ECE 385

Spring 2024

Experiment # 7

Lab7 : SoC on NIOS II in SV

Name: Jie Wang, Shitian Yang
Student ID: 3200112404, 3200112415

Prof. Chushan Li,
Prof. Zuofu Cheng
ZJU-UIUC Institute
April 12, 2024, Sunday D-225
TA: Jiebang Xia
Demo Point: 5/5

1.Introduction

In this lab, we used the basic capabilities of the NIOS II processor to learn System-On-Chip (SoC). Based on the DE2-Board and the Altera Cyclone IV device, we implemented memory-mapped I/Os and used a simple SoC that interfaces with peripherals such as onboard switches and LEDs. The NIOS II processor served as the system controller, managing tasks such as user interface and data I/O, while operations were offloaded to peripheral using SystemVerilog. We use the system to accomplish small tasks like LED blinking as well as an accumulator program. So, the system can keep adding until overflow.

2. NIOS-II System

A. Hardware Component

The NIOS II processor is a 32-bit IP-based CPU that can be programmed using high-level languages, such as C. In this lab, we used the NIOS II to control the blinking of LEDs and to accumulate values from the onboard switches. The processor was set up with an SDRAM controller and a Parallel I/O (PIO) block, allowing it to interact with the hardware peripherals and perform the required tasks.

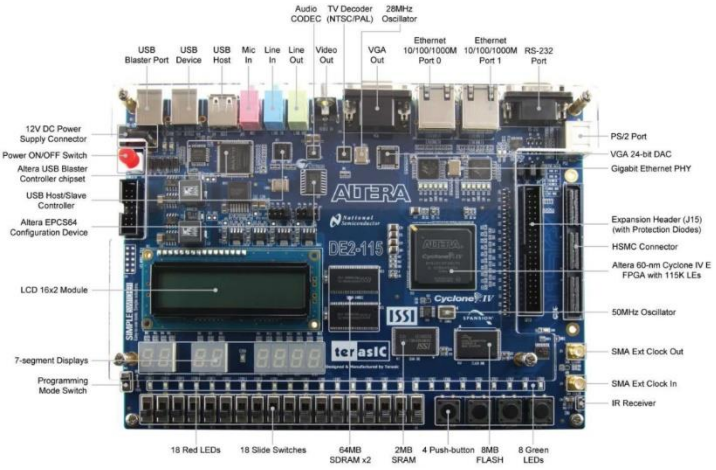


Fig-1: DE-2 FPGA Board with IP integrated on it.

The IP blocks include NIOS II, SDRAM controller, PLL and some PIO blocks.

- **NIOS II:** This is the central processing unit (CPU) that orchestrates the operation of the system, performing general C language computing tasks.
- **SDRAM Controller:** Manages memory operations between the NIOS II processor and the synchronous dynamic RAM (SDRAM) on the board
- **Parallel I/O (PIO) Blocks:** These are used for interfacing with simple peripherals. In this lab, two sets of PIO blocks are used:
- **LED Output PIO:** This block interfaces the NIOS II with the onboard green LEDs to display the value stored in the accumulator.
- **Switch Input PIO:** This block reads input from the onboard switches, which are used to input 8-bit numbers to be added to the accumulator.

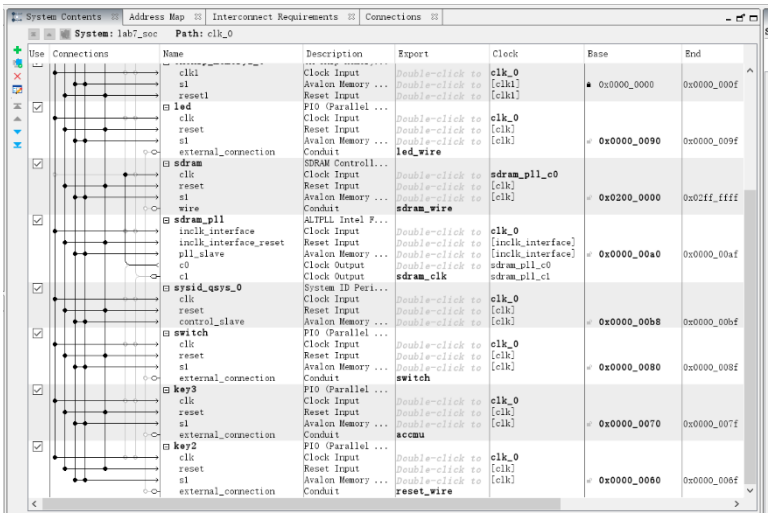


Fig-2: PIO connection for hardware, with base address list above.

