FIFO

Reference

- Complete Digital Design by Mark Balch
- FIFO Architecture, Functions, and Applications by Texas Instruments
- Simulation and Synthesis Techniques for Asynchronous FIFO Design by Clifford E. Cummings, Sunburst Design, Inc.

Introduction

- System On Chip with multi clocks
- Possible metastability

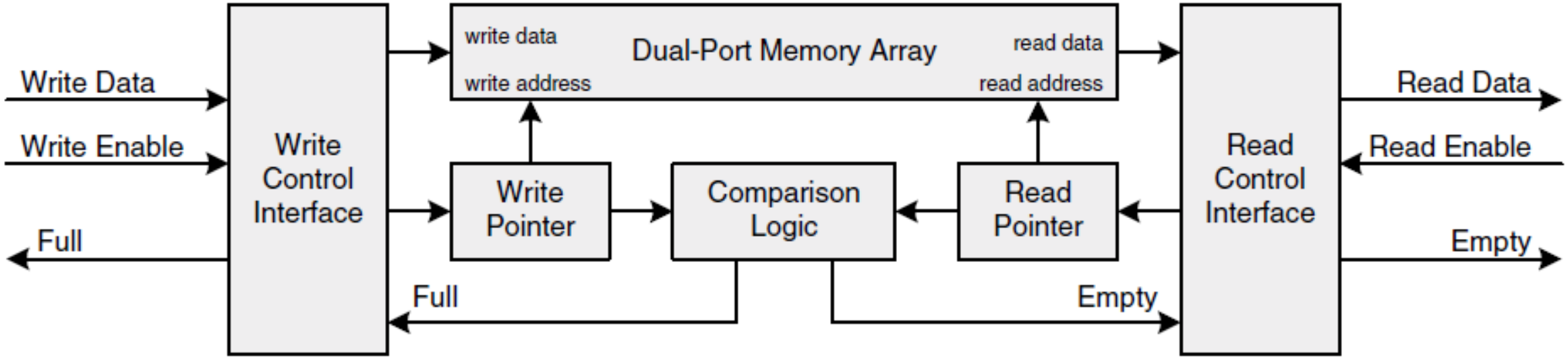
Introduction

- First-in-first-out (FIFO) memories are special-purpose devices that implement a basic queue structure that has broad application in computer and communications architecture.
- Unlike other memory devices, a typical FIFO has two unidirectional ports without address inputs: one for writing and another for reading.
- As the name implies, the first data written is the first read, and the last data written is the last read.
- A FIFO is not a random access memory but a sequential access memory.
  - Unlike a conventional memory, once a data element has been read once, it cannot be read again, because the next read will return the next data element written to the FIFO.
  - By their nature, FIFOs are subject to overflow and underflow conditions.
  - Their finite size, often referred to as depth
  - An overflow occurs when an attempt is made to write new data to a full FIFO.
  - An empty FIFO has no data to provide on a read request, which results in an underflow.

Basic FIFO Architecture

- A FIFO is not addressed in a linear fashion; rather, it is made to form a continuous ring of memory that is addressed by the two internal pointers.
- The fullness of the FIFO is determined not by the absolute values of the pointers but by their relative values.
- An empty FIFO begins with its read and write pointers set to the same value.
- As entries are written, the write pointer increments.
- As entries are read, the read pointer increments as well.
- If the read pointer ever catches up to the write pointer such that the two match, the FIFO is empty again.
- If the read pointer fails to advance, the write pointer will eventually wrap around the end of the memory array and become equal to the read pointer.
- At this point, the FIFO is full and cannot accept any more data until reading resumes.
- Full and empty flags are generated by the FIFO to provide status to the writing and reading logic.
- Some FIFOs contain more detailed fullness status, such as signals that represent programmable fullness thresholds.

