

ECE385 Experiment #7

Eric Meyers, Ryan Helsdingen

Section ABG; TAs: Ben Delay, Shuo Liu

March 16th, 2016

emeyer7, helsdin2

I. INTRODUCTION

This lab is meant to give a brief introduction to NIOS System on a Chip (SOC) with an SDRAM controller and PIO block. This platform will be used to make an LED blink and progress into the design of a very simple accumulator program in C to output LEDs according to switches on the Altera FPGA Board.

II. DESCRIPTION OF CIRCUIT

There are two parts to lab 7, the blinking LED circuit and more or less an extension of the blinking LED, the 8 bit accumulator. Both circuits are programmed in C through the NIOS environment, stored in the off-chip SDRAM, and linked to input/output through PIOs created inside Qsys. The blinking LED uses a PIO (Parallel I/O) block to control the state of the LEDs. The 8-bit accumulator has the same PIO for the LEDs as well as a PIO block for the 8 input switches.

The accumulator also has two input buttons, 'Accumulate' and 'Reset'. At high level programming, the accumulator is intended to input an 8-bit switch value and add it to the value of the output 8-bit value of the LEDs each time the 'Accumulate' button is pressed. The value overflows to 0 at 255. If the 'Reset' button is pressed, then the LEDs get cleared.

III. PURPOSE OF MODULES

The top-level module for lab 7 takes in a 50 MHz clock signal, 5 bits for keys/buttons, 8 bits for switches, and 32 data bits from the DRAM. The module outputs 8 LED bits and several other bits to interface with the SDRAM including the 32 data bits. The key input and output elements needed for this lab, however, are KEY[0] for hardware reset, KEY[2] for 'Reset' button,

KEY[3] for 'Accumulate' button, CLOCK_50 for 50 MHz clock input, 8 bit LEDG for LEDs, 8 bit SW for controlling the switches, and all SDRAM connecting bits.

IV. SCHEMATIC/BLOCK DIAGRAM

Please refer to "Section VIII: Figures" to view the Schematic/Block Diagrams for this lab. In Figure 1 is the top level module and Figure 4 shows the interconnections within this module. Unfortunately due to the processor and SDRAM being too complex to view on this report, they have been omitted. Figure 3 and Figure 2 show the key and switch PIO module that were added into the Lab7.sv file to create the accumulator. //

V. ANSWERS TO INQ QUESTIONS

INQ.6:1: What advantage might on-chip memory have for program execution?

The advantage would be that the program execution will have quicker access time thus speeding up the execution time (i.e. faster R/W times).

INQ.6:2: Note the bus connections coming from the NIOS II : Is it a Von Neumann, "Pure Harvard", or "Modified Harvard" Machine and why?

Because the connections for the instructions on the NIOS II Processor and the On-Chip Memory are going to the same place, this system is considered a Von Neumann Machine.

INQ.7:3: 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?

The LED PIO Module does not need any access to the program bus and is only outputting data. Therefore, it only needs access to the data bus.

INQ.7:4: Why does SDRAM require constant refreshing?

In a DRAM chip, every bit of memory data is stored as the presence or absence of an electric charge on a small capacitor. As time passes, the charges in these cells leak away, so without being refreshed the stored data would eventually be lost. This is the case with Asynchronous DRAM and also the case with Synchronous DRAM (SDRAM).

INQ.8:5: Justify how you came up with one GBit to your TA?

SDRAM Parameter	Value
Data Width	32
# of Rows	13
# of Columns	10
# Chip Selects	1
# of Banks	4

Rows * # Columns = Total Addressability/Bank

Total Address./Bank * # Banks = Total Address./Chip

Total Address./Chip * #Chips = Total Overall Address.

Total Overall Address. * Data Width = Total Memory

$$2^{13} * 2^{10} * 4 * 2 * 32 = 1\text{GBit}$$

INQ.8:6: What is the maximum theoretical transfer rate to the SDRAM according to the timings given?

Access Time(5.5ns) + Active to Write Delay (20ns) + Write Recovery Time (14ns) = 39.5ns.

We can write 32 bits in one write. 32 bits divided by 39.5 ns gives a maximum transfer rate of approximately 772.59 Mbits/second.

INQ.8:7: The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?

The SDRAM cannot run too slowly because it might not refresh quickly enough and lose its contents. Since the SDRAM runs off of electrical charge, it must be

refreshed quickly.

INQ.10:8: Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under SDRAM controller.

"At higher clock rates, a PLL is necessary to ensure that the SDRAM clock toggles only when signals are stable on the pins. If you use a PLL, you must tune the PLL to introduce a clock phase shift so that SDRAM clock edges arrive after synchronous signals have stabilized" -p.21 Embedded Peripherals IP User Guide

INQ.13:9: What address does the NIOS II start execution from? Why do we do this step after assigning the addresses?

The NIOS II starts executing from x10000000. This is the reset address of the processor and upon pressing a reset signal, this will set the execution address back to this.

