ECE 385

Spring 2024

Lab 6

Ziheng Li (zihengl5)

Xin Yang (xiny9)

Section AL1

Prof. Zuofu Chen

Introduction

Summarize Microblaze

The MicroBlaze processor acts as a 32-bit CPU that can be programmed with high-level languages like C. Its primary function in this lab is to serve as the system controller, handling tasks that do not require high performance, such as user interface management, data input, and output. While other parts that require high-performance are designed and implemented in FPGA logic (like screen drawing, for example).

Operation of the overall Week 2 design

In Week 2, the design extends the SOC system developed in Week 1 by adding USB and VGA peripherals. The main task for week 2's design is to integrate a USB controller (MAX3421E) communicated via SPI protocol, and setting up a VGA drawing techniques along with a VGA->HDMI converter for display.

USB: The lab involves interfacing the MicroBlaze processor with a USB on FPGA to manage devices such as a keyboard. We implement SPI communication to exchange data and control signals.

VGA/HDMI: The lab also includes generating VGA signals to control the movement of a ball bouncing on the screen, where movements are controlled by a USB keyboard, hich is then converted into HDMI signals.

Module descriptions

Module: mb_usb_hdmi_top.sv
Inputs: Clk, reset_rtl_0, [0:0] gpio_usb_int_tri_i, usb_spi_miso, uart_rtl_0_rxd
Outputs: gpio_usb_rst_tri_o, usb_spi_mosi, usb_spi_sclk, usb_spi_ss, hdmi_tmds_clk_n, hdmi_tmds_clk_p, [2:0]hdmi_tmds_data_n, [2:0]hdmi_tmds_data_p, [7:0] hex_segA, [3:0] hex_gridA, [7:0] hex_segB, [3:0] hex_gridB
Description: This module serves as the top-level module for the USB and HDMI interface. It implements USB, UART, HDMI, and HEX display functionalities. It also enables the program to draw the bouncing ball to screen.

Purpose: Its purpose is to provide internal signals and combine multiple hardware components into a single entity.

Module: hex_driver.sv
Inputs: clk, reset, in [4]
Outputs: [7:0] hex_seg, [3:0] hex_grid
Description: The module uses a loop to instantiate four nibble_to_hex converters, mapping each of the 4-bit inputs to their corresponding 7-segment display patterns. Then choose the hex_grid to determine which of the four LEDs is active.
Purpose: Convert binary into hexadecimal and then display them on the LEDs.

Module: ball.sv
Inputs: Reset, frame_clk, [7:0] keycode
Outputs: [9:0] BallX, [9:0] BallY, [9:0] BallS
Description: This module implements the logic for controlling the motion and position of a bouncing ball on a display screen. It uses key code input from a keyboard to update its movement by changing its direction and X Y coordinates for the next frame. The ball's position is updated at every frame clock cycle, and the ball reaches the 4 boundaries of the screen.
Purpose: The purpose of this module is to provide the logic for drawing a bouncing ball on a display screen. It also allows the user to control its motion using keyboard inputs.

Module: mb_block.bd
Inputs: clk_100MHz, gpio_usb_int_tri_i[0:0], reset_rtl_0, uart_rtl_0_rxd, usb_spi_miso
Outputs: [31:0] gpio_usb_keycode_0_tri_o, [31:0] gpio_usb_keycode_1_tri_o, [0:0]gpio_usb_rst_tri_o, uart_rtl_0_txd, usb_spi_mosi, usb_spi_sclk, [0:0]usb_spi_ss
Description: This module serves as the top-level module for the block design of mb_block with the related peripherals. It instantiates and connects the MicroBlaze, an AXI UART, multiples GPIOs for USB key codes, and an SPI interface for USB communication. Additionally, it also handles the clock and reset signals for the system.
Purpose: The purpose of this module is to modify the MicroBlaze system to support the USB interface and the MAX3421E chip.