Basic FIFO Architecture

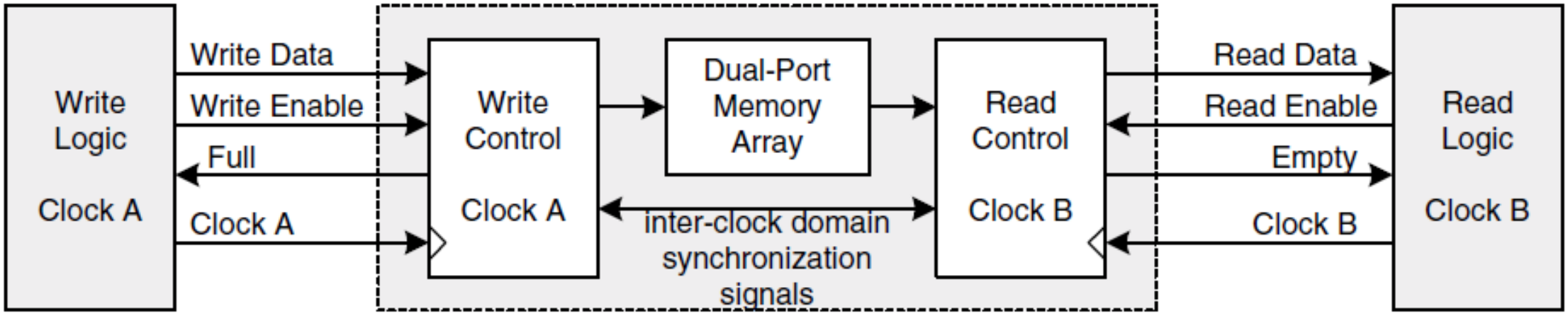


- A FIFO is created by surrounding a dual-port memory array—generally SRAM, but DRAM can also be used—with a write pointer, a read pointer, and control logic
- A FIFO can be synchronous or asynchronous

Basic FIFO Architecture

- One common role that a FIFO fills is in clock domain crossing.
- In such an application, there is a need to communicate a series of data values from a block of logic operating on one clock to another block operating on a different clock.
- Exchanging data between clock domains requires special attention, because there is normally no way to perform a conventional timing analysis across two different clocks to guarantee adequate setup and hold times at the destination flops.
- Either an asynchronous FIFO or a dual-clock synchronous FIFO can be used to solve this problem

Basic FIFO Architecture



Basic FIFO Architecture

- The dual-port memory at the heart of the FIFO is an asynchronous element that can be accessed by the logic operating in either clock domain.
- A dual-clock synchronous FIFO is designed to handle arbitrary differences in the clocks between the two halves of the device.
- When one or more bytes are written on clock A, the write-pointer information is carried safely across to the clock B domain within the FIFO via inter-clock domain synchronization logic.
- This enables the read-control interface to determine that there is data waiting to be read.
- Logic on clock B can read this data long after it has been safely written into the memory array and allowed to settle to a stable state.

