ECE 385

Fall 2023

Experiment #6

SOC with Microblaze in SystemVerilog

Ziyi Lin

Zhuoyang Lai
TA: Jerry Wang

1.Introduction

(a) MicroBlaze is a soft-IP based 32-bit CPU which can be programmed using a high-level language, in this lab C programming language. It is an example of a system on chip(SoC), and it can take over some low performance tasks, rather than directly programming the hardware. It is an example of modified-Harvard Architecture and support interrupt system.

(b) Firstly, we modify the C code design to enable the GPIO module to read and write data from and to the MAX3421E host, a USB host which is connected to either keyboard or mouse, enabling the whole system to send control signals to control the movement of the ball displayed. Secondly, vga_controller and vga_hdmi module enable the output of the hdmi signal to be displayed. color_mapper controls the color of the hdmi signal that is to be played. Thirdly, we modify the ball.sv to realize the logic that controls the ball to move.

2. Written Description and Diagrams of Microblaze System

a/b. Module descriptions:

i.Describe in words the hardware (e.g., every .SV module) including the provided .SV modules.

Module: VGA_Controller.sv
Inputs: pixel_clk, reset
Outputs: hs, vs, active_nblank, sync, [9:0] drawX, [9:0] drawY
Description: The vga_controller manages VGA display synchronization for a 640x480 resolution. It counts pixel positions and creates sync signals based on VGA timing. The module uses a 25MHz clock and use counter to generate horizontal sync pulse and vertical sync pulse.
Purpose: To produce control and sync signals for interfacing with a VGA display.

Module: ball.sv
Inputs: logic Reset, frame_clk, logic [7:0] keycode
Outputs: logic [9:0] BallX, BallY, BallS
Description: Controls the movement of a ball on a screen with a constant speed. The ball bounces when it hits the edges and can be directed using the keycode.
Purpose: To provide a dynamic visual representation of a ball that moves and bounces around a screen, as well as respond to user inputs for direction control.

Module: color_mapper.sv
Inputs: logic [9:0] BallX, BallY, DrawX, DrawY, Ball_size
Outputs: logic [3:0] Red, Green, Blue

Description: Maps a color to a pixel based on its position relative to a ball. If the pixel is within the ball (using a circle formula), the color is set to a specific value. Otherwise, it varies based on DrawX.
Purpose: Provide color information for a display given a ball's position and the current pixel being drawn.

Module: hex_driver.sv
Input: Clk, reset, [3:0]in[4]
Output: [7:0]hex_seg,[3:0]hex_grid
Description: hex display driver
Purpose: to display the output in hex form.

Module: mb_usb_hdmi_top.sv
Inputs: Clk, reset_rtl_0, gpio_usb_int_tri_i, usb_spi_miso, uart_rtl_0_rxd, uart_rtl_0_txd.
Outputs: gpio_usb_rst_tri_o, usb_spi_mosi, usb_spi_sclk, usb_spi_ss, hdmi_tmds_clk_n, hdmi_tmds_clk_p, hdmi_tmds_data_n, hdmi_tmds_data_p, hex_segA, hex_gridA, hex_segB, hex_gridB.
Description: the top-level module for a MicroBlaze-based test project, designed for interfacing with USB, HDMI, UART
Purpose: The purpose of this module is to provide a top-level interface for connecting and controlling various peripherals and modules within the project.

Module: mb_intro_top.sv
Inputs: clk, [15:0]sw, [3:0] btn, uart_txd
Outputs: [15:0] led, uart
Description: The module is designed to instantiate a MicroBlaze-based system with minimal additional logic, primarily interfacing with switches, buttons, LEDs, and UART communication.
Purpose: provide a basic starting point for the MicroBlaze tutorial

Block_design

Following IP Specific for 6.2

Module: gpio_usb_rst
Inputs: s_axi_aclk(clock), s_axi_aresetn(reset), gpio_io_i, gpio2_io_i, gpio2_io_t
Outputs: gpio_io_o, gpio_io_t, gpio2_io_o, gpio2_io_t //from datasheet
Description: the gpio receive reset signal from FPGA and transfer it to slave
Purpose: The AXI GPIO design provides a general purpose input/output interface to an AXI4-Lite interface.

Module: gpio_usb_int
Inputs: s_axi_aclk(clock), s_axi_aresetn(reset), gpio_io_i, gpio2_io_i, gpio2_io_t
Outputs: gpio_io_o, gpio_io_t, gpio2_io_o, gpio2_io_t //from datasheet
Description: the gpio integrate and interrupt signal.
Purpose: The AXI GPIO design provides a general purpose input/output interface to an AXI4-Lite interface.

