Chapter 7 Dealing with Asynchronous Boundaries – a few examples and considerations

1. Goal

The outside world is generally asynchronous – so even if we design a completely synchronous system, we often need to deal with asynchronous boundaries.

2. Some definitions:

(i) Setup time – the time of the D-input of a FF or latch must be stable **before** the active edge clocks the FF or latch.

(ii) Hold time – the time of the D-input of a FF or latch must be stable **after** the active edge clocks the FF or latch.

(iii) Settling time – the time it takes a FF that has come metastable to decide if its output is '0' or '1'.

(iv) Metastability – a phenomena whereby a FF can not decide whether the output is '0' or '1' for a long time because the data input of the FF changed logic state too close to the active clock edge of the FF clock.

3. Factors causing / affecting metastability

Asynchronous signals, duty cycle distortion on clocks, clock frequency, noise. By far, dealing with asynchronous signals causes us to deal with the problem of metastability most frequently.

(i) Asynchronous signals – we will have a metastability problem whenever we are required to synchronize an asynchronous signal.

➔ Such situation exists nearly everywhere in the real world.

➔ Examples: (1) Graphics cards in PC's that have their own processor and connect to a PCI bus

(2) Any voice switch where we are required to move the voice channel from one SONET/SDH or copper network to another SONET/SDH or copper network

(3) A mouse or keyboard or input peripheral devices connecting to a PC

(4) Move data trough a FIFO on a card that sits in a multi-service access switch or edge router – the line clock and system clock are often different and have no timing relationship to each other.

(ii) Duty cycle distortion on clocks – a phenomena where either or both of the high/low phases of the clock are high/low for slightly longer than the ideal clock.

➔ Often caused by different rise/fall times of logic gates/buffers that are driving the clock.

➔ Can cause metastability in a fully synchronous system if the depth of the logic is close to the clock period, such that any shrinkage in the clock period would cause timing to fail.

(iii) Clock frequency – in some situations where the clock frequency is so high that it is approaching the clock skew on the chip.

➔ We may have to worry about metastability even though the chip is a fully synchronous chip and everything is clocked using the same clock source.

➔ Solution: break down the design into blocks where you know metastability is not going to be a problem, and allow 2 clock cycles to perform operations between FFs where you metastability is a problem.

➔ Alternatively, place a FF in the middle of the path to ensure that there is no metastability problem

(iv) Noise – does not affect the overall performance of a synchronizer for low amplitude noise.

4. The Synchronizer – solving the metastability problem

If we put two FFs in cascade to sample an asynchronous data input, then we can greatly reduce the probability of the second sampling FF making a false decision and taking a long time to settle. This decision making function is called a Synchronizer, and it significantly reduces the probability of metastability on the output of the second sampling FF. The penalty to be paid for the second FF is an extra clock cycle of latency, and this is often not an issue.

<Diagram for synchronizer>

<Formula: from “Digital Integrated Circuits: A Design Perspective” by Jan M. Rabaey>

The robustness is determined by

1) Signal switching rate and rise time
2) Sampling frequency
3) Wait time T

Example:

5. **Synchronize point signals and busses**

➔ Point signals: never synchronize twice for two different pieces of logic. Rather, we synchronize once, and use the single synchronized signal to feed two different pieces of logic.

→ The reason is that inevitably there will be occasions where one synchronizer makes a different output decision than the second one, as we wire length feeding the data input to each synchronizer will be different and have a different delay.

➔ Bus signal: you can’t reliably synchronize a bus. Rather, we synchronize a control signal that validates the bus that guarantee that when the control signal is available, the value of the bus is stable and settled down.

6. **FIFOs and Synchronization of FULL/EMPTY Semaphores**

Consider a “transfer” based FIFO (as apposed to the circular FIFO), all the writer cares about is “Can I write?” and all the reader cares about is “Can I read”?