Asynchronous FIFO Design

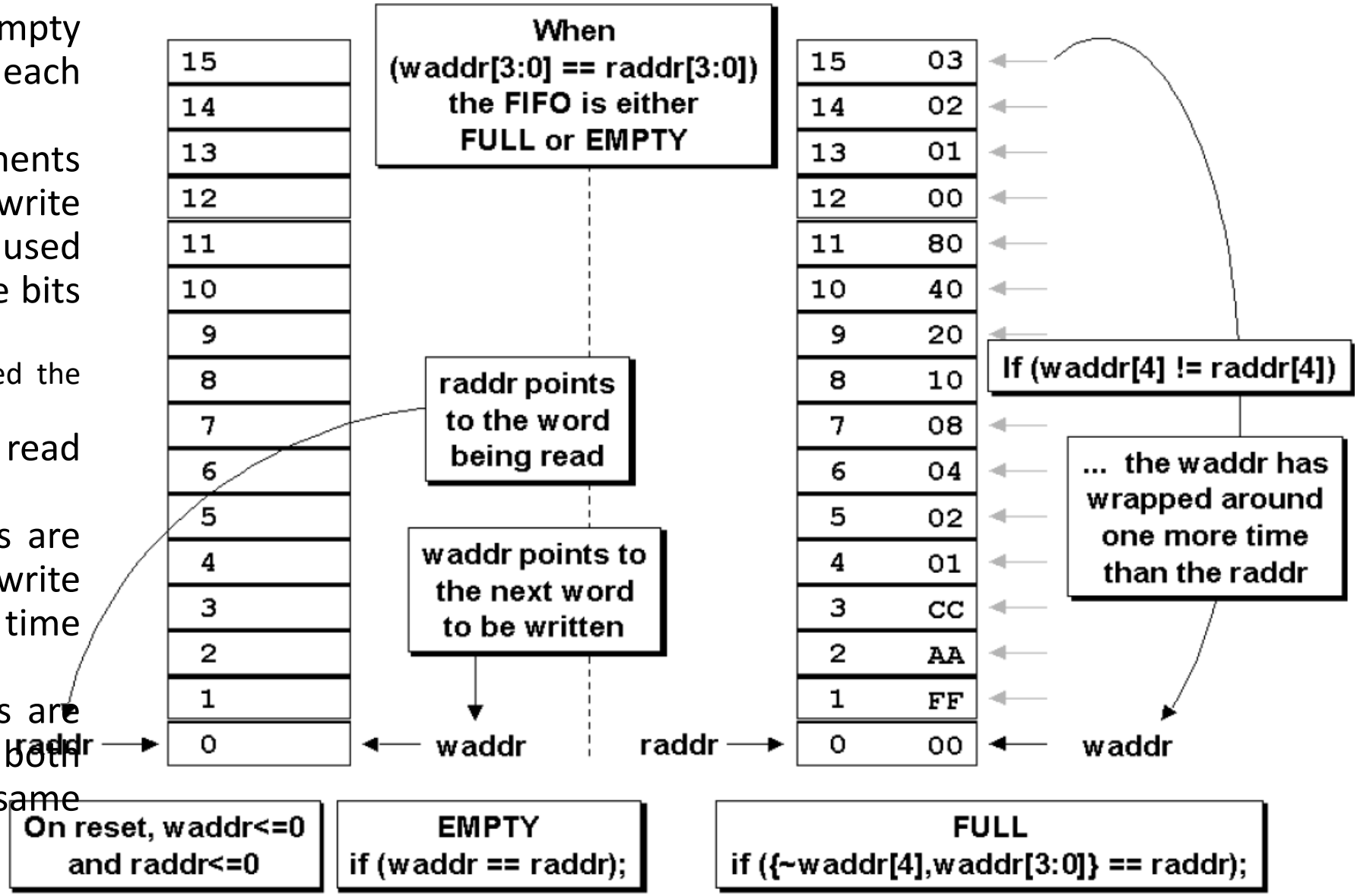
- Data values are written to a FIFO buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other
- Used to safely pass data from one clock domain to another asynchronous clock domain
- **Synchronous FIFO Pointer Design**
  - A FIFO where writes to, and reads from the FIFO buffer are conducted in the same clock domain
  - The implementation counts the number of writes to, and reads from the FIFO buffer to increment (on FIFO write but no read), decrement (on FIFO read but no write) or hold (no writes and reads, or simultaneous write and read operation) the current fill value of the FIFO buffer.
  - The FIFO is full when the FIFO counter reaches a predetermined full value and the FIFO is empty when the FIFO counter is zero.