Module: gpio_usb_keycode
Inputs: s_axi_aclk(clock), s_axi_aresetn(reset), gpio_io_i, gpio2_io_i, gpio2_io_t
Outputs: gpio_io_o, gpio_io_t, gpio2_io_o, gpio2_io_t //from datasheet
Description: the gpio receives keycode signal from slave and transfers it to master and interprets them as direction.
Purpose: The AXI GPIO design provides a general purpose input/output interface to an AXI4-Lite interface.

Module: spi_usb
Inputs: S_AXI_ACLK, S_AXI_ARESETN, Freeze, S_AXI_AWADDR[C_S_AXI_ ADDR_WIDTH-1:0], S_AXI_AWVALID, S_AXI_WDATA[C_S_AXI_ DATA_WIDTH - 1: 0], S_AXI_WSTB[C_S_AXI_ DATA_WIDTH/8-1:0] , S_AXI_WVALID, S_AXI_BREADY, S_AXI_ARADDR[C_S_AXI_ ADDR_WIDTH - 1:0], S_AXI_ARVALID, S_AXI_RREADY, io1_i
Outputs: S_AXI_AWREADY, S_AXI_WREADY, S_AXI_BRESP[1:0], S_AXI_BVALID, S_AXI_ARREADY, S_AXI_RDATA[C_S_AXI_ DATA_WIDTH -1:0], S_AXI_RRESP[1:0], S_AXI_RVALID,io0_o
Description: SPI is a communication IP to interface to external USB chip and HDMI peripheral.
Purpose: to interface to external USB chip and HDMI peripheral

Module: timer_usb_axi
Inputs: axi_aclk, axi_aresetn, axi_timer_in
Outputs: axi_timer_irq, axi_timer_out.
Description: timer module that interfaces to the AXI4-Lite interface.
Purpose: USB has many timeouts that require timekeeping in milliseconds

Module: xlconcat_0
Inputs: In0,In1,In2,In3
Outputs: dout[3:0]
Description: connect the interrupt signal.
Purpose: connect the interrupt signal and pack them together.

Following IP for both 6.1 and 6.2

Module: clk_wiz_1
Input: clk_100MHZ, reset
Output: clk_out1, locked
Description: The Clocking Wizard enables designers to generate clock signals with specific frequencies, can be reset via reset signal.
Purpose: setting the desired clock frequency without the need for manually coding

Module: mdm (Micro blaze debug module)
Input: debug signal
Output: debug data
Description: providing debugging and monitoring capabilities for MicroBlaze-based FPGA designs.
Purpose: facilitate the debugging and monitoring of MicroBlaze soft processors embedded in FPGA designs.

Module: MicroBlaze
Input: M_AXI_DP (Peripheral Data Interface, AXI4-Lite or AXI4 interface)
DLMB(Data interface, Local Memory Bus (BRAM only))
M_AXI_IP (Peripheral Instruction interface, AXI4-Lite interface)
ILMB (Instruction interface, Local Memory Bus (BRAM only))
S0_AXIS..S15_AXIS (AXI4-Stream interface slave direct connection interfaces)
M_AXI_DC (Data-side cache AXI4 interface)
M_ACE_DC (Data-side cache AXI Coherency Extension (ACE) interface)

M_AXI_IC (Instruction-side cache AXI4 interface)
M_ACE_IC (Instruction-side cache AXI Coherency Extension (ACE) interface)
Output: M_AXI_DP (Peripheral Data Interface, AXI4-Lite or AXI4 interface)
DLMB (Data interface, Local Memory Bus (BRAM only))
M0_AXIS..M15_AXIS(AXI4-Stream interface master direct connection interface)
M_AXI_DC (Data-side cache AXI4 interface)
M_ACE_DC (Data-side cache AXI Coherency Extension (ACE) interface)
Description: The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data and instruction accesses.
Purpose: compute, modify and read memory and finish tasks according to the instruction.

