ECE 385

EXPERIMENT #6

SOC with NIOS II in SystemVerilog

I. OBJECTIVE

In this experiment you will learn the basic capability of the NIOS II processor as the foundation of your System-On-Chip (SoC) projects. You will learn the fundamentals of memory-mapped I/Os and implement a simple SoC interfacing with peripherals such as the on-board switches and LEDs. The first week of the lab will have you configure a basic SoC design with a CPU, memory, and some basic peripherals (LEDs and switches). The second week will have you extend this system with USB and VGA peripherals.

II. INTRODUCTION

Week 1:

The goal of this lab is creating a NIOS II based system on the Altera MAX10 device. The NIOS II is an IP based 32-bit CPU which can programmed using a high-level language (in this class, we will be using C). A typical use case scenario is to have the NIOS II be the system controller and handle tasks which do not need to be high performance (for example, user interface, data input and output) while an accelerator peripheral in the FPGA logic (designed using SystemVerilog) handles the high-performance operations.

The Introduction to NIOSII and Qsys will give you a walkthrough of the Platform Designer tool, which is used to instantiate IP blocks (including the NIOS II). We will set up a minimal NIOS II device with an SDRAM (Synchronous Dynamic RAM) controller and a PIO (Parallel I/O) block to blink some LEDs using a C program running on the NIOS II to confirm it is working. You will then be asked to write a program which reads 8-bit numbers from the switches on the DE10-Lite board and sums into an accumulator, displaying the output using the LEDs via the NIOS II. This will involve instantiating another PIO block to read data from the switches and modifying the C program to input data, add, and display the data.

Please read the **INTRODUCTION TO NIOS II AND QSYS** (INQ. 1-22).

Your top-level circuit should have **at least** the following inputs and outputs:
General Interface:

Inputs

```
KEY[0]          : logic          // For Qsys-mapped hardware reset purpose
KEY[1]          : logic          // For accumulator accumulation ('Accumulate')
MAX10_CLK1_50 : logic            // 50 MHz clock input
LEDR            : logic [7:0]    // LED display of the accumulator
SW              : logic [7:0]    // Switches for the accumulation input
```

SDRAM Interface for Nios II Software:

Bidirectional ports (inout)

```
DRAM_DQ    : logic [15:0]   // SDRAM Data 16 Bits
```

Outputs

```
DRAM_ADDR       : logic [12:0]   // SDRAM Address 13 Bits
DRAM_BA         : logic [1:0]    // SDRAM Bank Address 2 Bits
DRAM_DQM        : logic [1:0]    // SDRAM Data Mask 4 Bits
DRAM_RAS_N      : logic;         // SDRAM Row Address Strobe
DRAM_CAS_N      : logic;         // SDRAM Column Address Strobe
DRAM_CKE        : logic;         // SDRAM Clock Enable
DRAM_WE_N       : logic;         // SDRAM Write Enable
DRAM_CS_N       : logic;         // SDRAM Chip Select
DRAM_CLK        : logic;         // SDRAM Clock
```

NOTE: For debugging, you may add LEDs, hex displays, switches, and/or buttons to the above lists. Note that you are provided a .QSF file for the pin assignments, so the pin assignment table is not included for this lab.

Week 2:

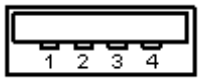
You will augment the Week 1 lab with the addition of a USB host controller (MAX3421E) which communicates to the DE10-Lite board via the SPI protocol. In addition, you will need to instantiate some modules to make use of the DE10-Lite's USB port to connect it to a VGA monitor.

How the USB Keyboard Works

The Universal Serial Bus (USB) standard defines the connection and communication protocols between computers and electronic devices. It also provides power supply to the connected devices. Due to the compatibility with a wide variety of devices, the USB standard has become prevalent since its introduction in the 1990s.

A USB cable has either a type A or a type B port on its ends. A USB port consists of four pins: VDD, D-, D+, and GND. Figure 1 shows the configuration. The VDD and GND pins are power lines, and D- and D+ are data lines. When data is being transmitted, D- and D+ take opposite voltage levels in a single time frame to represent one bit of data. On low and full speed

devices, a differential ‘1’ is transmitted by pulling D+ high and D- low, while a differential ‘0’ is a D- pulled high a D+ pulled low.



Type A

Type B

Pin	Name
1	VDD (5V)
2	D-
3	D+
4	GND

Figure 1

The DE10-Lite board does not come with a USB host port, but the ECE 385 shield which plugs into the board contains a Maxim MAX3421E USB host chipset. This chip communicates with the DE10-Lite board through the SPI protocol, which is a synchronous serial protocol and provides USB host functionality. A good portion of this assignment will be to implement the low-level SPI communication protocol between the NIOS II CPU and the MAX3421E.

