

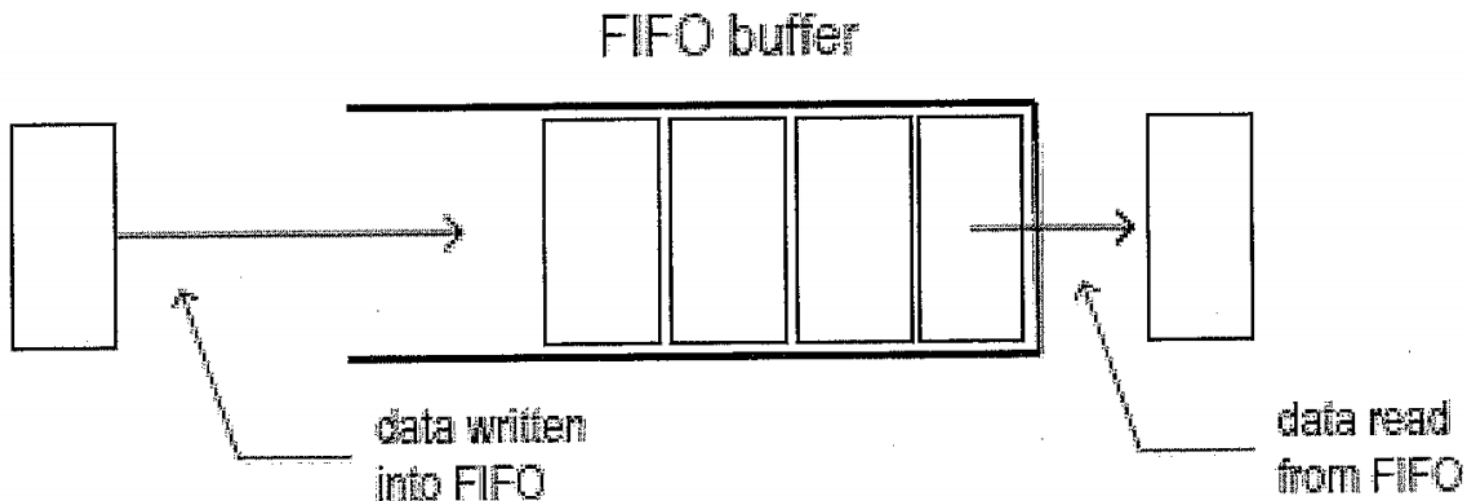
FIFO (**F**irst-**I**n **F**irst-**O**ut)

