ECE 385

Fall 2020

Experiment 7

SOC with NIOS II in SystemVerilog

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

In this lab, we used NIOS II to control the hardware directly using the software. NIOS II is a 32-bit CPU that can be programmed using high-level language such as C. We implemented first followed the instruction to make LED constantly blinking and then implemented a simple accumulator using C instead of SystemVerilog on FPGA by ourselves.

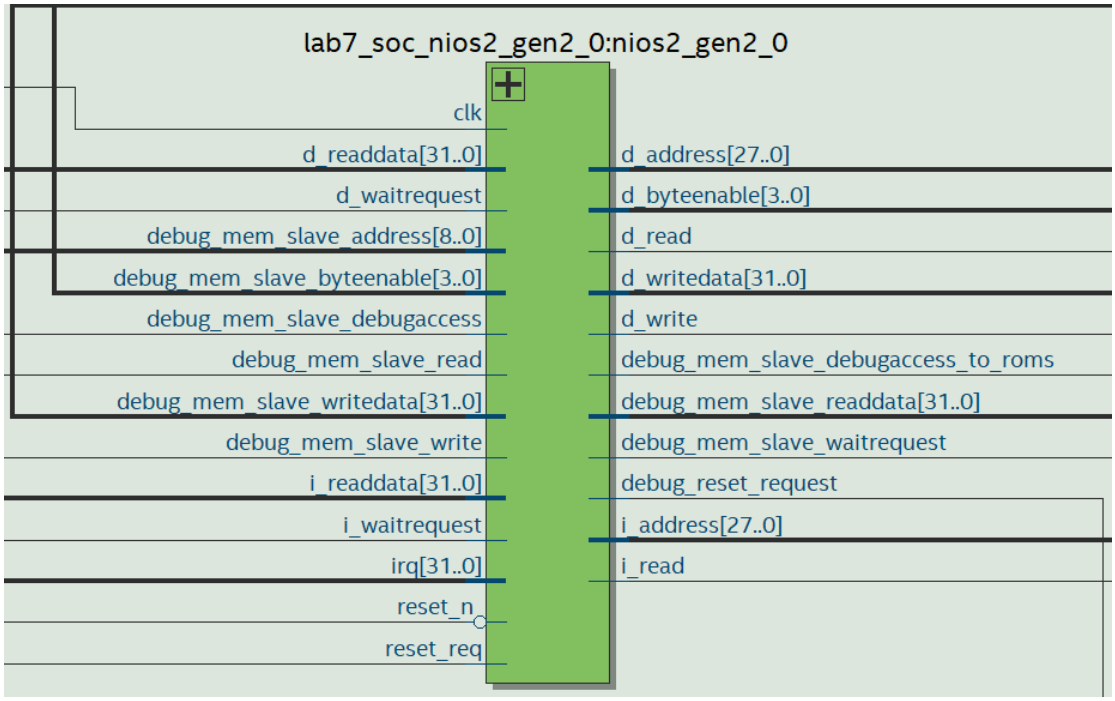
NOIS-II System:

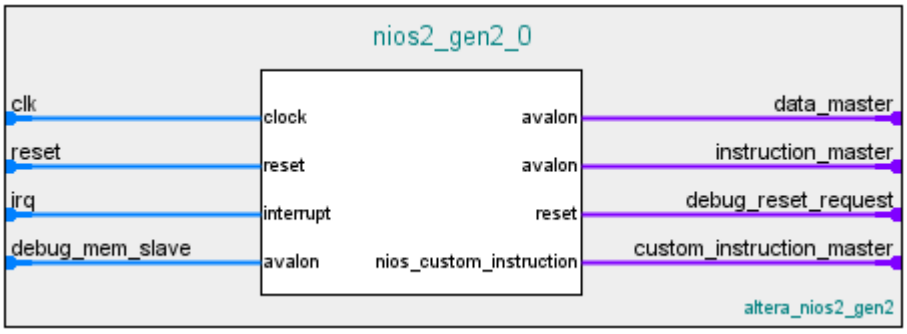
Description:

On the hardware side, the system has a default Clk signal that will be used as the clock for the rest of the modules. We also added several new modules, which are on-chip memory, the LED, the SDRAM, the PLL, and the link check. These are enough for the blink program, and the details of each module can be found below. For the accumulator program, we added SW and two buttons as the user inputs.