Module: axi_uart
Inputs: S_AXI_ACLK, S_AXI_ARESETN, Freeze, S_AXI_AWADDR[C_S_AXI_ ADDR_WIDTH-1:0], S_AXI_AWVALID, S_AXI_WDATA[C_S_AXI_ DATA_WIDTH - 1: 0], S_AXI_WSTB[C_S_AXI_ DATA_WIDTH/8-1:0] , S_AXI_WVALID, S_AXI_BREADY, S_AXI_ARADDR[C_S_AXI_ ADDR_WIDTH -1:0], S_AXI_ARVALID, S_AXI_RREADY
Outputs: IP2INTC_Irpt, S_AXI_AWREADY, S_AXI_WREADY, S_AXI_BRESP[1:0], S_AXI_BVALID, S_AXI_ARREADY, S_AXI_RDATA[C_S_AXI_
DATA_WIDTH -1:0], S_AXI_RRESP[1:0], S_AXI_RVALID //from datasheet
Description: UART is a hardware module that provides asynchronous serial communication. It is used for transmitting and receiving data between devices in a serial fashion. By using AXI4-Lite Interface Signals we can control UART.
Purpose: facilitate serial communication between digital devices.

Module:microblaze_0_axi_intc
Inputs: s_axi_i, interrupt_address
Outputs: s_axi_o, processor_ack
Description: generate interrupt requests that need to be efficiently managed.
Purpose: managing interrupts in complex SoC designs

Module: reset_clk_wiz_1_100M
Inputs: slowest_sync_clk, ext_reset_in, aux_reset_in, mb_debug_sys_rst, dcm_locked, mb_reset
Outputs: bus_reset

Description: The Processor System Reset module provides control over various reset signals within the processing system.
Purpose: ensures that the processing system starts from a known and stable state.

Following IP for specific for 6.1
Module: axi_gpio
Inputs: s_axi_aclk(clock), s_axi_aresetn(reset), gpio_io_i, gpio2_io_i, gpio2_io_t
Outputs: gpio_io_o, gpio_io_t, gpio2_io_o, gpio2_io_t //from datasheet
Description: axi_gpio has a memory address, when we dereference it, we can change the behavior of the LED.
Purpose: single output used to blink a single LED.

c. Describe in Lab 6.1 how the I/O works.

In Lab6.1, input is the switches for input, LEDs are used as output devices.
We use the GPIO module as a bridge from MicroBlaze to FPGA logic, GPIO modules are the input (to software), output (to FPGA fabric). GPIO has a memory address assigned. By using the pointer in the C code, we can dereference the address, read the input data or change the data corresponding to the output LED. Then the value will be reflected back to the LED.sv file and change the behavior of the LED.

d. Describe in words how the MicroBlaze interacts with both the MAX3421E USB chip and the ball motion components.

The input data of MAX3421E is computed inside the MicroBlaze and then the result is sent into gpio_usb_keycode. then the keycode will be interpretate as the direction signal inside the ball.sv module.

e. Describe in detail the VGA operation, and how your Ball, Color Mapper, and the VGA controller modules interact.

VGA writes the screen row by row from left to right from top to bottom, it will reset the vertical sync every 1/60 seconds, meaning the screen is refreshed every 1/60 second(60HZ). Ball sends the position of the ball and the size of the ball, also Ball takes signals from the keyboard, VGA controller sends out DrawX, DrawY signals to indicate where to draw, all these signals go into Color Mapper to judge what color should be drawn for the current pixel.

f. Describe the VGA-HDMI IP, how does HDMI differ from VGA, how are they similar?

VGA-HDMI IP takes two different clock signal, and vde, vsync, hsync from VGA_controller, R/G/B from color mapper, and output two differential outputs clk_p,clk_n, data_n and data_p.
HDMI uses digital signal while VGA uses analog signal. They both are used to transfer video signals.

