ECEN 749- Microprocessor System Design

Section 603

TA: Mr. Kunal Bharathi

LAB 3

Creating a Custom Hardware IP and Interfacing it with Software

Date of Performance: 09/19/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

Introduction

The lab will introduce with the process of creating and importing a custom IP module for the Zynq Processing System. It will focus on hardware/software co-design. It will also focus on creating and integrating an Integer multiplication peripheral into a microprocessor system and also focus on the software development part using SDK.

Design:

The Zynq chip is divided into two parts: PL(Programming logic) and PS (Processing system) . The processing system has a dual core ARM Cortex A9 processor and Programming logic uses Xilinx 7 series FPGA logic cells. The below figure highlights the same:

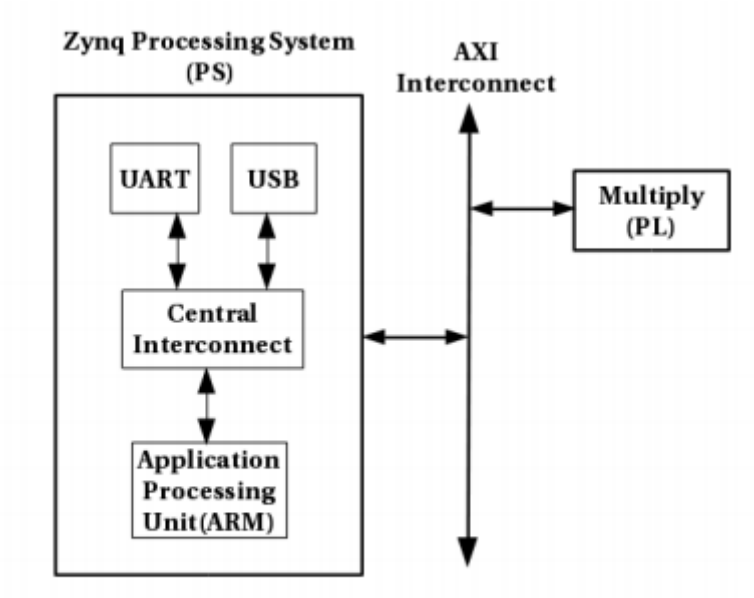

Figure 1: Zynq System Diagram

Procedure:

1. Launch Vivado and open new project file.
2. Create new RTL project and skip for 'Add Existing Window' and 'Add Constraints'. The constraints would be added later as a separate file.
3. Then select the Zybo board from the boards tab.
4. Connect the ZYBO board to the PC and switch on. Make sure it is in JTAG mode.
5. In the flow navigator, click on 'Create Block Design' and write name of the design.
6. Then add 'ZYNQ7 Processing System' IP.
7. Download the 'ZYBO Z7 B2 .tcl' file from /mnt/lab files/ECEN449 and in the Reconfiguration window click on 'Presets and Apply Configuration' and import the tcl file. Uncheck all the peripheral I/O pins.
8. Double click on 'Peripheral I/O Pins' and enable 'UART 1' .
9. This step is of creating Multiply IP package. Select ' Create a new AXI4 peripheral' and click on next.

10. Set the interface values as follows:

    Interface type : Lite
    Interface mode: slave
    Data Width: 32
    Number of Registers: 4
    We will need only three registers for the 'multiply' IP however the minimum number of registers allowed is 4
11. Check Edit IP and Click finish.
12. At this point, the peripheral that has been generated by Vivado is an AXI lite slave that contains 4 x 32 bit read/write registers
13. Open the Vivado window that contains the peripheral just created.
14. In the sources window , expand 'multiply v1 0' and double click on the 'multiply v1 0 S00 AXI.v' file to open it.
15. Insert the following Verilog code in the .v file.

```
reg [ 0 : C_S_AXI_DATA WIDTH−1] tmp_reg ;
 always @( posedge S AXI ACLK )
begin
if ( S_AXI_ARESETN == 1 ' b0 )
 begin
slv_reg2 <=0; //resetting the output register
tmp_reg <=0; //resetting the temporary register
end
else begin
tmp_reg <= slv_reg0 ∗ slv_reg1 ;// storing the multiplication in temp register
slv_reg2 <= tmp_reg ; // storing the result in temp register
end
end
```

16. In the package tab, click on 'Review and Package' and select 'Re-package'. Now Vivado will repackage the IP with added functionality. When Vivado finishes packaging the IP, close the project.
17. Open the diagram tab and add 'multiply' IP to the PS system. Select 'Run Connection Automation' and in the prompted window select 'All Automation' and click 'OK'. Once automation is completed, a layout is generated. Right click on the diagram and select 'Regenerate Layout' to re draw the layout design.
18. Create HDL Wrapper and then generate the bitstream.
19. Export the hardware design including the bitstream and launch SDK.
20. Open the SDK and create a new application project.
21. Then name the project and then select "Hello World!" and "Finish". This will generate the necessary templates files.
22. Edit the 'helloworld.c' source file to write values to the registers 'slv reg0' and 'slvreg1' and read the multiplication result from 'slv reg2'. The value in each of these registers must be printed.