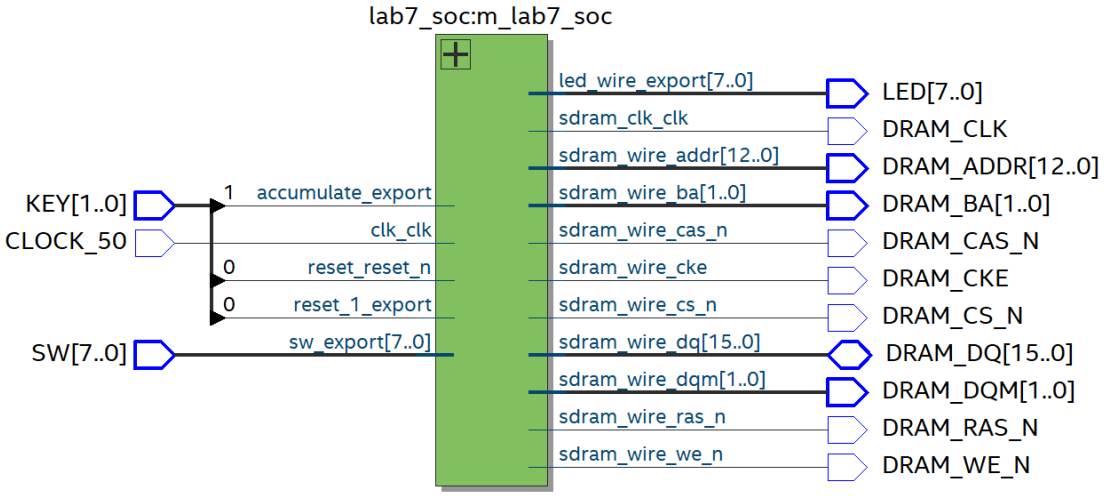
On the software, we had two C programs. The first is the blink grogram, which will make the last LED on the board constantly blink. The program gets the address of the LED and changes the LSB of the LED to high and low for a few clocks, so the last LED can blink on the board. The second program is the accumulator program. It will print the sum of the current value on the LED and the value on the switches. The program will do the calculation when “sum” button is pressed and clear the value of LED when “clear” button is pressed. The details about two programs can be found bellow.

NOIS-II Diagram:





Top Level Block Diagram:



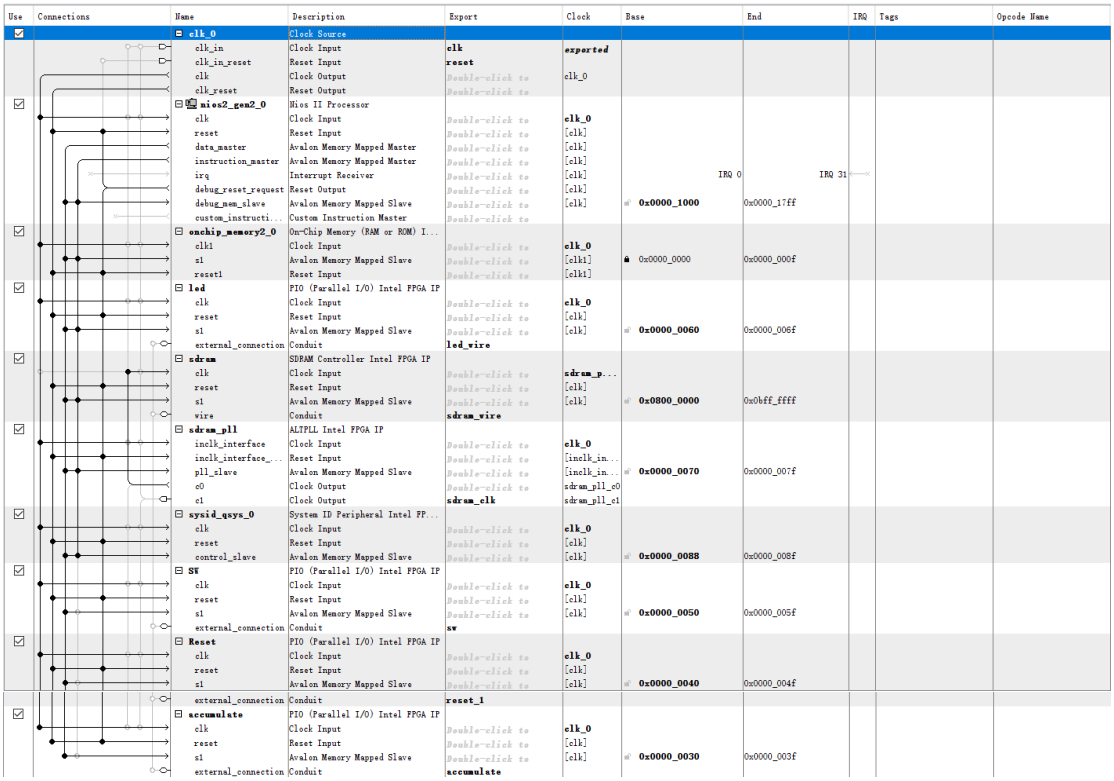
.sv Modules:

