Xilinx Platform Studio tutorial

Per.Anderson@cs.lth.se

April 12, 2005

This tutorial intend to show you how to create an initial system configuration. From Xilinx Platform Studio(XPS) version 6.1 this has been significantly simplified due to the system creation wizard.

note During the synthesis process large files will be created. This process is much faster if you store the project in a local drive. Do not forget to copy your project to your home directory at the end of the lecture.

note To save disk space you can remove temporary files from the project by **tools→clean→all** .

1 Creating a system architecture from scratch

This tutorial will guide you through the creation of a new design using Xilinxs Platform Studio(XPS). XPS is a GUI that helps you to specify your system, i.e.

- which processors, memory blocks and other FPGA IPs(peripherals) to use
- how the different IPs are connected
- the memory map, i.e. for addresses for memory mapped IO/peripherals

XPS also interface the tools used throughout the whole design flow. In this tutorial you will create a system consisting of three components, a MicroBlaze processor, a uart(serial port), and a memory block. The system functionality is just to write a message to the uart.

1.1 Preparations

Before you can start designing your system you need to download a configuration file for the boards we will use. The file, called XPS_library.zip, can be found on the lab home page, http://www.cs.lth.se/EDA380/labs/. Download it and unzip in a location of your choice. The path may not contain spaces!

1.2 Start Xilinx Platform Studio(XPS)

You can find the program in Windows start menu. **start→programs→ Xilinx Platform Studio 6.3i→Xilinx Platform Studio**

1.3 Create a new design

When you start XPS there will be a dialog from which you can start the base system builder wizard. It can also be launched from **Files→New Project→Base System Builder...** In the first dialog, specify the project location and name(the path cannot contain spaces). XPS will create a lot of files in the project directory, so create a new empty directory. During synthesis there will be a lot of file accesses, so to avoid file server overload, you should place the project on the local drive(c:). Do not forget to copy the files to your home directory at the end of the session. In the lower half of the dialog check the box "User Peripheral Repository search path..." and point to the XPS_library you unziped earlier. Click OK, select "I would like to create a new design", Click Next. Set the target board as follows:

Board Vendor	Memec Design
Board Name	Virtex-II V2MB1000 Development board
Board Revision	3

If you can not see Memec as vendor, either your paths contain spaces, or you selected the wrong user lib path above (select the directory above Memec_Design). Click Next. MicroBlaze is the only processor option, so click Next. keep the 100MHz clock, but you can remove the debug interface, Next. Now you can choose which board components(leds, buttons et.c.) you want to create interfaces. Keep the leds and buttons, but skip the DDR_SDRAM on the second page, next. You do not need extra peripherals for now, next. Keep the RS232 as std in and out. This is where the C IO functions, i.e. `printf()` and `scanf()`, read/write its data, next. Now you are done, click Generate and Finish.

Now you have created your initial system. All you need to do now is compile and run logic synthesis, as will be explained later. Now is a good time to explore XPS a bit. There are three main frames in the application window, the left frame let you navigate trough your system. In the "System" tab you find all hardware components, and in the "Application" tab you find all software (under source). There is a default source file, double click on it to have it opened in the right frame. Now go back to the "System" tab. An overview of the system is opened if you double click on the "PBD File: system.pbd" under "Project Files". To get a more detailed view of your system you can use the Add/Edit Cores dialog. This is found in the Project menu. There are several tabs that gives you all the system details. Investigate the different tabs and reflect on how the system looks. There are three buses, how are they connected. Hint, the Local Memory Bus (LMB) can only connect the processor and a memory block, the On-chip Peripheral Bus(OPB) is a general bus used to connect the CPU and peripherals. Also note that the memory blocks (BRAM) are dual ported and they are not directly connected to the bus. The controller to memory port connection can be found in middle right hand side of the "Bus Connections" tab.

2 Software compilation

There are two types of software, libraries and user programs. To generate the libraries select (**Tools→Generate Libraries**). This step also generates header files containing symbolic names for memory mapped ports/signals. Check in

the left frame, application tab, click on the + left of "processor:microblaze_0". There you will find a include file (xparameters.h) with symbolic names of all memory mapped peripherals/IO. Use the symbolic names in your C code, so you can change the memory map later without changing the C code.

The user programs are compiled in a similar way, **Tools→Build All User Applications** . You might want to remove the comment at the print outs in the source file. Note, the standard C function `printf` generates huge libraries, which will not fit in the memory. Instead use `print`, or `xil_printf` function to print. `xil_printf` is similar to `printf`, but much smaller and lacks some functions, i.e. floating point support.

3 Hardware Synthesis

Hardware synthesis is done in several steps(create the logic circuit (netlist), create the FPGA configuration (bitmap)), but you do not need to worry about this. Simply select **Tool→Generate Bitstream** and everything will be generated automatically for you. This will take long time, 5-15 minutes. During this step the tools use a User Constraint File(UCF), which can be found in the left frame in the system tab. This file bounds nets to pins and the names must be the same as in the port configuration of the peripherals.

