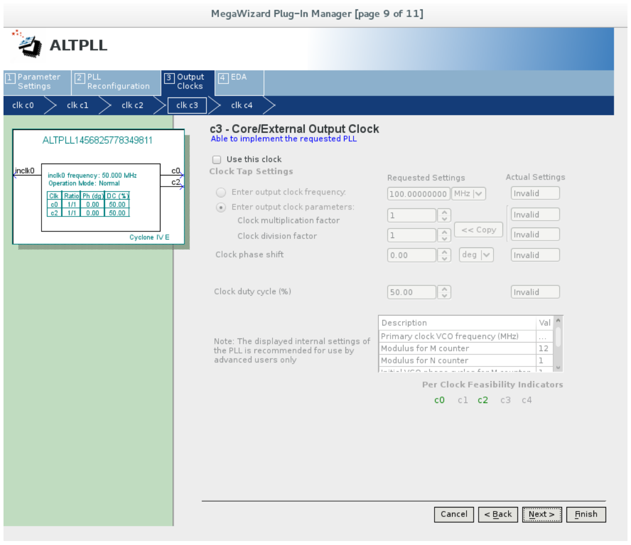
We will setup a PLL that will generate 2 clocks **@50 MHz**: one for internally driving the Nios II processor and its internal components, and a second one for driving the external SDRAM memory[3].

For doing so, search the component "Avalon ALTPLL"[4].

Note for Ubuntu and Debian users: The qsys interface may have problems with X-System when inserting the “altpll”. Two workarounds are proposed:

- When the configuration window of altdll opens, immediately resize 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.

1. 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.
2. Then in the tab "Inputs/Lock" un-check all the options.
3. Then in the tab "output clocks", select for the clk c0 a requested frequency of 50 MHz.
4. Do the same, with the same frequency for clk c2.


MegaWizard Plug-In Manager [page 9 of 11]

ALTPLL

1 Parameter Settings | 2 PLL Reconfiguration | 3 Output Clocks | 4 EDA

clk c0 | clk c1 | clk c2 | clk c3 | clk c4

ALTPLL1456825778349811

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00
c2	1/1	0.00	50.00

Cyclone IV E

c3 - Core/External Output Clock
Able to implement the requested PLL

Use this clock
Clock Tap Settings

	Requested Settings		Actual Settings
Enter output clock frequency:	100.00000000	MHz	Invalid
Enter output clock parameters:			
Clock multiplication factor	1	<< Copy	Invalid
Clock division factor	1		Invalid
Clock phase shift	0.00	deg	Invalid
Clock duty cycle (%)	50.00		Invalid

Note: The displayed internal settings of the PLL is recommended for use by advanced users only

Description	Val
Primary clock VCO frequency (MHz)	...
Modulus for M counter	12
Modulus for N counter	1

Per Clock Feasibility Indicators
c0 c1 c2 c3 c4

Figure 4: ALTPLL settings

[3]ERRATUM: When using the PLL for generating internal and SDRAM clock signals use C0 for internal and C2 for external clock. Please avoid using C1. C2 is directly connected to the output clk buffer due to this we use C2 for SDRAM clock. Not following this rule ends in malfunctioning of memory access.

[4]There is a tool-chain bug follow description in this section for more detail.

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". 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_ctrl”.

2.3.1 LEDs

From the component library, search for "PIO (Parallel I/O)" Double-click on it to add this IP as well.

- Set the “Width” to 8 bits. Ensure that the “Direction” is set to “Output”.
- 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".
- The default settings are acceptable.
- Rename as “jtag_uart”.

2.3.3 System ID

From the System Contents include the "System ID Peripheral". 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.