Module: VGA_controller.sv
Inputs: pixel_clk, reset
Outputs: hs, vs, active_nblank, frame, sync, [9:0] drawX, [9:0] drawY
Description: This module implements a VGA controller that generates the signals with proper frequency for displaying graphics on a VGA monitor. It generates the horizontal sync (hs) and vertical sync (vs) signals. It outputs the coordinates of the current pixel being drawn (drawX and drawY). Additionally, we add a frame signal that pulses once per frame in 60Hz, which can be used for the HDMI module to refresh the frames.
Purpose: The purpose of this module is to generate the timing signals and coordinate information required for displaying on a VGA monitor.

Module: clk_wiz_0.v
Inputs: reset, clk_in1
Outputs: clk_out1, clk_out2, locked
Description: This module is used to create a pixel clock (25MHz) and a 5x TMDS clock (125MHz).
Purpose: The purpose of this module is to provide two different clock signals

How the I/O works

We used the GPIO module as a bridge from MicroBlaze to FPGA logic. We could modify/read the LEDs and switches directly through a memory location, as specified below, rather than communicating through a driver. In our case for example, address 0x40000000 and 0x40010000 correspond to LEDs and switches accordingly. If we want to read/write two devices, we only need to dereference the address to read/write it.

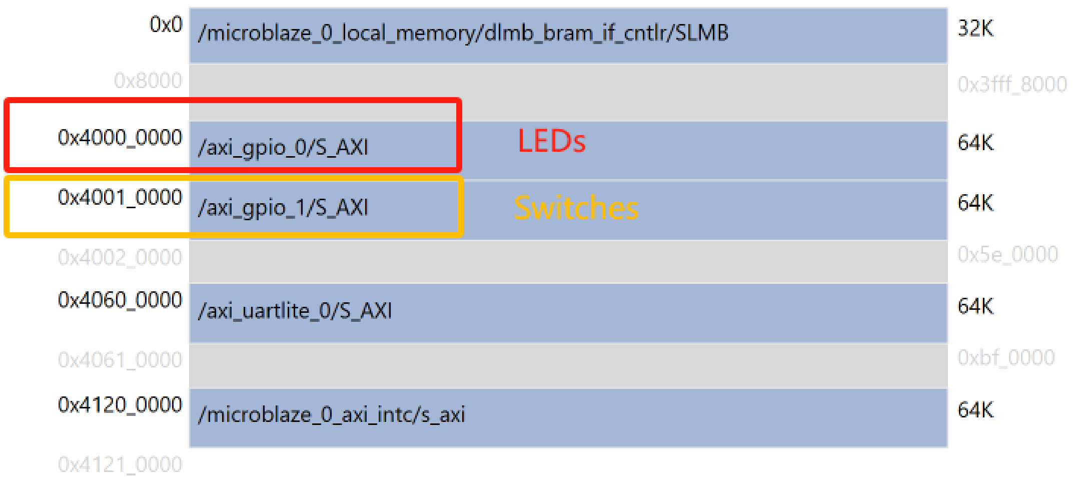


Fig.1 Address map

MicroBlaze & MAX3421E & Ball

MicroBlaze communicates with the MAX3421E USB through SPI transactions that includes the four functions we implement: MAXreg_wr, MAXbytes_wr, MAXreg_rd, MAXbytes_rd. They were used to read and write to USB devices through the MAX3421 chip according to the commands from MicroBlaze. Then the MicroBlaze is able to read the keycode from the USB port and send it to the ball module to control the movement of the ball, finally draw to the screen through VGA drawing module.

VGA operation

The principle of VGA can be considered as a drawing strategy that draws the image pixel by pixel(25Mhz). It updates the coordinates of Y when it finishes printing the complete row. It also outputs drawX/Y, the current pixel being drawn to the screen. The ball module basically updates the position information of the ball's center for the next frame. As for the color mapper will take this center information from the ball module and drawX/Y from the VGA controller to decide the color on each pixel of the next frame, outputting RGB values correspond to that pixel we are drawing to VGA drawing module.