Module: lab7.sv
Inputs: CLOCK_50 [1:0] KEY [7:0] SW
Outputs: [7:0] LEDG, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N ○ DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, [3:0] DRAM_DQM
Description: gets the switches input and buttons from FPGA and then outputs the control signals to the SDRAM to control the CPU and the output signals to LEDs.
Purpose: The top-level module that connects the NIOS II to software and board.

Module: lab7_soc.v
Inputs: clk_clk, reset_reset_n, accumulate_export, reset_1_export, [7:0] sw_export
Outputs: [7:0] led_wire_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [1:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n
Inout: [15:0] sdram_wire_dq
Description: This module links the all the declared components, which are LED,

reset button, run button, clock, switches and SDRAM.
Purpose: This file is generated by the Platform Designer base on the settings during initialization.

System Level Block Diagram:



Functionalities of each blocks:

clk_0:
The clock for the other modules, although some modules will have their own clocks.

nios2_gen2_0:
The NIOS II processor, which will be responsible for the computation of the program.

onchip_memory2_0:
The on-chip memory, which will be used by the processor above to do the read and write during computation.

led:
The LED outputs to the board. The address of the LED will be used to control the on/off of each LED.

sdram:
This is the DRAM module. It is connected to the processor so the processor can get the memory contents from it.

sdram_pll:
This module generates the clock for the sdram which will be 1ns slower than the main clock. The slower clock for sdram is used to prevent the sdram from read/write corruption.

sysid_qsys_0:
This module is used to check whether the hardware and the software are connected correctly.

SW:
The input switch from the board, which will be responsible for getting the user inputs.

sum:
The input sum button on the board, which will be responsible for the user "run" input signal. When pressed, the CPU will accumulate the switch value to the current sum.

clear:
The input clear button on the board, which will be responsible for the user clear input signal to clear the LED values.

INQ Questions:

Q: What are the differences between the Nios II/e and Nios II/f CPUs?

A: NOIS II/e is the economy version, which will use as few recourses as possible, and it only support JTAG debug and ECC RAM protection. While NOIS II/f is the performance version, which will focus on the speed of the performance, and it support much more features, such as instruction/data cache.

Q: What advantage might on-chip memory have for program execution?

A: On-chip memory is much faster to read and write than loading from external memory.

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

A: It is modified Harvard machine. It is Harvard machine rather than Von Neumann machine because the instruction and data are stored in separate blocks. But it is not pure Harvard machine because we can access the instructions as data.

Q 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?

A: LED only needs to access the data bus because LED is the output signals, and it only needs the data from CPU. On the other hand, the on-chip memory needs the access of both data and the instructions, so it would need the both data and the program bus.

Q: Justify how you came up with 512 Mbit?

A: Because there are 32M x 16 chips, and the results lead to 32*16 = 512 Mbit.

SDRAM parameter	Short name	Parameter value
Data Width	[width]	16
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1
# of Banks	[nbanks]	4

Q: What is the maximum theoretical transfer rate to the SDRAM according to the timings given?

A: SDRAM can access 2 bits of data in 5.4ns, so the maximum speed is 16 bits / 5.4ns
2 Bytes / 5.4s * 1000000 = 370.37 MB/s

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

A: SDRAM is works like capacitors, it needs to be constantly refreshed in order to keep the data. If SDRAM runs too slow, it will cause the data corruption.

Q: The clock going out to the SDRAM chip (clk c1) 1ns behind of the controller clock (clk c0). Why do we need to do this?

