Lab 09 Report: IP In Vivado HDL

Peter Mowen
peter.mowen@temple.edu

Summary

This lab explores creating custom intellectual property (IP) blocks for the Zybo. An IP block which displays the number of 1s in the first 15 bits of a 32-bit number will be created in Vivado HLx and used in Xilinx SDK.

Introduction

The basis for this lab is exercise 4A from *Zynq Book Tutorials*. In that exercise, an IP block was created that simply writes a number directly to the LEDs. The IP block uses the AXI bus to communicate between the processor and the FPGA. The IP block's main logic is written in Verilog. This Verilog code is then used to configure the FPGA fabric. This exercise was completed as part of the lab but will not be covered in this report. Instead, the updates to that code will be reported. It is assumed that if the updated code works, the original code on which it is based also works.

In this lab, the student will create a custom IP block which counts the number of ones in the first 15 bits of a 32-bit number and displays them on the LEDs. A sequence of ten numbers will be fed into the custom led controller and the number of ones in each number will be written to the LEDs for a sufficient amount of time to be read by the user.

Discussion

<u>Hardware Setup in Vivado HLx</u>

Below is a screenshot of the block diagram from Vivado HLx:

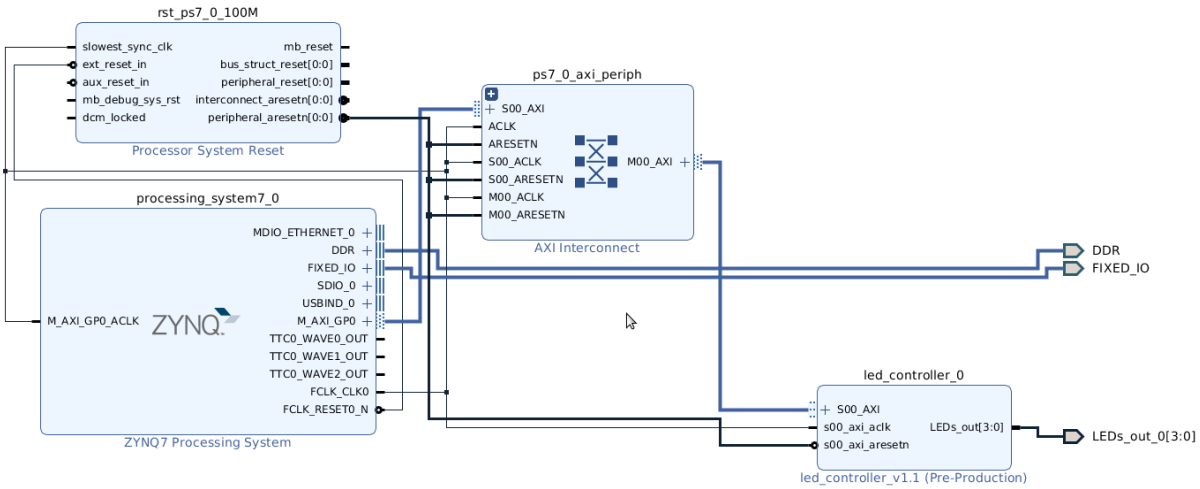

Figure 1: Block Diagram from Vivado HLx

The `led_controller_0` block is the custom IP block created for this lab. Note that the led controller block is connected to the processing system through the AXI peripheral IP block. This is the same bus which the GPIO IP blocks use.

The LED controller logic Verilog code was updated to meet the specifications of this lab and can be seen here:

```
400     // Add user logic here
401     reg [3:0] tempLEDs, i;
402     reg[31:0] tempSLV;
403     always @* begin
404         tempLEDs = 0;
405         tempSLV = slv_reg0;
406         for (i = 0; i < 15; i=i+1)
407             if (tempSLV[i]) tempLEDs = tempLEDs + 1;
408     end
409     assign LEDs_out = tempLEDs;
410     // User logic ends
```

