ECE 385
Lab 7
SOC with NIOS II in SystemVerilog

Li, Ziying
Kang, Xingjian

April 12, 2024

1 Introduction

In this lab, we used platform designer and Nios ii system in Quartus to build a accumulator. By inputting numbers using switchs to add the number to LEDs.

2 Written Description and Diagrams of NIOS-II System

2.1 Summary of Operation

2.1.1 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).

We add 3 PIO blocks: led, sw, key. We used their export to wire hardware with software. With a 32-bit modified Harvard RISC architecture and a C compiler, the NIOS II is an IP-based 32-bit CPU controller that serves as the system controller and can handle low-performance activities.

2.1.2 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.

This accumulator mainly used $*LED\_PIO+ = *SW\_PIO$ to accumulate the number input by the switch onto the number in LED every time the INC buttom is pressed.

2.2 Written Description of all .sv Modules

(1) **Module:** lab7.sv

Inputs: CLOCK_50, KEY[3:0], SW[7:0], DRAM_DQ[31:0]

Outputs: LEDG[7:0], DRAM_ADDR[12:0], DRAM_BA[1:0], DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQ[31:0], DRAM_DQM[3:0], RAM_RAS_N, DRAM_WE_N, DRAM_CLK

Description: Top level file in the project.

Purpose: This file connects software and hardware

(2) **Module:** lab7_soc.v

Inputs: clk_clk, key_wire_export[2:0], reset_reset_n, sdram_wire_dq[31:0], sw_wire_export[7:0]

Outputs: led_wire_export[7:0], sdram_clk_clk, sdram_wire_addr[12:0], sdram_wire_ba[1:0], sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_dq[31:0], sdram_wire_dqm[3:0], sdram_wire_ras_n, sdram_wire_we_n

Description: platform designer generated file.

Purpose: This file has the wire connection for hardware that can be connected to toplevel.

2.3 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.

