ECE385
Spring 2019
Experiment #7

SOC with NIOS II in SystemVerilog

Austin Lee (sal3)
Zuling Chen (zulingc2)
Tuesday 3PM ABE
Patrick Wang, Gene Shiue

Introduction

In Lab 7, we explored the System-on-Chip (SOC) functionality of the NIOS II processor on the Cyclone IV device. We learned how to combine SystemVerilog modules for the FPGA and instruction code in C for the NIOS II to create a functioning accumulator. In the process, we completed a tutorial on the usage of the Platform Designer tool to create instances of IP device blocks.

Written Description

Summary of Operation:

Our designed needed to have three PIO blocks added onto the essential blocks needed for the board to function. They were keys and switches, and then the output LEDs. The PIO blocks for the keys and switches served as input blocks that recieved data from any switch and key presses. These input block for switches was 8 bits wide while the keys were 4 bits wide. The LED outputs also used a 8 bit wide block.

Software Components:

For the software part of the lab, only one file was written. In our main.c file, we wait to see whenever a key was pressed. This input data came from the keys PIO block and then depending on which key was pressed, it would either reset the current accumulator to 0 or add on to the accumulator. Our main.c code also took inputs from the switch PIO block to know what number to add onto the accumulator. We implemented a isPressed variable and set it to 1 whenever a key was pressed so that the loop for adding didn't add the switch data more than once. After the key is released, we reset isPressed's value back to 0. To get around the overflowing of the accumulator at 255, we just took the current accumulator amount and used C's mod function. This would result in accumulator always being a number between 0 and 255, never reaching 256.

Module Descriptions

Module: Lab7 (Top-level file)
Inputs: CLOCK_50, [3:0] KEY, [7:0] SW;
Outputs: [7:0] LEDG, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
Inout: [31:0] DRAM_DQM;
Description: Top level file for Lab 7. Contains a single call to lab7_soc.
Purpose: As the top level entity, this module simply instantiates the SOC and thus all the other modules generated by .qsys file and connects the NIOS II system with the FPGA hardware.

Module: Lab7_soc
Inputs: clk_clk, [3:0] key_wire_export, reset_reset_n, input wire [7:0] sw_wire_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, [3:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n.
Inout: [31:0] sdram_wire_dq
Description: Top level file for the SOC. Controls all the inputs/outputs/inouts to and from the platform designer.
Purpose: This file serves as the connections between all the blocks made in the platform designer. It connects various signals between all the submodules created including our NIOS_GEN2 block, our PIOs created for the purposes of this lab and the SDRAM blocks.

Block Diagram(s)

