Winter 2024

ECE 315 Computer Interfacing

LAB 3: Zynq-7000 SPI Interface

DATES

Date	Section	Time	Demo Due Date	Report Due Date
12-Mar-2024	D21	2:00 to 4:50 PM	26-Mar-2024	26-Mar-2024 11:00 PM
13-Mar-2024	D31		27-Mar-2024	27-Mar-2024 11:00 PM
14-Mar-2024	D41		28-Mar-2024	28-Mar-2024 11:00 PM
15-Mar-2024	D51		05-Apr-2024	05-Apr-2024 11:00 PM

LEARNING OBJECTIVES

- To gain practical experience with the Serial Peripheral Interface (SPI) in both the controller (master) and peripheral (slave) modes.
- To gain practical experience with an SPI device, specifically the Pmod OLED screen, to delve into the fundamentals of embedded graphics programming.

HARDWARE PLATFORM AND SOFTWARE ENVIRONMENT

- The hardware platform is a Digilent Zybo Z7 development board. As with all the labs in ECE 315, CPU0 will run the FreeRTOS real-time kernel, this time with three non-idle tasks for the SPI interface.
- The fixed Zynq-7000 System-On-Chip (SoC) hardware configuration and the initial skeleton source file for Exercise 1 must be downloaded from the Lab 3 section of the lab eClass site. The skeleton source files contain the initial application code in C that you will be modifying to implement your designs in the two exercises.
- The Xilinx/AMD Pmod OLED (revision A) module will be utilized in this lab for Exercise 2, providing you an opportunity to work with graphical output via the SPI protocol and to enhance your understanding of digital communication and embedded graphics programming.

Winter 2024

ECE 315 Computer Interfacing

LAB 3: Zynq-7000 SPI Interface

INTRODUCTION

This laboratory exercise aims to provide students with a comprehensive understanding and hands-on experience in the utilization of the widely used Serial Peripheral Interface (SPI) communication protocol. This laboratory session is designed to facilitate an in-depth exploration of SPI in both its controller (master) and peripheral (slave) operational modes. Utilizing the advanced hardware platform provided by the Digilent Zybo Z7 development board, alongside the Xilinx/AMD SPI interface and the Pmod OLED (revision A) module, students will complete two exercises. These are intended to bridge theoretical knowledge with practical application, focusing on the manipulation of data within an SPI system and interfacing with an OLED display to generate dynamic graphical outputs. Furthermore, this laboratory seeks to empower students to apply their acquired knowledge towards the development of innovative applications or games, thereby fostering a deeper comprehension of embedded systems design and enhancing their problem-solving capabilities within a real-time operating system framework. This laboratory exercise is not merely an exercise in programming but an opportunity to cultivate the skills necessary for creative engineering problem-solving and innovation in interfacing and embedded systems engineering.

.

PRE-LAB **Marks 10%**

Go to the Lab 3 section in the lab eClass site for ECE 315 and download the zipped directory for the provided project. Unzip the directory into a suitable location. Locate, open, and study the main source .c and .h source files. Separate .c and .h files are provided for Part 1 and Part 2. Note that the FreeRTOS application inside **lab_3_part_1.c** consists of the main function, two queues, two global variables (for loopback enable, discussed later), and the three tasks **vUartManagerTask**, **vSpiMainTask** and **vSpiSubTask**, which have the following roles:

vUartManagerTask manages the serial interface between the terminal window of SDK on the host PC and the FreeRTOS system that is running on CPU0 in the Zynq-7000 SoC on the Zybo Z7. The task displays a menu of numbered commands to the user, receives command selections, and then displays the results in the terminal. This task also monitors the bytes that are sent back to the SDK terminal, looking for a message termination sequence (defined later).

vSpiMainTask manages the SPI1 interface in the Zynq-7000 SoC using controller (master) mode. This task receives data bytes from vUartManagerTask via FIFO1. The task can send data bytes to vSpiSubTask over a bidirectional SPI bus, which is implemented using four wires in the FPGA fabric, which connect the SPI0 interface with the SPI1 interface. In order to "push" characters over the SPI bus (to the peripheral task and back), vSpiMainTask can insert control characters into the data stream that goes to vSpiSubTask. These control characters can be passed from vUartManagerTask to vSpiMainTask. Any control characters that return from the peripheral task are removed from the character stream and they will not be displayed on the terminal. The master task can then send the remaining data bytes on to vUartManagerTask using FIFO2. Using a loopback mode that is enabled by a global variable, vSpiMainTask echoes data bytes back to vUartManagerTask without sending those data bytes over the SPI interface. When this loopback feature is disabled, the SPI connection is enabled between the vSpiMainTask and vSpiSubTask.

