TÉCNICO LISBOA

Projecto e Controlo em Lógica Digital

5th Laboratory

Microprocessor

The objective of this assignment is to implement a microprocessor inside the FPGA of the DE2 board with the aim to use it as an interface of the Oscilloscope. I tis intended to implement a NIOS II to run a small example program. This program should read the state of the switches and activate the LEDs accordingly. Moreover the state of the switches should be shown in the PC that is connected to the DE2 board.

Generically the steps are:

- Implement a microprocessor (and auxiliary hardware) using the QSYS tool of Quartus II.
- Program the microprocessor firmware in the DE2 board
- Make a small software project using the "NIOS II Software Build Tools for Eclipse"
- Write a small program in C that performs the required tasks.

The use of a microprocessor inside a FPGA consumes resources and, being a sequential system, it doesn't have the performance of a system implemented in Verilog. On the other hand it facilitates quite a lot the use of peripheral devices that are intrinsically controlled sequentially.

This implementation is somewhat time consuming being necessary to configure a vast set of parameters. To minimize the time it is given a recipe based in Altera tutorials. The recipe is tested for the DE2 board using Quartus II and Eclipse version 12.

The tutorials used are:

- "Introduction to the Altera Qsys System Integration Tool"
  Altera corporation
  ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/Introduction_to_the_Altera_Qsys_Tool.pdf
- " Using the SDRAM Memory on Altera's DE2 Board with Verilog Design"
  Altera Corporation
  ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_DE2_sdram_verilog.pdf
- "My First Nios II Software Tutorial"
  Altera corporation
  http://www.altera.com/literature/tt/tt_my_first_nios_sw.pdf

Tutorial for the system implementation

Start of the project

Start with a clean DE2_TOP. Beware that some tools do not allow to have spaces (and "strange characters") in the directory path. Suggestion: create in the c: a folder with a "simple name" and develop there the project.

Open the project in Quartus II

Defining the NIOS II system using the Qsys tool

Launch the Qsys tool from the "tools" menu. A new project should appear with only a clock.

Go to the folder "Clock Settings" and make sure that there exists a clock; external; 50 MHz.

Go back to the folder "System Contents".

Specify the Microprocessor

In the "Component Library" expand "Embedded Processors";

Select "NiosII processor"; click in "Add"[1]

Choose "Nios II/s"

In "Hardware Arithmetic Operation" choose "none" for "Hardware Multiplication Type" and don't select "Hardware Divide".

Click in "Finish" to go back to the main window.

Implement the SDRAM module

Add: "Memories and Memories controllers"→"External Memory Interfaces"→"SDRAM interfaces"→"SDRAM Controller"

In the configuration window choose:

- "Presets": "Custom"
- "Data Width": 16 bits
- Leave the default values for the others.
- Click on "Finish"

Implement the parallel ports

Add: "Peripherals"→ "Microcontroller Peripherals" → "PIO (Parallel I/O)"

In the configuration window choose:

- "Width": 8
- "Direction": "input"
- Click on "Finish"

Since this is an input port, change the name in the main window to "switches".

Add: "Peripherals"→ "Microcontroller Peripherals" → "PIO (Parallel I/O)"

In the configuration window choose:

- "Width": 8
- "Direction": "output"
- Click in "Finish"

Since this is an output port, change the name in the main window to "Leds".

Implement JTAG UART

This is a serial interface that will allow to communicate with the PC through the DE2 programming cable. One could implement other protocols like the RS232…

Add: "Interface Protocols" → "Serial" → "JTAG UART"

Do not change any configuration and click "Finish".

---

[1] Some errors should appear in the configuration window as we don't have yet the memory in which the microprocessor code will run. This will be made further down.

Connections

In the main window we have now all the components but the connections between them are still missing. Typically, the clock of all components is the "clock output" from the clk_0. The reset comes from the "reset output" of clk_0 and from the "reset output" of the "jtag debug module" (this is inside the NIOS II component! Do not mistake it with the JTAG UART).

- Connect the "clock output" of clk_0 to:
  - "clk" of nios II
  - "clk" of sdram
  - "clk" of leds
  - "clk" of switches
  - "clk" of jtag uart
- Connect the "reset output" of clk_0 and the "reset output" of "jtag debug module" to:
  - "reset" of nios II
  - "reset" of sdram
  - "reset" of leds
  - "reset" of switches
  - "reset" of jtag uart
- Connect "s1" from the memory to the "data master" and "instruction master" of niosII
- Connect "s1" of the leds and switches to the "data master" of nios II
- Connect "avalon jtag slave" of "JTAG UART" to "data master of nios II
- Connect the IRQ line of the JTAG UART to Nios II (select in the IRQ column) using port 5

Define addresses

One of Qsys columns is called "base". It contains the base values of the memory regions. It can be configured manually or by the system.

Configure the SDRAM memory to have as base address 0x00000000.

Block this address by clicking on the "lock"

Go to "System" and choose "Assign Base Addresses"

Take note of the given values. In particular the ones for the LEDs and switches.

Configure the Microprocessor memory

The NIOS II microprocessor uses two vectors to know where the instructions to be executed are. They are the "reset vector" and the "exception vector". They are implemented in the SDRAM. Now that the memory addresses are assigned we can configure NIOS II.