For SW Engineers

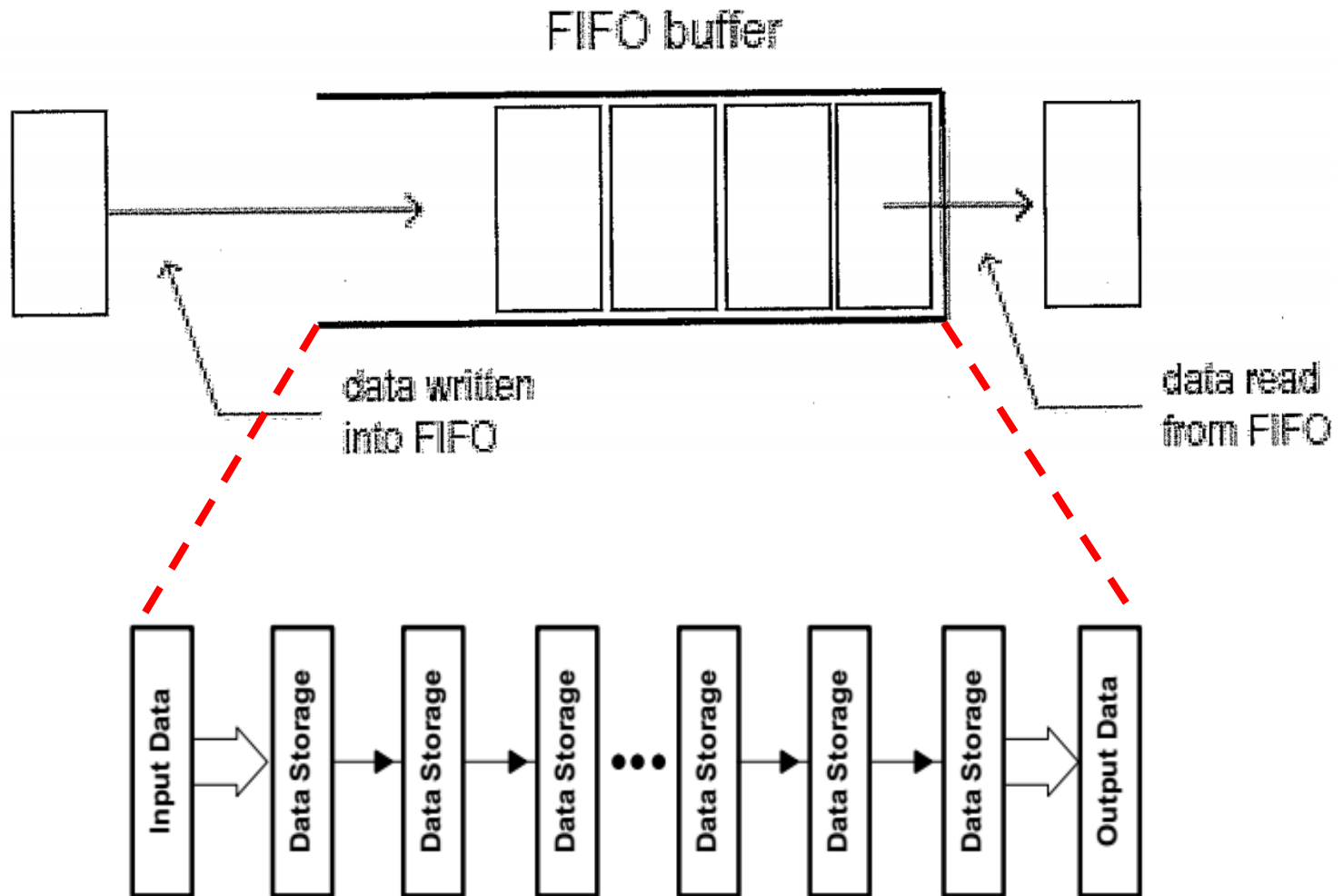
Tuan Nguyen-viet

FIFO Buffer

- FIFO memory is the buffer link of the system.
- Cache the continuous data stream (coming from Transmitter)
 - to prevent data loss
 - when entering and storing operations (at Receiver).
- A FIFO is a special type of buffer.
- The name FIFO stands for first in first out
 - and means that
 - the data written into the buffer first comes out of it first.
- Elastic storage b/w two sub-systems.



Data Flow



FIFO Types

There are three kinds of FIFO:

1. **Shift register** – FIFO with an invariable number of stored data words
 - and, thus,
 - the necessary **synchronism** between the read and the write operations
 - because a data word must be read *every time* one is written.
2. **Exclusive read/write FIFO** – FIFO with a variable number of stored data words
 - and, because of the *internal structure*,
 - the necessary **synchronism** between the read and the write operations
3. **Concurrent read/write FIFO** – FIFO with a variable number of stored data words
 - and possible **asynchronism** between the read and the write operation

FIFO Types (2)

- Two electronic sub-/systems always are connected to the input and output of a FIFO:
 - one that writes and
 - one that reads.
- 1. If certain timing conditions must be maintained between the writing and the reading systems,
 - we speak of **exclusive** read/write FIFOs
 - because the two systems must be **synchronized**.
- 2. But, if there are **no** timing restrictions in how the systems are driven, meaning that the writing system and the reading system can work **out of synchronism**,
 - the FIFO is called **concurrent** read/write.

Further information:

- (1) The first FIFO designs to appear on the market were **exclusive** read/write
 - because these were easier to implement.
- (2) **Nearly all** present FIFOs are **concurrent** read/write
 - because so many applications call for **concurrent** read/write versions.
 - **Concurrent** read/write FIFOs can be used in **synchronous systems** without any difficulty.

