SPI Controller

VLSI GURU

- Understand the design
  - SPI Controller overview?
    - What is SPI protocol?
  - why we need SPI Controller?
  - What external blocks SPI Controller interface with?
  - SPI Controller detailed functionality
  - SPI Controller Architecture
  - how many interface and Ports required with directions: Input, Output, Inout
  - Registers required?
  - Memories required?
- Does design require Parameters?
- Is design, combinational or sequential?
  - Sequential
    - How many processes are there in the design?
      - How many always blocks are required?
      - Does the process require state machine?
        - How many state, significance of each state.
        - Implementation style for state machine
- Implement the code
  - List down coding algorithm

SPI Controller

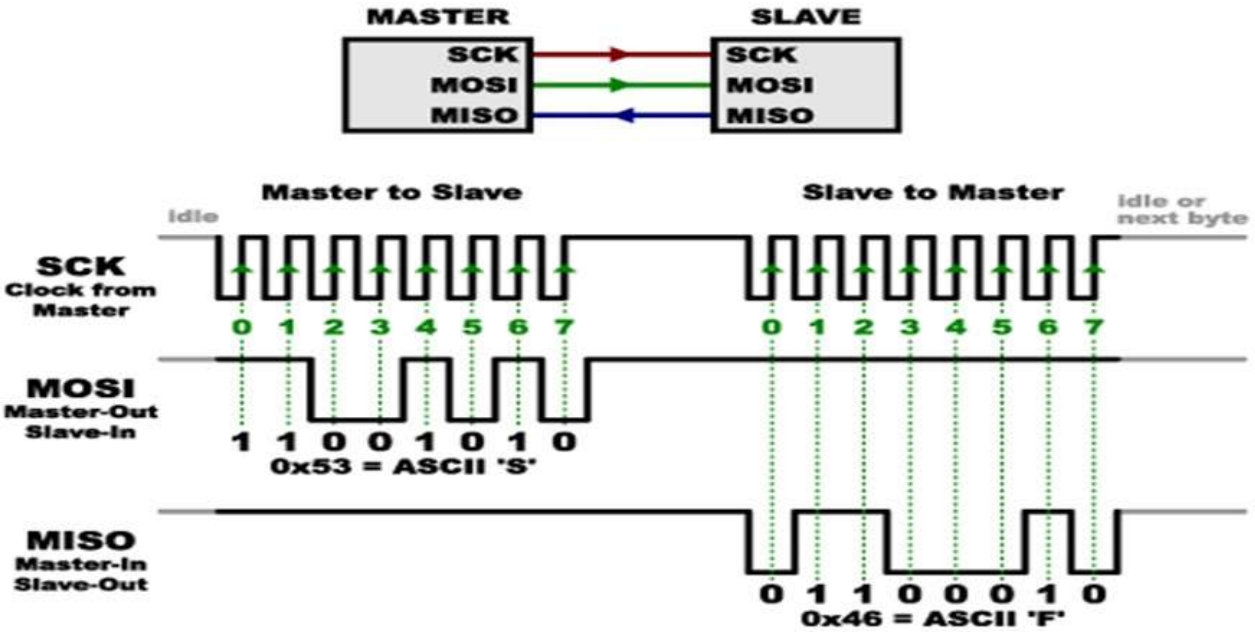
VLSI GURU

- SPI Protocol: Example
  - Near traffic junction, we see temperature, pollution, humidity, etc will be displayed. These systems have sensors, which give data(number) to microcontroller, which in turn displays the information.
  - Which protocol can be used for sensors giving data to microcontroller?
    - High performance? Low performance?
      - Performance is not important : Microcontroller requires this information once every 10 sec. Temperature is in range of 10 to 50, this will require only 6 bits, per minute 36 bits needs to be transferred. Compare this with USB ports requiring GB of transfer per minute.(high performance required
    - What is the priority?
      - Reducing the cost is the priority.
    - Hence we go with a protocol which gives low performance, low cost and low power consumption. SPI ideal for this requirement.
- Serial Peripheral Interface
  - Serial : Data is transmitted in serial manner
  - Peripheral: Data is transmitted to peripheral blocks
- Why we need SPI?
  - Low cost, Low power, Low performance
  - If the performance is not the criteria, we go for SPI protocol.
    - Ex: Sensing temperature in a electronic device
      - sense once a minute, how much data = 1 byte

SPI Protocol

VLSI GURU

- Read is happening to 0x53 address, slave is giving read data as 0x46
  - Master : SPI Controller
  - Slave: Temperature sensor with SPI interface
- SPI Ports: SCK, MOSI, MISO, CS
  - MOSI: Master Out Slave In



SPI Controller

VLSI GURU

- Design which interfaces with Processor or system components on one side and SPI slave(ex: temperature sensor, humidity sensor) on other side.
- Processor will configure SPI controller registers for initiating communication with SPI slaves.
- SPI controller gets processor transaction in form of APB(or AHB) transactions.
    - These transactions will write the registers in SPI controller
    - SPI controller will use these register values to drive SPI interface signals in SPI transaction format.
    - SPI slave can understand above transaction since it is coming in SPI format.
- In summary, SPI controller converts the APB transactions coming from processor in to SPI protocol format and viceversa.
    - SPI controller acting like intermediate component between microcontroller and SPI slave communication.
    - Microcontroller can't directly communicate with SPI slave, since it does not have SPI interface.

