WIRELESS OPEN-ACCESS
RESEARCH PLATFORM

Lab 1: Introduction to WARP Design Flows

Charles Camp

Rice University

WARP Project

Document Revision 7

November 22, 2007

1 Introduction

In the first half of this lab, you will use Xilinx Platform Studio and Base System Builder to construct a simple hardware & software platform, then test your design on the WARP FPGA Board. The instructions for this exercise are on the WARP web site.

In the second half of the lab you will implement a custom peripheral core using Xilinx System Generator and import that core back into your base system. You will also write and compile software to control the core.

Note: All files are stored in `C:\workshop\userN\` where `userN` is your user login location. This location will be referred to as `.\` for the rest of the lab.

2 Building Your Base System

Please visit http://warp.rice.edu/trac/wiki/Exercises/XPSIntro to complete this exercise.

The link will guide you through a step-by-step process of creating your own base system, but keep the following things in mind.

1. Use Xilinx Platform Studio 9.1i.
2. Save your project in `.\Lab1_EDK\xps\system.xmp`.
3. You'll use the remote server to implement the design and your local PC to download your FPGA bitstream and test the design.

3 Converting a System Generator Model to an OPB Peripheral

1. Close XPS (if it's still open).
2. Open MATLAB R2006b.
3. From the Matlab command window, change the working directory to `.\Lab1_EDK\`.
4. Choose File→ Open, and select the file `adder.mdl`.
5. This Simulink file is a System Generator model which implements a simple, unsigned, fixed-point adder with two 32-bit inputs and a 32-bit output.
6. Click the triangular play icon at the top of the model window.
7. Double-click the Simulink Scope block in the model.
8. Verify that the adder output waveform switches to value 12 at time $t = 1$.
9. Close the Scope window, then close the `adder` model.
10. In the Matlab command window, type `sysgen2opb('adder')`. This will prepare your model for conversion to an OPB-compliant peripheral.
11. Re-open the `adder` model.
12. Double click on the System Generator icon within the model. Verify that the System Generator paramters match those in Figure 1.

13. Click the Settings button, then the Folder icon, then navigate to `.\Lab1_EDK\xps\system.xmp`. Click Open, then OK. This step automates the export of your peripheral to an existing XPS project.
14. Click Generate, and wait for the Generation Completed message.