Exclusive Read/Write FIFOs

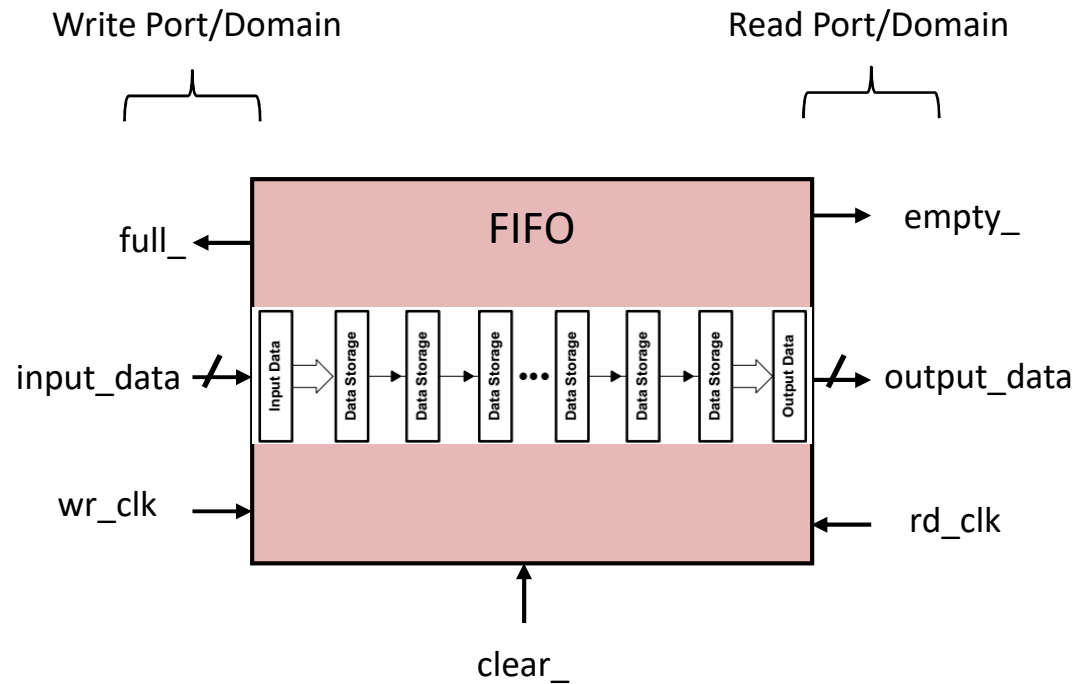
- In **exclusive** read/write FIFOs,
 - the writing of data is not independent of how the data are read.
- There are timing relationships between the write clock and the read **clock**.
 - For instance, overlapping of the read and the write **clocks**
 - could be prohibited.
- To permit use of such FIFOs between two systems that work **asynchronously** to one another,
 - an external circuit is required for synchronization.
 - But this synchronization circuit usually considerably reduces the **data rate**.