A: Because the data bus will be constantly updated and writing to SDRAM is slow. So we need to make the clock of SDRAM 1ns slower so that the SDRAM can use that time to correctly read the data from the data bus.

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

A: The start of the execution is at address 0x08000020. We need to this step because

we need to tell the processor where to start executing, and we also need to ensure that other modules have their own non-overlap memory blocks.

Q: 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).

```
5   int main()
6   {
7       int i = 0;
8       volatile unsigned int *LED_PIO = (unsigned int*)0x20;
9
10      *LED_PIO = 0; //clear all LEDs
11      while ( (1+1) != 3) //infinite loop
12      {
13          for (i = 0; i < 100000; i++); //software delay
14          *LED_PIO |= 0x1; //set LSB
15          for (i = 0; i < 100000; i++); //software delay
16          *LED_PIO &= ~0x1; //clear LSB
17      }
18      return 1; //never gets here
19  }
20
```

A: Volatile key word is used to tell the processor that the content of the target memory may change at any time, and do not panic when it changes its value.

The code first gets the LED pointer and clear its content to be all 0s. Lines 13 and 15 is used to delay the software clock, so that the LED can stay on for a few seconds instead of letting t blink to fast to be captured by human eyes. Line 14 sets the at LSB to be high and line 16 sets the LSB of LED to be high. This will be trapped in a while true loop, so the LED will be constantly blinking.
0x00000000 |= 0x1 ➔ LED = 0x00000000 OR 0x00000001 = 0x00000001 (line 14)
0x00000001 &= ~0x1 ➔ LED = 0x00000001 AND 0x11111110 = 0x00000000 (line 16)

Q: 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.

A:
.bss: static allocation of memory
Eg. static int a;

.heap: allocate memory denamically in heap
Eg. int* a = (int*) malloc(sizeof(int));

.rodata: read-only data
Eg. const int a = 385;

.rwdata: read-write data
Eg. int var = 385;

.stack: operations in stack
Eg. int length = strlen("length of string"); (strlen uses run time stack to get the length of a string)

.text: code text
Eg.
while (true){
    print("This report will get full credit!");
}

Q: Description of Accumulator code.

A:

```
5   int main()
6   {
7       int i = 0;
8       volatile unsigned int *LED_PIO = (unsigned int*)0x60; //make a po
9       volatile unsigned int *SW_PIO = (unsigned int*)0x50;
10      volatile unsigned int *RESET_PIO = (unsigned int*)0x40;
11      volatile unsigned int *ACCUMULATE_PIO = (unsigned int*)0x30;
12
13      unsigned int sum = 0;
14      *LED_PIO = 0; //clear all LEDs
15      while ( (1+1) != 3) //infinite loop
16      {
17          if (*RESET_PIO == 0) {
18              sum = 0;
19          } else {
20              if (*ACCUMULATE_PIO == 0) {
21                  sum += *SW_PIO;
22                  if (sum > 255) sum -= 256;
23              }
24          }
25          for (i = 0; i < 100000; i++); //software delay
26          *LED_PIO = sum;
27      }
28      return 1; //never gets here
29  }
30
```

Line 13 initialize the sum result and line 14 clear the LED. In the while loop, the program will monitor on the button values and change the sum value accordingly. If the reset button is pressed at line 17, the sum will be clear to be zero. If the run button is pressed at line 20, the sum will sum up the itself with the switch value. Line 22 deals with the LED value overflow. Then line 25 make the software delay for a few seconds for the hardware. Finally, line 26 will update the LED value by the sum.

Post-Lab Question:

A:

LUT	2212
DSP	0
Memory (BRAM)	36864 bits
Flip-Flop	1796
Frequency	86.29 MHz
Static Power	96.43 mW
Dynamic Power	47.25 mW
Total Power	161.08 mW

Conclusion:

Lab Manuel Issues:

Pro:

1. The INQ is very clear and easy to follow.

Con:

1. More explanation of NOIS would be appreciated.

Summary:

In this lab, experience the use of NIOS II, which can use software to control the hardware directly. This is very efficient when you want to implement something complicated because you can use high level language like C to write your program instead of in SystemVerilog. On the other hand, because you are using the software to control the hardware, we will need to link the hardware and software correctly, which can be complicated and time consuming.