Asynchronous FIFO Design

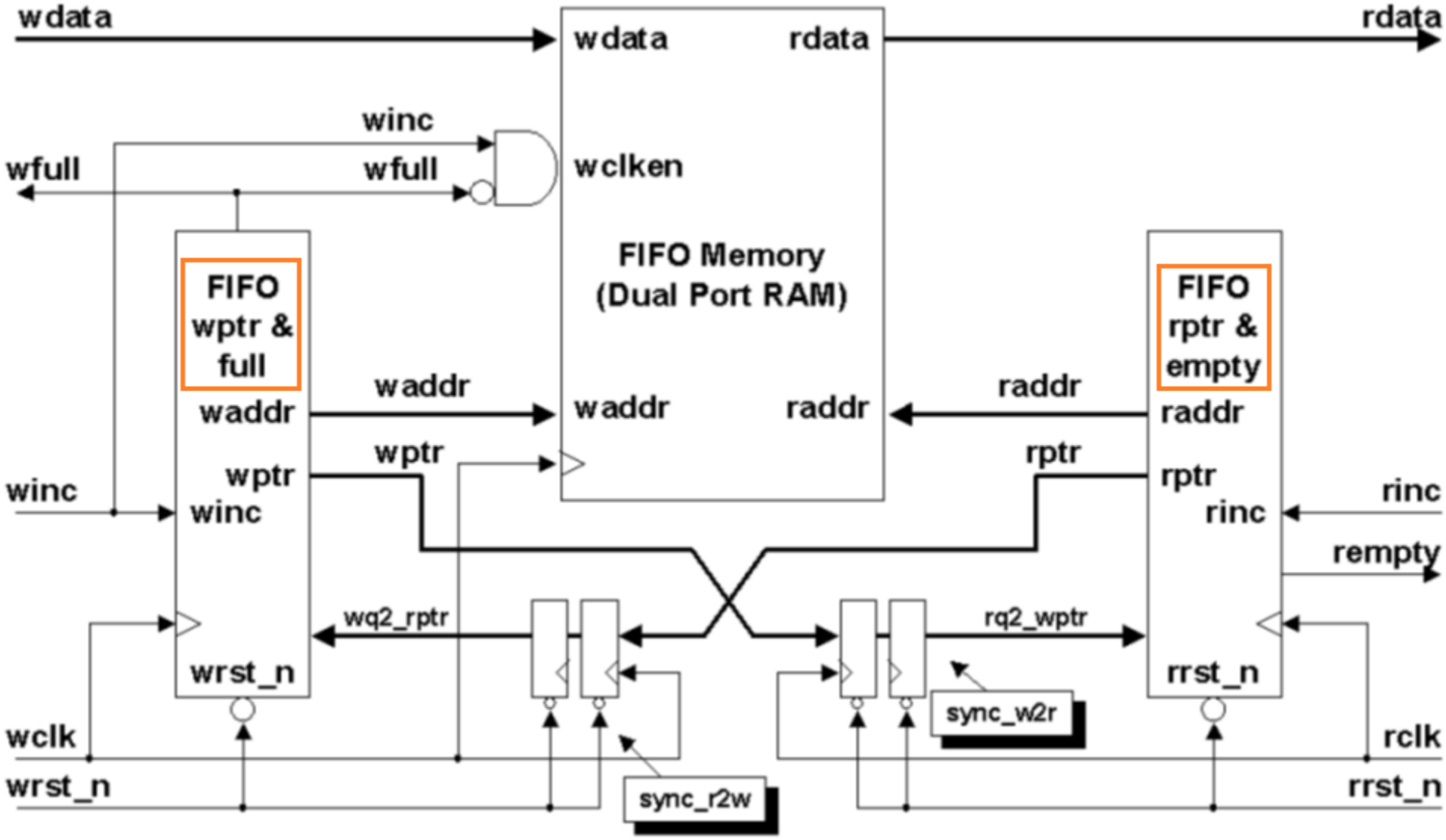
- **Asynchronous FIFO pointers**
- The write pointer always points to the next word to be written
- On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written
- Read pointer always points to the current FIFO word to be read
  - Receiver logic does not have to use two clock periods to read the data word
- On reset, both pointers are reset to zero
- The FIFO is empty when the read and write pointers are both equal.
- This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, having read the last word from the FIFO.
- A FIFO is full when the pointers are again equal, that is, when the write pointer has wrapped around and caught up to the read pointer.
- Problem! - The FIFO is either empty or full when the pointers are equal

Asynchronous FIFO Design

- Distinguish between full and empty by adding an extra bit to each pointer.
- When the write pointer increments past the final FIFO address, the write pointer will increment the unused MSB while setting the rest of the bits back to zero
  - The FIFO has wrapped and toggled the pointer MSB.
- The same is done with the read pointer.
- If the MSBs of the two pointers are different, it means that the write pointer has wrapped one more time that the read pointer.
- If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times



Asynchronous FIFO Design



Calculation of FIFO Depth

Case – 1 : $f_A > f_B$ with no idle cycles in both write and read.

Writing frequency = f_A = 80MHz.
Reading Frequency = f_B = 50MHz.
Burst Length = No. of data items to be transferred = 120.
There are no idle cycles in both reading and writing which means that, all the items in the burst will be written and read in consecutive clock cycles.