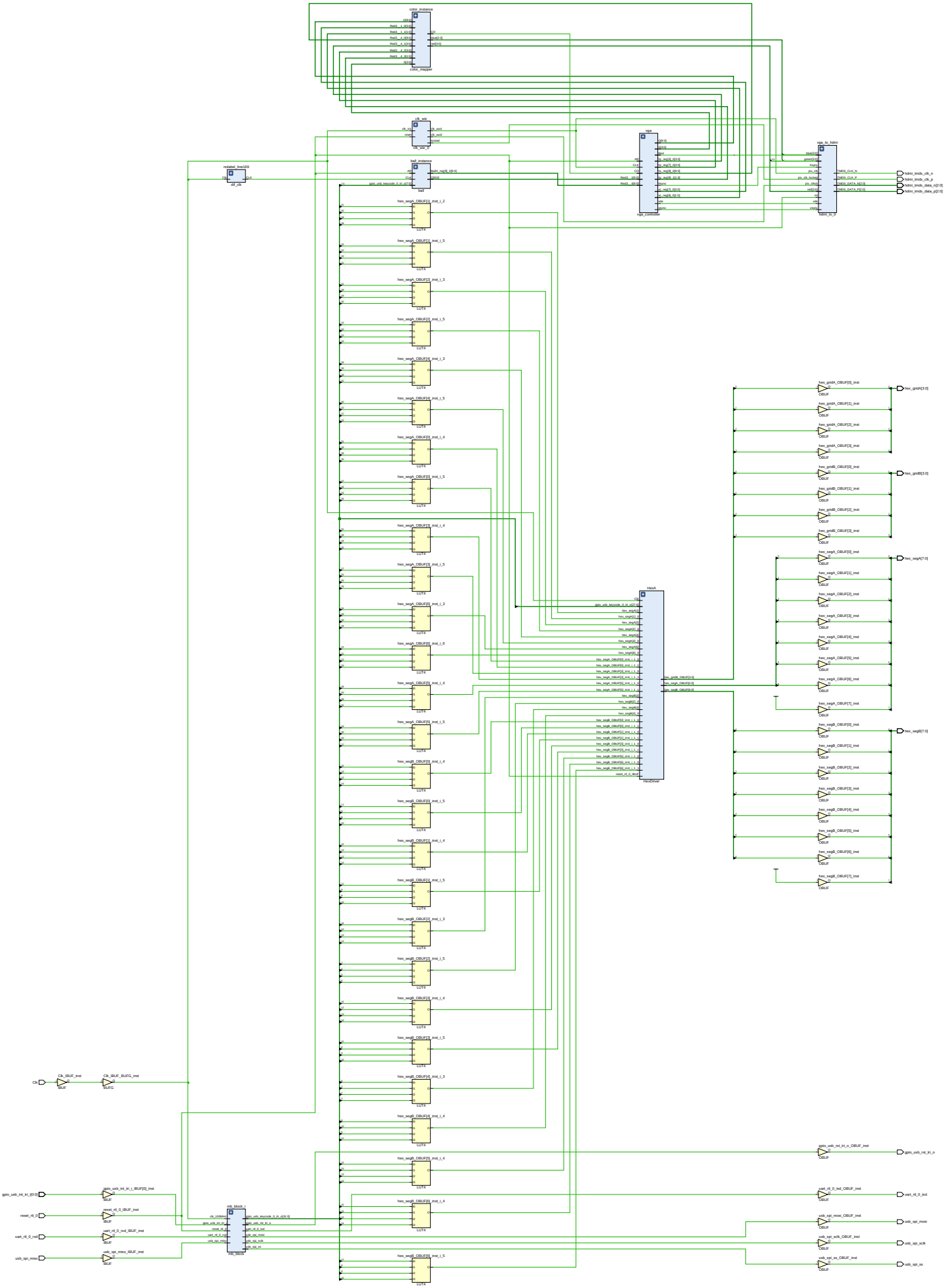

Figure 1（toplevel block diagram）

4.Description of software component of the lab

(a) The accumulator records a 16-bit data that performs data = data + SW, which SW is read from the switches, when Run signal is high, and will only perform this once before Run goes low. data = 0 will be performed after Reset goes high. If data + SW > 0xffff and Run from low goes to high, the console will print out a message indicating there is an overflow just happened.

(b) There are four signals inside the SPI protocol, CLK, MOSI, MISO, SS. CLK is the clock signal, MOSI means master out slave in and MISO means master in slave out. SS is referred to as slave select, it is an inverse of one-hot encoding vector. A slave is selected when the according bit goes low. It is also worth noting that SPI protocol supports Full Duplex Communication, which allows data to be transmitted and received simultaneously. Also, it is known that, at maximum, SPI can support MAX3421 with a clock at 26 MHZ. SPI transfers data from and to MAX3421 through a data buffer. Specifically, the write data buffer should first write reg + 2,
Indicating the address and the "2" means write, by setting 1 at bit position 1, then the data buffer contents the data of single or multiple bytes we want to write. In read operation, we need to prepare two same sized data buffers, and first write reg + 0 to send the address and indicate read, then receive single or multiple bytes of read data.

