EE 705: VLSI DESIGN LAB 2025
PROJECT- SPI-LITE MODULE

TAs: Shahrukh + Harshitha

Note:

1. Submission is **only** through Moodle in the form of a PDF file upload.
2. Keep the port names and module names as mentioned in the question (including the case—whatever is capital must be capital & whatever is small must be small).
3. Code must be in **Verilog**(case-sensitive)**.** keep the module and port name as same as mentioned.

---

1) Design the spi module (module name: **spi_lite**) for a RISCV processor, that comprises of the following ports:
   a. Input Ports:
      i. clk_i : System clock
      ii. rst_i: Synchronous Active high Reset
      iii. cfg_awvalid_i: Valid Address signal requires for Handshaking to enable write
      iv. cfg_awaddr_i: Address of particular buffer to be written. (Some default value of address as flag)
      v. cfg_wvalid_i : Unused Signal, Keep it for Uniformity.
      vi. cfg_wdata_i : Configuration data and actual data to be sent by SPI.
      vii. cfg_wstrb_i: Unused Signal, Keep it for Uniformity.
      viii. cfg_bready_i : For Hanshaking of response channel similar to AXI response channel.(you don't need to check error of response channel. Ie. cfg_bresp_o is always zero for simpilicity.)
      ix. cfg_arvalid_i : Handshaking signal to enable read and also affects cfg_awready_o.
      x. cfg_araddr_i : Address of particular buffer to be read.
      xi. cfg_rready_i: Handshaking signal required for read.
      xii. spi_miso_i: Master In Slave Out.
   b. Output Ports:
      i. cfg_awready_o: Handshaking signal required for write Enable and also affects cfg_wready_o.
      ii. cfg_wready_o : Handshaking signal required for write data.(core requires this.)
      iii. cfg_bvalid_o: Handshaking signal required for write response channel.
      iv. cfg_bresp_o: Response of write response channel.(keep it 2'b0)
      v. cfg_arready_o: Handshaking Signal for Read Address.(keep it low when cfg_rvalid_o is high)
      vi. cfg_rvalid_o: Handshaking signal for Read Data.
      vii. cfg_rdata_o: Data read from configuration buffers.
      viii. cfg_rresp_o: Signal to tell read response is OK. (keep it 2'b0).
      ix. spi_clk_o: Serial clock
      x. spi_mosi_o: Master Out Serial In
      xi. spi_cs_o: SPI chip_select
      xii. intr_o: Interrupt.

2) Functional Description of the SPI-Lite module:
    a. Read about SPI communication Protocol(see link). Current “SPI-Lite” is more complex. Ports description is Similar to AXI.(It will be better if You read AXI first then UART then SPI.)
    b. When cfg_awaddr_i is `SPI_DGIER, cfg_data_i writes interrupt on buffer.
    c. Similarly do configuration for each case { `SPI_IPISR , `SPI_IPIER , `SPI_SRR , `SPI_SRR , `SPI_CR, `SPI_SR }
3) Make a test bench to test the complete module, showing all the input and output signals properly
4) The Hardware needs to be tested on FPGA using VIO and ILA.
5) Ensure that your complete module is compliant with RISCV ISA (RV32I)
6) Draw a block diagram. It should clearly represent how the integration of the SPI module will happen with the other modules.

Understanding the SPI Register Definitions
Here’s a breakdown of some key registers and their functions:

1. SPI_DGIER (Global Interrupt Enable Register)

- Address: 0x1C
- **SPI_DGIER_GIE (Bit 31)** → Global Interrupt Enable (1 = enable, 0 = disable)
- Used to enable **global** interrupt signalling for SPI.

2. SPI_IPISR (Interrupt Status Register)

- Address: 0x20
- **Bit 2 (SPI_IPISR_TX_EMPTY)** → Indicates if TX FIFO is empty.
- This register stores interrupt flags.

3. SPI_IPIER (Interrupt Enable Register)

- Address: 0x28
- **Bit 2 (SPI_IPIER_TX_EMPTY)** → Enables the TX empty interrupt.
- Used to enable specific SPI interrupts.

4. SPI_SRR (Software Reset Register)

- Address: 0x40
- **Bits [31:0] (SPI_SRR_RESET)** → Writing to this register triggers a reset.