- Rename it to sysid[5]

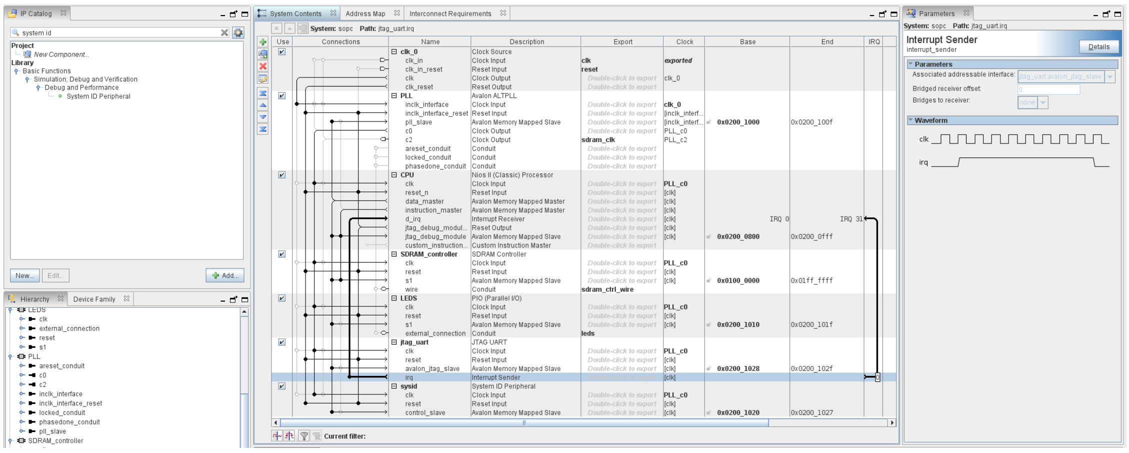
2.4 Connecting components

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 allowed 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 PLL to the 50 MHz clock output called "clk" on the top component called "clk_0". Refer to the figure in the next page. Afterwards, connect the PLL output c0 (pre-configured to 50 MHz) to all the components of the system.

[5] You can add only one system ID core to an SOPC Builder system, and its name is always **sysid**. `https://www.altera.co.jp/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii51014.pdf`



System: sopc Path: jtag_uart.irq

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				
✓	PLL	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...	0x0200_1000	0x0200_100f	
	c0	Clock Output	Double-click to export	PLL_c0			
	c2	Clock Output	sdram_clk	PLL_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 (Classic) Processor					
	clk	Clock Input	Double-click to export	PLL_c0			
	reset_n	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]			
	d_irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
	jtag_debug_modul...	Reset Output	Double-click to export	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0200_0800	0x0200_0fff	
	custom_instruction...	Custom Instruction Master	Double-click to export				
✓	SDRAM_controller	SDRAM Controller					
	clk	Clock Input	Double-click to export	PLL_c0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0100_0000	0x01ff_ffff	
	wire	Conduit	sdram_ctrl_wire				
✓	LEDS	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	PLL_c0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0200_1010	0x0200_101f	
	external_connection	Conduit	leds				
✓	jtag_uart	JTAG UART					
	clk	Clock Input	Double-click to export	PLL_c0			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0200_1028	0x0200_102f	
	irq	Interrupt Sender	Double-click to export	[clk]			0
✓	sysid	System ID Peripheral					
	clk	Clock Input	Double-click to export	PLL_c0			
	reset	Reset Input	Double-click to export	[clk]			
	control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0200_1020	0x0200_1027	

Figure 5: System overview Qsys

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, connect both resets to all the components of 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, the SDRAM memory, and the jtag_uart core. 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.

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 realize that clk and reset are already exported.

There are three interfaces to export in this design.

- On the LEDs (PIO peripheral) you export the "external connection" bus (connection to the LEDs).
- Export also the c2 port of the PLL. This will allow to set as output the clock generated for the external SDRAM.
- 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 the system look similar to the one shown in figure 5.

2.5 Assign Addresses and Interrupt Priorities to Peripherals

SOPC Builder provides two easy menu items that help clean up address map issues and interrupt priority issues. From the System menu, choose Auto-Assign Base Addresses. The tool will assign appropriate base addresses for the components by taking their widths into consideration.
Have a look to the "Address Map" tab, in order to observe the addresses assigned to each of your peripherals and memories.
From the System menu, choose Auto-Assign IRQs. 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.

Double click on the "CPU" to launch the “Nios II Processor Parameter Settings” GUI.

