ECE 385

EXPERIMENT #6

SOC with Microblaze in SystemVerilog

I. OBJECTIVE

In this experiment you will learn the basic capability of the Microblaze 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 to create a Microblaze based system on the Xilinx Spartan 7 device. The Microblaze is an IP based 32-bit CPU which can be programmed using a high-level language (in this class, we will be using C). A typical use case scenario is to have the Microblaze 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 Microblaze and Vitis will give you a walkthrough of the Block Design tool in Vivado, which is used to instantiate IP blocks (including the Microblaze). We will set up a minimal Microblaze device with an on-chip memory block and a PIO (Parallel I/O) block to blink some LEDs using a C program running on the Microblaze to confirm it is working. You will then be asked to write a program which reads 8-bit numbers from the switches on the Urbana board and sums into an accumulator, displaying the output using the LEDs via the Microblaze. 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 MICROBLAZE AND VITIS** (INMB).

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

Inputs

```
btn[3:0]        : logic          // Pushbuttons for Accumulate PIO
clk             : logic          // 100 MHz clock input
led             : logic [15:0]   // LED display of the accumulator
sw              : logic [15:0]   // Switches for the accumulation input

uart_rxd        : logic          // UART going from FPGA -> USB
uart_txd        : logic          // UART going from USB->FPGA
```

Note: because we have moved to the manufacturer provided pin-assignments starting with this lab, the capitalization and spelling of the pins may be different than what you are used to. This should not be a problem for the provided code, but double check the names once you extend the code to make sure your top-level port names match with the .xdc assignments.

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

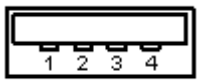
Week 2:

You will augment the Week 1 lab with the addition of a USB host controller (MAX3421E) which communicates to the Microblaze via the SPI protocol. In addition, you will need to instantiate some modules to make use of the Urbana board's USB port, enumerate a USB keyboard, and use it to control a ball which is drawn using VGA. A provided IP block converts the VGA signal to an HDMI signal, which is then used to display the ball on an external HDMI 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 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 Urbana board contains a Maxim MAX3421E USB host chipset. This chip communicates with the FPGA 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 Microblaze 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 ON THE MICROBLAZE (IUSB).

How the VGA/HDMI Monitor works

For detailed explanation on how the VGA monitor works, please refer to the lectures. We will discuss in detail the operation of the VGA protocol and how the ball and background fields are generated. Note that included as part of the provided code for this lab is an IP block which converts the VGA signal to an HDMI signal. The IUSB document also describes how to customize this IP.

The provided codes given on the ECE 385 website generate the horizontal sync, vertical sync, horizontal pixel, and vertical pixel location and will draw the ball in a stationary way.

Instructions for the lab

The goal of this circuit is to make a small ball move on the HDMI 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 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:

- The USB keyboard should output the key pressed via the serial terminal
- 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 provided .xdc file(s) 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 Microblaze, 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 Microblaze 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). Pressing ‘Reset’ (btn[0]) at any time clears the accumulator to 0 and updates the display accordingly (turns all the LEDs off). Pressing ‘Accumulate’ (btn[1]) loads the number represented by the switches into the CPU, adding it to the accumulator. The 16 switches (SW [15:0]) are read as an 16-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 INMB tutorial. Be prepared to give answers to any of the questions from your TA when demonstrating 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 Urbana Board, 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.

Week 2:

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

1. Modifying the Lab 6.1 block design 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 Microblaze 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 block design 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 Microblaze with the rest of your FPGA logic (e.g., to transmit the keycode and some debugging information, and a timer module for handling USB timeouts. You may delete the PIO for the switches and the on-chip memory block from Lab 6.1, although you should keep the Microblaze, the memories, and other essentials such as the UART.

For **step 2** you would need to download the files from the website and connect them together and input/output pins as required. We will discuss in the lecture how the VGA works, to give you a hint as to how to set up the VGA connections. You will also need to properly customize the VGA->HDMI IP and verify all the connections in order to display the video signal on the HDMI port.

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 documentation of the Vitis SPI driver and looking at examples that describe 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.

IV. LAB

Follow the Lab 6 demo information on the course website.

V. POST-LAB

1.) Refer to the Design Resources and Statistics in IVT and complete the following design statistics table.

LUT	
DSP	
Memory (BRAM)	
Flip-Flop	
Latches*	
Frequency	
Static Power	
Dynamic Power	
Total Power	

*The number of latches should be 0 in a fully synchronous FPGA design. Note that Vivado will create latches if your `always_comb` procedures do not synthesize into purely combinational logic. This indicates a **bug in your design** and should be fixed. Your TA will verify that your design has no latches during the demo.

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 Microblaze processor running on the Spartan 7 FPGA.
    b. Briefly summarize the operation of the overall Week 2 design
2. Written Description and Diagrams of Microblaze System
    a. For the following written description, assume the full second week design unless otherwise noted.
    b. Module descriptions:
        i. Describe in words the hardware (e.g., every .SV module) including the provided .SV modules. A guide on how to do this was shown in the Lab 2.2 report outline.
        ii. Additionally for this lab, you created many modules which are encapsulated in a block design. Describe every component in the block design, some of them like the AXI interconnect, you will need to do some research.
        iii. You can separately describe the ‘core’ components common to week 1 and 2 first, and then describe the additional components for week 2 separately.
    c. Describe in Lab 6.1 how the I/O works.
    d. Describe in words how the MicroBlaze interacts with both the MAX3421E USB chip and the ball motion components.
    e. Describe in detail the VGA operation, and how your Ball, Color Mapper, and the VGA controller modules interact.
    f. Describe the VGA-HDMI IP, how does HDMI differ from VGA, how are they similar?
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. 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. Written description of the SPI protocol and how it operates in the context of the MAX3421E.
    c. Describe the purpose of each function you filled in in the C code (you do not need to describe the functions you did not modify).

5. Answers to all INQ & Post lab questions
6. Document the Design Resources and Statistics in table provided in the lab.
7. 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?