ECE 385

Spring 2020

Experiment # 7

SOC with NIOS II in SystemVerilog

Megh Shah, Ishaan Patel
Lab Section: ABC, Tuesday 11:30 AM
TA: Gene Shiue, David Zhang

Introduction:

The NIOS II is a 32-bit CPU which is very versatile and has many use cases. The SOC consists of many different logic elements which can be programmed and utilized using a high-level language, such as C, to handle tasks and operate as a system controller. The FPGA board works in parallel with the NIOS II CPU as it handles the high-performance operations and logic, and transfers it to the NIOS II CPU which handles the user interface and the data input and output. Overall, by coherently utilizing both the NIOS II CPU and the FPGA board, we are able to perform high-levels of operations as well as being able to control and interact with the peripherals in real-time.

Pre-Lab Questions (Italicized Questions in INQ Tutorial):

1. The main difference between the NIOS II/e and NIOS II/f CPUs is speed. The "e" processor stands for "economy" and is designed to use the fewest logic elements and memory resources while the "f" stands for "fast" and is designed for maximum performance. Other differences include the "f" processor having hardware multiply/divide, shadow register sets, external interrupt controllers, and much more.
2. The advantage of using an on-chip memory for program execution is speed. Having an on-chip SRAM memory can allow the programs to be executed much faster due to SRAM memory having substantially greater read/write speeds than other forms of memory as well as being closer to the cpu core.
3. The NIOS II CPU utilizes a modified Harvard architecture. This is because even though we are using separate buses for data and instructions, we still have the option to retrieve instructions as data.
4. The LED peripherals only require access to the data bus as it does not compute or operate with anything related to the programs. It simply needs access to the data to ensure that it can display the correct value that will be stored in the accumulator.
5. SDRAM requires constant refreshing because it is a form of dynamic memory. It is made up of capacitors which lose their charge over time, so to ensure that the SDRAM maintains its data, it needs to be refreshed constantly.
6. The maximum theoretical transfer rate to the SDRAM is the number of bits with one access times the inverse of the access time. In our case, since we are able to retrieve 32-bits at a time and with an access time of 5.5 ns, the maximum theoretical transfer rate would be 176 bits/ns ((1/5.5) * 32Bits).
7. The SDRAM cannot run too slowly, such as below 50MHz, as it needs to be constantly refreshed to maintain the data and values that it is holding. If there is too much time delay before the SDRAM is refreshed, it will lose its data.
8. The SDRAM clock operates at a phase shift of -3ns to accommodate for metastability and propagation delay. The SDRAM chip needs to make sure that

whatever read/write operation the processor is doing is completed and the values are stable first.

9. The NIOS II CPU starts execution from 0x1000_0000 as that is the beginning address for the SDRAM in platform designer.

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

These parameters were selected based on knowing that the total amount of memory should be 128 Mbytes. Due to this fact, we had to come up with a combination that resulted in a total memory of that value while keeping the data width at 32-bits as the NIOS II CPU is based on 32-bits.

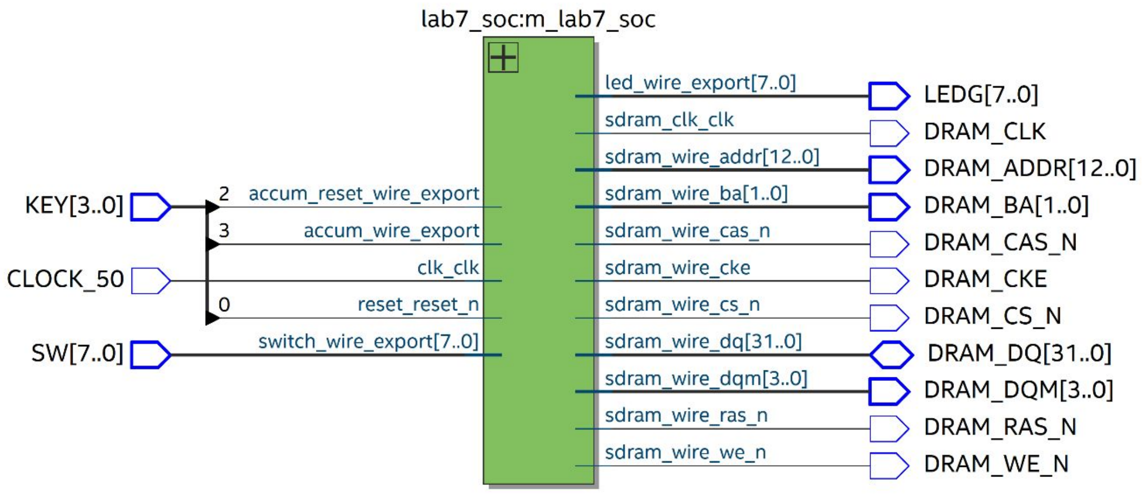
Written Description and Diagrams of NIOS-II System:

a. Summary of Operation –
   i. Regarding the hardware component of the lab, most of the hardware was designed and simulated using the Platform Designer offered by Quartus. In the Platform Designer, we created many different hardware and logic elements that were needed to simulate the lab. First, we ensured that there was a clock present as it is important to ensure a synchronous source to drive all other elements. Other hardware that was added to ensure functionality were the NIOS II Processor which is essentially the main component and connects to all other hardware, and the On-Chip Memory which acts as a type of RAM or ROM to hold temporary data and instructions. We also created a PIO output block which serves as our LED to display what value the accumulator is holding. On top of this, we instantiated an SDRAM controller with its own SDRAM clock to connect and operate with the physical SDRAM chip. We also added hardware support for switches and the accumulator, as well as the reset for the accumulator, which would be exported, so it could be used with the C program.