- Edit the Nios II processor
- In "Reset Vector" choose
  - "Reset Vector memory": sdram_0.s1
  - Do not change the default values neither for the offset nor for the reset vector (It should correspond to the base address of the memory)
- In "Exception Vector choose
  - "Exception Vector Memory": sdram_0.s1
  - Do not change the default values neither for the offset nor for the reset vector (It should correspond to the base address plus the offset of the memory)[2]
- Click on "finish"

Export ports to the FPGA (for the DE2_TOP)

It is necessary to export the ports corresponding to the LEDs, to the switches and to the SDRAM for the project. In the export column it should appear greyed out "click to export"…

- Click on "click to export" on the line "external connection" of the "switches" port and give it a name, e.g. "switches".
- Click on "click to export" on the line "external connection" of the "leds" port and give it a name, e.g. "leds".

[2] After this configuration the memory errors should disappear.

- Click on “click to export” on the line “wire” of the “sdram_0” port and give it a name, e.g. “sdram”.

The given names will appear as port names of the Verilog module that will be created by the tool.

Generate the verilog module

Save the system. Make sure you are saving it in the correct directory

Next, go to the folder “generation”. Select “none” in “Create Simulation Model” and in “Create testbench Qsys system”

Click on “Generate”[3]

Exit Qsys and go back to Quartus II

Integrate the system in DE2_Top

Generate the necessary clock for the SDRAM

The clock skew (phase difference) depends on the physical properties of the DE2 board. For the correct operation of the SDRAM chip it is necessary that its clock, DRAM_CLK, precedes the NIOS II system clock, CLOCK_50, by 3 nanoseconds. This can be achieved by using a PLL. Use the “Mega Wizard” to create a PLL that has as input the 50 MHz clock and as outputs two clocks with the same frequency: The first one advanced by 3 ns (clock for the SDRAM) and the second one in phase with the input clock (clock for the NIOS II).

Instantiate the module generated by Qsys

In Quartus II open the generated verilog file. At this point the file should not be included in the project. Open it from the directory to where it was generated. The file should be inside of a subdirectory of DE2_TOP. For instance nios_system/synthesis/nios_system.v

Identify the module definition and the ports of the module. The necessary connections to be made should be obvious except for two details on the SDRAM signals:

- “sdram_ba” is a 2 bit vector on the NIOS system while in the DE2_TOP there are two one bit signals (DRAM_BA_1 and DRAM_BA_0). Concatenate them.
- “sdram_dqm” is a 2 bit vector on the NIOS system while in the DE2_TOP there are two one bit signals (DRAM_UDQM and DRAM_LDQM). Concatenate them.

Don’t forget to connect the SDRAM clock and the NIOS clock to the outputs of the PLL. Don’t forget the Reset, the LEDs and switches ports. Remove the assignment from DRAM_DQ to high impedance.

Compile the project. (It might be necessary to add the Qsys file to the project. In such case use nios_system.qsys).

Program the DE2 board with the project. Quartus should give you a warning stating that the project is using a core with a limited license.

Accept and leave the window open so you can continue using the core. Do not exit Quartus.

Software application

Open NIOS Software Build Tools for Eclipse.

Create a directory for Eclipse Workspace under the project directory.

[3] If necessary, the system can be reconfigured by reopening the Qsys.

Create the "Board Support Package"

The Board Support Package is the part of the software that will contain information about the NIOS system implemented in the FPGA.

File→New→Nios II Board Support Package

- Give a name to the project. E.g. soft_bsp
- In "SOPC Information file name" choose the file ".sopcinfo" that was generates. In this case the "nios_system.sopcinfo". When the file is read it should appear in "CPU" the implemented cpu (nios2_qsys_o in this case)
- Leave the default values and click finish.

Create the application

File→New→Nios II application

- Give the project a name. E.g. soft
- In BSP location search for the generated bsp (soft_bsp)
- Leave the default values and click finish

In the project explorer click with the right button in the application (soft) and do New→Source File.

Name the file, e.g. app_main.c

Click in finish.

Create the C code

The LEDs and the switches behave as memory spaces that can be written and/or read. In this case the memory addresses are defined by the Qsys tool. (e.g. on the test compilation the LEDs are in 0x00801000 and the switches are in 0x00801010. Each with 8 bits, a byte, which corresponds to char.

Start by defining the LEDs and switches:

```
#define switches (volatile char*) 0x00801010
#define leds (char*) 0x00801000
```

In the file app_main.c create the function main which will be executed by the processor:

```
int main (int argc, char** argc1, char** argv2)
```

Next, create a continuous loop, e.g. while(1), in which the content pointed by the variable switches is copied to the memory space addressed by the variable leds:

```
*leds=*switches;
```

Save the file![4]

Compile and run

Click with the right button on top of the application (soft) and "build project"

Click with the right button on top of the application (soft) and:

"Run As → Nios II Hardware"

[4] Be aware that the Eclipse might not warn you that the files need to be saved and it will compile without your latest changes.

When the switches are “changed” the LEDs should also change. This means that the LEDs should reflect the state of the switches. You can also try a getchar and a printf inside the while loop. The NIOS console behaves as stdout e stdin…