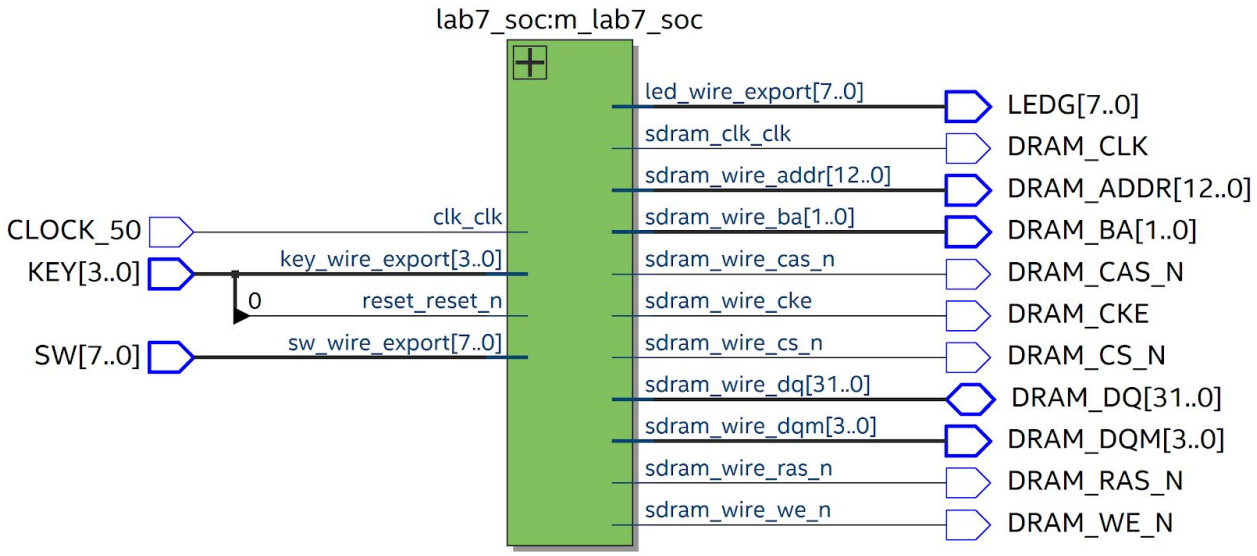

Figure 1: Top Level Lab 7

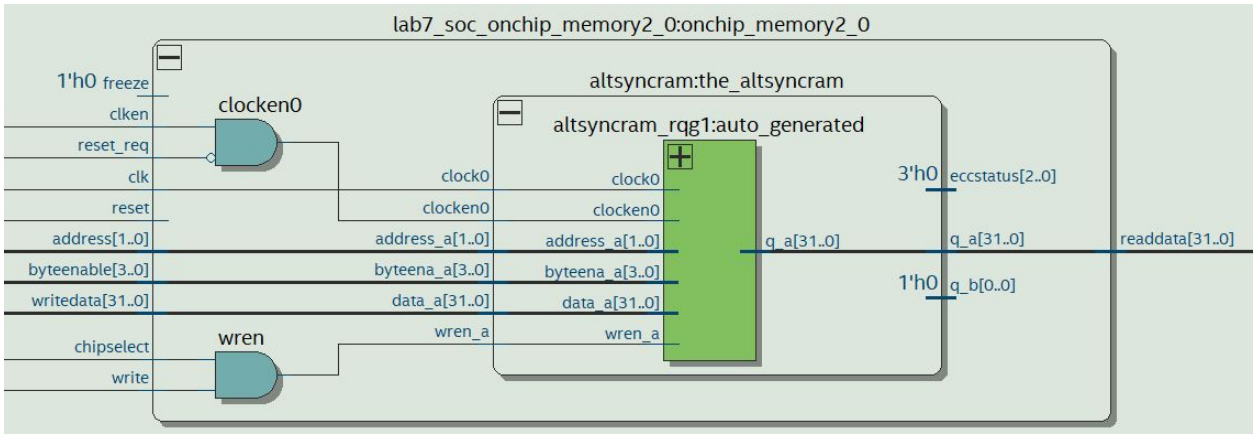

Figure 2: On-Chip Memory

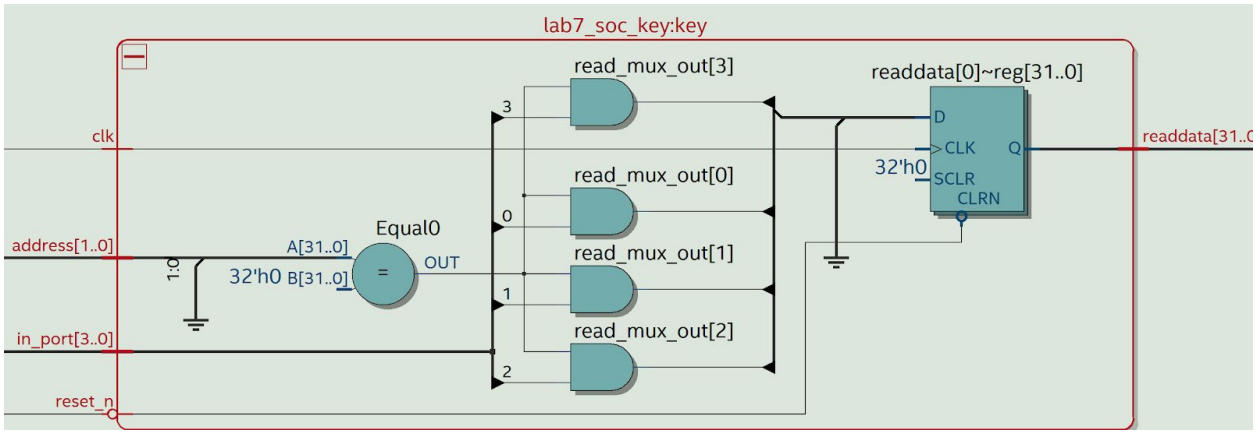

Figure 3: Key PIO

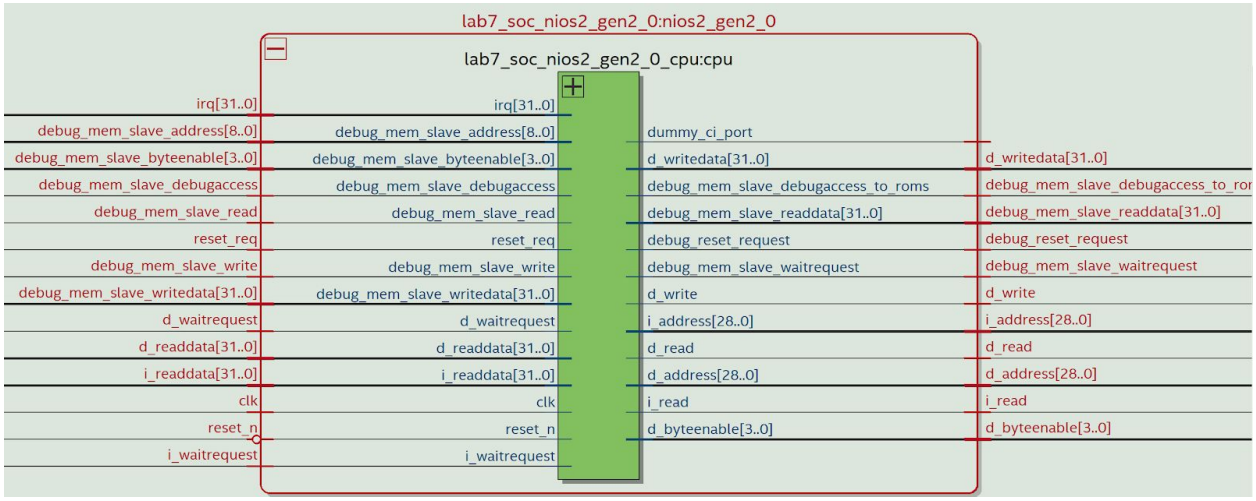

Figure 4: NIOS II

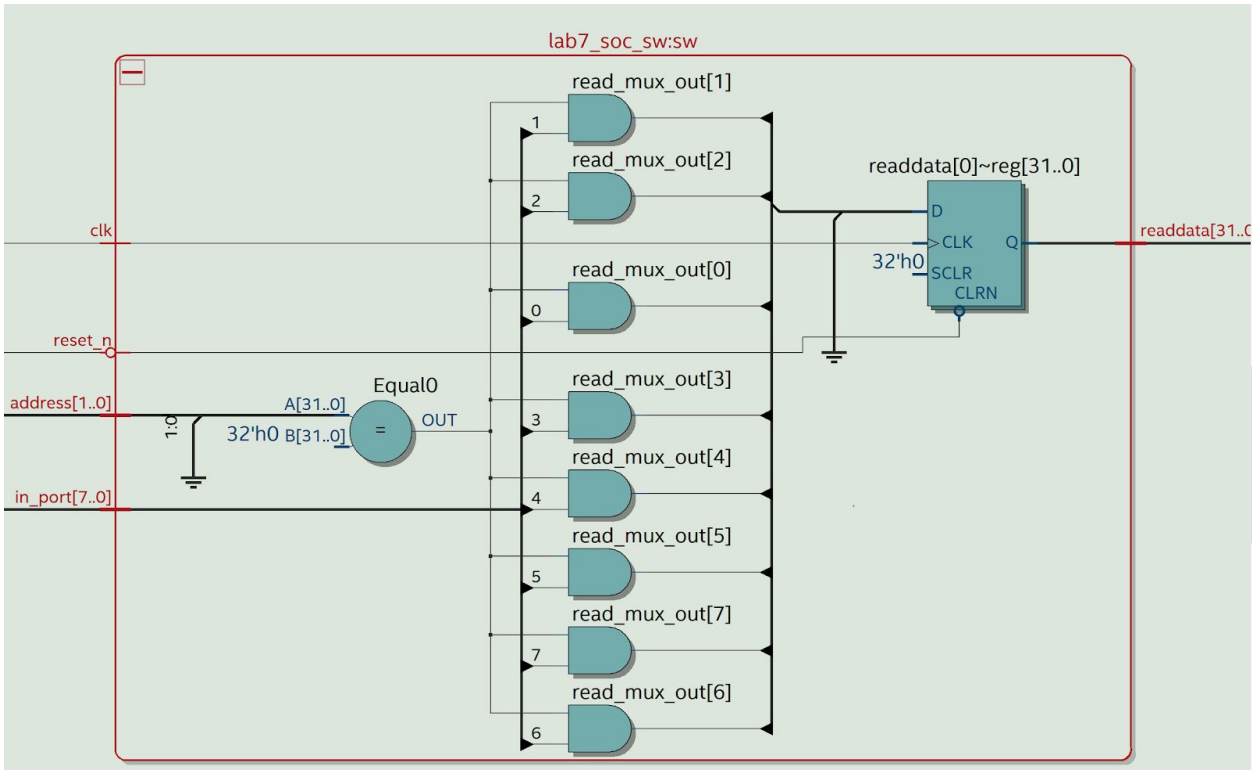

Figure 5: Switches PIO

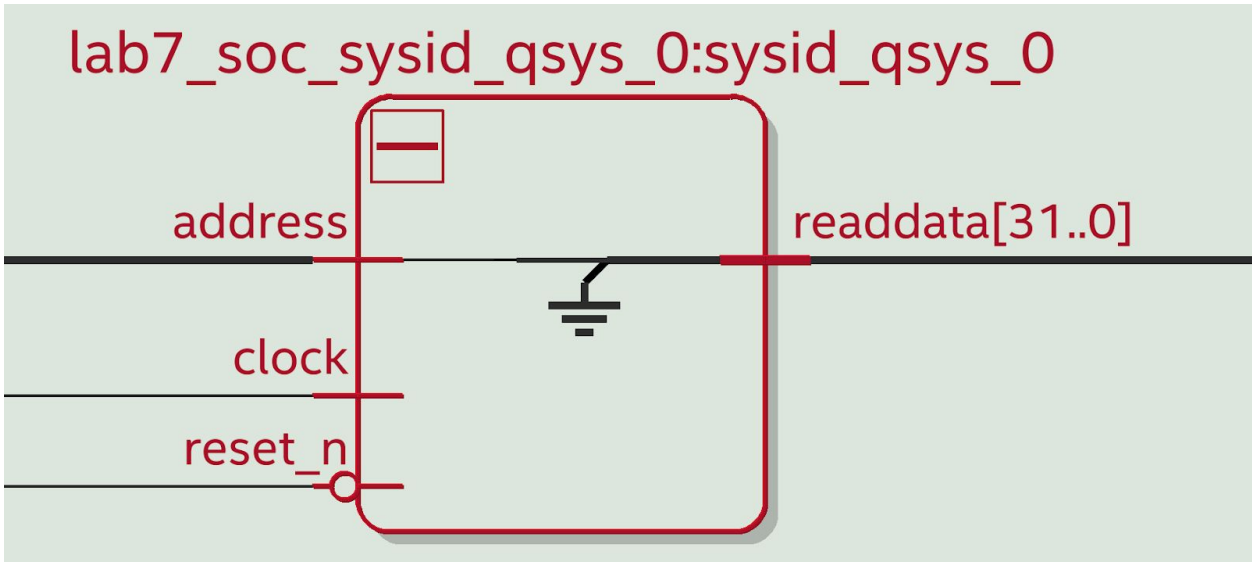

Figure 6: Sysid_qys_0

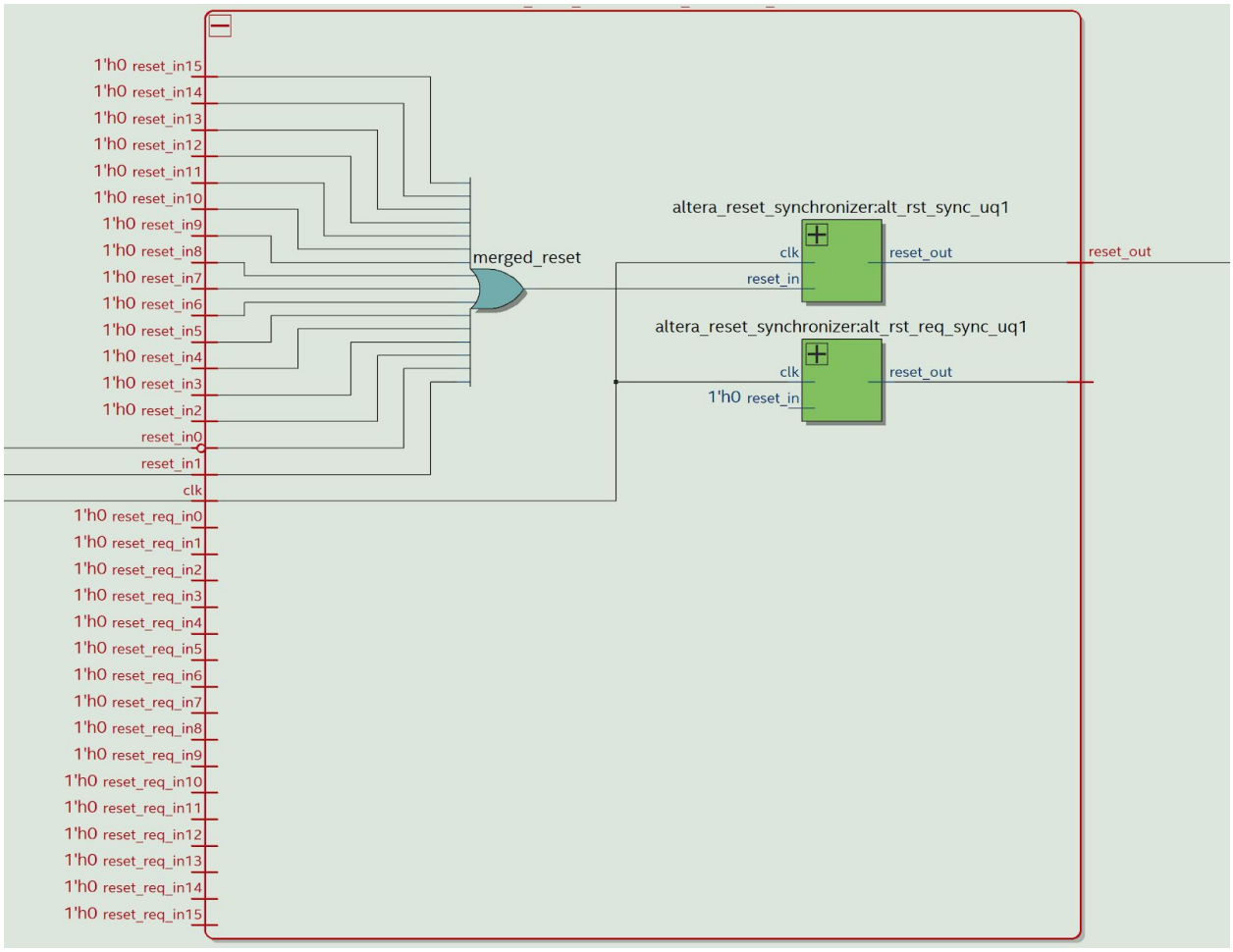

Figure 7: Reset Controller

Figure 8: SOC Connections

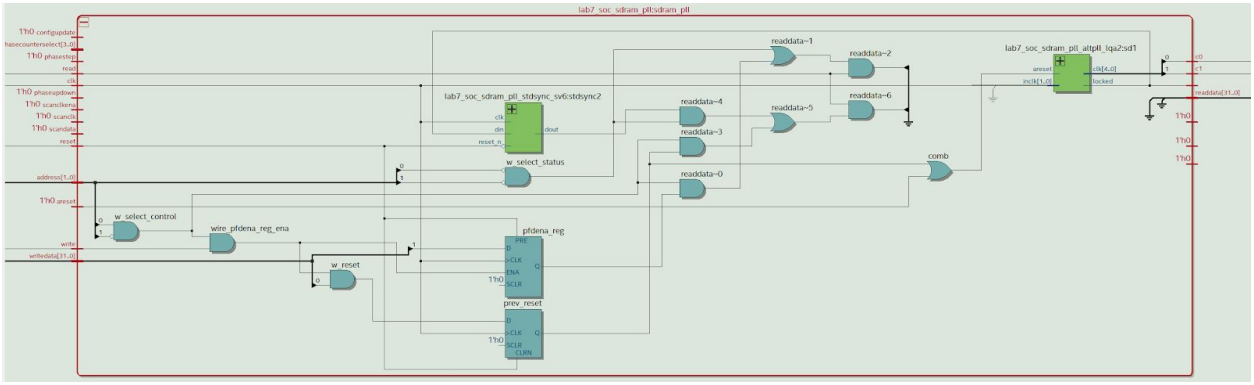

Figure 9: SDRAM

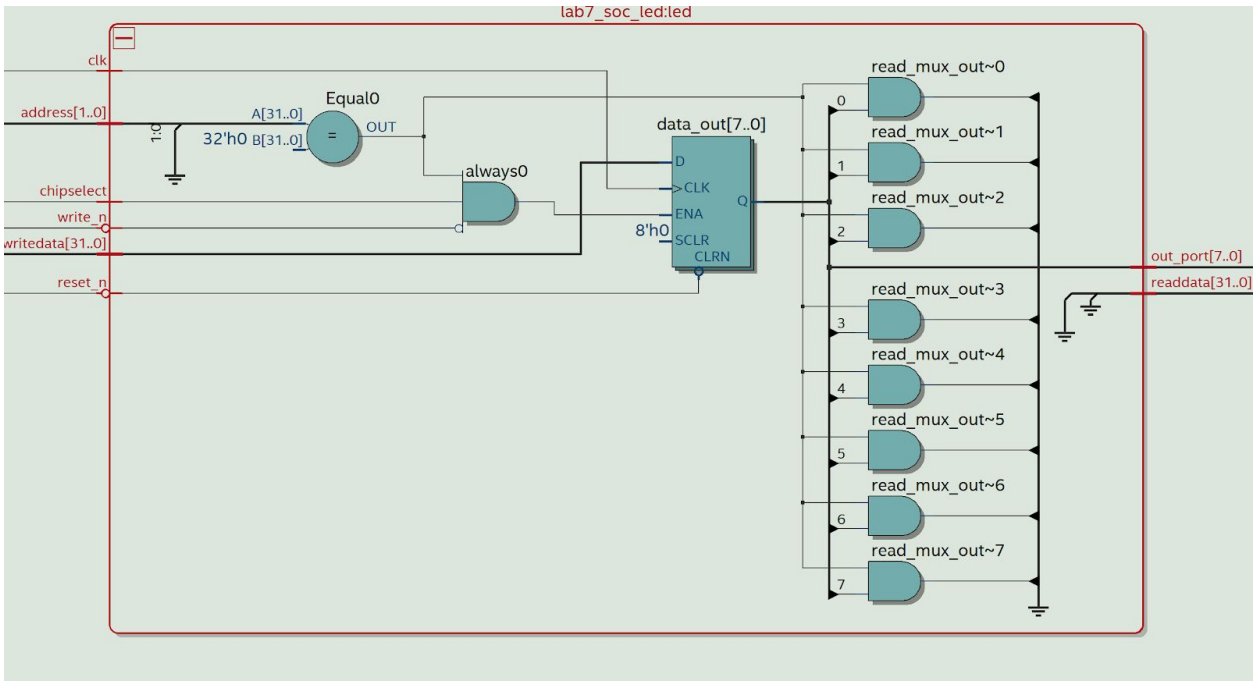

Figure 10: LEDs

Tutorial Questions

1. What are the differences between the Nios II/e and Nios II/f CPUs?
    a. The Nios II/e processor is an economy version of the processor. There shouldn't really be a difference in this lab because we are only performing simple functions like making the LED blink. The Nios II/e version does not support hardware