In this example, we are moving small packets of data through the FIFO, one at a time. And there is only one FIFO. ➔ We need a single status bit to tell when the FIFO is full and when it is empty. ➔ We call such a status bit a FULL/EMPTY semaphore (F/E). We can represent the F/E using a single D-FF:

The F/E in the Synchronizer block needs to know

a) When to set the F/E high?
b) When to set the F/E low?

However, WR_DONE and RD_DONE are asynchronous to each other. Which clock should we use to clock the FULLEMPTYB semaphore?

So, we use _______ for the F/E FF, then we need to synchronize the _________ signal.

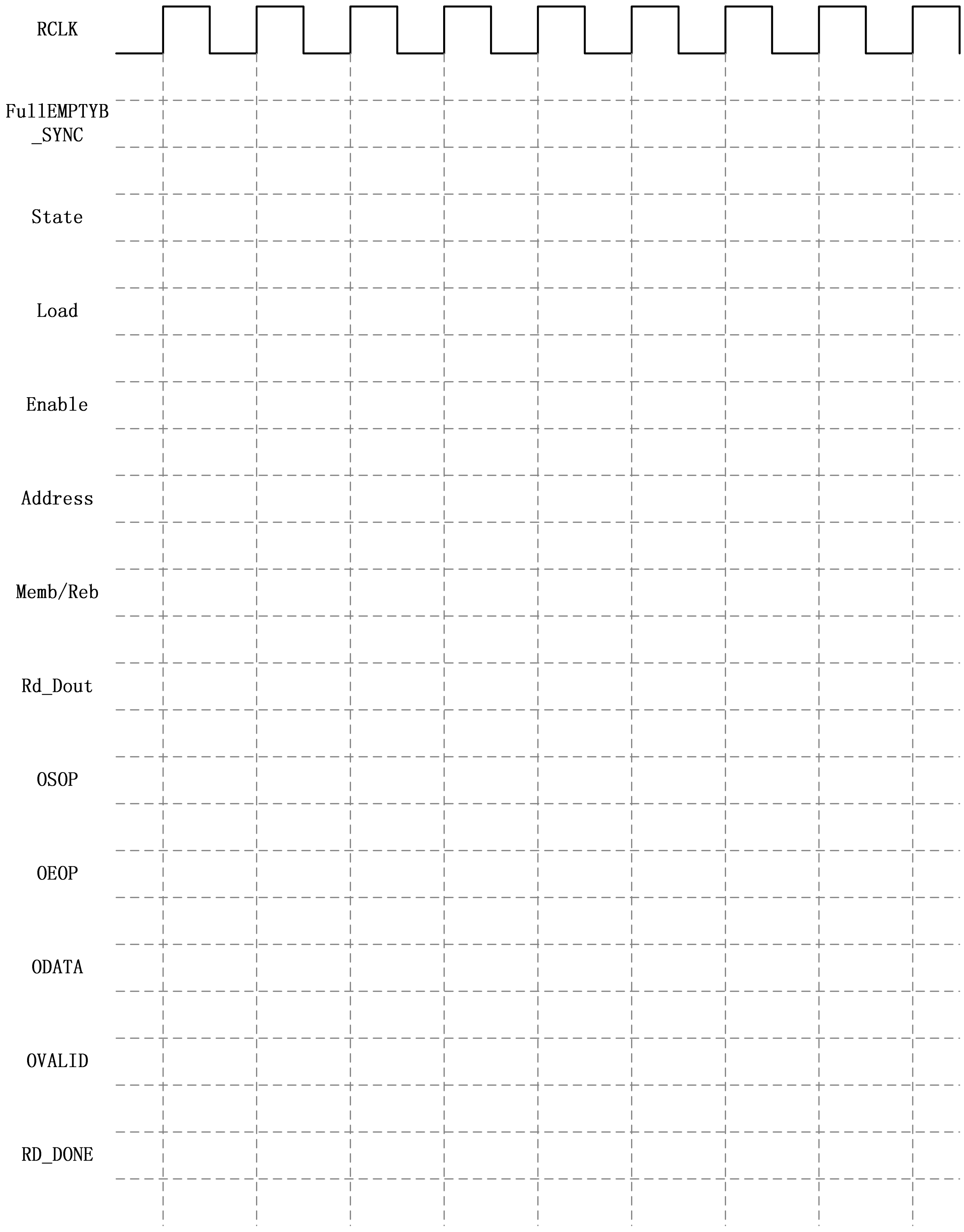
For RD_DONE is level signal, it is fine. But if RD_DONE is a pulse, then we might have problem. Example:

There are a couple of ways to prevent this:

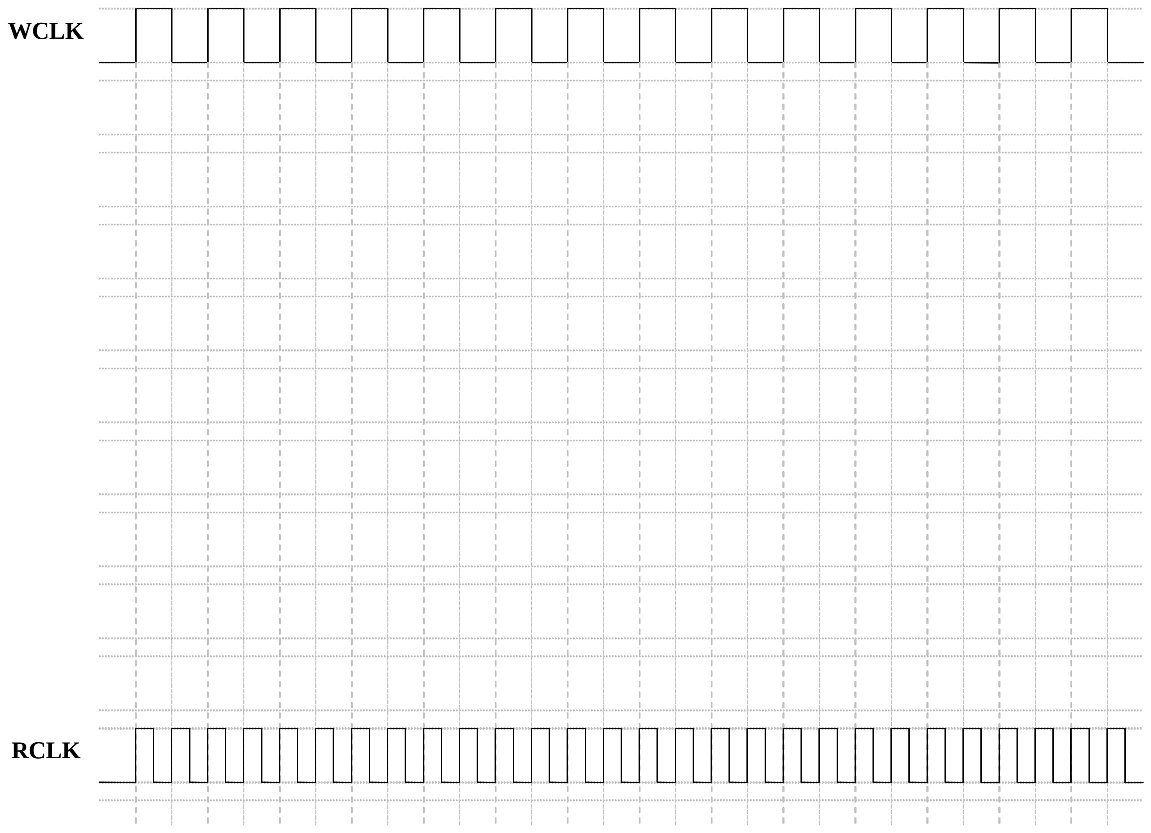
1) Edge detector

2) Synchronous sample/hold circuit

For the FULLEMPTYB signal



Read operation — Three Bytes Case


WCLK
RCLK