3.1 Merge software and Hardware flows

To merge software binaries and the bit stream from the hardware synthesis run **Tools→Update Bitstream** . This places the executables in the block ram.

3.2 Run the final implementation on the FPGA

Start a windows hyperterminal **start→programs→ accessories→communications→hyperterminal** . Give it a name and chose *com1* in the *Connect using* frame. Use port settings according to the uart parameters:

Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

This is the configuration of the RS232 peripheral, feel free to change the speed.

Now you are about to run rout system and you need a FPGA board, connect it and turn it on. Download the bitstream, **Tools→Download** . Now the system starts to run. It blinks some leds and if you added some printouts you should get the text in the hyperterminal. If not, add a print in the source file and download again. Press the button labelled *RESET* on the FPGA board to restart the program (will give a new printout).

XPS keeps track of most dependencies among the source files, so if you change anything downloading to the FPGA will rebuild all parts that are affected. You can force XPS to rerun a step by choosing **Tools→Clean→...** .

4 That was simple, what is next

Now you have created and run your first system. Use the time on the first lab to explore the XPS program. The following labs will contain much more work and you will not be able to finish unless you have prepared before, i.e. created the system. In the next lab you will need a system consisting of a MicroBlaze, a RS232 (uartlite) and a timer (opb_timer). If you have time you can start to create this system. Read the documentation of the timer. To find the documentation, add a timer to a design(in the Add/Edit Hardware dialog). Go to the "Parameters" tab and chose the timer. Click on the "Open PDF Doc" button.

5 Dual processor and FSL communication

Lets continue by creating a dual processor system, this will be needed in lab 3. Start by creating a system containing only one MicroBlaze and a RS232(uartlite) using the basic system build wizard. To add a second processor you need to manually duplicate all components and parameters in the processor/memory subsystem. Use the Add/Edit Cores dialog. In the peripherals tab, add one more instance of each peripheral except the opb_uartlite (2 `microblaze`, 2 `bram_block`, 4 `lmb_bram_if_ctrl`(this is not the bus, but the logic between the memory and the bus)). Name them so you can differentiate the two subsystems.

Continue to the "Bus connections" tab. The OPB should be shared between the processors, but you need two more LMBs. Create them and connect the right processor ports(here d is for data and i is for instruction). You also need to connect the added LMB controller ports to the new BRAM ports.

Set the address space in the next tab. The processor boots from address 0x00000000, so this address must be in accessable from the ilmb port of the processor. Use the same memory addresses for the local memory for both processor.

Next continue to the ports. For the added LMB buses and MicroBlaze you must connect the clock and reset ports to the nets called `sys_clk_s` and `sys_rst_s`. Make sure to use the right net names. The default names will not work.

In the last tab you must set the parameter `C_EXT_RESET_HIGH` to 0 for the added buses(LMB).

Finally you also need a program to run on the second processor. In the left frame, select the "Applications" tab. Right click on "Software Projects" and add a new project for the new processor. If you want to use STDIN or STDOUT you must bind them in the "Library/OS Parameters" tab, "Software Platform Settings" dialog, in the Project menu. This is a simple way to check so both processors are live, but be warned, if you have concurrent access to the same uart from the two processors, data will be lost.

5.1 FSL communication

When that both CPUs are running it is time to do some useful things. In the current configuration the processors can not communicate. For this purpose

we use Fast Simplex Links (FSLs). FSL is a point to point bus(unidirected). Each MicroBlaze can have up to 8 FSL links. Change the parameters of the MicroBlaze instances, so each have one FSL link. Also add two FSL buses in the bus tab and connect the FSL buses with the right ports, a port name including 's' means slave(read) and 'm' means master(write). You also need to connect the clk and rst ports, and set the `C_EXT_RESET_HIGH` as you did for the LMBs you added earlier.

There are macros for reading and writing from/to the FSL. To use them include the `mb_interface.h`. The commonly used macros are `microblaze_bwrite_datafsl(data, port)` and `microblaze_bread_datafsl(data, port)`. Both are blocking, `data` is the data to read/write. For the read operation it must be a variable, which then will be updated to contain the value read, `port` is the FSL port number, starting from 0. Example programs (assuming there is a FSL from MicroBlaze0(master) to MicroBlaze1(slave):

MicroBlaze 0:

```
int main(void){
  for(i=0; i<10; i++){
    microblaze_bwrite_datafsl(i, 0);
  }
}
```

MicroBlaze 1:

```
int main(void){
  for(i=0; i<10; i++){
    microblaze_bread_datafsl(data, 0);
    xil_printf("%d\n\r", data);
  }
}
```