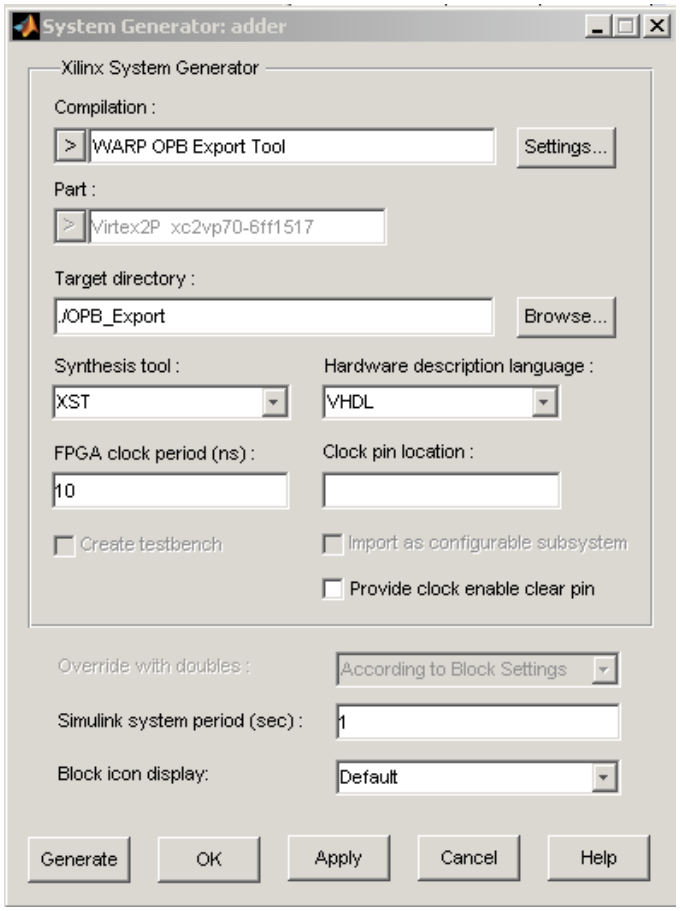

Figure 1: System Generator parameters for OPB core generation

4 Importing the OPB Peripheral

1. Open Xilinx Platform Studio and reopen the previous project `.\Lab1_EDK\xps\system.xmp`.
2. In the IP Catalog tab of the Project Information Area, expand the Project Local pcores entry.
3. Right click on `adder_opbw` and select Add IP.
4. From the Bus Interface view of the System Assembly tab, expand the entry for `adder_opbw`. Connect the core to the OPB by clicking the hollow green circle on the left.
5. Repeat steps 3 and 4 to add two more adder cores (three total). When you've added three cores, your Bus Interface view should look like Figure 2.
6. Switch to the Addresses view and click Generate addresses. This constructs the hardware project's memory map.
7. From the 'Device Configuration' menu at the top of the XPS window, select 'Update Bitstream'.

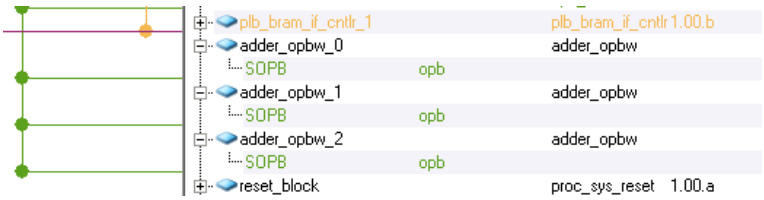

Figure 2: Adder cores attached to the OPB

8. This will compile the hardware component of your base system, including the three adders, and compile the board support libraries and peripheral drivers.
9. After several minutes, the compilation will finish (watch for "`Memory Initialization completed successfully`" in the console window).

5 Using the OPB Peripheral

1. View file `adder.h` in directory `.\Lab1_EDK\xps\drivers\adder_opbw\src`. This file defines macros that simplify the task of accessing registers in the adder core. In particular, we'll need the following macros from this file:

```
#define adder_WriteReg_Op1(BaseAddress, Value) \
  XIo_Out32((BaseAddress) + (adder_Op1_OFFSET), (Xuint32)(Value))

#define adder_WriteReg_Op2(BaseAddress, Value) \
  XIo_Out32((BaseAddress) + (adder_Op2_OFFSET), (Xuint32)(Value))

#define adder_ReadReg_Sum(BaseAddress) \
  XIo_In32((BaseAddress) + (adder_Sum_OFFSET))
```

2. View file `.\Lab1_EDK\xps\ppc405_0\include\xparameters.h`. This file defines key parameters, including address locations, for each core in the system. We need this file to locate the base addresses of the three instances of our adder core. Look for lines like these (your addresses may be different):

```
#define XPAR_ADDER_OPBW_0_BASEADDR 0x7FC40000

#define XPAR_ADDER_OPBW_1_BASEADDR 0x7FC20000

#define XPAR_ADDER_OPBW_2_BASEADDR 0x7FC00000
```

3. Modify your C file from the base system to utilize the first adder in your system. The code below is a good starting point:

```
#include "adder.h"
#include "xparameters.h"

int main() {
  unsigned int result;
  unsigned int operand_a = 3;
  unsigned int operand_b = 5;

  adder_WriteReg_Op1(XPAR_ADDER_OPBW_0_BASEADDR, operand_a);

  adder_WriteReg_Op2(XPAR_ADDER_OPBW_0_BASEADDR, operand_b);

  result = adder_ReadReg_Sum(XPAR_ADDER_OPBW_0_BASEADDR);

  xil_printf("Adder Says : %d + %d = %d\r\n",operand_a,operand_b,result);

  return 0;

}
```

4. From the Device Configuration menu at the top of the XPS window, select Update Bitstream.

5. Generate Linker Script as your program has changed and Update Bitstream once again.
6. Download the bitstream using iMPACT on your local PC.
7. View the terminal output in Tera Term.
8. Try modifying your code to utilize all three adder cores. You should only have to modify the base address values in your C code to target different cores.

6 Optional Further Exercises

Here are few other ideas that you can play with as you get comfortable with using the tools.

1. Use the terminal (Tera Term Pro) to input two numbers and show return their sum. `XUartLite_RecvByte(STDIN_BASEADDRESS)` allows you to read values from the terminal.
2. Use the hex displays to show your output.
3. Make a calculator, where one can subtract, multiply etc.