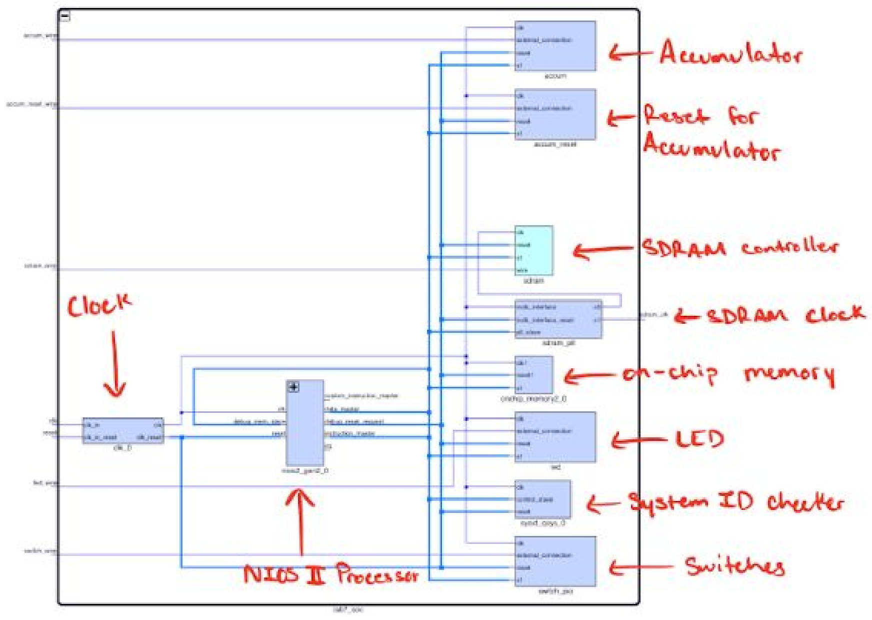
ii. Regarding the software component of the lab, we had to ensure that the exports, such as wires, from the NIOS II CPU was connected to the software side, so our code would actually impact the hardware. Regarding the blinker code and the C program, we had to create a variable LED whose address points to the physical address of the actual LED on the FPGA board. The accumulator was simply of type unsigned int that added whatever value it had with the values set in the switches. We ensured that we added delays and tests to avoid human error and maximize functionality.

b. Written Description of all .sv Modules -
   i. **lab7.sv** – The top level file that connects all ports together. Connects the wires and exports from the NIOS II processor with the I/O ports on the FPGA board, such as keys and LEDS. Also, connects the clock as well as the physical SDRAM chip.
   ii. **lab7soc.v –** Contains the wires and hardware export from the NIOS II CPU. These outputs are then connected with logic elements created in Quartus to ultimately connect to the FPGA board hardware.
   iii. **lab7.sdc –** Constraints the timings of ports and hardware, such as keys and switches, by adding or a set delay to account for metastability and propagation delays.

c. Top Level Block Diagram -



d. System Level Block Diagram -



Clock – Used to control and drive hardware in the system on chip

NIOS II Processor – Connects to all hardware to control what each element needs to do
Accumulator – Used to store the value of the accumulator in the C program

Accumulator Reset – Resets the value of the accumulator to zero

SDRAM Controller – Contains instructions which the NIOS II processor executes from

SDRAM Clock – Special clock used for the SDRAM to avoid metastability and **propagation delays**

On-Chip Memory – Used to store data and values. Needs to be constantly refreshed or else will lose data

LED – PIO output module which reads data and displays the value

System ID Checker – Checks whether the hardware and software are compatible

Switches – PIO input module that takes in user input

Post-Lab Questions:

LUT	2291
DSP	0
Memory (BRAM)	36,864
Flip-Flop	615
Frequency	81.63 Mhz
Static Power	102.03 mW
Dynamic Power	40.22 mW
Total Power	195.67 mW

The only problem that we had was when we pressed the accumulate button, the FPGA would add multiple times. The reason for this is because we didn't wait for the button to be depressed. The way we fixed this problem is by making the processor go into a while loop until the button is fully depressed. This will force only one accumulation. This problem was a good introduction into the world of bare metal electronics and programming.

Conclusion:

The functionality of our design was very straightforward. We did not have many issues or errors since we prepared well beforehand. The only issue we ran into was a logical issue, as mentioned above, and was fixed fairly easily. There was nothing that was ambiguous or unnecessarily difficult regarding this lab. All documentations and instructions needed to complete this lab were provided and were through and well-detailed. Overall, this lab provided us with great understanding regarding SoCs. We now thoroughly understand what an SoC is and how it operates as we delve deeper into FPGA software and hardware and begin to utilize more and more of its potential.