Sol :

- ✓ Time required to write one data item = $\frac{1}{80\ MHz}$ = 12.5 nSec.
- ✓ Time required to write all the data in the burst = 120 * 12.5 nSec. = 1500 nSec.
- ✓ Time required to read one data item = $\frac{1}{50\ MHz}$ = 20 nSec.
- ✓ So, for every 20 nSec, the module B is going to read one data in the burst.
- ✓ So, in a period of 1500 nSec, 120 no. of data items can be written.
- ✓ And the no. of data items can be read in a duration of 1500 nSec = $\left(\frac{1500\ nSec}{20\ nSec}\right) = 75$
- ✓ The remaining no. of bytes to be stored in the FIFO = 120 – 75 = 45.
- ✓ So, the FIFO which has to be in this scenario must be capable of storing 45 data items.

So, the minimum depth of the FIFO should be 45.

Calculation of FIFO Depth

Case – 2 : $f_A > f_B$ with one clk cycle delay between two successive reads and writes.

- Scenario is no way different from the previous scenario, because, there will be one clock cycle delay between two successive reads and writes.
- So, the approach is same as the earlier one

Calculation of FIFO Depth

Case – 3 : $f_A > f_B$ with idle cycles in both write and read.

Writing frequency = f_A = 80MHz.
Reading Frequency = f_B = 50MHz.
Burst Length = No. of data items to be transferred = 120.
No. of idle cycles between two successive writes is = 1.
No. of idle cycles between two successive reads is = 3.

Sol :

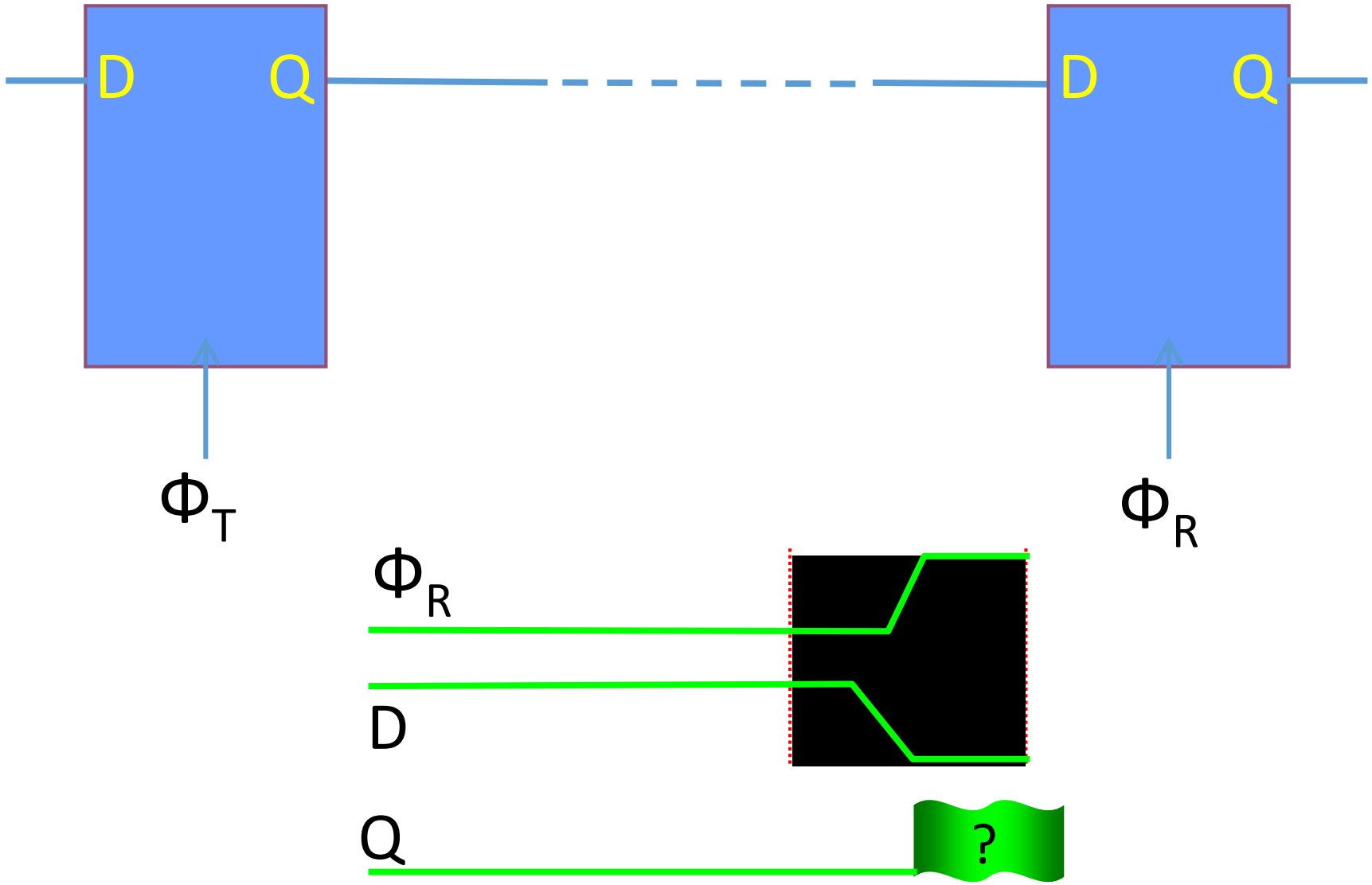
- ✓ The no. of idle cycles between two successive writes is 1 clock cycle. It means that, after writing one data, module A is waiting for one clock cycle, to initiate the next write. So, it can be understood that for every **two** clock cycles, one data is written.
- ✓ The no. of idle cycles between two successive reads is 3 clock cycles. It means that, after reading one data, module B is waiting for 3 clock cycles, to initiate the next read. So, it can be understood that for every **four** clock cycles, one data is read.
- ✓ Time required to write one data item = $2 * \frac{1}{80\ MHz}$ = 25 nSec.
- ✓ Time required to write all the data in the burst = 120 * 25 nSec. = 3000 nSec.
- ✓ Time required to read one data item = $4 * \frac{1}{50\ MHz}$ = 80 nSec.

