***Intro to System-on-Chip Design Course***

**LAB 09**

**PMod Expansion Header**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817608)

[1.1 Lab overview 1](#_Toc83817609)

[2 Learning Objectives 1](#_Toc83817610)

[3 Requirements 2](#_Toc83817611)

[4 Provided files 2](#_Toc83817612)

[5 Hardware 3](#_Toc83817613)

[5.1 HDL Source Files 3](#_Toc83817614)

[5.2 Memory Map 3](#_Toc83817615)

[6 Software 4](#_Toc83817616)

[6.1 API Development 4](#_Toc83817617)

[6.2 Create API file 5](#_Toc83817618)

[6.3 Application development 5](#_Toc83817619)

[6.4 Example program 6](#_Toc83817620)

[7 Extension work 7](#_Toc83817621)

[7.1 Extra tasks for this lab: 7](#_Toc83817622)

# Introduction

## Lab overview

This lab is an extension to the previous lab. In this lab, we will use the Expansion headers of the Basys 3 development board to expand our SoC with additional joystick peripherals and develop the final application further with a new API.

In this lab we will:

* Add the AHB-SPI bridge to the SoC in Vivado,
* Develop an API to handle communication with the AHB-SPI bridge that connects the Joysticks to the system,
* Develop the application from the previous lab (or your own application) to demonstrate the new features.

# Learning Objectives

* Implement an AHB to Dual SPI bridge on the FPGA to connect two joysticks peripheral to the AHB bus.
* Develop a two-player snake game using C program which implements timer handler to control and display/ detect the two snakes coordinates.
* Modify a C program to read the control inputs from two joysticks.

# Requirements

This lab requires the following hardware and software:

* **Hardware:**
  + **Diligent BASYS 3** FPGA board connected to computer via **MicroUSB cable.** A constraints file for this board is also provided.
  + **VGA-compliant monitor** and **VGA cable** to connect your board
* **Software**
  + Xilinx Vivado
  + Keil uVision
  + TeraTerm

# Provided files

* You will need the files from the previous labsexcept AHBLITE\_SYS.v.
* The following files are provided with this Lab

|  |  |
| --- | --- |
| **Name** | **Description** |
| AHBLITE\_SYS.v | New top level definition including instantiation of the SPI peripherals. |
| AHB2DUALSPI.v | AHB-SPI bridge top level including the control registers accessed via the AHB bus. |
| basys\_3\_constraints.xdc | New constraints file adding the PMod header to the previous file. |
| spiCtrl.v | Digilent provided source file controlling SPI communication. |
| SPImode0.v | Digilent provided source file that connects to and manages the SPI bus as master. |

* You also need to download the following files provided by Digilent from <https://www.instructables.com/id/How-to-Use-the-PmodJSTK-With-the-Basys3-FPGA/?_ga=2.114543111.1085112373.1566463814-1794713755.1562769977>

|  |  |
| --- | --- |
| **Name** | **Description** |
| spiCtrl.v | Digilent provided source file controlling SPI communication. |
| SPImode0.v | Digilent provided source file that connects to and manages the SPI bus as master. |

* Open ***basys\_3\_constraints.xdc*** and add the following at the end to include connections to the PMod header used in this lab.

#Joystick connected to PMod A

set\_property -dict {PACKAGE\_PIN J1 IOSTANDARD LVCMOS33} [get\_ports SS\_1]

set\_property -dict {PACKAGE\_PIN L2 IOSTANDARD LVCMOS33} [get\_ports MOSI\_1]

set\_property -dict {PACKAGE\_PIN J2 IOSTANDARD LVCMOS33} [get\_ports MISO\_1]

set\_property -dict {PACKAGE\_PIN G2 IOSTANDARD LVCMOS33} [get\_ports SCLK\_1]

#Joystick connected to PMod B

set\_property -dict {PACKAGE\_PIN A14 IOSTANDARD LVCMOS33} [get\_ports SS\_2]

set\_property -dict {PACKAGE\_PIN A16 IOSTANDARD LVCMOS33} [get\_ports MOSI\_2]

set\_property -dict {PACKAGE\_PIN B15 IOSTANDARD LVCMOS33} [get\_ports MISO\_2]

set\_property -dict {PACKAGE\_PIN B16 IOSTANDARD LVCMOS33} [get\_ports SCLK\_2]

# Hardware

## HDL Source Files

This module uses the Digilent JSTK2 PMod expansion board. This module uses an SPI serial bus to receive instructions from a master and send information about the Joystick’s state and position. Below is a diagram of the SoC:

Diagram

Description automatically generated

For the precise details about the SPI bus, please view the following link:

<https://www.mouser.co.uk/datasheet/2/690/pmodjstk2_rm-1099496.pdf>

## Memory Map

The updated memory map for the system is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Peripheral** | **Base address** | **End address** | **Size** |
| SRAM | 0x0000\_0000 | 0x00FF\_FFFF | 16MB |
| VGA | 0x5000\_0000 | 0x50FF\_FFFF | 16MB |
| UART | 0x5100\_0000 | 0x51FF\_FFFF | 16MB |
| Timer | 0x5200\_0000 | 0x52FF\_FFFF | 16MB |
| GPIO | 0x5300\_0000 | 0x53FF\_FFFF | 16MB |
| 7-segment display | 0x5400\_0000 | 0x54FF\_FFFF | 16MB |
| Joystick 1 | 0x5500\_0000 | 0x55FF\_FFFF | 16MB |
| Joystick 2 | 0x5600\_0000 | 0x56FF\_FFFF | 16MB |

In the top-level file AHBLITE\_SYS.v, two peripherals are instantiated in parallel. The SPI peripheral runs at 66.67KHz so, to reduce latency (and save power by reducing the run-time of the ISR), two SPI peripherals are run in parallel to double the bandwidth.

In software, the order of execution would be as follows:

* Start the data transfer on Joystick 1,
* Start the data transfer on Joystick 2,
* Wait for Joystick 1 to finish,
* Wait for Joystick 2 to finish.

***Note: Ideally, this may not be the best way to implement this type of game.***

# Software

## API Development

Below is the register map for each of the SPI peripherals:

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Base address** | **Size** | **Description** |
| Write Command | +0x00 | 4 bytes | Contain the command to be sent to the JSTK SPI slave. 8-bit value defined in the datasheet. |
| X Position | +0x04 | 4 bytes | 10-bit value representing the X position of the peripheral. |
| Y Position | +0x08 | 4 bytes | 10-bit value representing the Y position of the peripheral. |
| Status | +0x0C | 4 bytes | 8-bit value containing the state of the peripheral push-buttons. |
| Control | +0x10 | 4 bytes | Control register used to initiate data transfers and monitor ongoing requests. |

The content of the control register is described below:

|  |  |
| --- | --- |
| **Bit Number** | **Use** |
| 7 | Unimplemented |
| 6 | Unimplemented |
| 5 | Unimplemented |
| 4 | Unimplemented |
| 3 | Unimplemented |
| 2 | Unimplemented |
| 1 | Command finished. 1 while a command is ongoing. |
| 0 | Start command. Command is sent on the rising edge. |

## Create API file

Under the “Device” folder, create a header file called “jstk\_driver.h” and a source file called “jstk\_driver.c”.

Write the functions in “jstk\_driver.c” and include all the function definitions and parameters in “jstk\_driver.h”.

Suggested functions are as follows:

|  |  |
| --- | --- |
| **API Functions** | **Description** |
| void jstkSendPositionCmd() | Initialize the write\_command register and begin the transfer with the control register. |
| void jstkWaitForCmd() | Wait for the control register to indicate the command has finished before continuing. |

The PMOD JSTK2 datasheet lists the following requirements for the system:

* Wait 15us from bringing CS low before sending the first byte of data,
* 25us between bringing CS high and bringing CS low.

In the context of this API, this means:

* There is a 15us delay for the control register to update. This wait should be reflected in the code.
* There must be a 25us delay before sending the next command.

When timing it is practical to use the value of the timer. The periodic pulses can be counted and used to determine when the timing period has been met.

## Application development

In this section we will suggest ideas to develop on the Interrupt-driven method from Lab 11. The UART handler is not explicitly used since the keyboard input is not used to control the snake game but the UART port can still be used to send data for the purposes of debug.

* Main program (Reset Handler)
  + Initialize the SoC.
  + Initialize the game.
  + Enter the sleep-on-exit mode.
* Timer interrupt handler
  + Read the state of both Joysticks using the jstk\_driver API.
  + Update the directions of both snakes and move them one step.
  + Detect if the target is reached or if the snake hits the wall

The Timer ISR should execute the following steps in sequence:

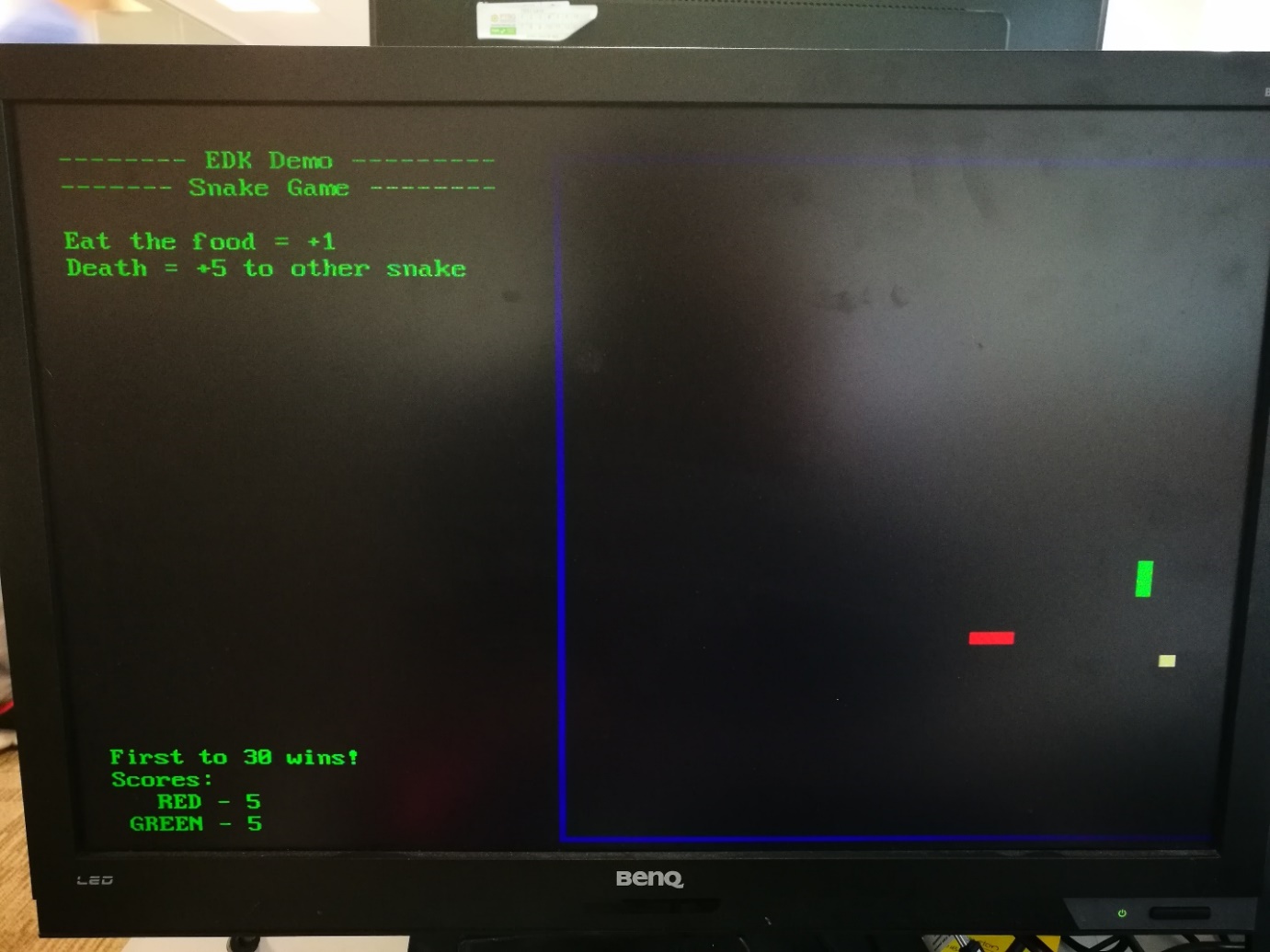
* Read the Joystick Peripherals,
* Update the direction of the snake,
* Change the position of the snake,
* Check for collisions (Target, other snake, wall),
* Update the state of the game (including updating the display).

Since there are two snakes, the rules of the game should be changed. These are the rules that have been implemented in our example but do not have to be followed in your application.

If a snake eats the food, the player gains one point. If the snake dies, the other player gains five points. When one snake has 30 points, the game ends and they are declared the winner.

* If one snake eats the food, that snakes gains +1 point.
* If a snake dies, the other snake gains +5 points.
* The first snake to 30 points wins and the scores are reset when the system is reset

## Example program



This is how demo example program should look.

# Extension work

## Extra tasks for this lab:

This extension lab has gone through the process of creating and controlling additional peripherals through drivers in the C language. There are many available PMod extension boards available through Digilent, which cover a range of application such as:

* WiFi Transceivers,
* GPS Receivers and Accelerometers,
* Audio Input and Output.

A complete list can be found at the following link:

<https://store.digilentinc.com/pmod-modules-connectors/>