vSpiSubTask manages the SPI0 interface in the Zynq-7000 using sub-node (slave) mode. This task exchanges data with vSpiMainTask over the bidirectional SPI bus. This task will be modified by you to count and return the number of bytes and messages received from vSpiMainTask. The number of bytes must be communicated using the following message format: "***The total number of characters received over SPI: <number>\n. The number of messages received over SPI so far: <number1>\n***". The task will send messages in this format to the SDK terminal, via vSpiMainTask and vUartManagerTask, that provide the byte count and message count messages after the character bytes that were originally typed into the terminal by the user. This count message will be sent after the detection of the termination sequence "**\r%\r**" from the user.

As pre-lab work, construct a system architecture diagram that illustrates the data connections and control relationships among the SDK terminal on the host, the UART1 interface, the SPI1 interface, the SPI0 interface, the three tasks that are running in the FreeRTOS environment, the two queues, and all the global variables (loopback enable/disable variables). This diagram is also to be included in your report.

PART 1 - IMPLEMENTING SPI LOOPBACK

Marks 30%

Step 1: Open the provided Vivado project file with Vivado and execute the Open Block Design command in the IP INTEGRATOR submenu. Make a screenshot copy of the displayed diagram and include it in your report. Note the connections between the two SPI interfaces, SPI1 and SPI0, and explain the significance of these connections and their role in the overall design architecture.

Step 2: Synthesize the design by executing the command Generate Bitstream in the PROGRAM AND DEBUG submenu. When the synthesis steps have all completed (it will take some time), execute the command File –> Export –> Export Hardware (be sure to specify the "Include bitstream" option before you click OK). Once the synthesized hardware design has been exported for software development, execute the command File -> Launch SDK. In the Project Explorer window of SDK, create an empty FreeRTOS application project and copy the source files provided in the resources for part 1. The source code for this exercise is inside **lab_3_part_1.c**. The file **initialization.h** contains the function prototypes and declarations. The SPI Main Read/Write, and SPI Sub Read/Write functions are defined inside **initialization.c.**

Step 3: Select the project subdirectory in the Project Explorer window and execute the Run As -> Launch on Hardware (System Debugger) command to compile the project, download it to the Zybo Z7 board, and then start executing the software. The target system should display a menu of numbered commands in the terminal window. Each command is selected by typing the corresponding number followed by Enter.

Step 4: Verify that the command "**Toggle UART Loop-back Mode**" functions correctly. The loopback feature should start off with loopback disabled. Issuing the command should enable a loopback path in vUartManagerTask. With the loopback enabled, text that is typed into the terminal window will be returned immediately for display in the terminal. Text entry is ended by typing the termination sequence (Enter, %, Enter) or (Enter, <command>, Enter). Ending text entry also causes the loopback path to be disabled, the command to be ended, and termination of the message output in the terminal. Information messages will be displayed in the terminal to confirm when loopback has been enabled and disabled.

Step 5: Verify that the command "**Toggle SPI Loop-back Mode**" functions correctly. The loopback feature should start off with loopback disabled. Issuing the command should enable a loopback path in vSpiMainTask . With the loopback path enabled, text that is typed into the terminal window will be returned immediately for display in the terminal. Text entry is ended by typing the termination sequence (Enter, %, Enter) or (Enter, <command>, Enter).

Ending text entry also causes the loopback path to be disabled. Information messages will be displayed in the terminal to confirm when loopback has been enabled and disabled.

Step 6: Verify that the default behavior in vSpiSubTask works correctly. When the loopback is disabled using command Loopback vSpiMainTask , the external SPI connection is activated. However, the initial template from the eClass will not display any output for this case. Students need to write the code within the designated commented sections in the file to cause output to be displayed.