SPI Controller

VLSI GURU

- SPI Controller overview?
  - SPI Controller has 2 interfaces
    - **SPI controller interfaces with processor using APB interface, interfaces with SPI Slave using SPI interface.**
    - APB Interface: Used by processor to program address and data registers
      - address and data registers act like buffers
    - SPI Interface: Used to drive above address and data in serial manner
    - SPI Controller will have a **control register**. This register has en bit,
      - Whenever this bit is programmed to 1, SPI controller initiates a read to SPI slave(sensor). We program this register with en=1 once every minute, so that sensor reads will happen once every minute.
      - Data comes to SPI master, processor will read this data and displays it to the display board.
- why we need SPI Controller?
  - SPI Controller is not a source or not a sink of the data, all information comes from processor by programming address, data registers.
  - SPI Controller will only drive this address and data in serial manner
- Essentially, SPI Controller is only converting APB protocol information to SPI protocol format.

SPI Controller

VLSI GURU

- What external blocks does SPI Controller interface with?
  - One side with Processor
    - Program the address and data registers
    - Program the control register
  - Other side with SPI Slave
    - SPI Controller will issue addr +Wr_rd information to the SPI Slave
    - SPI Slave will either capture the data or will return the data
- Similar to SPI controller, for every peripheral protocol, corresponding controller is required so that peripheral can be interfaced with the processor.
  - USB2.0 requires USB2.0 controller (UHCI, OHCI, etc)
  - PCIe requires PCIe controller
  - MMC requires MMC controller.
- SPI Controller detailed functionality
  - Processor programs the registers
  - Processor configures ctrl_reg to trigger the transfer and to indicate how many transactions to do.
  - SPI Controller converts APB information in to SPI tx format and viceversa.
- SPI Controller Architecture
  - Not required
- SPI Interfaces
  - Processor: APB
  - SPI Slave: SCLK, Mosi, Miso, CS: Chip Select

SPI Controller

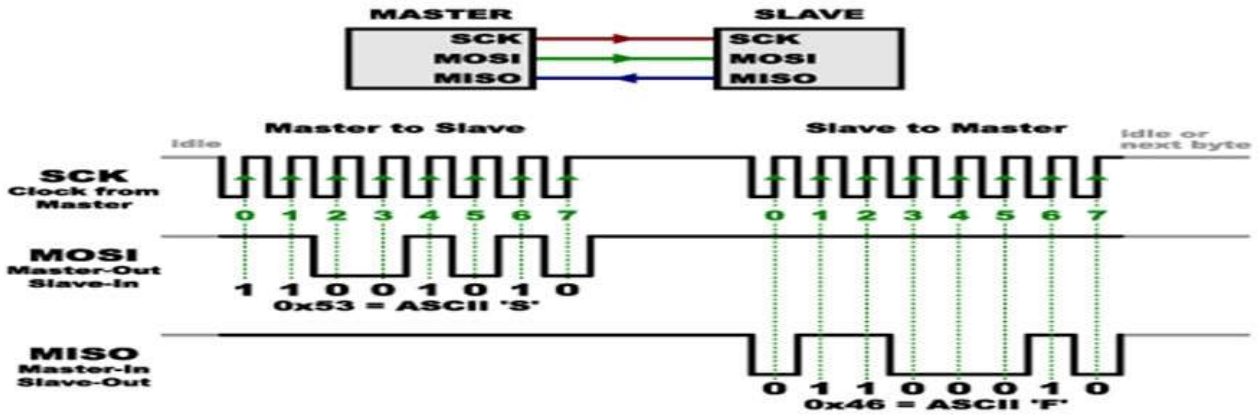
VLSI GURU

- SPI system Architecture
  - One master connected to multiple slaves
  - At any time master can do transaction to only one slave
    - Chip select indicates that target slave
    - CS[3:0] : 4 slaves
    - CS= 4'b0100 => 2nd slave is selected
      - 2nd slave will keep itself ready
    - CS=4'b0110 => Illegal scenario
      - 2 slaves can't be selected at same time
    - CS does not indicate write/read, it only indicates which slave is selected