Calculation of FIFO Depth

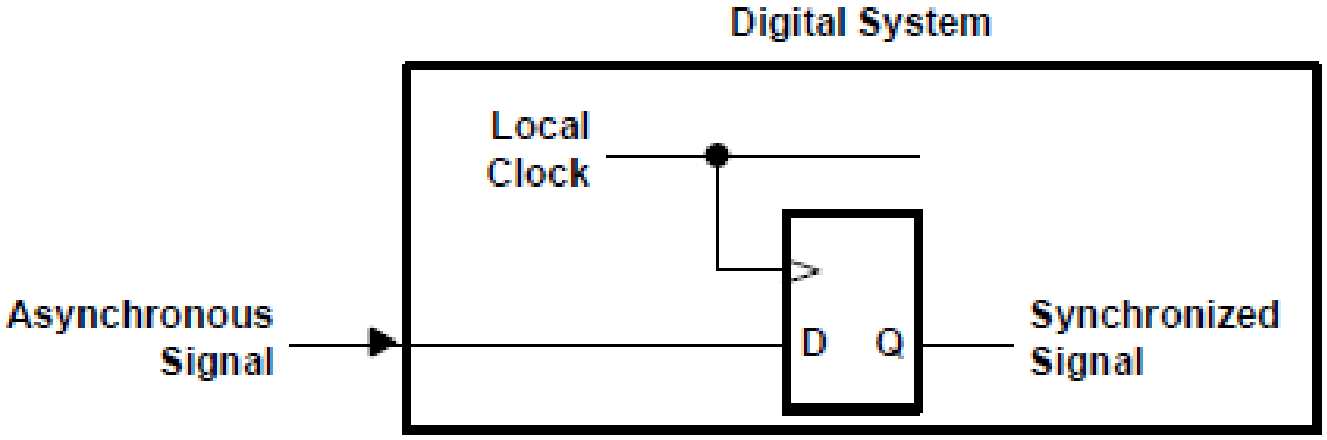
- ✓ So, for every 80 nSec, the module B is going to read one data in the burst.
- ✓ So, in a period of 3000 nSec, 120 no. of data items can be written.
- ✓ The no. of data items can be read in a period of 3000 nSec = $\left(\frac{3000\ nSec}{80\ nSec}\right) = 37.5 \simeq 37$
- ✓ The remaining no. of bytes to be stored in the FIFO = 120 – 37 = 83.
- ✓ So, the FIFO which has to be in this scenario must be capable of storing 83 data items.

