ECEN 749 Lab 3 Report

Tong Lu
UIN 621007982

October 9, 2018

Introduction

In this lab, we created an integer multiplication peripheral and integrated it with ZYNQ processor system. Then we created a C program to interact with the multiplication peripheral.

Procedure

1. Launch Vivado and create a block design with ZYNQ processor.
2. Re-customize the IP design and enable UART1 on peripheral pins for USB communication with PC.
3. Created a multiplication IP witn AXI lite slave peripheral that contains 4×32 bit read/write registers.
4. Add multiplier custom logic into the existing Verilog IP code, and repackage the IP.
5. Launch SDK and create a C program (see helloworld.c in Appendix) to test out the multiply functionality on top of the hardware platform.
6. Save the C application, connect the FPGA, program the FPGA, configure and run the C program on FPGA.
7. Use picocom to monitor USB1 port at buadrate of 115200.

Result

All the programs was finished and demonstrated to TA. The programs are working well and meet all the requirement on lab manual (Figure 1).

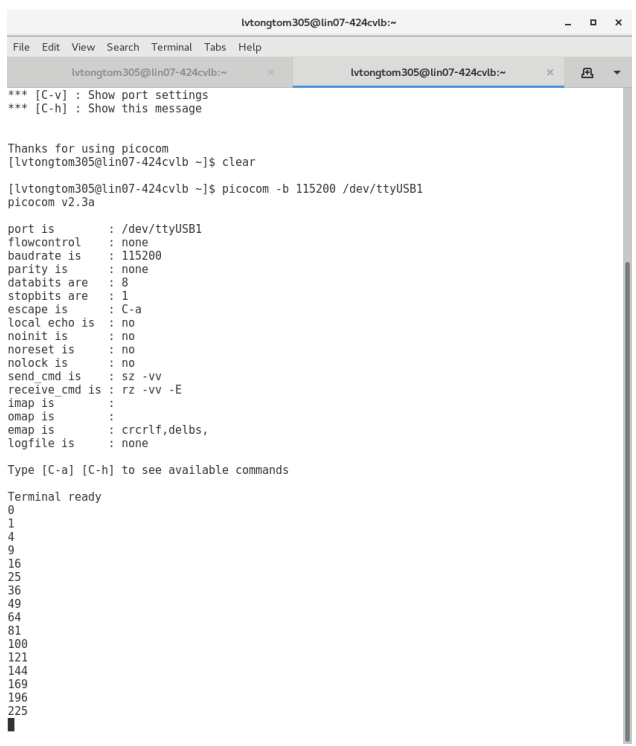

Figure 1. USB1 Port Printout

Conclusion

In this lab, I learned to create a ZYNQ processor system along with a customize IP design communicating on AXI peripheral and develop software program based on the platform. This lab is a good introduction for me and it will help me a lot in the future projects.

Answer to Questions

(a) **What is the purpose of the tmp_reg from the Verilog code provided in lab, and what happens if this register is removed from the code?**

The temp_reg works like a buffer to store the output of multiply block. If we don't have such a buffer, it is possible that the output from reg_2 could be read out before the multiplication operation complete.

(b) **What values of 'slv_reg0' and 'slv_reg1' would produce incorrect results from the multiplication block? What is the name commonly assigned to this type of computation error, and how would you correct this? Provide a Verilog example and explain what you would change during the creation of the corrected peripheral.**

When two large numbers multiply, the reg_2 may **overflow**. For example, if we try to perform

$$0xFFFFF \times 0xFFFFF = 0xFFFFE00001$$

reg_2 won't have enough bits to hold the result. To correct this issue, we can simply increase the size of reg_2:

```
[63:0] slv_reg2;
```

Appendix

```c
/*****************************************************************************
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ----------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ----------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "multiply.h"

int main() // main function
{
    init_platform(); // Initalize the platform
    int i = 0; // Initalize iterator varible for the for loop below
```

```c
    u32 result; // Initalize the result varible
    for (i = 0; i <= 16; i++) { // Run for loop from 0 to 16
        MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, MULTIPLY_S00_AXI_SLV_REG0_OFFSET, i); // Write i into register0 of the multiply IP
        MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, MULTIPLY_S00_AXI_SLV_REG1_OFFSET, i);// Write i into register1 of the multiply IP
        result = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, MULTIPLY_S00_AXI_SLV_REG2_OFFSET); // Read out the result from reguster2
        printf("%u\n", result); // Print out the result on the UART1 port
    }
    cleanup_platform(); // Clean up the platform
    return 0; // Return and Quit
}
```

src/helloworld.c