Step 7: You are now to modify vSpiSubTask so that, as well as echoing characters that are typed at the keyboard, when the message termination sequence (Enter, %, Enter) is detected by this task in the stream of characters, vSpiSubTask will generate a string that will be sent immediately after the received bytes have been echoed over SPI back to vSpiMainTask. The generated string will state "***The total number of characters received over SPI: <number> \n . The number of messages received over SPI so far: <number1>\n.***", where <number> is the total number of bytes and <number1> is the total number of messages so far that were received by vSpiSubTask over the SPI0 interface. The special control characters that are added by vSpiMainTask into the stream of characters sent by the user at the SDK terminal will not be counted as received characters, but they will be returned to the vSpiMainTask. To make this work, you will also need to add the code inside vSpiMainTask and vUartManagerTask, as instructed in the source template file.

PART 2 - SPI OLED SCREEN INTERFACE Marks 30%

In this part of the lab, you will learn to interface with the Xilinx/AMD Pmod OLED (revision A), featuring a 128×32 pixel graphic OLED display. This module allows for complex graphical output via the SPI protocol, providing a hands-on opportunity to work with digital communication and embedded graphics programming at a high level.

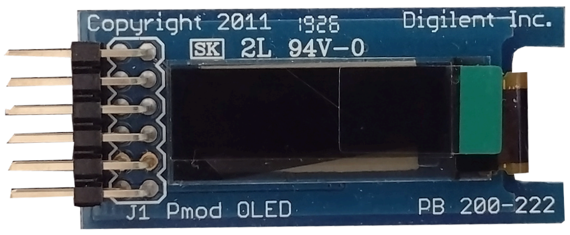

Figure 1: Pmod OLED screen

The Pmod OLED module communicates with the host board using the Serial Peripheral Interface (SPI) protocol. It requires the Chip Select (CS) line to be held low to enable data and command transmission. The state of the Data/Command (D/C) pin determines whether the transmitted bytes are interpreted as commands or data by the display controller. Specifically, when the D/C pin is pulled HIGH, the module interprets the bytes as data, allowing for the direct manipulation of pixels on the display. Conversely, when the D/C pin is pulled LOW, the bytes are transferred to the command register, enabling the execution of commands by the display controller. This flexibility allows for direct pixel manipulation, enabling the drawing of lines, shapes, and characters, as well as the rendering of bitmap images.

Pin	Signal	Description	Pin	Signal	Description
1	CS	Chip Select	7	D/C	Data/Command Control
2	MOSI	Master-Out-Slave-In	8	RES	Power Reset
3	NC	Not Connected	9	VBATC	Vbat Battery Voltage Control
4	SCK	Serial Clock	10	VDDC	Vdd Logic Voltage Control
5	GND	Power Supply Ground	11	GND	Power Supply Ground
6	VCC	Power Supply (3.3V)	12	VCC	Power Supply (3.3V)

Table 1: Pmod OLED pinout

To assist in application development, the manufacturer AMD/Xilinx provides the **PmodOLED_v1_0** library, which offers a suite of SPI functions and drivers for screen interfacing. However, due to performance issues with some functions, we've introduced custom replacements. The files **OLEDControllerCustom.c** and **OLEDControllerCustom.h** contain optimized versions of **OLED_DrawLineTo();** and **OLED_RectangleTo();**, superseding the original **OLED_LineTo();** and **OLED_DrawRect();**. While you are encouraged to familiarize yourself with the standard library functions in **PmodOLED.h**, please use the improved custom functions for these specific operations to ensure proper operation.

You are provided with a basic demo application that demonstrates the use of the Pmod OLED for drawing figures and displaying text. Your first task is to replace the existing functionality of the demo application entirely. Implement a new feature where a horizontal line sweeps across the screen from top to bottom, and a vertical line sweeps from left to right. These lines should move at different speeds. To achieve this, utilize the **OLED_DrawLineTo** function in a loop, iterating through every column for the vertical line and every row for the horizontal line, respectively.

Specifications:

- **Horizontal Line Sweep:** The line should start at the top of the screen and move down to the bottom, once it reaches the bottom it should start moving up back to the top.
- **Vertical Line Sweep:** The line should start on the left side of the screen and move to the right, when it reaches the end it changes its direction to the left.
- **Speed Variation:** Ensure that the horizontal and vertical lines sweep across the screen at noticeably different speeds.

