ECE 385

Spring 2020

Experiment 7

SOC with NIOS II in SystemVerilog

Yuding Wang(3170111700)
Zhuoting Han(3170111699)
Lab/D-221
Jinghan Huang

❏ Introduction

❏ Summarize the basic functionality of the NIOS-II processor running on the Cyclone IV FPGA.

The NIOS II can be the system controller and handle tasks which do not need to be high performance (for example, user interface, data input and output) while an accelerator peripheral in the FPGA logic (designed using SystemVerilog) handles the high-performance operations.

❏ Written Description and Diagrams of NIOS-II System

❏ Summary of Operation

❏ Describe in words the hardware component of the lab (Describe what IP blocks are used beyond the ones necessary for the NIOS to run. In this lab, the only additional IP blocks used are the PIO blocks).

The hardware component consists of sdram controller, on chip memory and clock phase shifter. The sdram controller allows us to describe the dimensions of the on board sdram, such specs as the number of rows/ columns, and the number of banks. The on chip memory is used to store valuable on chip memory, but here it is just a placeholder. Finally, the phase shifter is used to adjust the clock on the SDRAM so that the sdram chip can read data at the correct times.

❏ Written Description of all .sv Modules

❏ A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Qsys generated file lab7_soc.v!

Module: PLL
Input: inclk_interface, inclk_interface_reset
Output: c0, c1
Description: Phase lock loop
Purpose: Generate a clock whose frequency is different from the system clock or shift the clock.

Module: SDRAM
Input: clk, reset, wire
Output: wire
Description: Synchronous dynamic random-access memory
Purpose: Save data in Avalon bus.

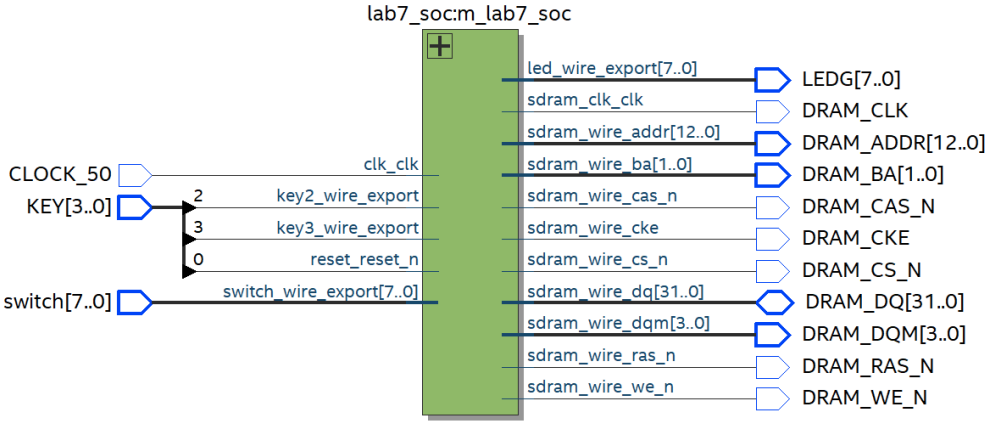
Module: LED
Input: clk, reset
Output: external_connection
Description: A PIO peripheral IP.
Purpose: Control LED.