VGA-HDMI

The VGA control outputs horizontal/vertical sync pulse to the HDMI IP. In addition to that, VGA-HDMI IP also takes the digital RGB signal from color_mapper. HDMI differs from VGA for it is digital rather than analog. VGA transmits separate color and sync signals in an analog format, while HDMI transmits digital data packets containing both audio and video information. Meanwhile, HDMI provides better image quality and higher refresh rate than VGA. For similarities, both of them are capable of delivering video signals to display devices like monitors.

Top Level Block Diagram

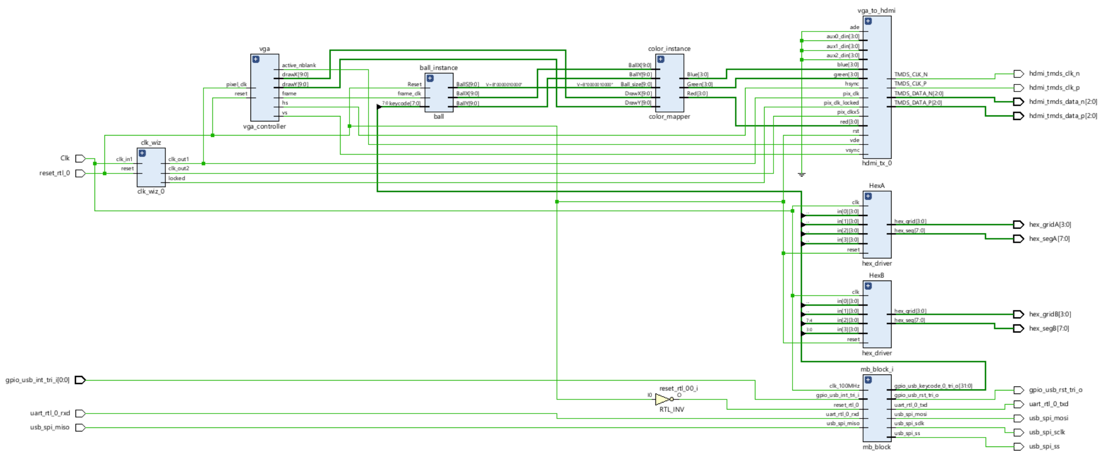


Fig.2 RTL diagram for Lab 6.2

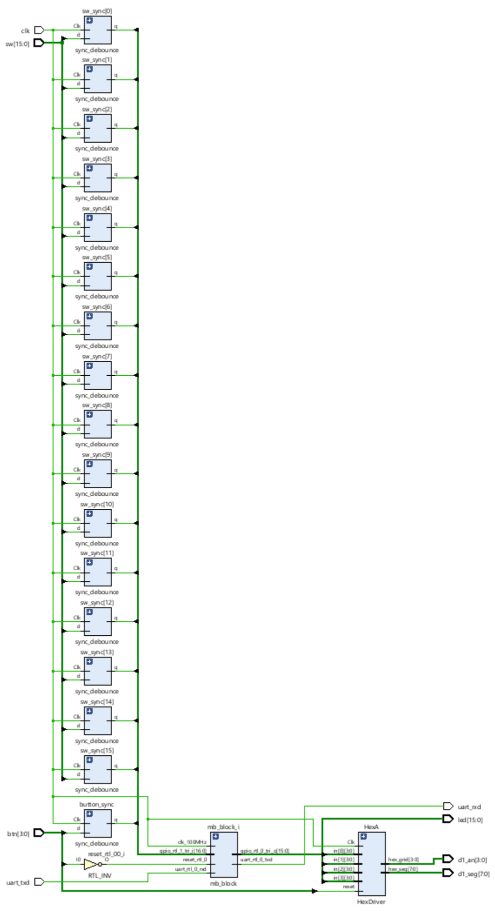


Fig.3 RTL diagram for Lab 6.1

Software (Accumulator)

First, we bound the address 0x40000000 (from address map of block design) to led_gpio_data. Second, we bound the address 0x40010000 to {btn[1], SW} to read the button and switch data. The technique applied here is polling. The main function used an infinite loop to keep trying to get the button signal. When we press the add button, thousands of rising edges are detected and might cause duplicated adding. Thus we introduce a flag to make sure the add signal is up just once. The flag is set to 0 to prevent repeating the addition until the condition changes. The flag is reset to 1 when the specific bit(add) in led_gpio_data2 is no longer being press, and indicating the next addition is valid. If the accumulator is greater than 65535, the value gets reset by minus 65535 to handle overflow and prints an error statement in the Vitis serial terminal.

