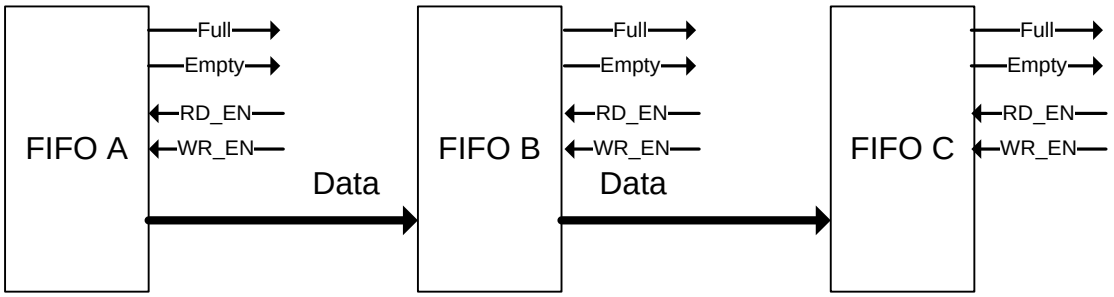
Working with Interfaces

.plan

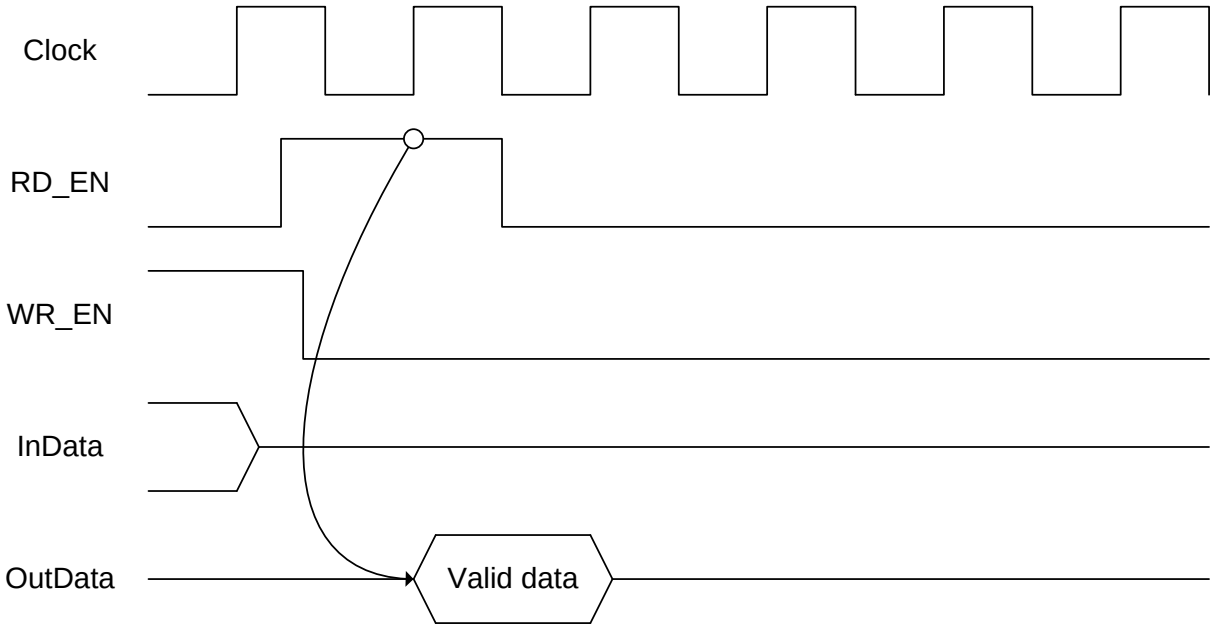
1.) Interface driven design.
    a. When modules are given to you.
        i. Signal conditioning “signal massaging”
    b. When writing your own modules.
        i. Handshake-based interfaces.
2.) Advantages of a clean interface.
    a. Remove combinational logic in between modules.
    b. Eliminate your own timing assumptions about your modules.
3.) What makes a **bad** interface?
4.) What makes a **good** interface?