So, the minimum depth of the FIFO should be 83.

The problem: metastability



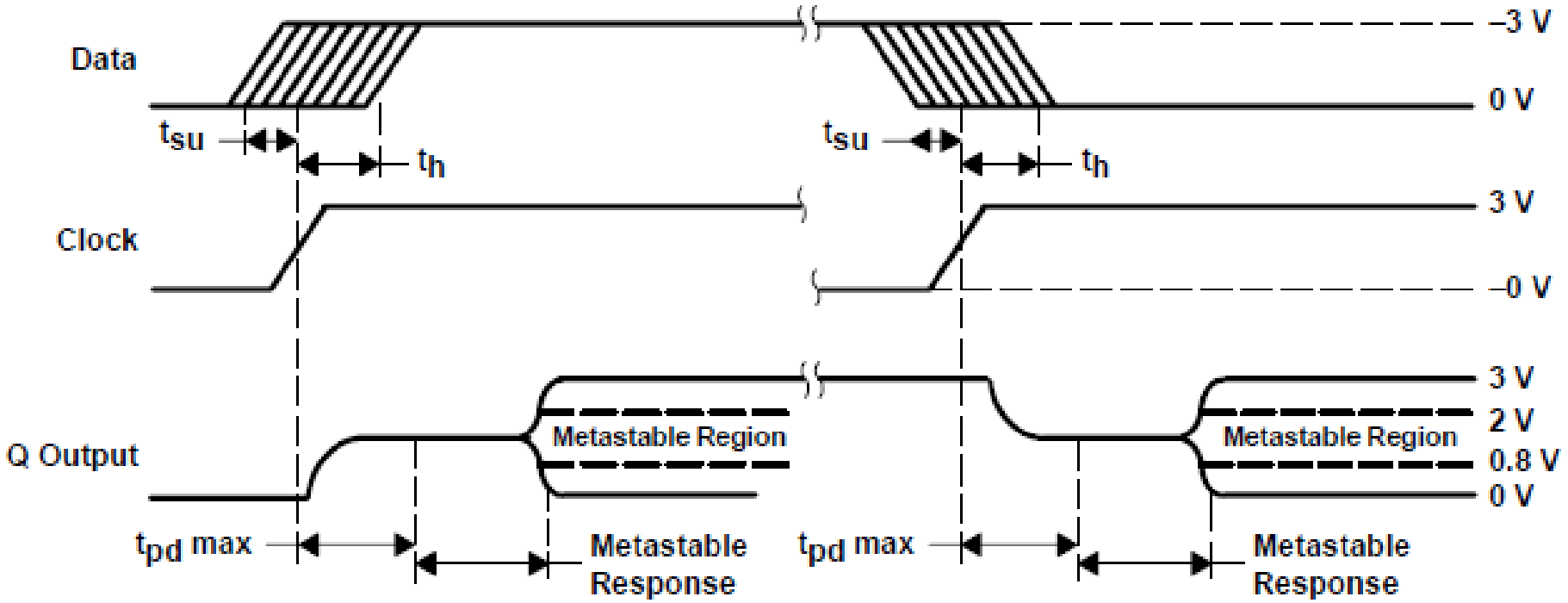
Metastability of Synchronizing Circuits



$$\text{MTBF} = \frac{1}{f_{in} \times f_{clk} \times t_d}$$

For $f_{clk} = 1$ MHz, f_{in} of 1 kHz, and $t_d = 30$ ps:

$$\text{MTBF} = \frac{1}{1\ \text{kHz} \times 1\ \text{MHz} \times 30\ \text{ps}} = 33.3\ \text{s}$$



Metastability of Synchronizing Circuits