Concurrent Read/Write FIFOs

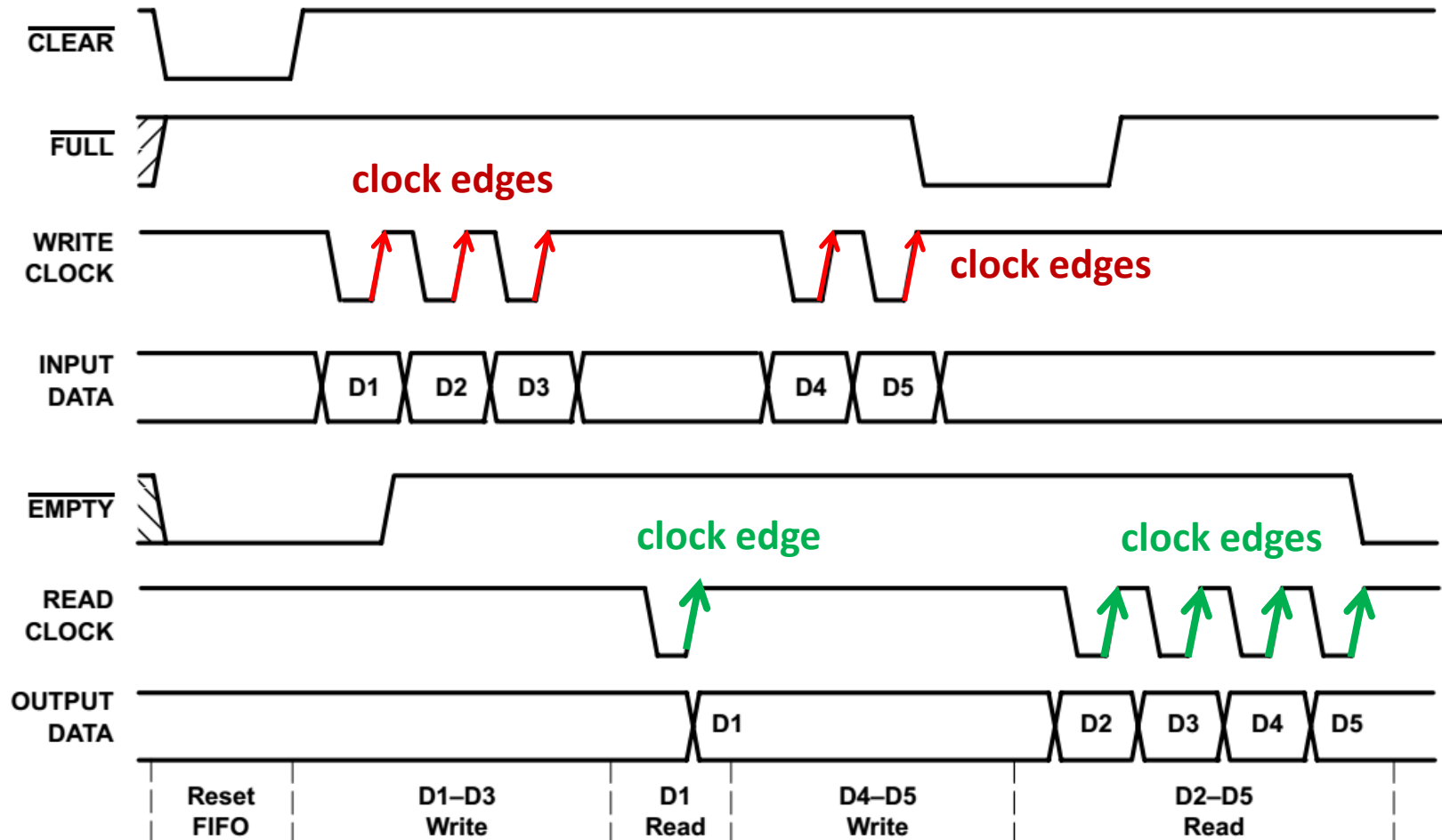
- In **concurrent** read/write FIFOs,
 - there is no dependence between the writing and reading of data.
- Simultaneous writing and reading are possible in overlapping fashion or successively.
 - This means that two systems with different frequencies can be connected to the FIFO.
- The designer need not worry about synchronizing the two systems because this is taken care of in the FIFO.
- **Concurrent** read/write FIFOs,
 - depending on the control signals for writing and reading,
 - fall into two groups:
 1. Synchronous FIFOs
 2. Asynchronous FIFOs

Asynchronous FIFOs

Asynchronous FIFOs



Timing Diagram for **Asynchronous** FIFO of **Length 4**



Timing Diagram for **Asynchronous** FIFO (2)

WRITE: When a data word is to be written into an **asynchronous** FIFO,

- it is first necessary to check
 - whether there is space available in the FIFO.
- This is done by querying the FULL status line (the signal **full_**).
 - If free space is indicated (**full_ = logic 1**),
 - the data word is applied to the data inputs and written into the FIFO
 - by a clock edge on the WRITE CLOCK input (**wr_clk**).

READ: The EMPTY status output (**empty_**) has to be queried before reading,

- because data can be read out
 - only if it is stored in the FIFO (**empty_ = logic 1**).
 - Then, a clock edge is applied to the READ CLOCK input (**rd_clk**),
 - causing the first word in the **data queue** to appear on the data output.