INQ.18:10: 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).

The volatile keyword indicates to the compiler that the code should not be optimized to removed any "unused" variables. If the volatile keyword is not used then the program will be stuck in an infinite loop.

The set and clear functions work by using bitwise operators to "set" (turn to active high or active low) or "clear" (vice-versa) a given bit to turn it "on" or "off".

INQ.19:11: Look at the various segment (.bss, .heap, .rodata, .rdata, .stack, .text), what does each section mean? Give an example of C code which places data into each segment.

The .bss is a segment will all zeros. All variables that are statically declared but not assigned will be allocated in bss so that they are initialized to zero.

The .heap is the segment that is allocated for use by the program. Any variable/pointer that is assigned to heap memory will use the "new" keyword and will be allocated here. User is responsible for disposing of this data or else memory leaks will occur.

The .rodata is a segment of memory similar to the .bss in that they both contain statically allocated variables. Instead of being initialized to zero however, it will be initialized with whatever the user defines. The rodata is read-only. Similar to the rodata, there is the rwdata which is simply read-write privileged.

The .stack segment contains all memory allocated by a given program. The stack is local to the process itself and will contain all local variables, parameter values, etc.

The .text segment is read only and contains the actual executable code. This is the low-level assembly language that is then translated into machine language.

VI. POST LAB

Refer to Table I for the design statistics table of this lab.

Resource	Value
LUT	2215
DSP	0
Memory (BRAM)	64,512
Flip-Flop	574
Frequency	77.05 MHz
Static Power	102.04 mW
Dynamic Power	42.54 mW
Total Power	198.03 mW

TABLE I: Design Statistics

VII. CONCLUSION

For experiment 7, the team was successfully able to configure the NIOS II based system on the Altera Cyclone IV. The blinking light demo proved functionality between the Qsys wizard, NIOS II device, SDRAM controller, and PIO (Parallel I/O).

The accumulator was also successfully constructed by the team by constructing another PIO block for the eight switches on the DE2 and adjusting the main.c program to handle the input, output, and the overall functionality of the accumulator.