SPI Timing diagram observations

VLSI GURU

- addr is issued by Master to Slave on MOSI
- addr/data are transmitted in LSB first manner
- clock will not be run(will be high), when addr/data is not transmitted
- last bit(MSB) of address phase indicates write(1) or read(0)
- MOSI and MISO by default will be '1' if no information to be sent
- SCLk is always driven by M->S
- both master and slave should have fixed delay based pattern, where they know when write/read data phases should start.
- addr and data sampling happens on +edge of clock
- addr and data must be transmitted without any breaks.
- read to slave at @53 is returning data of 46



SPI Controller coding

VLSI GURU

- Different phases in Timing diagram
  - IDLE
  - ADDRESS
  - IDLE_BW_ADDR_DATA
  - DATA
  - When we implement the code, above will become the states of the FSM.
- How write transaction happens?
  - SPI Ctrl will issue address and data on MOSI port.
- How read transaction happens?
  - SPI Ctrl will issue address on MOSI
  - SPI Slave will return data on MISO
- registers
  - addr_regA : list of addresses where master wants to perform access to SPI slave
    - Array size indicates the maximum number transactions we want to do at a time.
  - data_regA : registers that will hold either data to write to slave or data coming from slave.
    - at max all these values can be accommodated using 8 bits
      - temperature : 8 bits
      - wind speed => 290Km/hour => 8 bits is sufficient? (290 will require 9 bits)
        - We will use multiplication factor of 2
          - Sensor will send speed/2 number, at receiver I will do *2 factor
          - if I get 30 => actually means wind speed 2 * 30 = 60Km/hr
          - 8 bits can indicate 0 to 512 Km/hr speed, but it can only indicate even numbers.
  - ctrl_reg
- Memories? No
- Parameters? Yes MAX_TXS : How many maximum transactions SPI controller can do

SPI Controller : ctrl_reg decoding

VLSI GURU

- State machine : Yes
  - states?
  - S_IDLE
  - S_ADDR
  - S_IDLE_BW_ADDR_DATA
  - S_DATA
  - fields to keep track of where last transaction happened, so next transfer should start from that point

SPI Controller : Register programming

VLSI GURU

- Lets say registers are programmed as below (take all values in hexadecimal)
  - addr_regA = 53, 65, 32, 78, 97, 86, 52, 94
    - In all these fields MSB=0, hence all these indicate 8 read transactions.
  - data_regA = 12, 34, 54, 67, 98, 88, 17, 83
    - data_regA content does not hold any significance since all the transactions are read, data will come from SPI slave, that data will be stored to data_regA.

  - addr_regA = 53, D5, 32, E8, 97, F6, 52, 94
    - 5 reads(53, 32,97,52,94) , 3 writes(D5,E8,F6 => 55, 68, 76)
  - data_regA = 12, 34, 54, 67, 98, 88, 17, 83
- we transmitted **3 transactions**: (53,12), (65,34), (32,54)
- **#200 units delay**
- now I want to transfer **2 more**: (78, 67), (97, 98)
- some one has to keep track where **last tx happened**
- **ctrl_reg[0] = 1** : Trigger the transfer or initiate the transfer
- ctrl_reg[3:1] : How many transfers to do(with +1 mapping)
  - 2 => do 3 transfers
  - 0 => do 1 transfer
  - 7 => do 8 transfers (I still need only 3 bits to do 8 transfers also)
- **ctrl_reg[6:4] : What is the last transaction index**
  - In above example after 3, 2 transactions, cur_tx_index=5 (ctrl_reg[6:4] = 5)
- ctrl_reg[7] : error has occurred in the design.

SPI Controller

VLSI GURU

- When registers are there, there should be address. Each register should have unique address. Similar to how address was assigned to priority_regA in interrupt controller example(REG_START_ADDR to REG_START_ADDR+'hF).
  - addr_regA[7:0] => **8 unique addresses**
    - 8'h00, 01,02,03..07 (for 8 registers)
  - data_regA[7:0] => 8 unique addresses
    - 8'h10,.....8'h17 (for 8 registers)
    - why not start this from 8'h08?
      - 08 to 0F are reserved for future enhancements of design
      - In next version of design, if we need to use 16 address registers instead of 8 in current design, then we can assign addr_regA range from 8'h00 to 8'h0F.
      - In most of the designs, some address spaces will be kept reserved, these are meant to be used for future enhancements.
  - ctrl_reg
    - 8'h20
      - Same, 8'h18 to 8'h1F are kept reserved for data_regA size increase in next design versions.

SPI Controller coding

- Code spi_ctrl module with following ports
  - APB, SPI
- Declare parameters
- Declare the ports with directions and sizes
- Declare the registers
- Start implementing the processes
  - one process for programming the registers
    - Reuse from intr_ctrl examples.
  - Other process for implementing SPI protocol