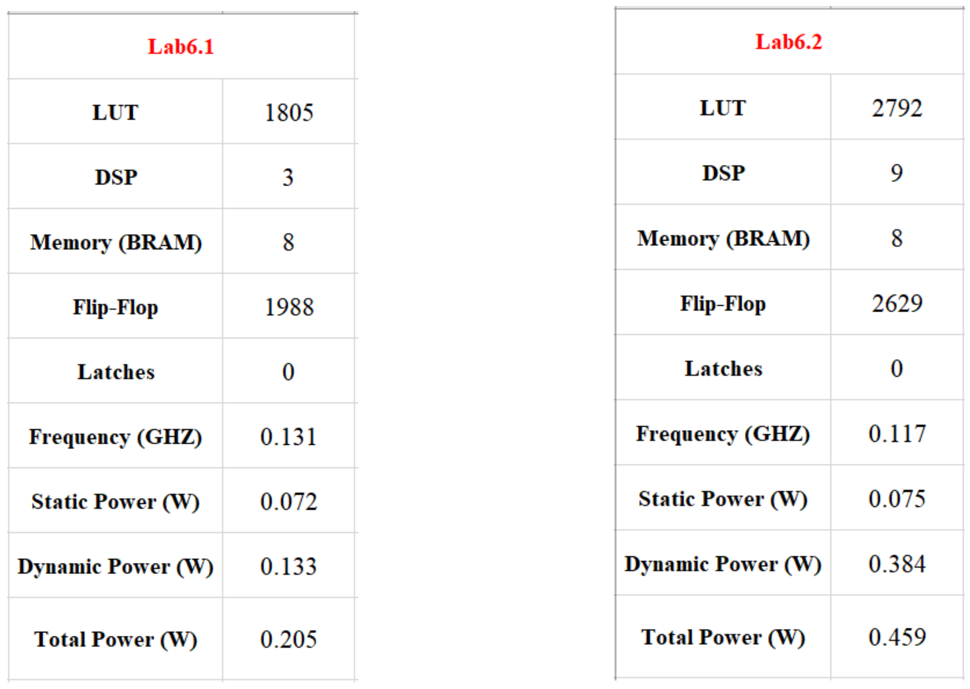
SPI Protocol

SPI is used as a serial communication protocol to transfer data between MicroBlaze and MAX3421E. Here the MicroBlaze is the master and the MAX3421E is the slave. The master sends out MOSI to the slave while the slave sends out MISO to the master. They get synchronized using SCLK from the master. The master sends commands and data to the MAX3421E, and receives responses and data from the MAX3421E, then the process of USB communication data exchange is complete.

Functions in C code

The purpose of MAXreg_wr is to enable writing a single host register. The purpose of MAXbytes_wr is to enable writing multiple bytes and return a pointer to a memory position after last written. The purpose of MAXreg_rd is to enable reading of a single host register. The purpose of MAXbytes_rd is to enable reading multiple-bytes register and return a pointer to a memory position after last read. Note the for all of these functions, we need to send out the register number that we want to read from/write on prior to any data transaction.

INMB & Design Resources



Lab6.1	
LUT	1805
DSP	3
Memory (BRAM)	8
Flip-Flop	1988
Latches	0
Frequency (GHZ)	0.131
Static Power (W)	0.072
Dynamic Power (W)	0.133
Total Power (W)	0.205

Lab6.2	
LUT	2792
DSP	9
Memory (BRAM)	8
Flip-Flop	2629
Latches	0
Frequency (GHZ)	0.117
Static Power (W)	0.075
Dynamic Power (W)	0.384
Total Power (W)	0.459

Fig.4 Design resources

Select the "Microcontroller" Preset and then modify the "Local Memory" to 32KB. You should do some research and figure out what are some primary differences between the various presets which are available.