Timing Diagram for **Asynchronous** FIFO (3)

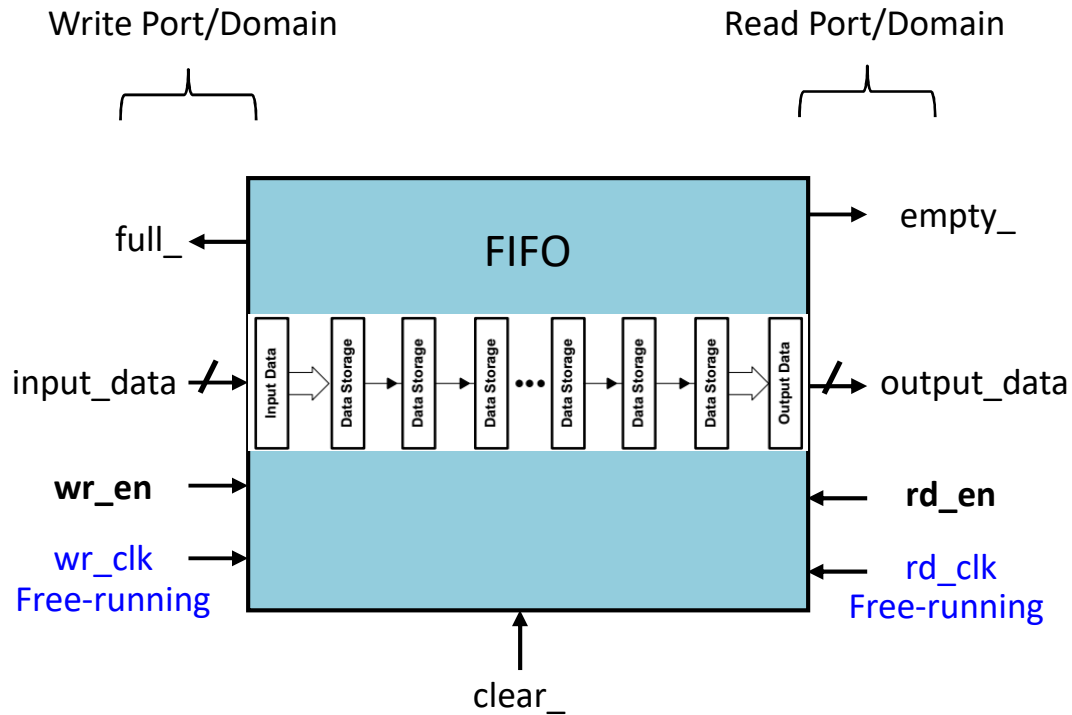
The resetting (the signal **clear_ = logic 1**) of the FIFO that is always necessary at the beginning.

Then, three data words are written in.

- The data words D1 through D3 appear *one after the other* on the INPUT DATA inputs
 - and **clock edges** are applied to WRITE CLOCK (**wr_clk**) for transfer of the data.
- Once the first data word has been written into the FIFO,
 - the EMPTY signal (**empty_**) changes from low level (**logic 0**) to high level (**logic 1**).
- Another two data words are written into the FIFO
 - before the first read cycle.
- The subsequent reading out of the first data word with the aid of a **clock edge** on READ CLOCK (**rd_clk**) does not alter the status signals.
- With the writing of another two data words, the FIFO is full (**full_ = logic 0**).
 - This is indicated by the FULL signal (**full_ = logic 0**).
- Finally, the four data words D2 through D5 remaining in the FIFO are read out.
- Thus, the FIFO is empty again (**empty_ = logic 0**),
 - so the EMPTY status line (**empty_**) shows this by low level (**logic 0**).

