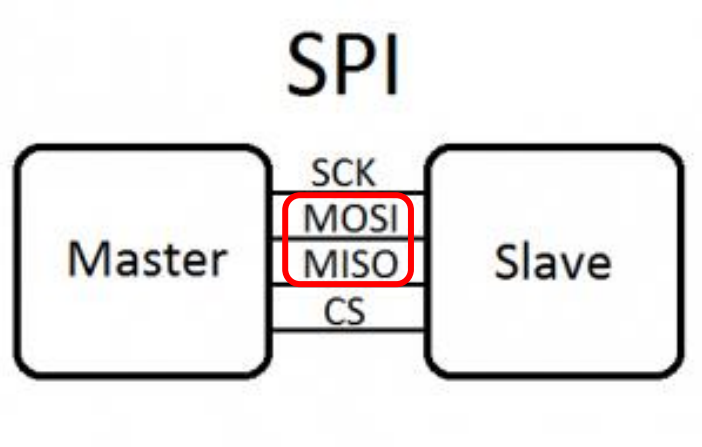
LAB -3 : Using SPI on Zynq-7000

ECE – 315 : COMPUTER INTERFACING – WINTER 2022

SPI (Serial Peripheral Interface) - Overview

- SPI (Serial Peripheral Interface) can act as a Master / Slave.
- In this lab, two SPI controllers are being used, SPI 0 and SPI 1. Zynq has two SPI peripherals.
- SPI 0 is Master, and SPI 1 is Slave. Master and Slave simultaneously communicate with each other (full-duplex).
- For communication they used MISO (Master In Slave Out) and MOSI (Master Out Slave In) ports.
- Master and Slave used in the template code send one byte to each other.
- We are using loopback mode for SPI 0 and SPI 1. So, the data send by SPI 0 is also received back.



Reference: electrosome.com

Overview of the Template code (SDK) --- [1]

Exercise - 1

1. The tasks run on CPU 0 only. Four tasks : TaskUartManager(), TaskSpi0Master(), and TaskSpi1Slave()

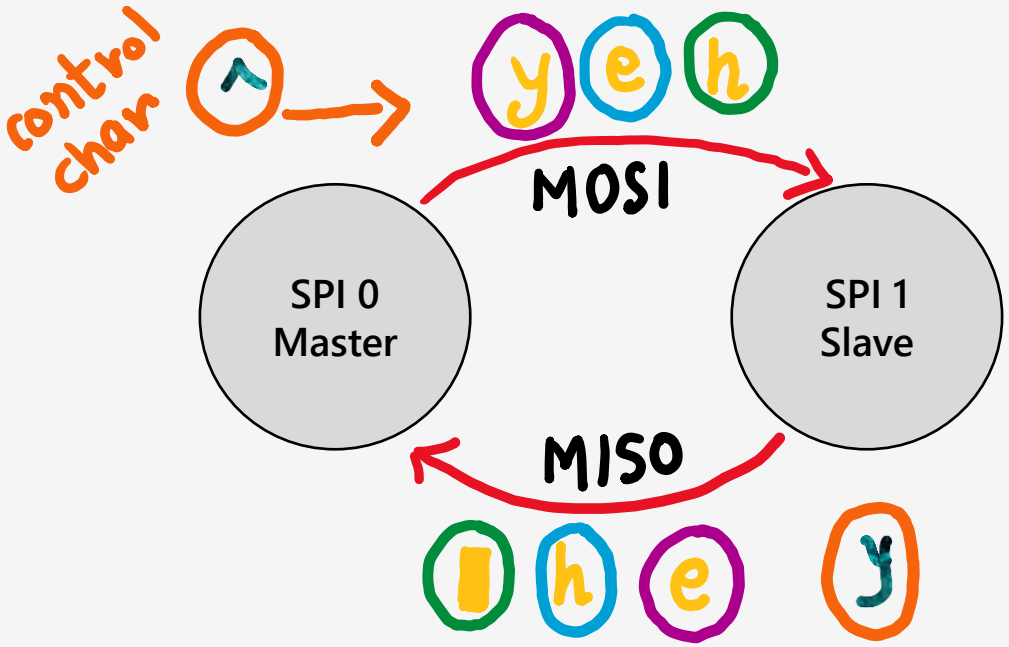
2. TaskUartManager() – Prints the USER menu. UART polls to check if there is any received data. If yes, is it a command? Yes, process it and take the respective action. Command 1 enables/disables loopback mode for this task. It is controlled using the variable `task1_uart_loopback_en.` Command 2 enables/disables loopback mode for TaskSpi0Master(). It is controlled using the variable `spi_master_loopback_en.`

3. On start of the application, enter the command : Say “enter” -> “1” -> enter. This will enable the UART loopback that is by disabled at the beginning. Same goes for command no. 2. Commands will be detected using the sequence “\r number \r”. When in loopback mode for command 1 or 2, “\r#\r” will disable the echo of text entry and disables the loopback mode too. Loopback can also be disabled by entering the command again.

4. TaskSpi0Master() - Uses FIFO 1 and FIFO 2 to execute the loopback when command 2 is entered by the user. FIFO1 connects task 1 to task 2. FIFO 2 connects task 2 back to task 1. When loopback mode for menu option 2 is disabled, SPI connection starts. Any data entered by the user is transmitted by SPI0 to SPI 1 interface. The characters that are coming back from SPI 1 to SPI 0 are then echoed back to the console via TaskUartManager.

Overview of the Template code (SDK) --- [2]

5 TaskSpi1Slave() manages the SPI 1 in slave mode. Currently, this task is receiving the bytes from the Master Interface SPI 0. Data arriving in the receive FIFO can be read after checking status bits (to confirm new data is available) on SPI 0. The same data is then written in the Tx FIFO of slave SPI.



Sending text “hey” from Master to Slave: MOSI and MISO transfers simultaneously. So, when you first write ‘h’ to Slave, you will receive garbage character. Next send ‘e’ to Slave. Now you will receive ‘h’ back on Master.

Keeps going on…..

To receive the last character, in this case ‘y’ back on Master, you insert a control character such as ^ to receive it. We are using $ as a control character in the lab.

This is because the Master and Slave are connected via Shift Registers such that it is a circular buffer. They operate in Serial In Serial Out (SISO) mode.

Overview of Step – 7 (From manual : Exercise – 1)

1. Currently, SPI slave is only echoing the characters back. Modify the TaskSpi1Slave() so that characters entered by user are echoed back when the termination sequence "\r#\r" is received from the user. Calculate the number of bytes received by TaskSpi1Slave(). After echoing the characters on the console, the string "The number of characters received over SPI: <number>", where <number> is the bytes received.

   The special control characters send from TaskSpi1Master() by the user are not to be considered when counting the received bytes in TaskSpi1Slave().

Exercise – 2 - Overview

1. TaskLoadCPUGen() generated artificial CPU load by executing a bitwise complement operation on a dummy data. The operation is repeated *loop_count* number of times.

2. TaskLoopCountProcessor() is responsible for updating the *loop_count* variable and providing enough delay period for this loop_count to stabilize.

3. TaskPrintRunTimeStats() prints the amount of time each task spends in the running state. The %value for IDLE task and CPU load Generator task will vary with the *loop_count* values.

GOAL:

Find out the "loop_count" values for IDLE task utilization of 40%, 50%, 60%, 70%,80% and 90%.