Figure 2: LED controller logic Verilog code

First, two 4-bit registers, `tempLEDs` and `i`, are created; `tempLEDs` to hold the sum of the number of ones in `slv_reg0` and `i` to be an iterator used for iterating through the first 15 bits of `slv_reg0`. The 32-bit register `tempSLV` holds a temporary copy of `slv_reg0`, which is a register holding the number passed to the controller from SDK. After creating these registers, there is an `always` combination block. In this combination block, the `tempLEDs` register is initialized to zero, the contents of `slv_reg0` are copied into `tempSLV`, and a `for` loop is used to iterate over the first 15 bits of `tempSLV`. If the i^{th} register contains a 1, then `tempLEDs` is incremented. Outside the combination block, the `LEDS_out` wire is assigned the contents of the `tempLEDs` register. `LEDs_out` is a 4-bit output wire added during the tutorial exercise. It gets tied to the LEDs through the constraints file.

Below is a screenshot of the constraints file for this block diagram:

Diagram | Address Editor | led_constraints.xdc *

/home/pmmowen/Programming/Vivado/zynq_book_tutorial_exercises/led_controller/led_controller.srcs/constrs_1/new/led_constraints.xdc

```
1 set_property PACKAGE_PIN M14          [get_ports {LEDs_out_0[0]}]
2 set_property IOSTANDARD LVCMOS33      [get_ports {LEDs_out_0[0]}]
3 set_property PACKAGE_PIN M15          [get_ports {LEDs_out_0[1]}]
4 set_property IOSTANDARD LVCMOS33      [get_ports {LEDs_out_0[1]}]
5 set_property PACKAGE_PIN G14          [get_ports {LEDs_out_0[2]}]
6 set_property IOSTANDARD LVCMOS33      [get_ports {LEDs_out_0[2]}]
7 set_property PACKAGE_PIN D18          [get_ports {LEDs_out_0[3]}]
8 set_property IOSTANDARD LVCMOS33      [get_ports {LEDs_out_0[3]}]
```

Figure 3: Constraints file

This file tells Vivado HLx how to map the bits of `LEDs_out_0` to the physical hardware. It maps each bit to a physical pin. Below is a picture of the LEDs on the Zybo:

Figure 4: LEDs on Zybo

Note that beside each LED name is a pin number and that these pin numbers are the `PACKAGE_PIN` properties from the constraints file. The means bit 0 of `LEDs_out_0` gets mapped to LD0, bit 1 gets mapped to LD1, etc. In this way, the 4-bit number stored in `LEDs_out` is displayed on the LEDs.

C Code

Below is a screenshot of the C code in its entirety:

lab09.c | system.mss

```
1  /* Generated driver function for led_controller IP core */
2  #include "led_controller.h"
3  #include "xparameters.h"
4  #include "xil_io.h"
5  #include "xil_printf.h"
6
7  #define printf xil_printf
8
9  // Define delay length
10 #define DELAY 50000000
11
12 /*  Define the base memory address of the led_controller IP core */
13 #define LED_BASE XPAR_LED_CONTROLLER_0_S00_AXI_BASEADDR
14
15 int sequence[] = {// base 10        binary          LEDS
16                   1,      // 0b0000000000000001 => 0001
17                   15,     // 0b0000000000001111 => 0100
18                   3,      // 0b0000000000000011 => 0010
19                   255,    // 0b0000000011111111 => 1000
20                   6,      // 0b0000000000000110 => 0010
21                   30,     // 0b0000000000011110 => 0100
22                   2,      // 0b0000000000000010 => 0001
23                   63,     // 0b0000000000111111 => 0110
24                   511,    // 0b0000000111111111 => 1001
25                   32767,  // 0b1111111111111111 => 1111
26                };
27
28 /* main function */
29 int main(void){
30     u32 led_val = 0;    // unsigned 32-bit variables for storing current LED value
31     int n = sizeof(sequence) / sizeof(int);     // number of elements in the sequence
32     int i,k;    // iterators for sequence and time-wasting for loop
33
34     /* Loop forever */
35     while(1){
36             for (i = 0; i < n; i++)
37             {
38                 printf("sequence[%d] = %d\n", i, sequence[i]);
39                 led_val = sequence[i];
40                 /* Write value to led controller IP core using generated driver function */
41                 LED_CONTROLLER_mWriteReg(LED_BASE, 0, led_val);
42                 /* run a simple delay to allow changes on LEDs to be visible */
43                 for(k=0;k<DELAY;k++);
44             }
45         }
46     return 1;
47 }
48
```

Figure 5: Lab 9 C code

On line 2, the led_contriller.h file is added to the code. This file contains functions which can be used to interact with the custom led controller IP block. This code is generated by Vivado HLx based on the custom IP block. The preprocessor definition on line 15 maps the base address of the custom IP block to `LED_BASE`. The contents of this address end up in `slv_reg0`. This is similar to how GPIO objects are addresses.