multiplication and division, unlike the Nios II/f CPU. The Nios II/e version requires less FPGA resources than the Nios II/f version does and runs at a higher max frequency.
2. What advantage might on-chip memory have for program execution?
    a. On chip memory makes it so that read/write times are faster than if we were to process all the data and instructions through hardware components.
3. Note the bus connections coming from the NIOS II; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?
    a. This machine is a modified Harvard machine because it is able to read and write at once, where as a Von Neumann can only read or write at once.
4. 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 case?
    a. The peripheral for LEDs only needs access to the bus because its purpose is to receive data from the bus. It does not input any data into the program. The on-chip memory however needs access to both buses as it needs to send and receive data.
5. Why does SDRAM require constant refreshing?
    a. Memory is stored in bits through transistors and capacitors, but the capacitor will leak charge. Therefore, constant refreshing is used to ensure that no memory is lost by reading and re-writing the existing data onto itself.
6. What is the maximum theoretical transfer rate to the SDRAM according to the timings given?
    a. The maximum theoretical transfer rate is 720 MB/s.
7. The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?
    a. Given that SDRAM requires constant refreshing in order to maintain data, an SDRAM running at a frequency too low would refresh at a slower rate as well. Memory could be lost due to charge leakage.
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 behind of the controller clock (clk c0). Why do we need to do this?