Module: Switch

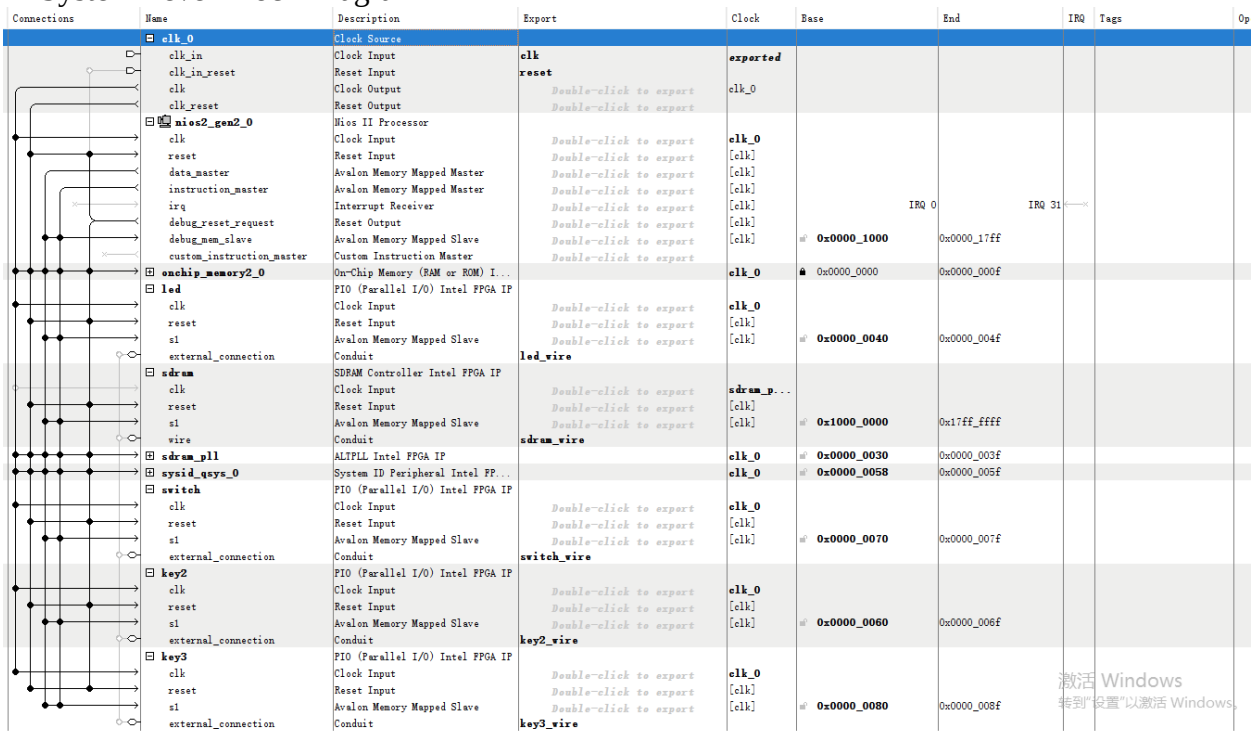
Input: clk, reset, external_connection
Output: external_connection
Description: A PIO peripheral IP.
Purpose: Input switch from FPGA to SoC.

❑ Top Level Block Diagram

❑ This diagram should represent the placement of all your modules in the top level. Please only include the top level diagram and not the RTL view of every module . The Qsys view of the NIOS processor is not necessary for this portion.



❑ System Level Block Diagram



Nios2_gen2_0: processor
Onchip_memory2_0: on-chip memory
Led: A PIO peripheral IP that outputs and controls LED

SDRAM: Save data in Avalon bus
SDRAM_pll: Generate a clock whose frequency has a shift from the clock
Sysid_qsys_0: Set a sys id
Switch: Input switch from FPGA to SoC
Key2: reset button
Key3: accumulate button

❑ Describe in words the software component of the lab. One of the INQ questions asks about the blinker code, but you must also describe your accumulator.

Blinker: For the LED blinker code, we first define where the address of the LED PIO is located, there we can communicate with actual hardware. Then, we first clear the LED by writing to the memory a 0 to clear all 8 LEDs. Following, we declare an infinite while loop that executes the blinking. The two for loops are used as a software delay. The code "*LED_PIO |= 0x1;" means we or the 8 bit data stored at the LED memory location with a binary 1 to ensure that the LSB is going to be a 1 which turns on the right most LED. Similarly, the code "*LED_PIO &= ~0x1;" does a bit wise AND of the LED memory with a binary 0 to ensure that all the LEDs are turned off.

Accumulator: For the accumulator code, we declared more volatile unsigned integer pointers such as the switch, accumulator reset as well as the accumulator to store the memory location of each hardware component. We also declared a pause integer to ensure that each button press only executes once. In the while loop, we have 3 if statements. The first if statement means if the accumulator button is pressed, the sum is reset. The second if statement means if the accumulator button is pressed and the pause integer is 0 then we accumulate the amount of the switch to the sum, we then change the pause integer to a 1. The final if statement means if we release the accumulator button, we then change the value of the pause integer back to a 0. At the end of the code, we would then store the sum into the LED pointer to write the sum to the LED.