The ten-number sequence to be sent to the LED controller starts on line 15. Note that the binary representation of each number and the expected LED output are noted in comments beside each number.

Line 30 creates the variable which will be fed into the LED controller and initializes it to zero. Line 31 creates a variable containing the number of items in the sequence. It is setup in such a way that the sequence could be modified without having to change this variable. Next, two iterators are created. Finally, the main `while` loop is entered.

Inside the `while` loop, there are nested `for` loops. The outer loop iterates through each element of `sequence`. Each element is printed to the terminal, then assigned to `led_val`. On line 41, `led_val` is written to the LED controller using the `LED_CONTROLLER_mWriteReg()` function. The first argument in this function is the address to which `led_val` is to be written. The second argument is used to offset the address, zero indicating no offset. The last argument is the variable to be written into `LED_BASE`. Since the contents of this address end up in `slv_reg0`, the number of ones in led_val is counted, then written to the LEDs.

Results

The predicted pattern was displayed on the LEDs as can be seen in the following demonstration video:

https://youtu.be/CleUBNh4g_U

Conclusions

This lab successfully introduced how to create custom IP blocks in Vivado HLx. A custom block was created to count the number of 1s in the first 15 bits of a 32-bit number. This custom block was then used in a project to do just that. A sequence of ten numbers was fed from the processing system through this block and the correct number of 1s in each number was written to the LEDs.

Appendix

Verilog Code

```
// Add user logic here
      reg [3:0] tempLEDs, i;
      reg[31:0] tempSLV;
    always @* begin
        tempLEDs = 0;
        tempSLV = slv_reg0;
        for (i = 0; i < 15; i=i+1)
            if (tempSLV[i]) tempLEDs = tempLEDs + 1;
    end
    assign LEDs_out = tempLEDs;
      // User logic ends
```

C Code

```
/* Generated driver function for led_controller IP core */
#include "led_controller.h"
#include "xparameters.h"
#include "xil_io.h"
#include "xil_printf.h"

#define printf xil_printf
```

```
// Define delay length
#define DELAY 50000000

/*	Define the base memory address of the led_controller IP core */
#define LED_BASE XPAR_LED_CONTROLLER_0_S00_AXI_BASEADDR

int sequence[] = {//	base 10			binary			LEDS
			1,		// 0b0000000000000001 => 0001
			15,		// 0b0000000000001111 => 0100
			3,		// 0b0000000000000011 => 0010
			255,		// 0b0000000011111111 => 1000
			6,		// 0b0000000000000110 => 0010
			30,		// 0b0000000000011110 => 0100
			2,		// 0b0000000000000010 => 0001
			63,		// 0b0000000000111111 => 0110
			511,		// 0b0000000111111111 => 1001
			32767,		// 0b1111111111111111 => 1111
		};

/* main function */
int main(void){
	u32 led_val = 0;  // unsigned 32-bit variables for storing current LED value
	int n = sizeof(sequence) / sizeof(int);		// number of elements in the sequence
	int i,k;   // iterators for sequence and time-wasting for loop

	/* Loop forever */
	while(1){
			for (i = 0; i < n; i++)
			{
				printf("sequence[%d] = %d\n", i, sequence[i]);
				led_val = sequence[i];
			/* Write value to led_controller IP core using generated driver function */
				LED_CONTROLLER_mWriteReg(LED_BASE, 0, led_val);
				/* run a simple delay to allow changes on LEDs to be visible */
				for(k=0;k<DELAY;k++);
			}
		}
	return 1;
}
```