ECE 315 Computer Interfacing

Lab #3: Interfacing Using the Zynq-7000 SPI Interface

Winter 2022

Lab Dates, Demo Due Dates, and Report Dates

This lab exercise will be held on March 1 (LAB H21), March 2 (LAB H31), March 3 (LAB H41), and March 4 (LAB H51). You must demonstrate your designs at the beginning of the first session of Lab #4 (Mar. 15, Mar. 16, Mar. 17, and Mar. 18, for sections H21, H31, H41 and H51 respectively). Your lab report must be uploaded by 11:00 pm on the same day that the demonstration is due for lab sections H21, H31, H41 and H51.

Objectives

- To gain experience with using the serial peripheral interface (SPI) in both the master (controller) and slave (peripheral) modes.
- To gain experience with creating artificial load on a CPU and then measuring the resulting load using the FreeRTOS function **`vTaskGetRunTimeStats()`**.

Hardware Platform and Software Environment

- The hardware platform is a Digilent Zybo Z7 development board. A Digilent Cora Z7 will also work. 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.

Documentation

- Digilent Zybo Z7 Reference Manual: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual

- The FreeRTOS Reference Manual: https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.
- Chapters 6 of the ECE 315 lecture slides.
- Documentation and examples for Xilinx's SPI driver functions:
  https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/spi/src
  https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/spi/examples

Laboratory Exercise Instructions

Pre-lab work

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 .c and .h source files. Separate .c and .h files are provided for Exercise 1 and Exercise 2. Note that the FreeRTOS application inside *spi_lab3_main.c* consists of the **`main`** function, two queues, two global variables (for loopback enable, discussed later), and the three tasks **`TaskUartManager`**, **`TaskSpi0Master`** and **`TaskSpi1Slave`**, which have the following roles:

- **`TaskUartManager`** manages the serial interface between the terminal (also called the console) 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).

- **`TaskSpi0Master`** manages the SPI0 interface in the Zynq-7000 SoC using master mode. This task receives data bytes from **`TaskUartManager`** via **`FIFO1 (queue1)`**. The task can send data bytes to **`TaskSpi1Slave`** over a bidirectional SPI bus, which is implemented using four wires in the FPGA fabric, that connects the SPI0 interface with the SPI1 interface. In order to "push" characters over the SPI bus (to the slave task and back), `TaskSpi0Master` can insert control characters into the data stream that goes to `TaskSpi1Slave`. These control characters can be passed from `TaskUartManager` to `TaskSpi0Master`. Any control characters that return from the slave 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 to `TaskUartManager` using `FIFO2 (queue2)`. Using a loopback mode that is enabled by a global variable, `TaskSpi0Master` echoes data bytes back to `TaskUartManager` without sending those data bytes over the SPI interface. When this loopback feature is disabled, the SPI connection is enabled between the `TaskSpi0Master` and `TaskSpi1Slave.`

- `TaskSpi1Slave` manages the SPI1 interface in the Zynq-7000 using slave mode. This task exchanges data with `TaskSpi0Master` over the bidirectional SPI bus. **This task will be modified by you to count and return the number of bytes received from `TaskSpi0Master`.** The number of bytes must be returned using the following message format: **`The number of characters received over SPI: <number>\n.`** By return we mean that the task will send messages to the SDK terminal, via **`TaskSpi0Master` and `TaskUartManager`**, that provide the byte count message after the message bytes that were originally typed into the terminal by you 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 systematic diagram that illustrates the data connections and control relationships among the SDK terminal on the host, the UART1 interface, the SPI0 interface, the SPI1 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.**

Note

Please take the time to carefully read the initial comments and explanations provided in the source and header files for Exercises 1 and 2.

Exercise 1: Verify the given system and add the byte count message (Marks: 35%)

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, SPI0 and SPI1, and describe them in your report.

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, go to the *src* subdirectory of the Lab 3 project subfolder and verify that it contains the expected .c source files and .h header files. The source code for this exercise is inside *spi_lab3_main.c.* The file *initialization.h* contains the function prototypes and declarations. The SPI Master Read/Write, and SPI Slave Read/Write functions are defined inside *initialization.c*.

Step 3: Select the software 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 (or Cora Z7) board, and then start executing the software. The target system should display a menu of numbered commands in the terminal window at the bottom of the SDK desktop. Each command is selected by typing in *Enter,* and the corresponding number followed by *Enter*.