*Following answers were found in AMD MicroBlaze Processor Embedded Design User Guide, see reference for details.

Microcontroller preset: Microcontroller preset suitable for microcontroller designs. Area optimized, with no caches and debug enabled.

Real-time preset: Real-time preset geared towards real-time control. Performance optimized, small caches and debug enabled, most execution units.

Application preset: Application preset design for high performance applications. Performance optimized, large caches and debug enabled, and all execution units including floating-point.

Note the bus connections coming from the MicroBlaze; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?

MicroBlaze is a modified Harvard machine, because we could access the memory that contains instructions as access data. For example, in Lab 6.1, we could access the led and switches directly through a memory location rather than driver.

What does the "asynchronous" in UART refer to regarding the data transmission method? What are some advantages and disadvantages of an asynchronous protocol vs. a synchronous protocol?

Asynchronous data transmission means the data is sent without synchronization to a common clock. Moreover, data is framed between start and stop bits to indicate the beginning and end of a transaction.
Advantages of asynchronous: Compared to synchronous, asynchronous protocol is easier to implement since it requires less complex hardware. In addition, it is more flexible since data could be transmitted through various rates as long as there's start&stop bits.
Disadvantages of asynchronous: Compared to synchronous, the usage of start&stop bits reduce the effective data transmission rate. Moreover, it is more likely to have errors due to lack of a common clock.

You should have learned about interrupts in ECE 220, and it is obvious why interrupts are useful for inputs. However, even devices which transmit data benefit from interrupts; explain why. Hint: the UART takes a long time (relative to the CPU) to transmit a single byte.

The advantages of using interrupts over polling is that interrupt requires less CPU source. In an interrupt-driven approach, the device tells the CPU when it needs attention, saying, "I have something for you to check." This allows the CPU to operate more efficiently, as it only addresses device requests as they occur. On the other hand, polling requires CPU to check each device, asking, "Do you have something for me?" This method is less efficient as it consumes more CPU resources, keeps querying devices regardless.

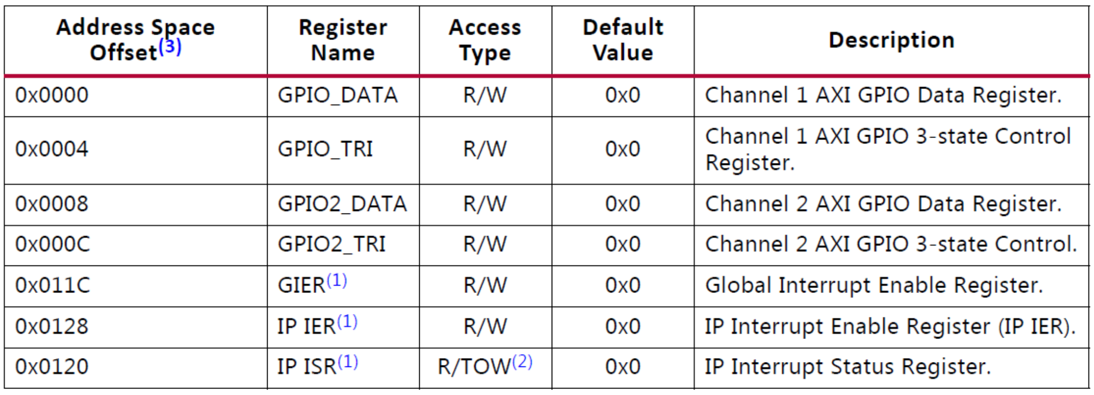
Look at the various segments (text, data, bss), what does each segment mean? What kind of code elements are stored in each segment? Also note the size of the executable in bytes. Remember that we configured 32Kbytes of on-chip memory to use as our program memory. Why does the provided code, which does very little, take up so much program memory? Hint: try commenting out some lines of code and see how the size changes.

*Following answers were adapted from https://mirzafahad.github.io/2021-05-08-text-data-bss/, see reference for details.