❑ Answers to all 11 INQ Questions

❑ The Prelab question (part A) is one of these 11 questions. The questions are reproduced at the end of this document for convenience.

INQ Questions

1. What advantage might on-chip memory have for program execution?

- The memory is more efficient and read and write times are faster than off chip memory since it is physically closer to the processor.

2. Note the bus connections coming from the NIOS II; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?

- It is a modified Harvard. Inside the core, different data are saved in different caches and we can fetch different data in separate buses at the same time. Outside the core, they are saved in the same place.

3. Note that while the on-chip memory needs access to both the data and program bus, the led peripheral only needs access to the data bus. Why might this be the case?

- The LED is an output peripheral, which only receives data. It has nothing to do with the program while on-chip memory needs to know where to read and write data.

4. Why does SDRAM require constant refreshing?

- SDRAM uses capacitors to store memory. If not refreshed, the voltage will decay and lose the data.

5. Make sure this is consistent with your above numbers; you will need to justify how you came up with 1 Gbit to your TA.

32*2^13*2^10*4 = 1Gbit

SDRAM Parameter	Short Name	Parameter Value
Data Width	[width]	32
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1
# of Banks	[nbanks]	4

6. What is the maximum theoretical transfer rate to the SDRAM according to the timings given?

- 1/5.5ns * 32bit = 5.41Gbps

7. The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?

- If the SDRAM runs too slow, the chip won't be able to refresh the data in required time, which results in data loss.

8. Make another output by clicking clk c1, and verify it has the same settings, except that the phase shift should be -3ns. This puts the clock going out to the SDRAM chip (clk c1) 3ns ahead of the controller clock (clk c0). Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under SDRAM controller.

- The physical difference in location requires that there be 3ns difference between the SDRAM clock and the controller clock.

9. What address does the NIOS II start execution from? Why do we do this step after assigning the addresses?

- The NIOS II base address is 0x00001000. The processor needs to know where to begin when it start and reset.

10. You must be able to explain what each line of this (very short) program does to your TA.Specifically, you must be able to explain what the volatile keyword does (line 8), and how the set and clear functions work by working out an example on paper (lines 13 and 16). This question is referring to the blinker code.

- Line 7: initialize the variable ‘i’ to 0.

Line 8: initializes the volatile unsigned integer pointer to the address of the LEDs. The volatile keyword tells the compiler that the value in that address can change anytime
Line 10: deference the pointer and set it to zero
Line 11: establish an infinite loop
Line 13: set up a for loop with nothing to execute. This acts as a software delay
Line 14: Set the LSB in the memory of LED_PIO to 1. Turn on the right most LED.
Line 15: set up a for loop with nothing to execute. This acts as a software delay
Line 16: Set the LSB in the memory of LED_PIO to 0. Turn off the right most LED.
Line 18: this line should never be reached. Lines 11-17 are in an infinite loop. If the main function returns 1, an error has occurred.

11. Look at the various segment (.bss, .heap, .rodata, .rwdata, .stack, .text), what does each section mean? Give an example of C code which places data into each segment, e.g. the code: const int my_constant[4] = {1, 2, 3, 4} will place 1, 2, 3, 4 into the .rodata segment

- .bss: statically allocated variables not specifically initialized or initialized to 0 e.g. int i;
- .heap: dynamically allocated variables and objects e.g. int *j = malloc(sizeof(int));
- .rodata: static constants -- “read only” e.g. const int j = 3;
- .rwdata: variables-- “read/write” e.g. int j = 3;
- .stack: statically allocated variables and function calls e.g. func(i, j);
- .text: the actual code e.g. int main() { //code here }

❑ Postlab Question

❑ Document the Design Resources and Statistics in following table.

LUT	2302
DSP	0
Memory (BRAM)	10,368
Flip-Flop	1991
Frequency	82.25 MHz
Static Power	102.06 mW
Dynamic Power	40.60 mW
Total Power	204.03 mW

❑ Conclusion

❑ Discuss functionality of your design. If parts of your design didn’t work, discuss what

could be done to fix it

- Our design can make an on board LED blink and keep on adding to an accumulator.

❑ Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.