A USB keyboard is a Human Interface Device (HID). HIDs usually do not require massive data transfers at any given moment, so a low-speed transmission would suffice. (Other USB devices such as a camera or a mass storage device would often need to send large files, which would require bulk transfers, a topic not covered in this lab.) Unlike earlier standards such as PS/2, a USB keyboard does not send key press signals on its own. All USB devices send information only when requested by the host. To receive key press signals promptly, the host needs to constantly poll information from the keyboard. In this lab, after proper configuration, the MAX3421E will constantly send *interrupt requests* to the keyboard, and the keyboard will respond with key press information in *report descriptors*. A descriptor simply means a data structure in which the information is stored.

Table 1 shows the keyboard input report format (8 bytes). In this format, a maximum of 6 simultaneous key presses can be handled, but here we will assume only one key is pressed at a time, which means we only need to look at the first key code. Each key code is an 8-bit hex number. For example, the character A is represented by 0x04, B by 0x05, and so on. When the key is not pressed, or is released, the key code will be 0x00 (No Event).

Byte	Description
0	Modifiers Keys
1	Reserved
2	Keycode 1
3	Keycode 2
4	Keycode 3

5	Keycode 4
6	Keycode 5
7	Keycode 6

Table 1

A more detailed explanation of how the keyboard works (and all the key code combinations) can be found in the INTRODUCTION TO USB AND EZ-OTG ON NIOS II (IUQ). The USB 2.0 Specification and the HID Device Class Definition (refer to the ECE 385 course website) are two documents that define all the behavior of a USB keyboard.

How the VGA Monitor works

For detailed explanation on how the VGA monitor works, please refer to the lectures. We will discuss in detail operation of the VGA protocol and how the ball and background fields are generated. Note that if you do not have access to a monitor which accepts a VGA input, you will need to either purchase such a monitor, or purchase an adapter from VGA -> HDMI which you can then plug into a television.

Some sample codes are given on the ECE 385 website which generate the horizontal sync, vertical sync, horizontal pixel, and vertical pixel location.

Instructions for the lab

The goal of this circuit is to make a small ball move on the VGA monitor screen. The ball can either move in the X (horizontal) direction or the Y (vertical) direction. (Remember that on the monitor, Y=0 is the top and Y=479 is the bottom!)

When the program starts, a stationary red ball should be displayed in the center of the screen. The ball should be waiting for a direction signal from the keyboard. As soon as a direction key (W-A-S-D) is pressed on the keyboard, the ball will start moving in the direction specified by the key.

W - Up
S - Down
A - Left
D - Right

When the ball reaches the edge of the screen, it should bounce back and start moving in the opposite direction.

The ball will keep moving and bouncing until another command is received from the keyboard. When a different direction key is pressed, the ball should start moving in the newly specified direction immediately, without returning to the center of the screen. NOTE: The ball should never move diagonally, and once set into motion by the initial key press, should never come to a stop.

Sample SystemVerilog code for the ball is given on ECE 385 web site. The sample code only implements the bouncing of the ball in the Y direction. You must add support for motion in the X direction and response to keyboard input.

Summarizing, complete working code for a ball, moving and bouncing in the Y direction, can be found on the ECE 385 website. You must add the following features:

- A keyboard entity that outputs the code of the last received key
- Motion and bouncing in the X and Y direction
- Immediately changing the ball's motion using the direction keys (W,A,S,D) (The ball should respond to the scan code)
- All these functions should work in any sequence without having to reset the circuit

For the detailed list of pin assignments, refer to the DE10-Lite.QSF file provided along with the Lab 6 materials.

III. PRE-LAB

Week 1:

A. Download the provided codes for Lab 6.1 on the ECE 385 course website. Follow the INQ tutorial to complete the NIOS II, memory, and the controller setup by performing the tasks as described in the tutorial. Your LED [0] should start blinking on your board as soon as the binary has been transmitted to the NIOS II CPU.

B. Modify the hardware and the software setup of the Lab 6 project to perform accumulation on the LED using the values from the switches as inputs. The LEDs should always display the value of the accumulator in binary and the accumulator should be 0 on startup (all LEDs off). The accumulator should overflow at 255+1 to 0. (255 + 1 → 0, 255 + 2 → 1, etc.) Pressing ‘Reset’ (KEY[0]) at any time clears the accumulator to 0 and updates the display accordingly (turns all the LEDs off). Pressing ‘Accumulate’ (KEY[1]) loads the number represented by the switches into the CPU, adding it to the accumulator. The 8 right-most switches (SW

[7:0]) are read as an 8-bit, unsigned, binary number with up being 1, down being 0. Push buttons should only react once to a single actuation.

C. Answer the *italicized* questions and fill in the table from the INQ tutorial. Be prepared to give answers to any of the questions from your TA when demoing as a quiz question. This is to ensure that you try to research what the settings do instead of simply trying to "make the picture look like your screen".

Hints: Unit test the input and output. The output should already work, but make sure you can turn on and off every segment. If you have problems, check the schematic for the DE10-lite, and make sure you are toggling the correct pins.

For this, and the rest of the class, you may use the C standard libraries (stdlib.h) or the C++ equivalents, this can save you a lot of work when coding in C.

You will need to bring the following to the lab:

1. Your code for the Lab.

2. Block diagram of your design with components, ports, and interconnections labeled.

Week 2:

The bulk of this lab can be broken down into four distinct tasks.