B. Software Component

(1) Blinker Code:

This simple program is initially used to test if the LEDs can be controlled by the processor. The first step in the blinker code is to define the memory address of the LED PIO block. The LEDs are initially cleared to ensure they start in a known state.
This is done by writing 0 to the LED's memory-mapped PIO address, which turns off all the LEDs. The main part of the blinker code runs inside an infinite loop, ensuring the blinking continues indefinitely. The LED is set (turned on) by using a bitwise OR operation. This operation sets the least significant bit (LSB) of the LED register to 1, turning on the rightmost LED.
The code use a for-loop to construct delay, which iterates a large number of times to create a noticeable delay between turning the LED on and off.

(2) Accumulator Program:

At first , the accumulator is set to zero, and this value is displayed on the LEDs. Each press of the 'Accumulate' button causes the current value represented by the switches (interpreted as an unsigned 8-bit binary number) to be added to the accumulator. The result is then displayed on the LEDs. The accumulator is designed to overflow gracefully, meaning that exceeding the maximum value (255) wraps the value around starting from zero. Pressing the 'Reset' button sets the accumulator back to zero and updates the LED display accordingly.
You can refer **Appendix A** for the modified Accumulator program.

C. Written Description of all .sv Modules

This part is included in Appendix B.

D. Top Level Block Diagram

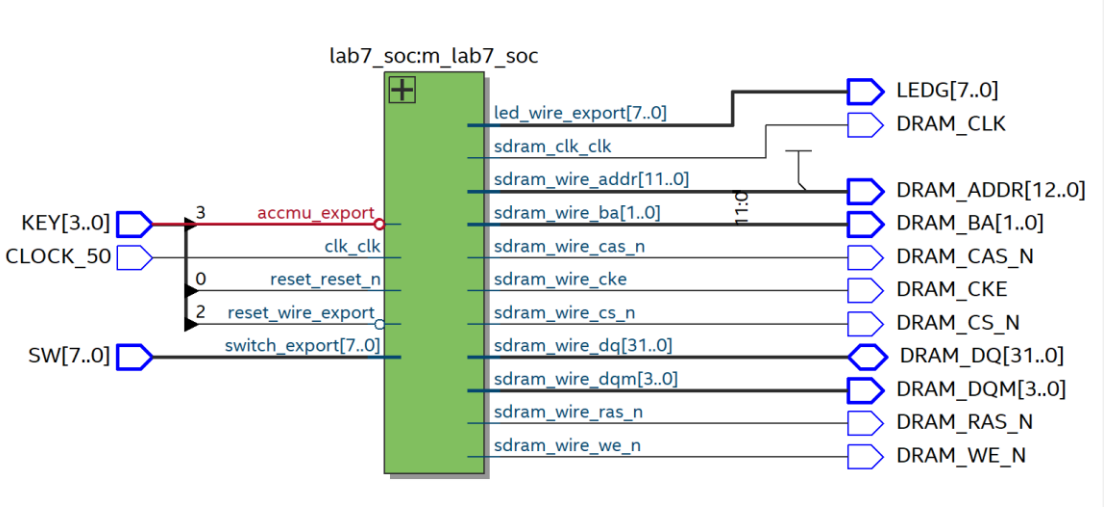


Fig-3: RTL Viewer of the Lab7 SV Module

3. Answers to INQ Questions

What are the differences between the Nios II/e and Nios II/f CPUs?
The Nios II/e is an economical version of the processor optimized for minimal resource and logic element usage, operating at slower speeds due to a higher clock cycle count per instruction, unlike the faster Nios II/f variant.

What advantage might on-chip memory have for program execution?
On-chip memory offers higher access speeds and operates at higher frequencies due to its proximity to the CPU, as opposed to off-chip memory, which has a longer data path and slower access times.

Is the Nios II a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?
Nios II is a "modified Harvard" architecture as it has shared memory for instructions and data but employs separate buses for instruction and data transfer, differentiating it from "pure Harvard" and Von Neumann architectures.

Why does the LED peripheral only need access to the data bus, unlike on-chip memory?
LED peripherals only require data bus access for displaying data, whereas on-chip memory needs to store both instructions and data, necessitating access to both the data and program bus.

Why does SDRAM require constant refreshing?
SDRAM comprises capacitors and transistors that naturally decay over time. Regular refreshing is necessary to maintain data integrity and prevent loss due to charge leakage.

Maximum Theoretical Transfer Rate to SDRAM: Given an access time of 5.5ns and a 32-bit data width, the maximum transfer rate to the SDRAM is 32 bits / 5.5ns, which equals 727MB/s.

Minimum SDRAM Operating Frequency: The SDRAM must maintain a minimum operating frequency of 50 MHz. This is due to the presence of capacitors that require frequent refreshing to retain their voltage level and thus preserve the correctness of the stored data.

SDRAM parameter	Short name	Parameter value(fill in from datasheet)
Data Width	[width]	32 bits
# of Rows	[nrows]	13 bits
# of Columns	[ncols]	10 bits
# of Chip Selects	[ncs]	1 bits
# of Banks	[nbanks]	4 bits

Table 1: Parameter of SDRAM

Phase Shift for SDRAM Clock: Creating an additional output with a phase shift of -3ns aligns the SDRAM chip clock (clk c1) 3ns ahead of the controller clock (clk c0). This phase shift is crucial as it allows time for the SDRAM controller to stabilize data access and ensures that control signals are valid at the address pins.

NIOS II Execution Start Address: The NIOS II processor begins execution from the SDRAM address x0200 0000. This step is necessary to ensure that the processor starts with the correct instruction sequence and avoids accessing non-instructional memory regions.

Understanding the Provided Program:
Please see **Section 2-B: Software Component.**

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

segment	meaning	example
.bss	region of uninitialized parameter	int A;
.heap	dynamically allocated parameter.	int pointer = (int)malloc(sizeof(int));
.rodata	region of read only constant variables	const int A = 0;
.rwdata	region of data that can be changed	int A =0;
.stack	allocated variables and function calls	int func(int a, int b)
.text	region of string	char A = ”hello”;

Table 2: various segment datatype

6. Bug Log

- Description of all bugs encountered, and corrective measures taken:

1. We spent a few hours fixing the system environment configuration. Due to some mysterious reason, the eclipse is quite hard to run on the Quartus II- 20.1. But after a few Q&A with ChatGPT, we successfully debug and work it out.
2. The Accumulator program fails to operate, because I accidentally add an infinite loop, causing the code to loop forever. After I check and fix it, it works correctly.
3. We forget to add a KEY 2 for the functionally reset state. With the help of our classmate, Yitao Cai, we complete it.

7.Conclusion

In conclusion, the lab 7 provided a hands-on experience with the NIOS II processor and its integration within an SoC. The functionality of the design was successful, with the LEDs accurately reflecting the state of the accumulator. Although we spent a long day preparing the midterm exam and the lab 7 demo, we still managed to finish it on time. The lab experience was valuable for understanding the role of the NIOS II processor in managing SoC tasks and the importance of peripheral interfacing.