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_ctrl 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

Click on "Generate" "Generate HDL", select VHDL instead of Verilog and click on the "Generate" button. SOPC Builder 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 SOPC Builder by clicking the Exit button and click Save when it asks if you'd like to save the system.

3 Back to QuartusPrime

3.1 Including the SoPC on the Top-Level

Once the SoPC has been successfully generated we can come back to QuartusPrime in order to include the SoPC as a block in our block-diagram.
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 the inputs and outputs, right-click on the sheet, select insert ↪ select symbol.
Expand the "project" folder and select the SoPC built earlier on Qsys. *If not found browse the recently generated sopc sub-folder.* Click OK to finish the action.
On the schematic sheet, place your SoPC block, and connect the clk, reset, and LEDs to the previously created pins.
Add the inputs and outputs required to connect the external memory. It is important to keep the same port names and directions as shown in figure 6. To add missing ports. Click on SoPC block (to select it). Next, right-click and choose “Generate Pins for Symbol Ports”. Rename them as shown in figure 6. The names are in the “pins_all.tcl” file, also.

3.2 Routing input and output pins

Create a file named pins_all.tcl and add it to the project. The TCL file will be used to assign correct location to the ports. To get the necessary information browse to the board documentation. The documentation is public available on HuCE Wiki. The URL is encoded by the QR-Code, which is located at front cover of the development board's box.
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 the design fit exactly with the names defined in this tcl script.