1. Modifying the Lab 6.1 Platform Designer to add the peripherals (PIO and SPI) required for communication with the MAX3421E
2. Putting together the top-level entity in SystemVerilog to include all the files given to you (including color_mapper, vga_controller and ball)
3. Modifying the 4 functions in MAX3421E.c to allow the USB driver running on the Nios II to communicate with the MAX3421E via the SPI peripheral.
4. Edit ball.sv to satisfy the requirements described in the instructions section above

You will be using Platform Designer setup from your previous Lab 6.1 for **step 1**. Your task is to add at the minimum a SPI peripheral, some PIO ports which go to the MAX3421 (e.g. for the reset and interrupt pins), some more PIO ports to interface your Nios II with the rest of your FPGA logic (e.g. to transmit the keycode and some debugging information), and a JTAG UART for printf (also essential for debugging). You may delete the PIO for the switches and the on-chip memory block from Lab 6.1, although you should keep the Nios II, the SDRAM controller, and the associated PLLs as well as the SystemID block.

For **step 2** you would need to download the files from the website and connect them together and to input/output pins as required. We will discuss in lecture how the VGA works, to give you a hint as to how to set up the VGA connections – there is also a tutorial on the Lab 6 wiki.

Step 3 will involve filling in 5 functions in the MAX3421E.c with appropriate calls to the SPI device driver (recommended). You can also directly program the SPI registers, but this is more complicated. Pseudocode for what needs to happen within each function is provided. This will involve reading and understanding the section of the Intel Embedded Peripherals Handbook that describes the SPI Controller, as well as reading the section on the SPI interface on the MAX3421E datasheet. This is likely the most difficult portion of this lab.

For **step 4** you will need to add if/else conditions to take care of all the other cases, i.e. right and left edge conditions, and key command conditions.

Demo Point Breakdown:
This is the breakdown for partial credit, if you can get the keyboard and VGA controller working together perfectly with all the conditions met, you will get full demo points without having to demo the individual steps.

- Display the last received key (scan code) from the keyboard (1 point)
- Somehow show that the ball can move in X as well as Y direction without reprogramming the FPGA (perhaps by using switch inputs as stimulus) (1 point)
- Somehow show that your keyboard code can identify the 4 different directions and light a different LED for each direction (1 point)
- Get the ball to respond to the keyboard (1 point)
- Somehow show that the ball can bounce when moving in the X as well as Y direction without reprogramming the FPGA. Ball does not move diagonally (1 point)

IV. LAB

Follow the Lab 6 demo information on the course website.

V. POST-LAB

1.) Refer to the Design Resources and Statistics in IQT.30-32 and complete the following design statistics table.

LUT	
DSP	
Memory (BRAM)	

Flip-Flop	
Frequency	
Static Power	
Dynamic Power	
Total Power	

Document any problems you encountered and your solutions to them, and a short conclusion.

VI. REPORT

Write a report, you may follow the provided outline below, or make sure your own report outline includes at least the items enumerated below.

1. Introduction
    a. Summarize the basic functionality of the NIOS-II processor running on the MAX10 FPGA.
    b. Briefly summarize the operation of the USB/VGA interface
2. Written Description and Diagrams of NIOS-II System
        i. Describe in words the hardware component of the lab, in this lab, only the Platform Designer module is here.
        ii. Describe in Lab 6.1 how the I/O works.
        iii. Describe in words how the NIOS interacts with both the MAX3421E USB chip and the VGA components
        iv. Written description of the SPI protocol.
        v. Describe the purpose of the each function you filled in the C code (you do not need to describe the functions you did not modify.
        vi. Describe in detail the VGA operation, and how your Ball, Color Mapper, and the VGA controller modules interact.
3. Top Level Block Diagram
    a. This diagram should represent the placement of all your modules in the top level. Please only include the top-level diagram and not the RTL view of every module.
4. Written Description of all .sv Modules

   a. A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Platform Designer generated file for your Nios II system! When describing the generated file, you should describe the PIO blocks added beyond those just needed to make the NIOS system run (i.e. the ones needed to communicate with the USB chip and other components). The Platform Designer view of the Nios II system is helpful here.
5. System Level Block Diagram (this is new for labs 7-9)
   a. The Platform Designer view of the SoC module should be found here, describe the functionality of each block (including those which are part of the SoC, such as the memories).
   b. Note that this is not trivial, as there are many components within the Platform Designer view.
   c. You can separately describe the 'core' components common to Lab 6.1 and 6.2 first, and then describe the specific modules for week 1 and week 2.
6. Describe in words the software component of the lab.
   a. One of the INQ questions asks about the blinker code, but you must also describe your accumulator.
   b. Describe the code you needed to fill in for the USB/SPI portion of the lab for Lab 6.2
7. Answers to all INQ Questions
      i. The Prelab question (part A) is one of these questions
   b. Post-lab Question
8. Document the Design Resources and Statistics in table provided in the lab.
9. Conclusion
   a. Discuss functionality of your design. If parts of your design did not work, discuss what could be done to fix it?
   b. Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester?