Fundamentals of Reconfigurable Computing ECGR6090/8090


# CREATING CUSTOM CORES

## 1 Objective

The goal of this lab is to guide you through the process of adding a custom core. The custom core will act as a simple accelerator allowing the processor to pass the calculation to the core. This allows the processor to perform other tasks. At the end of the lab you will have familiarity with:

- using the *Create and Import Peripheral Wizard* to create a template core
- modifying the template core to include multiple registers
- writing VHDL to perform calculations on the values stored in the registers
- creating a standalone C program to drive your new core

## 2 Getting Started

This lab requires you to create a new base system. Refer to the **Base System Builder Wizard Tutorial** on the Class Website. Create a project build directory:

```
bash-2.05$ mkdir lab2
bash-2.05$ mkdir lab2/build
bash-2.05$ cd lab2/build
```

1. We will be using Xilinx's Virtex-II Pro ML310 Revision D
2. Set the Reference Clock to 100 MHz, Processor's Clock to 300 MHz and the Bus to 100 MHz
3. Turn off JTAG Debugging
4. Add 64 KB of Data and 64 KB of Instruction On-Chip Memory (OCM)
5. Include an RS232 Uart with a baudrate of 9600, 8 Data bits and no parity bits

   Check the box for **Use Interrupt** with the UART

| IP Type | Instance | Name | Address | Base Address | High Address | Size | Lock | Bus Connection |
|---|---|---|---|---|---|---|---|---|
| opb_v20 | opb | | | | | U | | |
| ppc405 | ppc405_0 | MDCR | DSOCM_DCR | | | U | | No Connection |
| ppc405 | ppc405_0 | MDCR | ISOCM_DCR | | | U | | No Connection |
| plb2opb_bridge | plb2opb | SDCR | DCR | | | U | | No Connection |
| plb_v34 | plb | SDCR | | | | U | | No Connection |
| plb2opb_bridge | plb2opb | SPLB | RNG1 | | | U | | plb |
| plb2opb_bridge | plb2opb | SPLB | RNG2 | | | U | | plb |
| plb2opb_bridge | plb2opb | SPLB | RNG3 | | | U | | plb |
| plb2opb_bridge | plb2opb | SPLB | RNG0 | *0x00000000* | 0x7fffffff | 2G | | plb |
| opb_uartlite | RS232_Uart | SOPB | | *0x40600000* | 0x4060ffff | 64K | | opb |
| opb_intc | opb_intc_0 | SOPB | | *0x41200000* | 0x4120ffff | 64K | | opb |
| linear_eq_calc_core | linear_eq_calc_core_0 | SOPB | | *0x70000000* | 0x7000FFFF | 64K | | opb |
| dsbram_if_cntlr | docm_cntlr | DSOCM | | *0xa0900000* | 0xa090ffff | 64K | | docm |
| isbram_if_cntlr | iocm_cntlr | ISOCM | | *0xffff0000* | 0xffffffff | 64K | | iocm |

Figure 1: Address Range for linear_eq_calc_core

6. Remove any other unnecessary cores not mentioned in this list
7. Remove the Peripheral *plb_bram_if_cntlr_1* PLB BRAM Controller
8. Uncheck both the Memory and Peripheral selftests since we will be writing our own application

## 2. Adding A Custom Core Tutorial

In order to off-load calculations from the PowerPC to the FPGA we must first create a custom core which will perform the specific calculation. Refer to the **Adding A Custom Core Tutorial** on the Class Website to use the *Create or Import Peripheral Wizard*. For this lab we will be creating a core with the following configuration:

- Name the Core: **linear_eq_calc_core** (version 1.00.a is fine)
- Attach the Core to the **On-Chip Peripheral Bus (OPB)**
- For IPIF Services - only enable (check) **User logic S/W register support**
- Add **Four (4) 32-bit width** registers with **Posted Write Behavior**
- Select default options (Click Next) and the Finish.
- Add the new core to the design giving it the address range **0x70000000 to 0x7000FFFF**

  [HINT: Look at **Figure 1**]

  Do **NOT** click Generate Addresses
- In order for the PowerPC to communicate with your new core you **may need to modify** the address range for the **PLB2OPB Bridge** to include your core's entire address range.

  [HINT: Again, Look at **Figure 1**]

# 3 Assignment

## 1. Hardware Modifications

To modify the User Logic of your new core, follow the **Simple VHDL Tutorial** and add a process named **LINEAR_EQ_PROC**. In the tutorial you learn how to create a *process* to add 5 to a register and store the result in another register. Instead of simply adding 5, make the process perform:

```
y = (m*x) + b
```

Unlike C, we do not declare variables **y, m, x,** and **b**. Instead, in VHDL, we will use registers. When you created the Hardware Core you created four slave registers (slv_reg0 - slv_reg3). These registers will allow the PowerPC to write the values for **m** (slv_reg0), **x** (slv_reg1), and **b** (slv_reg2). The process you will add to your core will constantly be calculating **y** (slv_reg3). Then in order to read the result (**y**) the PowerPC simply needs to read from slv_reg3.

## 2. Software Application

Create an Application called **LinearEq** that will:

1. write a constant value of **4** to **m** (slv_reg0)
2. write a constant value of **9** to **b** (slv_reg2)
3. prompt the user for the value of **x** (slv_reg1)
4. print the result from the linear_eq_calc_core: **y** (slv_reg3)
5. repeat steps 3 and 4 until the user enters 0 or an invalid character

An example of the output:

```
Enter Integer x (or 0 to Quit):
You entered: 3
Given m=4
Given b=9
y=(4*3)+9=21
```

## 3. Software Application - Getting Started

To get started with the Software Application add the following to the beginning of your lineareq.c file:

```
#include "xparameters.h"
#include "xutil.h"
#include "stdio.h"

// -----------------------------------------------------------------------
// getnum(): Gets integers from user.  Returns Integer or 0 if invalid input
// -----------------------------------------------------------------------
```

```
int getnum() {
  char x[5];
  int y = 0;
  iscanf("%s",x);
  y = atoi(&x);
  return y;
}
// Your main function begins here...
int main(void) {
  // Declare Variables
  ...
```

Now complete the remainder of the Application so that it functions as described above.

**Attention:** For your application to function with the On-Chip Memory Bus you need to **Generate a Linker Script**. This is accomplished by:

1. Right Click on **Project: LinearEq** (your application)
2. At the bottom of the drop down menu select:

   **Generate Linker Script...**
3. Click **OK** to Generate the linker script (Don't change any of the default settings that appear in the new window)

This will create: *LinearEq_linker_script.ld* in your build directory of your lab02. It specifies the Addresses for the Data OCM Controler and Instruction OCM Controler. We will not modify this file for lab 2.

**Attention:** When testing your application you will not see the text you enter into the terminal:

```
Enter Integer x (or 0 to Quit):
```

This is OK (it is actually a problem with Minicom) Do not spend anytime trying to fix it. Instead we add the line below it so the user can see what they typed:

```
Enter Integer x (or 0 to Quit):
You entered: 3
```

# 4 Grading

This lab assignment is due **Friday, October 12 by 5pm (EST)**. To receive credit, you must meet with either the T.A. or the instructor and demonstrate that you've completed the lab. By default, this can be done in Woodward Hall room 237 (the Unix lab). Alternative times are possible but need to be arranged in advance. Be prepared to answer in person any questions in the lab or recompile your program as part of demonstrating that you completed the lab. Do not wait until the last minute to begin the project; extensions will not be granted.