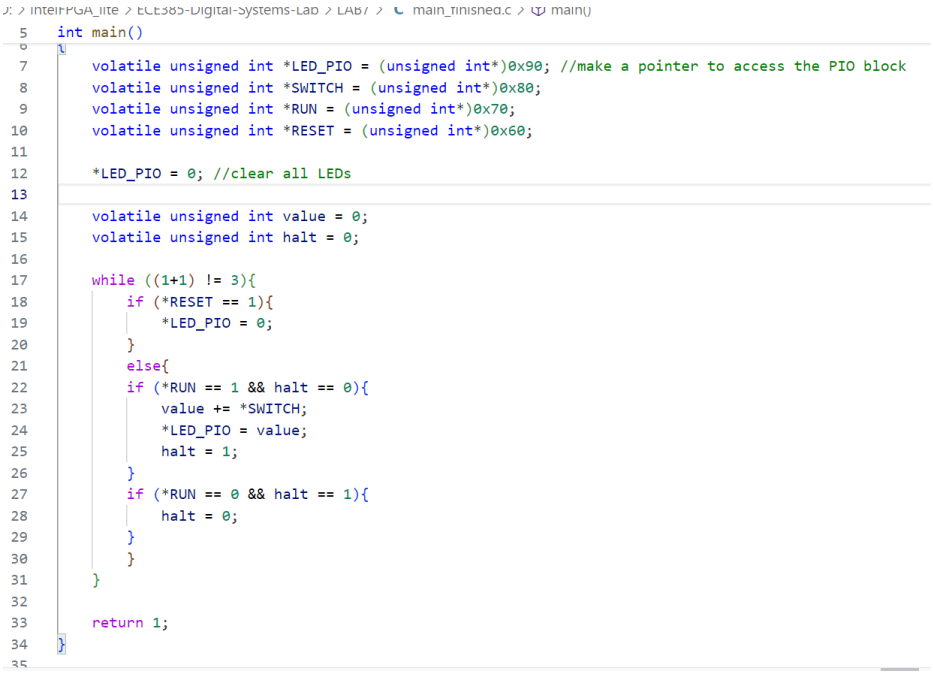
8. References

[1] KTTECH. (2017, January 31). ECE 385 Lab 7: SoC with NIOS II in SystemVerilog. Retrieved from https://kttechnology.wordpress.com/2017/03/03/ece-385-lab7-soc-with-nios-ii-in-system-verilog/ . Teaching Assistant Blog

[2] ECE385 Faculty. (n.d.). Lab 7 description
[3] ECE385 Faculty. (n.d.). Introduction to SystemVerilog (pdf)
[4] 青山，知乎（n.d）无法打开 Nios II Build Tools for Eclipse 解决方案 https://zhuanlan.zhihu.com/p/642007493
[5] Terasic Wiki. (n.d.). Getting Start Install WSL. Retrieved from https://www.terasic.com.tw/wiki/Getting_Start_Install_WSL
[6] Reddit(2021.) . Nios/Eclipse Error when trying to make Nios2 with BSP file from template https://www.reddit.com/r/FPGA/comments/l4wrbv/nioseclipse_error_when_trying_to_make_nios2_with/
[7] Intel (04/15/2022) *Why do I see errors like wsl dos2unix create-this-app;./create-this-app --no-make or make: command not found when running Nios® II Software Build Tools for Eclipse on Windows?*.https://www.intel.com/content/www/us/en/support/programmable/articles/000090071.html
[8] ECE385 Faculty. (n.d.). Introduction to Quartus Prime in the lab manual.

9. Appendix

A. Accumulator Program



```
int main()
{
    volatile unsigned int *LED_PIO = (unsigned int*)0x90; //make a pointer to access the PIO block
    volatile unsigned int *SWITCH = (unsigned int*)0x80;
    volatile unsigned int *RUN = (unsigned int*)0x70;
    volatile unsigned int *RESET = (unsigned int*)0x60;

    *LED_PIO = 0; //clear all LEDs

    volatile unsigned int value = 0;
    volatile unsigned int halt = 0;

    while ((1+1) != 3){
        if (*RESET == 1){
            *LED_PIO = 0;
        }
        else{
        if (*RUN == 1 && halt == 0){
            value += *SWITCH;
            *LED_PIO = value;
            halt = 1;
        }
        if (*RUN == 0 && halt == 1){
            halt = 0;
        }
        }
    }

    return 1;
}
```

