SPI continued

- Lines:
  - Serial clock (SCK)
  - MOSI
  - MISO
  - Chip select for each responder($\overline{cs}$)

- SPI is implemented with shift registers (left/right shift bit)
- Shift bits across lines with each clock edge
  - Detect rising or falling edge
    - Depending on polarity
      - Idle high or idle low
- Transmit(tx) : (from controller)
  - Controller loads a byte into a register
  - Shift bits across, most significant bit(MSB) first
  - Responder shifts recv^d bits into its register
    - Where do the responder bits go?
      - Transmit to the controller
      - We always transmit bits
- Receive(rx) : (to controller)
  - Controller transmits dummy bits to receive responder data
  - How does the responder know to send?
    1) If this is the only function, then chip select ($\overline{cs}$) will loads the byte into register
    2) Multi-byte protocol (e.g. E^2)
       - Controller transmiting one or more bytes (e.g. addr)
       - Responder loads byte into register and wait for more transmit(SCK edges)
         - Controller <u>must</u> wait for responder to finish
- Issues: lots of responders use lots of I/O pins
  - Chip selects
- Solution: daisy chain the responders and use 1 $\overline{cs}$ for all
  - Often used to expand I/O with a bunch of shift registers
- E.g.

  image

  - To transmit or receive, the controlle must send n bytes for n registers
  - What happens to the I/O as we shift?
    - Output: output lines will “toggle” with each shift
      - To whatever was in the previous location/line

    - Input: all bits that are read will be the same value

- Fix: shift registers include a latch input
  - Activate a transfer of bits to/from register starting with a latch
  - Latch output when done shifting