For your final task, you are now required to develop a small application or game. This project should make use of two or more input peripherals (e.g., keypad, buttons, switches, keyboard) for user interaction, with the Pmod OLED screen serving as the output peripheral.

Requirements:

- **Display:** Your application must display both text and graphics.
- **Input Interaction:** The input devices should influence the display in real-time. For example, you could use buttons to change displayed information (such as score or time) or to control graphical elements on the screen.

- **Creativity:** You are encouraged to be creative with your application or game design. It could be anything from a simple interactive story to a classic game like "Snake" or "Pong," adapted for the OLED display.

Assessment Criteria:

- **Functionality:** Correct implementation of the line sweep and interactive application/game.
- **Creativity and Complexity:** Originality of the application/game design and effective use of the OLED display and input peripherals.
- **Code Quality:** Readability, structure, and documentation of the code.

Function name	purpose
OLED_GetPos();	Fetch the current graphics drawing position.
OLED_MoveTo();	Set the current graphics drawing position.
OLED_SetCursor();	Set the character cursor position to the specified location.
OLED_PutString();	Write the specified null terminated character string to the display and advance the cursor.
OLED_DrawPixel();	Set the pixel at the current drawing location to the specified value.
OLED_RectangleTo();	Draw a rectangle bounded by the current location and the specified location.
OLED_DrawLineTo();	Draw a line from the current position to the specified position.
OLED_SetFillPattern();	Set a pointer to the current fill pattern to use.
OLED_FillRect();	Fill a rectangle bounded by the current location and the specified location.
OLED_Update();	Update the OLED display with the contents of the memory buffer.
OLED_ClearBuffer();	Clear the display memory buffer.

Table 1: Useful functions for completing Part 2

QUESTIONS

- How does activating loopback mode (UART and SPI) impact the interaction and data flow between **vUartManagerTask**, **vSpiMainTask** and **vSpiSubTask**?
- What difficulties did you encounter while setting up SPI communication between the main and sub devices, and what solutions did you implement?
- Describe the challenges that you faced when interfacing with the Pmod OLED screen and describe how you resolved them.
- Does input from peripherals affect the dynamic graphics on the Pmod OLED screen? Give examples from your game.
- Reflecting on this lab's activities, how have the tasks and experiments deepened your understanding of designing embedded systems and operating within a real-time OS framework?

RESOURCES

- FreeRTOS Documentation.
- Zybo Z7 Reference Manual.
- Pmod OLED module Reference Manual.

REPORT REQUIREMENTS

Cover Page:

- ➔ Include course and lab section numbers, lab number, and due date.
- ➔ List all student names in your lab team. Please omit ID numbers to respect privacy.

Abstract:

- ➔ Summarize the lab's objectives, the hardware used and the exercises in your own words.

Design Section:

- ➔ Outline your solutions with a concise explanation of your code for each exercise.
- ➔ Include task flow diagrams. Avoid big sections of code snapshots.

Testing Suite:

- ➔ Provide a table with test descriptions, expected and actual results and a rationale for each test case.

Conclusion:

- ➔ Assess if the given objectives were met and summarize the lab's findings.
- ➔ Discuss any issues if the lab was incomplete or unsuccessful. What were the cause(s) of the issue and how could you resolve them if you had more time.

Formatting and Submission:

- ➔ Submit the report as a single PDF file with lab number and student names in the filename.
- ➔ Ensure correct spelling and grammar.
- ➔ Use a readable font size (larger than size 10).
- ➔ Diagrams may be hand-drawn with a straightedge and scanned at 300 pixel per inch resolution.
- ➔ Source code in your report should be syntax-highlighted to enhance readability. Syntax highlighting differentiates code elements such as keywords comments and strings.
- ➔ Adopt a professional format with consistent headings and subheadings.
- ➔ Only submit your modified source code files in one zip folder. Do not zip the entire workspace.
- ➔ Adhere to the submission deadlines as indicated on the first page of the lab handout.

MARKING SCHEME

Students will be graded based on the form and general quality of the report (clarity, organization, tidiness, spelling, grammar) for the lab. The Pre-lab work is worth 10%. The lab demos are worth 60% and the report is worth 30%. The demonstrations for Parts 1 and 2 are worth 30% each.