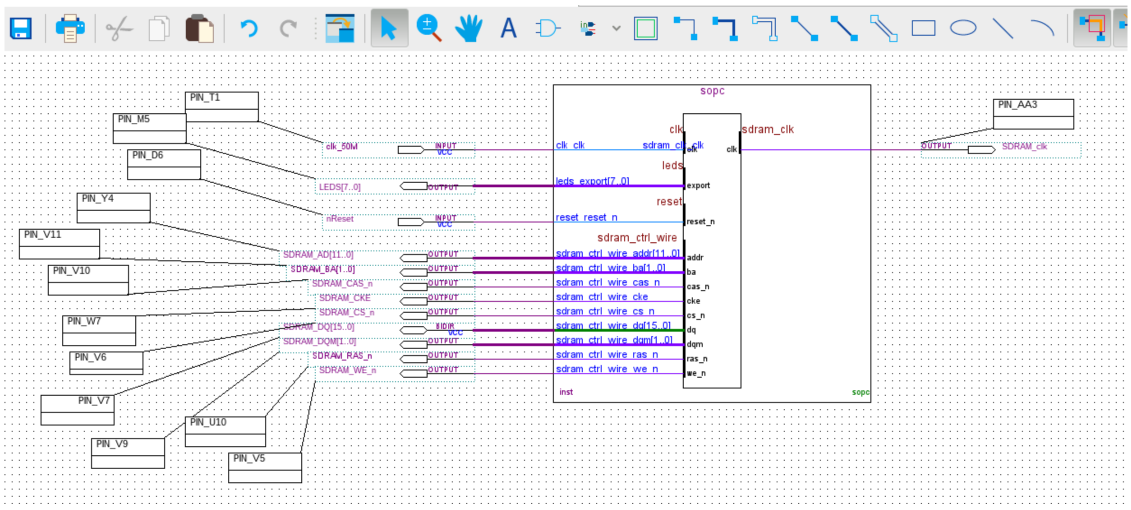


Figure 6: System top-level interface

```
set_location_assignment PIN_T1 -to clk_50M
set_location_assignment PIN_D6  -to nReset

set_location_assignment PIN_M5 -to LEDS[0]
set_location_assignment PIN_N6 -to LEDS[1]
set_location_assignment PIN_N5 -to LEDS[2]
set_location_assignment PIN_P6 -to LEDS[3]
set_location_assignment PIN_P5 -to LEDS[4]
set_location_assignment PIN_R6 -to LEDS[5]
set_location_assignment PIN_R5 -to LEDS[6]
set_location_assignment PIN_T5 -to LEDS[7]

set_location_assignment PIN_AA3  -to SDRAM_clk

set_location_assignment PIN_Y4   -to SDRAM_AD[0]
set_location_assignment PIN_Y3   -to SDRAM_AD[1]
set_location_assignment PIN_W6   -to SDRAM_AD[2]
set_location_assignment PIN_Y6   -to SDRAM_AD[3]
set_location_assignment PIN_Y8   -to SDRAM_AD[4]
set_location_assignment PIN_W10  -to SDRAM_AD[5]
set_location_assignment PIN_W8   -to SDRAM_AD[6]
set_location_assignment PIN_AA4  -to SDRAM_AD[7]
set_location_assignment PIN_Y10  -to SDRAM_AD[8]
set_location_assignment PIN_Y7   -to SDRAM_AD[9]
set_location_assignment PIN_U7   -to SDRAM_AD[10]
set_location_assignment PIN_AA5  -to SDRAM_AD[11]
set_location_assignment PIN_AB4  -to SDRAM_AD[12]

set_location_assignment PIN_V7   -to SDRAM_DQ[0]
set_location_assignment PIN_T8   -to SDRAM_DQ[1]
set_location_assignment PIN_U8   -to SDRAM_DQ[2]
set_location_assignment PIN_T9   -to SDRAM_DQ[3]
set_location_assignment PIN_V8   -to SDRAM_DQ[4]
set_location_assignment PIN_T10  -to SDRAM_DQ[5]
set_location_assignment PIN_U9   -to SDRAM_DQ[6]
set_location_assignment PIN_T11  -to SDRAM_DQ[7]
set_location_assignment PIN_AA7  -to SDRAM_DQ[8]
set_location_assignment PIN_AA8  -to SDRAM_DQ[9]
set_location_assignment PIN_AB7  -to SDRAM_DQ[10]
set_location_assignment PIN_AA9  -to SDRAM_DQ[11]
set_location_assignment PIN_AB8  -to SDRAM_DQ[12]
set_location_assignment PIN_AA10 -to SDRAM_DQ[13]
set_location_assignment PIN_AB9  -to SDRAM_DQ[14]
set_location_assignment PIN_AB10 -to SDRAM_DQ[15]

set_location_assignment PIN_V9   -to SDRAM_DQM[0]
set_location_assignment PIN_AB5  -to SDRAM_DQM[1]

set_location_assignment PIN_V11  -to SDRAM_BA[0]
set_location_assignment PIN_U11  -to SDRAM_BA[1]

set_location_assignment PIN_W7   -to SDRAM_CKE
set_location_assignment PIN_V6   -to SDRAM_CS_n
set_location_assignment PIN_U10  -to SDRAM_RAS_n
set_location_assignment PIN_V10  -to SDRAM_CAS_n
set_location_assignment PIN_V5   -to SDRAM_WE_n
```

Now run the tcl script by selecting "Tools" $\rightarrow$ "Tcl scripts...". On the window "TCL scripts", open the "Project" folder, select "pins_all.tcl" and click run.
Then you are ready for generating the FPGA configuration file. From "Processing" select "start compilation". This may take some minutes...

3.3 Configure the FPGA with your new processor

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

- On Quartus run Tools $\rightarrow$ Programmer.
- On the programmer window click on "Hardware Setup". On "currently selected hardware" select USB-Blaster[***]. Then close.
- 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

Lets come back to Qsys, and from there click on "Tools" $\rightarrow$ "Nios II software build tools for 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 $\rightarrow$ New $\rightarrow$ "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 to the files on the BSP. More precisely, have a look on **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" $\rightarrow$ "Source File", name it "counter.c"[6].

Edit the file *counter.c* with the following program:

[6] I prefer an additional sub-folder called "src" to store the source files. You may want to create such a folder too.

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

#define DELAY 1000000

int main(void)
{
  int counter = 0;
  unsigned int wait;

  printf("Lets start counting \n");
  IOWR_8DIRECT(LEDS_BASE,0,0);

  while(1)
  { counter ++;
    printf("counter = %d \n",counter);
    IOWR_8DIRECT(LEDS_BASE,0,counter);
    // silly busy wait
    for(wait = 0; wait < DELAY; wait++)
              asm volatile ("nop");
  }

}
```

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" $\rightarrow$ "nios II hardware".

Sometimes the board is not recognized 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.