(c) MAXreg_wr is used to write data to MAX3421, it first selects the only slave MAX3421, which we should set SLAVEMASK = 1, which actually set SS = 0 to enable the slave MAX3421, also, we need to set a data buffer which first transfer reg + 2 and the data byte, note that reg is the address we want to write to, and 2 stand for write operation by putting 1at bit position 1 then, deselect the slave as the end of this operation. MAXbytes_wr does the similar operation, but the data buffer sent has multiple bytes. MAX_reg_rd needs a similar data buffer to send address to slave and also needs a receive data buffer of the same size to read data from slave. MAXbytes_rd allows multiple bytes to be read. Notably, those operations are performed by calling a XSpi_transfer function, which supports multiple bytes data transfer.

5. INQ

q1:You should do some research and figure out what are some primary differences between the various presets which are available.
A simple microcontroller running bare-metal applications; a real-time processor featuring cache and a memory protection unit interfacing to tightly coupled on-chip memory running FreeRTOS; and finally, an application processor with a memory management unit running Linux. Notably, microcontroller preset uses least logic cells and utilization, and can be run at maximum frequency 220 MHZ. On the contrary, Application uses the most logic cells and utilization and can be run at maximum frequency 150 MHZ.

q2: Note the bus connections coming from the MicroBlaze; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?

Modified Harvard, because modified harvard system uses same memory for instruction and data, but they split cache. And this design uses this structure.

q3: 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?

They don't share the same clock signal which controls the starting time of transmission.
Advantage of asynchronous:

1. No need for shared clock, which decrease the use of pins
2. Flexibility, because they can communicate with same rate without having same clock signal
3. Simpility, it allows a simpler design.
4. It can resilient data interrupt by using start and stop bits

Disadvantage of asynchronous:

1. Higher cost by adding start and stop bits
2. Only same data rate can be allowed to transmitted
3. Limit distance are allowed
4. Not suitable for high data rate design

Advantage of synchronous:

1. Higher data rate
2. Efficiency, no need for start and stop bits
3. The shared clock provides a consistent timing reference, making synchronous communication suitable for real-time applications

Disadvantage of synchronous:

1. More pins required
2. Complexity
3. Clock skew

q4: 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.

It allows the CPU to perform another task while waiting for data to be transmitted by UART, if interruption is not involved, the CPU will have to repeatedly check the status of UART to see whether the data is already transmitted, which will have a huge cost of time.

q5: Why are the UART and LED peripherals only connected to the data bus?

They only need to access through the data bus because they don't involve instruction operations like fetch, decode and execute. This is an advantage of Harvard design.

q6: 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 18), and how the set and clear functions work by working out an example on paper (lines 30 and 33).

For volatile, it tells the compiler not to optimize or reorder accesses to that variable.
LED signal is set to 1 by OR operation to set the last bit of 32 bits data 1,
Data = Data || 0x00000001. The LED signal is set to 0 by AND 0 by setting the last bit of 32 bit data. Data = Data && (~0x00000001)
For the accumulator store the value summing into the accumulator, displaying the output using the LEDs via microblaze. When we press 'Reset' (btn[0]) at any time clears the accumulator to 0. Pressing 'Accumulate' (btn[1]) loads the number represented by the switches into the CPU, adding it to the accumulator. In C code we use pointer led_gpio_data. Memory 0x40000000 is mapped to led_gpio_data.
q7: Look at the various segments (text, data, bss), what does each segment mean? What kind of code elements are stored in each segment?

Text: Machine code generated by the compiler from your source code. This includes functions, loops, and all other executable instructions.
Data: Global variables with explicit initial values. Static variables (both inside and outside of functions) with explicit initial values.
Bss: Global variables that don't have an initial value (implicitly initialized to zero). Static variables (both inside and outside of functions) that don't have an initial value (implicitly initialized to zero).

q8: 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

Because in order to compile the code, we need to import source code, which means including a lot of more code.

q9: Make sure you understand the register map on page 10. If the base address is 0x40000000, how would you access GPIO2_DATA (for example?).

0x400000000 + 8 = 0x40000008

6.

LUT	**2792**
DSP	**9**
Memory(BRAM)	**8**
Flip-Flop	**2625**
Latches	**0**
Frequency	**122.339HZ**
Static Power	**0.075w**
Dynamic Power	**0.381**
Total Power	**0.456**

7. Conclusion

(a) by connecting to the USB keyboard, the FPGA board can take up to 6 inputs as commands from the keyboard, and WASD is used as a control signal to control the movement of the ball on screen, the ball will bounce once it touches the border of the display.
(b)nothing to be included.