ECE 385

Experiment #7

Spring 2018

SOC with NIOS in SystemVerilog

Sahil Shah/sahils2
Samir Kumar/samirkk2
ABC - Tues, 11:30-2:20
Vibhakar Vemulapati

Introduction:

This lab is designed as an introduction to QSYS and the NIOS II SoC. This lab shows basic functionality and design if the NIOS II processor. The second part of this lab shows how to connect peripherals like on-board LEDs, switches, and keys. The NIOS II processor can be programmed with a high level language (like C) to perform low performance operations.

Summary of Operation:

The IP blocks instantiated outside of those required to run the NIOS II processor were the PIO blocks. The only PIO module used in part one of the lab was for the on-board led. The second part of the lab required use of 2 one bit keys and 18 bit switch inputs and 8 bit led outputs. All of these PIO blocks connected to the data master on the NIOS II. One of the keys was used as a reset button and the other was used as to control when to accumulate the value in the switches. The switches were used to control the number added to the value in the LEDs.

The software program for this lab controlled the accumulation process. On the press of Key 3, the data held in the switches was added to the data stored in the LEDs. The data in both is accessed by dereferencing each pointer, whose value was given by Qsys. Then, to change the data, the pointers are dereferenced then edited to the new value. When Key 2 was pressed, the data in the led was accessed via dereferencing the pointer and setting the value to 0. The value in the switches was left unchanged by the program when reset was pressed.

Written Description of all Verilog and SystemVerilog Files:

Module: lab7.sv
Inputs: CLOCK_50
[3:0] KEY
[7:0] SW
[31:0] DRAM_DQ
Outputs: DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_ RAS_N, DRAM_WE_N, DRAM_CLK
[1:0] DRAM_BA
[3:0] DRAM_DQM
[7:0] LEDG
[12:0] DRAM_ADDR
[31:0] DRAM_DQ
Description: This top_level module instantiates the lab7_soc module created by Qsys
Purpose:

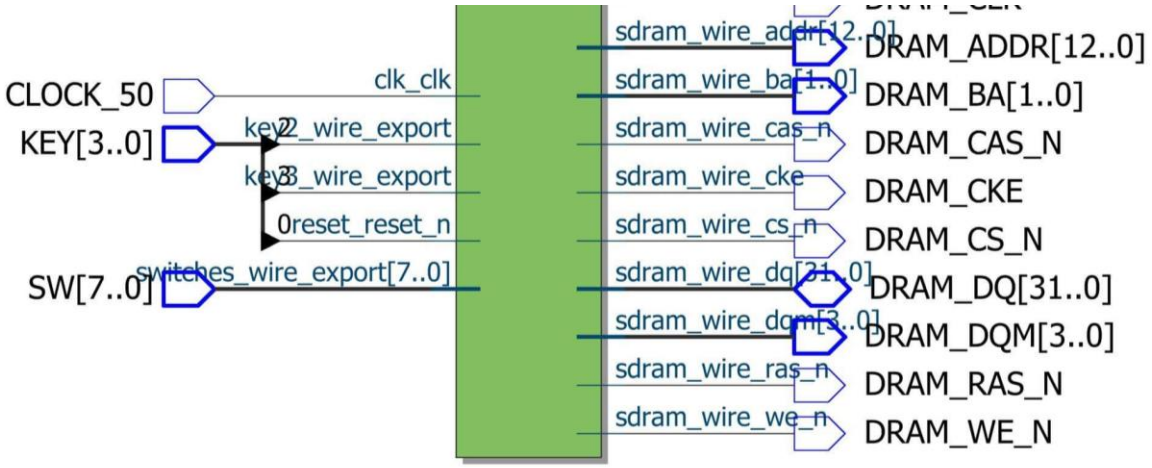
Module: lab7_soc.v
Inputs: clk_clk, key2_wire_export, key3_wire_export, reset_reset_n
[7:0] switches_wire_export
[31:0] sdram_wire
Outputs: sdram_clk_clk, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_ras_n
[3:0] sdram_wire_dqm
[7:0] led_wire_export
[12:0] sdram_wire_addr