Fig: main.c for the accumulator program

Function: main

Inputs:

Memory-mapped input/output registers:

***SWITCH:** Pointer to a memory-mapped input location containing the current state of switches.

***RUN:** Pointer to a memory-mapped input location indicating condition is active (1) or not (0).

***RESET:** Pointer to a memory-mapped input location indicating whether the reset condition is active (1) or not (0).

Outputs:

***LED_PIO:** Pointer to a memory-mapped output location controlling LEDs.

Description:

This function is designed to operate within an embedded system context where LEDs are controlled based on the state of switches and certain control signals (RUN and RESET). The function continuously checks the state of these controls and switches to adjust the display of LEDs accordingly. This functionality is often used in systems that require user interaction and feedback via simple interfaces such as LEDs and switches.

Operation:

Initialization:

Input/output (PIO) blocks are initialized to specific memory addresses.

LED outputs are initially cleared

Main Loop:

The function enters an infinite loop, making it suitable for embedded firmware where the

application must run indefinitely until manually stopped or reset.
Reset Handling:
Continuously checks if the RESET signal is active. If it is, all LEDs are turned off (*LED_PIO = 0).
Run and Halt Logic:
If the RUN signal is active and the system is not already "halted" (i.e., halt == 0), it performs the following actions:
Adds the current switch values (read from *SWITCH) to a running total stored in value.
Updates the LEDs to reflect the new value of value.
Sets halt to 1 to indicate that the system has processed this set of switch inputs.
If the RUN signal becomes inactive and the system was previously "halted" (i.e., halt == 1), it resets halt to 0, allowing new switch inputs to be processed when RUN becomes active again.
Continuation Condition:

B. System Verilog Description

lab7.sv Top-Level Module

The lab7 module serves as the top-level integration of the Nios II system and the FPGA fabric, ensuring that all components in the digital system interact seamlessly.
Inputs

- CLOCK_50: The 50 MHz system clock input.
- KEY [3:0]: A 4-bit input representing the push-buttons on the board.

Outputs

- LEDG [7:0]: An 8-bit output to the green LEDs on the board.
- DRAM_*: Various outputs to control the DRAM, including address lines, bank address, command lines, data bus, and clock.

Bidirectional

- DRAM_DQ [31:0]: A 32-bit bidirectional data bus for the DRAM.

Description
This module instantiates the lab7_soc module and wires up the external I/O, clock, and reset signals to the internal components. The accmu_export input is connected to the negation of KEY[3], allowing for active-low button press detection. The clk_clk input is driven by the CLOCK_50 signal, supplying the system clock to the SoC. The reset_reset_n and reset_wire_export inputs are connected to KEY[0] and the negation of KEY[2] respectively, to handle system and peripheral resets. The LED output led_wire_export and the switch input switch_export are wired directly to the LEDG output and SW input ports for user interaction. All DRAM-related signals are connected to their respective outputs, facilitating memory operations for the system.
This configuration of lab7 ensures that the FPGA's pins are appropriately mapped to the internal logic of the SoC, effectively creating the hardware platform on which the software will run.

lab7_soc.v Module

The lab7_soc module acts as the central hub for interfacing and controlling the various components within the Lab 7 system-on-chip (SoC).

Inputs

- accmu_export: Input signal for the accumulator.
- clk_clk: System clock input.
- reset_reset_n: Active-low reset signal for the entire SoC.
- reset_wire_export: Additional reset signal, possibly connected to an external reset button.
- switch_export: Input signal from the user switches.

Outputs

- led_wire_export: Output signal to the green LEDs used to display binary values.
- sdram_clk_clk: SDRAM clock output.
- sdram_wire_*: Multiple SDRAM interface signals including address, bank address, command signals, and data bus.

Description

This module serves as the processing unit of the system, utilizing a Nios II processor to send data and instructions across the SoC. The software application is executed on this processor. The on-chip SDRAM is interfaced to store data generated by the CPU, and the module also handles user input from switches and displays data on LEDs.