VIII. FIGURES

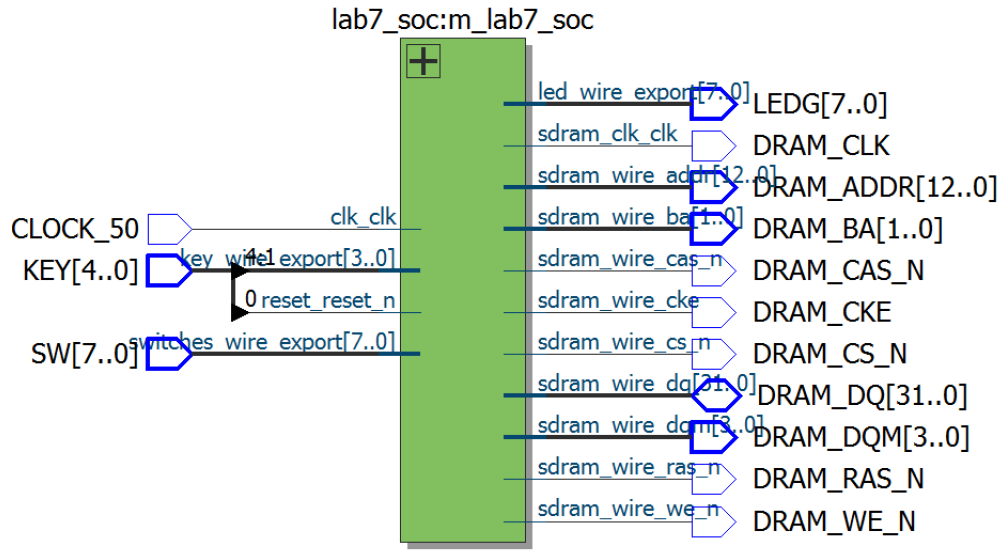


Fig. 1: Lab7 Top Level Block Schematic

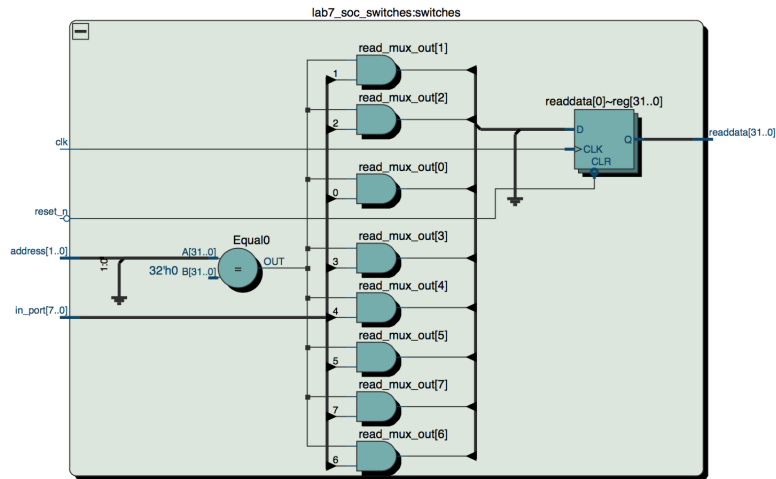


Fig. 2: Switches PIO Module

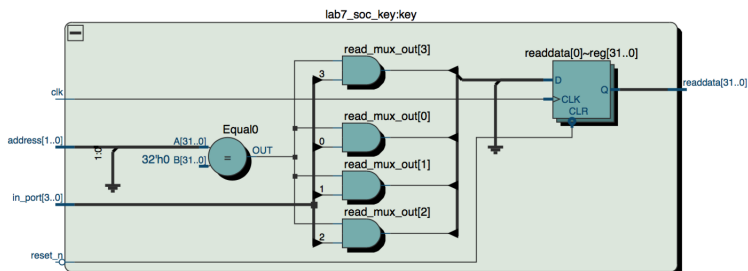


Fig. 3: Key PIO Module

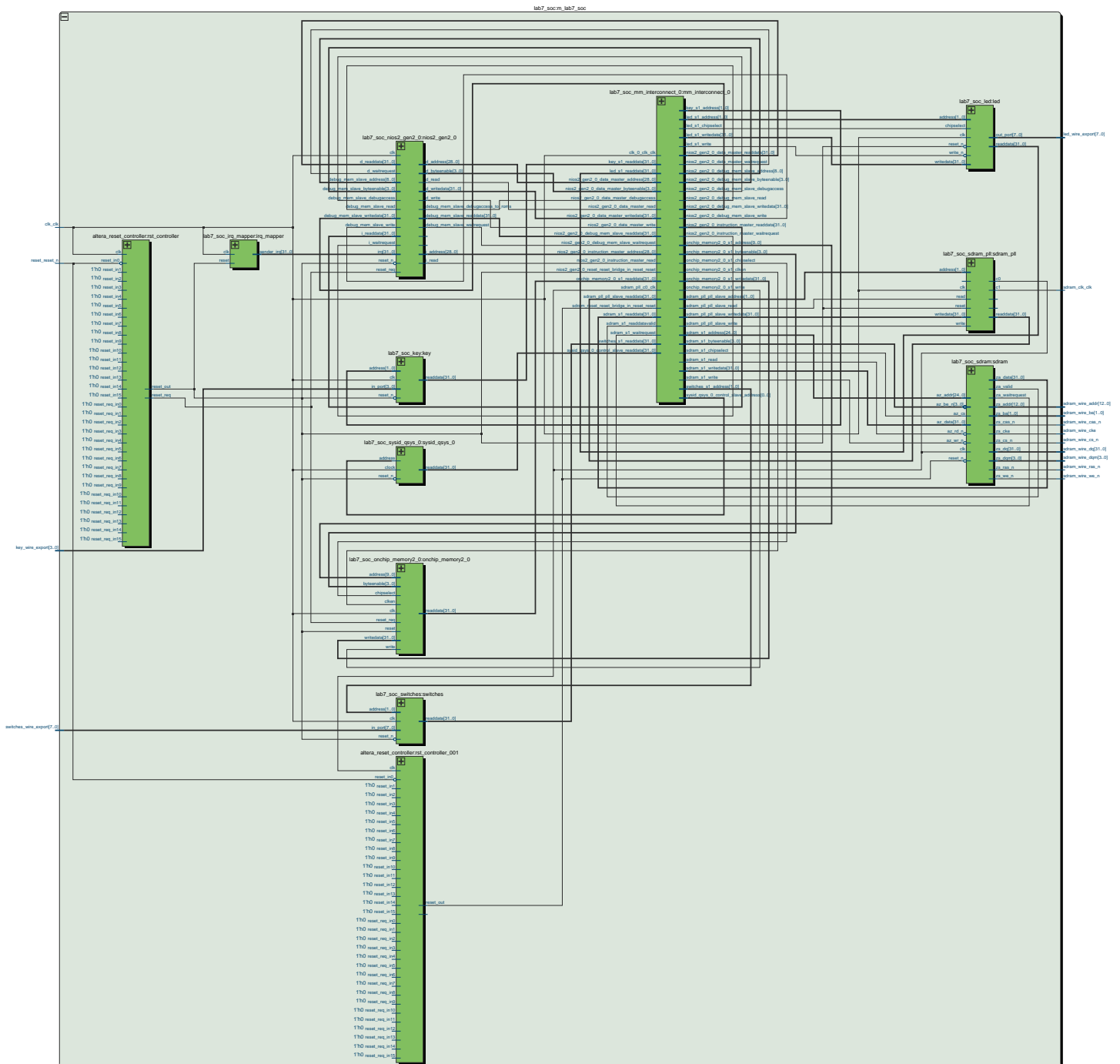


Fig. 4: Lab 7 Module Interconnections