Synchronous FIFOs

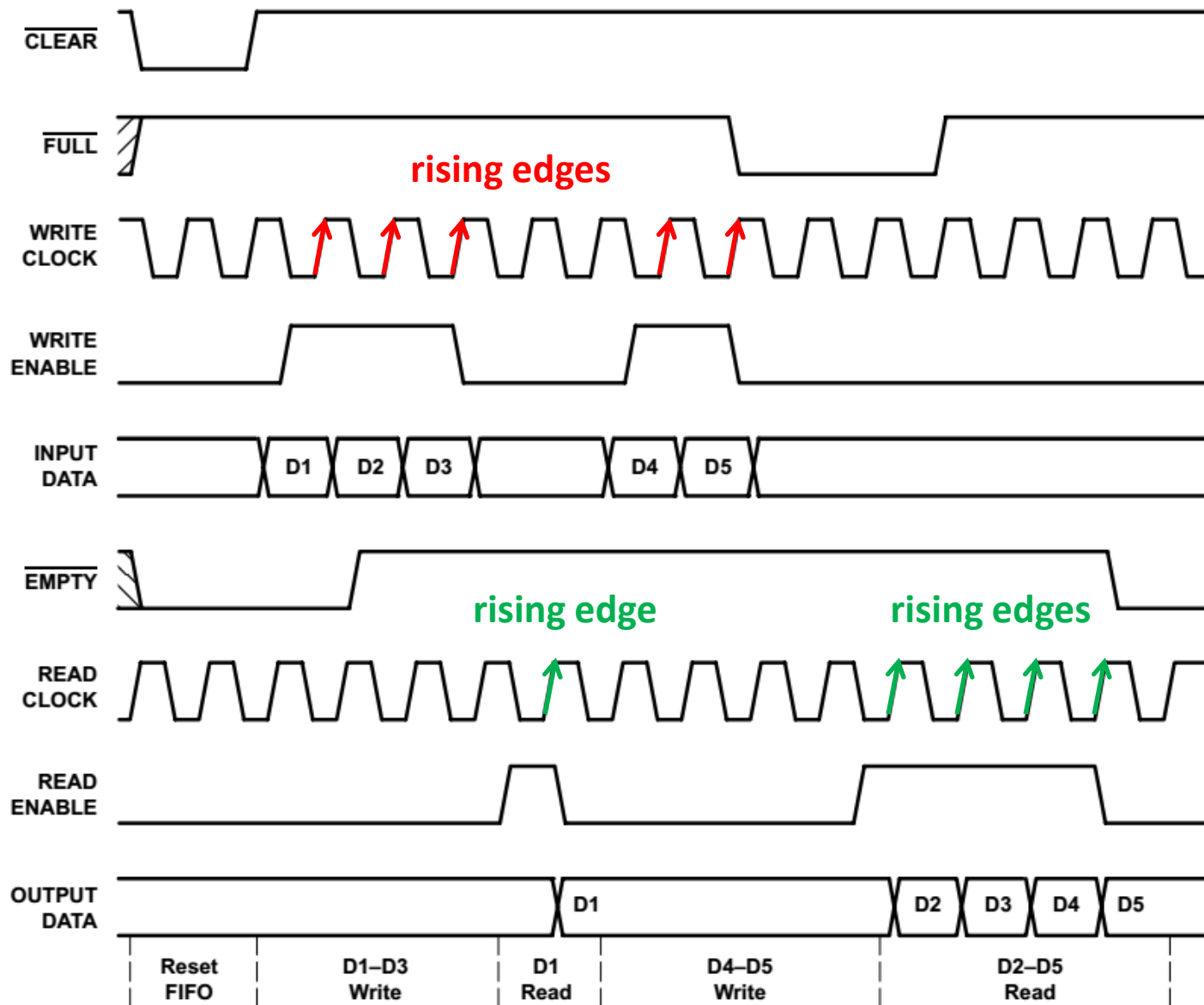
Synchronous FIFOs



Synchronous FIFOs

- **Synchronous** FIFOs are controlled based on methods of control proven in processor systems.
- Every digital processor system works synchronized with a **system-wide clock signal**.
 - This system timing continues to run even if no actions are being executed.
 - **Enable** signals (**_en**), also often called **chip-select** signals (**_cs**),
 - start the synchronous execution of write and read operations
 - in the various devices, such as memories and ports.
- Writing is controlled by the WRITE ENABLE input (**wr_en** = **1**) synchronous with WRITE CLOCK (**wr_clk**).
 - The FULL status line (**full_**) can be synchronized entirely with WRITE CLOCK (**wr_clk**) by the free-running clock.
- Data words are read out by a high level on the READ ENABLE input (**rd_en** = **1**) synchronous with READ CLOCK (**rd_clk**).
 - Here, too, the free-running clock permits 100 percent synchronization of the EMPTY signal (**empty_**) with READ CLOCK (**rd_clk**).