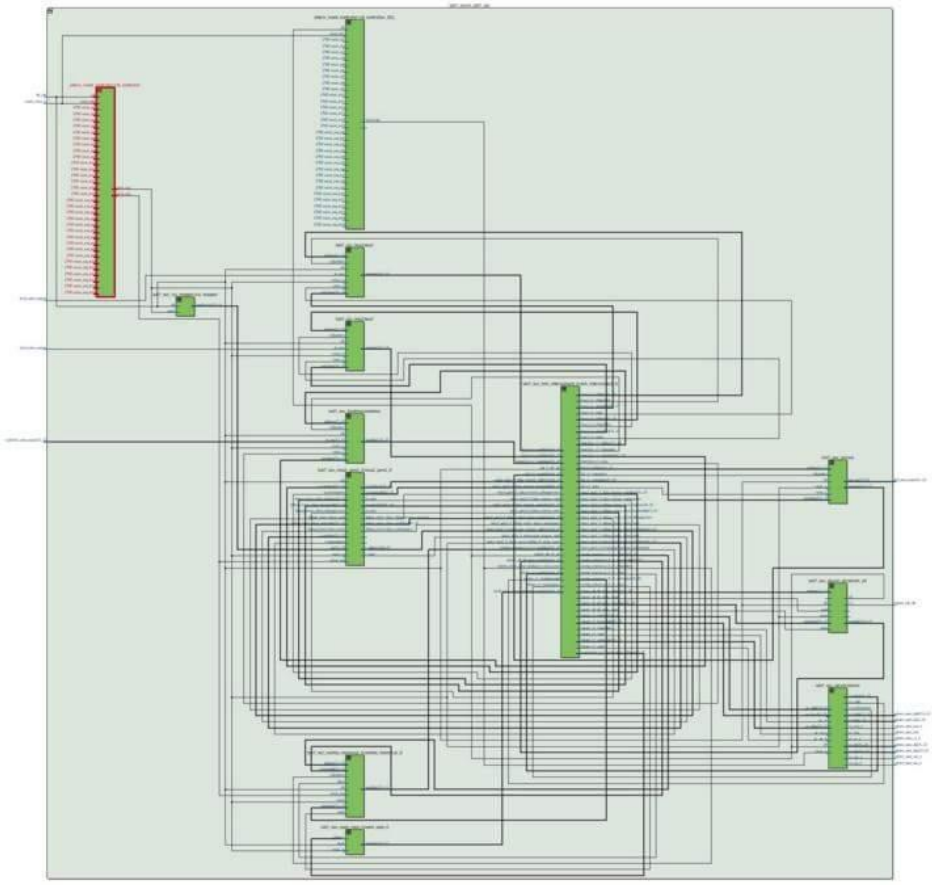
[31:0] sdram_wire_dq

Description: this file is the verilog file created when the qsys designed is generated

Purpose: This file contains the HDL code for the designed created in the Qsys. The code in this file builds the SoC

Block Diagrams:





INQ **Questions:**

1. **What advantage might on-chip memory have for program execution?**
    a. On-chip memory does not have the delay associated with off-chip memory. Off chip memory delay comes from the physical distance between the chip and memory on the board.
2. **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 a modified Harvard machine because the contents of the instruction can be accessed as if it were memory
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?**
    a. The led peripheral only needs to display the values in the data bus and nothing else.
4. **Why does SDRAM require constant refreshing?**
    a. SDRAM requires constant refreshing because dynamic RAM uses capacitors to store memory. If not refreshed, the capacitors lose their charge which results in a loss of memory for the RAM chip
5. **Make sure this is consistent with your above numbers; you will need to justify how you came up with 1GB with your TA (picture of table below**

SDRAM PARAMETER	SHORT NAME	Parameter Value (fill in from the data sheet)
DATA WIDTH	[width]	16
# of ROWS	[nrows]	13
# of COLUMNS	[ncols]	10
# of CHIP SELECTS	[ncs]	2
# of BANKS	[nbanks]	4

6. **What is the maximum theoretical transfer rate to the SDRAM according to the timings given?**
    a. Maximum theoretical transfer rate is 86.53 MHz.

7. **The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?**
    a. If the capacitors in SDRAM are not refreshed at a sufficient frequency, they lose charge and therefore data.
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.**
    a. 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?**
    a. The NIOS II base address is 0000 1000.
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.**
    a. Line 7: initialize the variable 'i' to 0.
    b. 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
    c. Line 10: deference the pointer and reset the value that it points to (set it to zero
    d. Line 11: establish an loop that runs on the conditional 1+1 != 3. This is always true so an infinite loop is created
    e. Line 13: set up a for loop with nothing to execute. This acts as a software delay
    f. Line 14: Set the value pointed to by LED_PIO to the result of the OR operation the the current value in the LEDs and 0x1. This sets the value of the LED to 1.
    g. Line 15: same as line 13
    h. Line 16: Set the value pointed to by LED_PIO to the result of the OR operation the the current value in the LEDs and 0x0. This sets the value of the LED to 0.
    i. 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**
    a. .bss: statically allocated variables not specifically initialized
        i. int j;

b. .heap: dynamically allocated variables and objects
    i. int *j = malloc(sizeof(int));
c. .rodata: static constants -- “read only”
    i. const int j = 3;
d. .rwdata: variables-- “read/write”
    i. int j = 3;
e. .stack: statically allocated variables and function calls
    i. int j = 4;
    ii. myfunc();
f. .txt: the actual code
    i. int main() {
    //code here
    }

Post-Lab: Design Resources and Statistics

	Slow 85C	Slow 0C
LUT	2184	
DSP	0	
Memory (BRAM)	10.368 kB	
Flip-Flop	1909	
Frequency (MHz)	79.92	86.53
Static Power (mW)	102.08	
Dynamic Power (mW)	45.09	
Total Power (mW)	208.79	

Conclusion:

Our design worked without issue. The only issue at times was understanding why connections were made the way they were shown in the INQ. Connecting the extra PIO blocks in part 2 of the lab simply entailed doing the same thing for the PIO block for the LEDs but setting the status to input instead of output.

We found that Eclipse was very buggy and would crash randomly. Also we got errors such as hash or timestamp errors. While vague, we ended up asking around and figured out what to do to fix those problems. Sometimes however, we got those errors even though there were no mistakes in the code. So after closing and reopening the code, the error went away all of a sudden.