Text segment, also known as code segment, stores the executable instruction (our code).
Data segment, also known as initialized data segment, stores global variables and static variables that are initialized.
BSS segment, also known as uninitialized data segment, stores global and static variables that are not being initialized.

Inside the main file, we include standard input/output library (#include <stdio.h>) to support printing statements, which consumes much program memory.

Make sure you understand the register map on page 10. If the base address is 0x40000000, how would you access GPIO2_DATA (for example?).



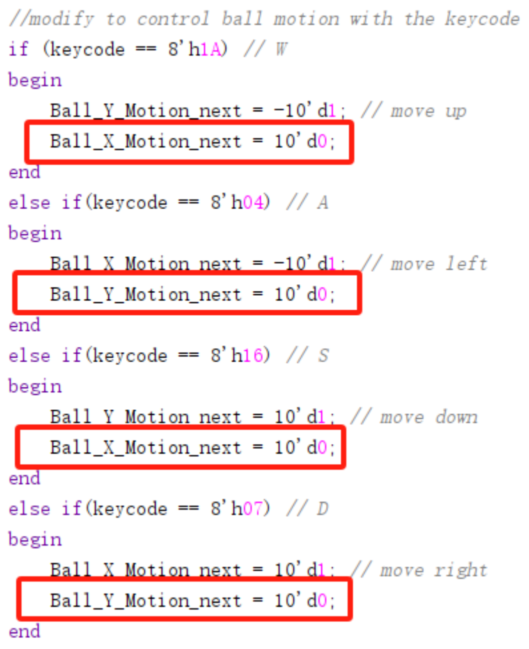
Address Space Offset[3]	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER[1]	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER[1]	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR[1]	R/TOW[2]	0x0	IP Interrupt Status Register.

Fig.5 Address space offsets

From the above graph, we could see that GPIO2_DATA is located +0x0008 from the base address, hence reading/writing 0x40000008 can access GPIO2_DATA.

Functionality

In our design, we did not read the requirement saying that the ball should not be moving diagonally. In order to achieve that, we need explicitly specify the moving direction of both X and Y, and setting one of the unrelated direction to zero, as shown below. For example, if we press W, the ball should move upward with a negative Y motion and should not move in X direction. Other than that, we successfully achieve all the functionalities of the lab 6.1 & 6.2.



```
//modify to control ball motion with the keycode
if (keycode == 8'h1A) // W
begin
    Ball_Y_Motion_next = -10'd1; // move up
    Ball_X_Motion_next = 10'd0;
end
else if(keycode == 8'h04) // A
begin
    Ball_X_Motion_next = -10'd1; // move left
    Ball_Y_Motion_next = 10'd0;
end
else if(keycode == 8'h16) // S
begin
    Ball_Y_Motion_next = 10'd1; // move down
    Ball_X_Motion_next = 10'd0;
end
else if(keycode == 8'h07) // D
begin
    Ball_X_Motion_next = 10'd1; // move right
    Ball_Y_Motion_next = 10'd0;
end
```

Fig.6 Code snipe from Ball.sv

Conclusion

In this experiment, we successfully implemented and enhanced a System-on-Chip design using the Microblaze processor on the Xilinx Spartan-7 FPGA. The first part involved configuring SOC with CPU memory and basic peripherals, focusing on memory-mapped I/O to control LEDs and read switches. The second part of the lab expand the setup by adding several GPIO blocks to enable the control from USB keyboard. In addition, the lab also teach us the fundamental logic of VGA drawing. We found the communication between Microblaze and MAX3421E is somehow confusing for beginners, especially the 4 modes for Xspi_Transfer() function. It would be nice if we could have more documentation/example for that.

Reference

MicroBlaze Processor Embedded Design User Guide (UG1579). AMD Technical Information Portal. (n.d.). https://docs.amd.com/r/en-US/ug1579-microblaze-embedded-design/MicroBlaze-Configuration-Wizard-Welcome-Page

Mirza, F. (n.d.). *Text, data, BSS, and Dec*. Fahad Mirza - Fahad Mirza - Embedded Engineer. https://mirzafahad.github.io/2021-05-08-text-data-bss/