Step 4: Verify that the command *Loopback TaskUartManager* functions correctly. The loopback feature should start off with loopback disabled. Issuing the command should enable a loopback path in `TaskUartManager`. 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, 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 *Loopback TaskSpi0Master* functions correctly. The loopback feature should start off with loopback disabled. Issuing the command should enable a loopback path in **`TaskSpi0Master`**. With the loopback path enabled, text that is typed into the terminal 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 behaviour in **`TaskSpi1Slave`** works correctly. When the loopback in disabled for command *Loopback TaskSpi0Master,* 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 at the designated commented sections in the file to cause output to be displayed.

Step 7: You are now to modify **`TaskSpi1Slave`** 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, a string will be generated by **`TaskSpi1Slave`** that is sent immediately after the received bytes have been echoed over SPI back to **`TaskSpi0Master`**. The generated string will state ": **`The number of characters received over SPI: <number> \n`** ", where <number> is the total number of bytes that were received by **`TaskSpi1Slave`** over the SPI1 interface. The special control characters that are added by **`TaskSpi0Master`** 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 master task. To make this work, you will also need to add the code inside **`TaskSpi0Master and TaskUartManager`** as instructed in the source template file.

Exercise 2: Calibrate the load generator experimentally (Marks: 35%)

In this exercise you will modify the *load_gen_main.c* file from the eClass, which generates an artificial CPU load that is controlled by the global `u32` variable `loop_count`. For this exercise, students must create a new Application Project in SDK and add the downloaded source code file inside "src" directory of the project. **Please refer to the PDF file "Important_modification_for lab3 in Xilinx SDK BSP folder" on the eClass for incorporating a required Xilinx BSP flag change for this exercise.** Note that the source code contains three tasks: `TaskCpuLoadGen`, `TaskLoopCountProcessor`, and `TaskPrintRunTimeStats`.

- `TaskCpuLoadGen` generates a fake load on the CPU by executing a useless bitwise complement operation on dummy data, which safely consumes several cycles of the CPU time at a constant fixed FreeRTOS delay of 1 tick time. This task will repeat the bitwise complement operation for `loop_count` times using a loop.
- `TaskLoopCountProcessor` is responsible for updating the loop count value and adjusting the appropriate delay period that gives enough time to the CPU to execute the `TaskCpuLoadGen` task.
- `TaskPrintRunTimeStats` prints the amount of time each task spends in the running state. The percentage value corresponding to the IDLE task and CPU load generation task will vary based on the `loop_count` value.

Step 1: Modify `TaskLoopCountProcessor` to change the `loop_count` value and adjust an appropriate delay period for each `loop_count` value.

Step 2: Modify `TaskCpuLoadGen` to perform the bitwise complement operation for `loop_count` number of times.

Step 3: Modify `TaskPrintRunTimeStats` to print the percentage values of time consumed by each task in running state. Display this tabular formatted data on the terminal.

Step 4: Different values of `loop_count` will cause different load factors on the CPU, but you do not yet know the relationship between those `loop_count` values and the corresponding percentages of IDLE time. By trial-and-error experiments, determine the values of `loop_count` that correspond to IDLE time percentages of 40%, 50%, 60%, 70%, 80% and 90%.

Demonstrations

The marks for Exercise 1 and 2 include a demonstration that you will give to a TA at the start of Lab #4. You are to demonstrate the operation of the SPI connection that echoes the characters and displays the byte count string message on the terminal. For Exercise 2, you are to demonstrate that when you enter the experimentally determined `loop_count` values, you obtain IDLE task loads of 40%, 50%, 60%, 70%, 80% and 90%. The TA will ask you to try some other `loop_count` values.

Report Requirements

Your report must include the final versions of the commented source files that you used to implement Exercises 1 and 2. Your report should briefly describe in your own words your two final designs (including the unchanged tasks and the modified tasks) and include the system diagram that illustrates the control relationships and data flows among the terminal window, tasks, queues, global variables, and the hardware interfaces (UART1, SPI0 and SPI1).

The report must take the form of **one file in pdf format** that is uploaded by the report deadline to eClass. Students can attach the code files with the report and submit all the work as one zip folder on eClass.

Marking Scheme

The form and general quality of the report (clarity, organization, tidiness, spelling, grammar, sufficient explanatory comments in the code) will be considered as well when grading the reports. The demonstration for Exercises 1 and 2 will be worth 55%, and the report is worth 45% . Partial marks will be awarded for partially working designs.