    a. The offset of 3ns between the two clock cycles is to ensure that no glitches occur. We make it so that the SDRAM clock edges only happen after the inputs have been stabilized.

9. What address does the NIOS II start execution from? Why do we do this step after assigning the addresses?
    a. We start the NIOS II execution from Address x10000000. We do this step after assigning the address so that the processor has a place to go for reset and special cases. This also prevents memory from overlapping each other and it helps to tell the program where every address is relative to the starting address.
10. Explain what each line in the C code does.
    a. Line 5: Makes a pointer to the LED PIO address
    b. Line 7: Makes a pointer to the Switch PIO address
    c. Line 9: Makes a pointer to the KEY PIO address
    d. Line 13: Instantiates the sum
    e. Line 15: Instantiates isPressed variable
    f. Line 17: Instantiates LED PIO value to 0
    g. Line 19: Starts infinite while loop
    h. Line 21-23: If KEY PIO is 2 (reset), sum becomes 0
    i. Line 25-31: If KEY PIO is 1 and isPressed is 0, sum adds data from SWITCH PIO and new sum value is modded 256
    j. Line 31: isPressed is set to 1 so the sum adding only occurs once
    k. Line 35-37: Resets isPressed to 0 after KEY PIO is no longer 1
    l. Line 39-45: Makes the LED blink
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 : Static double variable, //This is like making a global variable
    b. .heap ptr(int*)malloc(sizeof(float)) //creating a standard heap
    c. .rodata: const int i = 10000 //this data is read, so it should be constant
    d. .rwdata : int i = 12323; //this data is read/write, so it need not be constant
    e. .stack char c = function(); //Standard stack
    f. .text char yes[] = “print this” //this just stores text

SDRAM parameters

SDRAM parameter	Short name	Value
Data Width	[width]	32
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1

# of Banks	[nbanks]	4

Post-lab

LUT	2157
DSP	0
Memory(BRAM)	10368
Flip-Flop	1950
Frequency	72.4 MHz
Static Power	104.05 mW
Dynamic Power	42.38 mW
Total Power	198.84 mW

Conclusion

We were unable to configure the FPGA board for one of the laptops we used; it was not detected by the software in the Run Configuration window. After moving all the code onto our other laptop however there was no issue and it worked on the first try. We wonder if there was some un-debuggable system incompatibility issue with the FPGA that may have caused our previous demo in the previous lab to fail as well. Otherwise, the program worked smoothly but not exactly as designed. For the demo, instead of pressing one key to add the switches data onto the total sum, we had to press KEY3, KEY2, and KEY1 simultaneously. This bug was later fixed in our code. Other than that minor detail, we were able to successfully achieve all the objectives of this lab.