Dealing with a Bad Interface Specification

Your mission is to design a system that passes data from FIFO A→B→C:

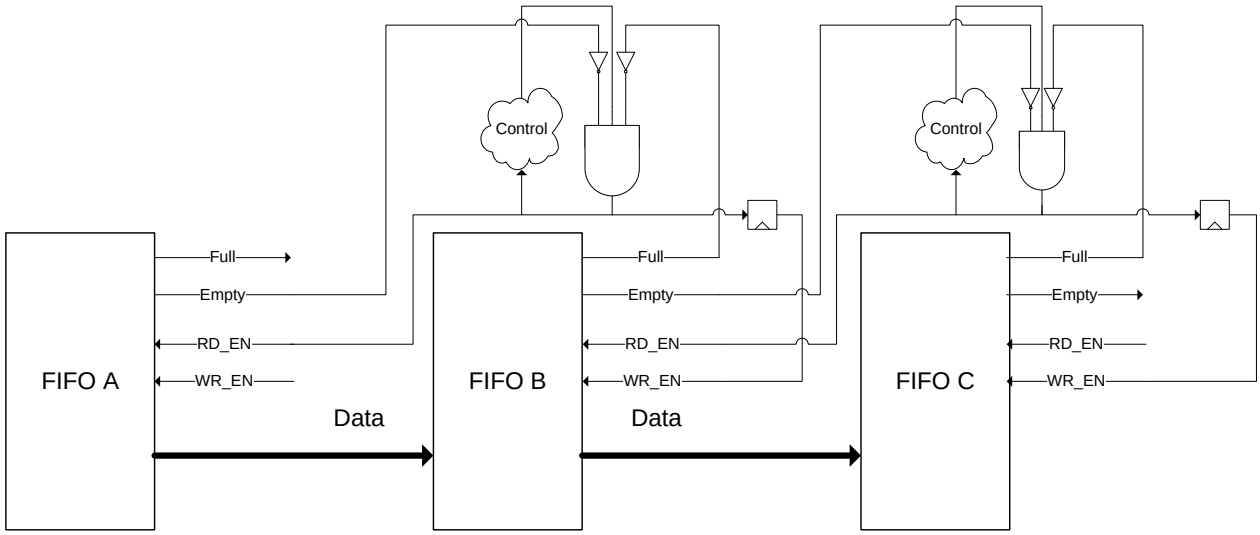


Each FIFO has a `RD_EN`, `WR_EN`, `Full`, and `Empty` signals (sound familiar?). The `RD_EN` and `WR_EN` signals have the following timing characteristics:



Your job: using gates and registers, create the logic in between these FIFOs that will ensure that the data going into FIFO A → FIFO C. In other words, if you continuously present FIFO A with data, ensure that it passes its data → B, which then passes its data → C.

Solution



Issues

- Basic data transport
  - Ensure that data only moves when the **source has data** and the **sink has space.**
  - Separate the RD_EN and WR_EN signals by a cycle so that data is valid on the Data line when the WR_EN rising edge happens.
- Preventing "word drop"
  - When the sink reaches its full state, we still write in an extra word (which the sink does not ever receive)
  - **This issue causes the circuit to fail.**

Discussion

Work through some timing examples to see why the basic data transport scheme used with this FIFO chain works.

Of more importance is the "word drop" issue. To see when this happens, draw a timing diagram where full and empty are both low for multiple cycles. Then, bring the full line into the sink high. Notice what happens to the system: you still read out an extra word (which is lost)! To fix this problem, add control logic (typically a counter) that controls data flow with another signal which pulses every certain number of cycles to limit data flow. This works because the "word drop" problem happens when full/empty are both low (so data will pass) for multiple cycles. If you limit the rate at which data flows to 1 word/X cycles, then you never pass consecutive words, and words are never dropped. More on this in discussion!