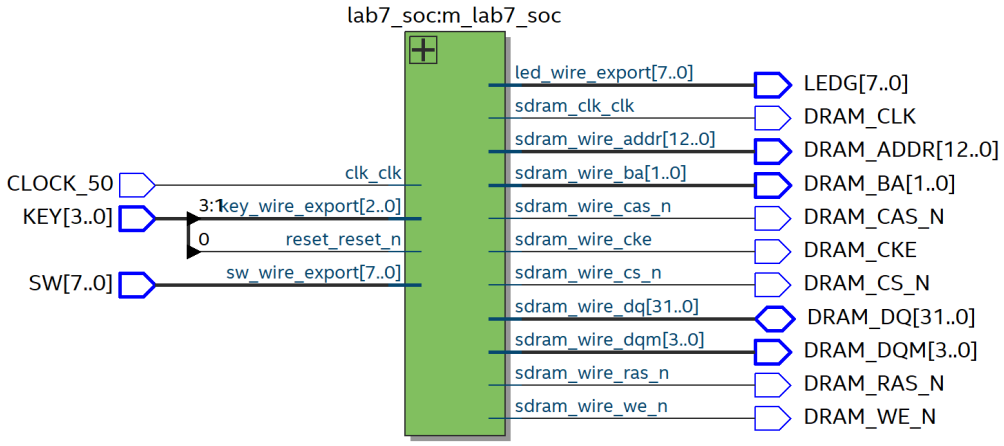

Figure 1: Enter Caption

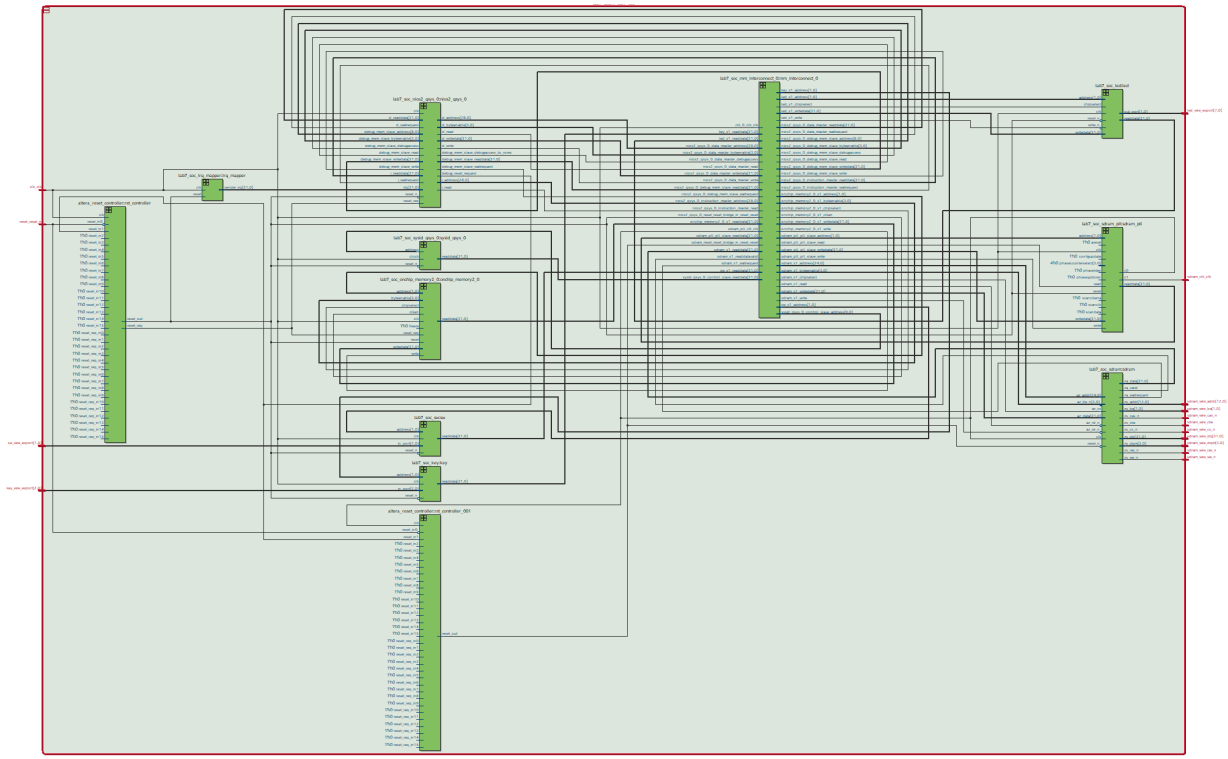

Figure 2: Enter Caption

3 Answers to all 11 INQ Questions

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

On-chip memory can significantly **speed up program execution**. This is because it reduces the time taken to fetch data from memory, as on-chip memory is physically closer to the processor, leading to **lower latency** and **higher bandwidth**. This is particularly beneficial for applications that require frequent memory access.

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

The NIOS II is a **Modified Harvard Architecture** machine. It has separate instruction and data caches (like a Harvard architecture), but also allows data to be accessed as instructions and vice versa (like a Von Neumann architecture)[1].

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 peripheral only needs access to the data bus because it only needs to receive data (such as on/off signals or brightness levels). It does not need to execute instructions, which would require access to the program bus. This simplifies the design and reduces resource usage.

4. Why does SDRAM require constant refreshing?

SDRAM requires constant refreshing because it uses capacitors to store bits. Capacitors leak charge over time, so without refreshing, the stored information would be lost. Refreshing recharges these capacitors and thus maintains the data integrity.

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.

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

$32 * 2^{13} * 2^{10} * 4 = 1Gbit$

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

$32bit * (1/5.5ns) = 5.818Gb/s$

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

The data need to be refreshed within a certain time. When frequency gets lower, time interval get larger and data may lose because of not-in-time refresh.

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.

Because it takes time for the controller to get the address, data, and control signals to be valid at SDRAM, we do need a clock that is a few time delayed of the controller clock.

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

NIOS II start execution from the start of the SDRAM, which we can see in the Reset Vector. In this lab, we start at 0x02000000.

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.

volatile keyword: The volatile type variable will read the memory and change as memory change. With "volatile unsigned int *LED_PIO = (unsigned int*)0x20;", we get stable access to the memory that is connected to LED.

set and clear: $*LED\_PIO| = 0x1$; makes the least significant bit in LED be 1 so that we cant set the right most led light up. $*LED\_PIO\& = 0x1$; makes every bit for LED be 0 with AND 0.

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 means the region of uninitialized parameter, for example, int A;

.heap means dynamically allocated parameter, for example, int *p = (int*)malloc(sizeof(int));

.rodata means region of read only constant variables, for example, const int A = 0;

.rwdata means region of data that can be changed, for example, int A =0;

.stack means allocated variables and function calls, for exmaple, int function(int A, int B);

.text means region of string, for example, char A = "ovo";

4 Postlab Question

LUT	2879
DSP	0
Memory (BRAM)	36864
Flip-Flop	1979
Frequency	90.54 MHz
Static Power	102.06 mW
Dynamic Power	41.86 mW
Total Power	205.32 mW

5 Conclusion

5.1 Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

After we adding key and sw PIOs and assign base addresses, the address for LED also changes, but we did not notice, so the LEDs did not respond. But we finally found the bug and fixed it.

5.2 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.

In the "13.1 Introduction to NIOS II and QSYS", the platform designer part, there is a block named "nios2_qsys_0", but in page INQ. 12, it become "nios_gen2_0", they are actually the same block, but confused us.