5. SPI_CR (Control Register)

- Address: 0x60
- Controls SPI operations:
    - SPI_CR_SPE (Bit 1) → SPI Enable
    - SPI_CR_MASTER (Bit 2) → 1 = Master Mode, 0 = Slave Mode
    - SPI_CR_CPOL (Bit 3) → Clock Polarity
    - SPI_CR_CPHA (Bit 4) → Clock Phase
    - SPI_CR_TXFIFO_RST (Bit 5) → Reset TX FIFO
    - SPI_CR_RXFIFO_RST (Bit 6) → Reset RX FIFO
    - SPI_CR_LSB_FIRST (Bit 9) → 1 = LSB First

6. SPI_SR (Status Register)

- Address: 0x64
- Indicates SPI status:
  - SPI_SR_RX_EMPTY (Bit 0) → RX FIFO empty
  - SPI_SR_RX_FULL (Bit 1) → RX FIFO full
  - SPI_SR_TX_EMPTY (Bit 2) → TX FIFO empty
  - SPI_SR_TX_FULL (Bit 3) → TX FIFO full

7. SPI_DTR (Data Transmit Register)

- Address: 0x68
- Used for writing data to be transmitted.
- Data Width: **8-bits** (SPI_DTR_DATA_W = 8)

8. SPI_DRR (Data Receive Register)

- Address: 0x6C
- Used for reading received SPI data.
- Data Width: **8-bits** (SPI_DRR_DATA_W = 8)

9. SPI_SSR (Slave Select Register)

- Address: 0x70
- Controls **slave selection** (SPI_SSR_VALUE).
- Data Width=1, Deafult value=1

For Example , To define a particular Macro write in this way :

```
 `define SPI_IPIER    8'h28
      `define SPI_IPIER_TX_EMPTY      2
      `define SPI_IPIER_TX_EMPTY_DEFAULT    0
      `define SPI_IPIER_TX_EMPTY_B        2
      `define SPI_IPIER_TX_EMPTY_T        2
       `define SPI_IPIER_TX_EMPTY_W        1
      `define SPI_IPIER_TX_EMPTY_R        2:2
```

ADDITIONAL Materials and hints:

a. https://www.youtube.com/watch?v=0nVNwozXsIc&t=119s
b. cs.sfu.ca/~ashriram/Courses/CS295/assets/notebooks/RISCV/RISCV_CARD.pdf
c. Include this in your code:

```
module spi_lite
(
   // Inputs
    input         clk_i
   ,input         rst_i
   ,input         cfg_awvalid_i
   ,input  [31:0]  cfg_awaddr_i
   ,input         cfg_wvalid_i
   ,input  [31:0]  cfg_wdata_i
   ,input  [3:0]   cfg_wstrb_i
```

```
        ,input         cfg_bready_i
        ,input         cfg_arvalid_i
        ,input  [31:0]  cfg_araddr_i
        ,input         cfg_rready_i
        ,input         spi_miso_i

        // Outputs
        ,output        cfg_awready_o
        ,output        cfg_wready_o
        ,output        cfg_bvalid_o
        ,output [1:0]   cfg_bresp_o
        ,output        cfg_arready_o
        ,output        cfg_rvalid_o
        ,output [31:0]  cfg_rdata_o
        ,output [1:0]   cfg_rresp_o
        ,output        spi_clk_o
        ,output        spi_mosi_o
        ,output        spi_cs_o
        ,output        intr_o
      );



      //This always block can be part of SPI-Lite , It is  given for your understanding.

always @ *
begin
   data_r = 32'b0;

   case (cfg_araddr_i[7:0])

   `SPI_DGIER:
   begin
      data_r[`SPI_DGIER_GIE_R] = spi_dgier_gie_q;
   end
   `SPI_IPISR:
   begin
      data_r[`SPI_IPISR_TX_EMPTY_R] = spi_ipisr_tx_empty_in_w;
   end
   `SPI_IPIER:
   begin
      data_r[`SPI_IPIER_TX_EMPTY_R] = spi_ipier_tx_empty_q;
```

```
end
`SPI_SRR:
begin
end
`SPI_CR:
begin
   data_r[`SPI_CR_LOOP_R] = spi_cr_loop_q;
   data_r[`SPI_CR_SPE_R] = spi_cr_spe_q;
   data_r[`SPI_CR_MASTER_R] = spi_cr_master_q;
   data_r[`SPI_CR_CPOL_R] = spi_cr_cpol_q;
   data_r[`SPI_CR_CPHA_R] = spi_cr_cpha_q;
   data_r[`SPI_CR_MANUAL_SS_R] = spi_cr_manual_ss_q;
   data_r[`SPI_CR_TRANS_INHIBIT_R] = spi_cr_trans_inhibit_q;
   data_r[`SPI_CR_LSB_FIRST_R] = spi_cr_lsb_first_q;
end
`SPI_SR:
begin
   data_r[`SPI_SR_RX_EMPTY_R] = spi_sr_rx_empty_in_w;
   data_r[`SPI_SR_RX_FULL_R] = spi_sr_rx_full_in_w;
   data_r[`SPI_SR_TX_EMPTY_R] = spi_sr_tx_empty_in_w;
   data_r[`SPI_SR_TX_FULL_R] = spi_sr_tx_full_in_w;
end
`SPI_DRR:
begin
   data_r[`SPI_DRR_DATA_R] = spi_drr_data_in_w;
end
`SPI_SSR:
begin
   data_r[`SPI_SSR_VALUE_R] = spi_ssr_value_q;
end
default :
   data_r = 32'b0;
endcase
  end
```