23. Include the xparameters.h and multiply.h in the source file.
24. Now program the FPGA and click on 'Run As' and then select 'Launch on Hardware(GDB)'. Vivado will launch this application on ZYBO Z7-10 board.
25. To view the output, picocom terminal is used. The following steps are followed:
    Open a terminal window and type the following:
    $ source /opt/coe/Xilinx/Vivado/2015.2/settings64.sh
    $ picocom -b 115200 /dev/ttyUSB1
26. The baud rate of the picocom should be 115200.

C Programming Code

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

int main()
{
    init_platform();
    unsigned int result;/*variable to store the multiplication
result*/
    unsigned int i; //variable for count

//multiplication is done by a loop.
//1 will be multiplied by 15, 2 by 16 and so on.
   for (i=0;i<16;i++)
   {
    MULTIPLY_mWriteReg(0x43C00000,0,i);/*writes the value into
register*/
    MULTIPLY_mWriteReg(0x43C00000,4,16-i);

    result= MULTIPLY_mReadReg(0X43C00000,8);/*reads the value after
multiplication and stores it in result*/

   xil_printf("Result of the multiplication of %d and %d =
%d\r\n",i,(16-i),result);
   }
    cleanup_platform(); //used to clean the program.
    return 0;
}
```

Output:

```
bash-4.2$ source /opt/coe/Xilinx/Vivado/2015.2/settings64.sh
bash-4.2$ picocom -b 115200 /dev/ttyUSB1
picocom v2.2

port is        : /dev/ttyUSB1
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Type [C-a] [C-h] to see available commands

Terminal ready

Result of the multiplication of 0 and 16 = 0
Result of the multiplication of 1 and 15 = 15
Result of the multiplication of 2 and 14 = 28
Result of the multiplication of 3 and 13 = 39
Result of the multiplication of 4 and 12 = 48
Result of the multiplication of 5 and 11 = 55
Result of the multiplication of 6 and 10 = 60
Result of the multiplication of 7 and 9 = 63
Result of the multiplication of 8 and 8 = 64
Result of the multiplication of 9 and 7 = 63
Result of the multiplication of 10 and 6 = 60
Result of the multiplication of 11 and 5 = 55
Result of the multiplication of 12 and 4 = 48
Result of the multiplication of 13 and 3 = 39
Result of the multiplication of 14 and 2 = 28
Result of the multiplication of 15 and 1 = 15
```

Result:

The major task in this lab was designing the Zynq base system and then designing the multiply IP peripheral. The understanding of xparameters.h and multiply.h was also critical in order to understand the write and read syntax and the logic of the code. Using picocom as a output console was also learnt in this lab. This lab helped in gaining knowledge regarding declaration of registers and how to work with them.

Conclusion:

The lab served its purpose of familiarizing with the purpose of creating and importing a custom IP module with the help of multiplication. It also helped in understanding the hardware software co design methodology.

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?

Ans. The purpose of the tmp reg or temporary register is to avoid the metastable state. This situation arises when the device on which the code is running is just matching the maximum performance of the device (especially related to clock cycle). In such cases it might happen that a multiplication instruction may not get completed within one clock cycle. So the value stored will not be a stable or expected output, but a random metastable value. In this case, if we directly store the value in the final output register without any temp, a wrong output will be seen at the end of first clock cycle. To avoid this, we use a temp register so that the metastable state is detected previously, and we get an additional clock cycle. By this time, the correct output is in the temporary register and we get the desired output.

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.

Ans. The slv_reg0 and slv_reg1 will produce incorrect results when the output would get overflow i.e. the multiplication of two 32 bit numbers can produce output which is greater than 32 bits. If this happens, the output cannot be stored in a single register accurately. This type of error is generally called logical and overflow errors. We can correct the overflow error by using a 64 bit temporary register. As the maximum value of multiplication of 32 bit registers can be 64 bit, we will store the value of multiplication in the temporary register and then load the lower 32 bit register in slv_reg2 and the upper 32 bit register in slv_reg3. Below is the Verilog code of the same:

```
reg [0 : 63] tmp_reg;
reg [0:63] tmp_reg2;
always@(posedge S_AXI_ACLK) begin
  if(S_AXI_ARESETN == 1'b0) begin
    slv_reg2 <= 0;
    tmp_reg <= 0;
  end
  else begin
    tmp_reg <= slv_reg0 * slv_reg1;
    slv_reg2 <= tmp_reg && 0x00000000ffffffff;//
     tmp_reg2 <=tmp_reg && 0xffffffff00000000;//
     slv_reg3 <= tmp_reg2>>16;

  end
end
```

Since we have stored the output in two registers, we need to read the values from the two registers and display result accordingly.