Timing Diagram for a Synchronous FIFO of Length 4



Timing Diagram for a **Synchronous FIFO (2)**

- WRITE CLOCK (**wr_clk**) and READ CLOCK (**rd_clk**) are free running.
- 1. The writing of new data into the FIFO is initialized by a low level on the WRITE ENABLE line (**wr_en = 1**).
 - The data are written into the FIFO with the next **rising edge** of WRITE CLOCK (**wr_clk**).
- 2. The READ ENABLE line (**rd_en = 1**) controls the reading out of data synchronous with READ CLOCK (**rd_clk**).
- All status lines within the FIFO can be synchronized by the two free-running-clock signals.
 - The FULL line (**full_**) only changes its level synchronously with WRITE CLOCK (**wr_clk**),
 - even if the change is produced by the reading of a data word.
 - Likewise, the EMPTY signal (**empty_**) is synchronized with READ CLOCK (**rd_clk**).
- A **synchronous** FIFO is the only **concurrent** read/write FIFO
 - in which the status signals are synchronized with the driving logic.

FIFO Architectures

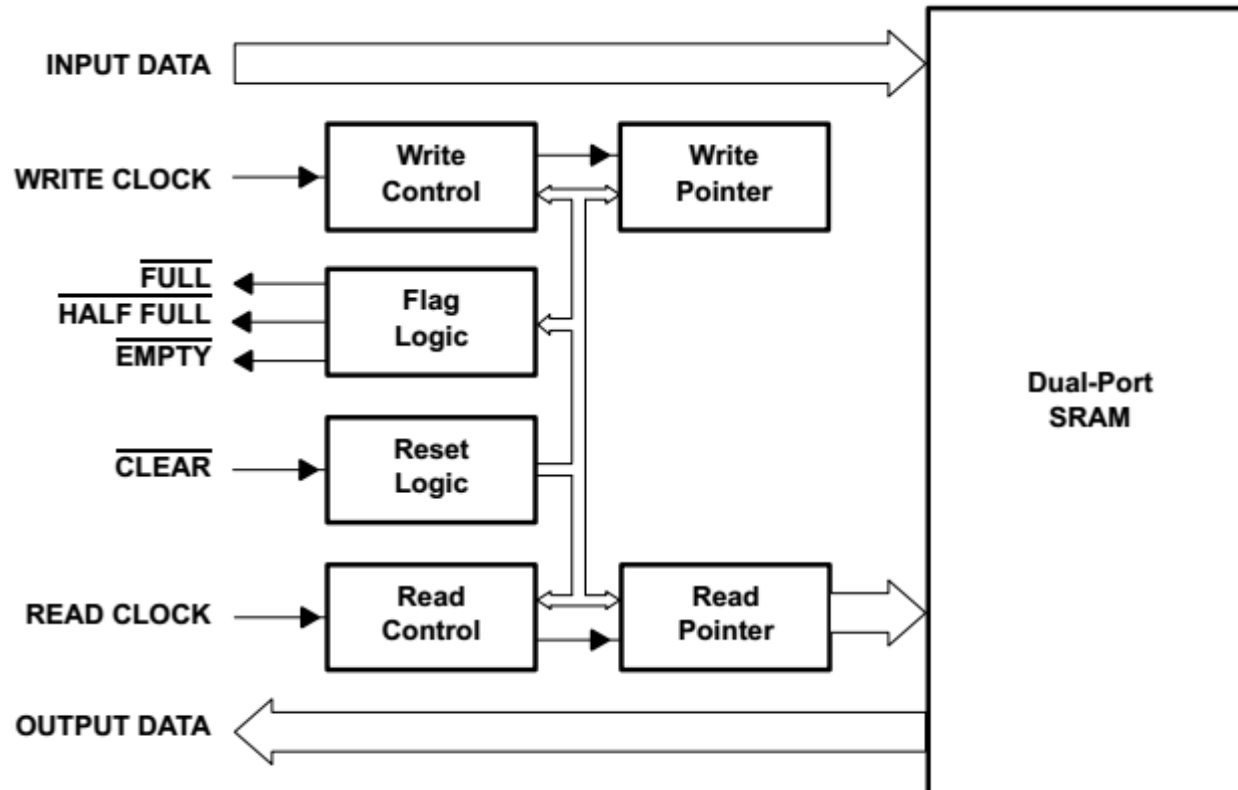
FIFO Architectures

- **Fall-Through FIFOs**
 - Fall-through FIFOs were the first FIFO generation.
 - The customers queuing at the checkout point of a supermarket could easily have been the model for this variant.
 - The first customer goes right up to the checkout point, while all others queue behind.
 - Once the first customer has paid and left the front of the queue, the other customers all move up one place.
- **FIFOs With Static Memory**
(see the next slides)

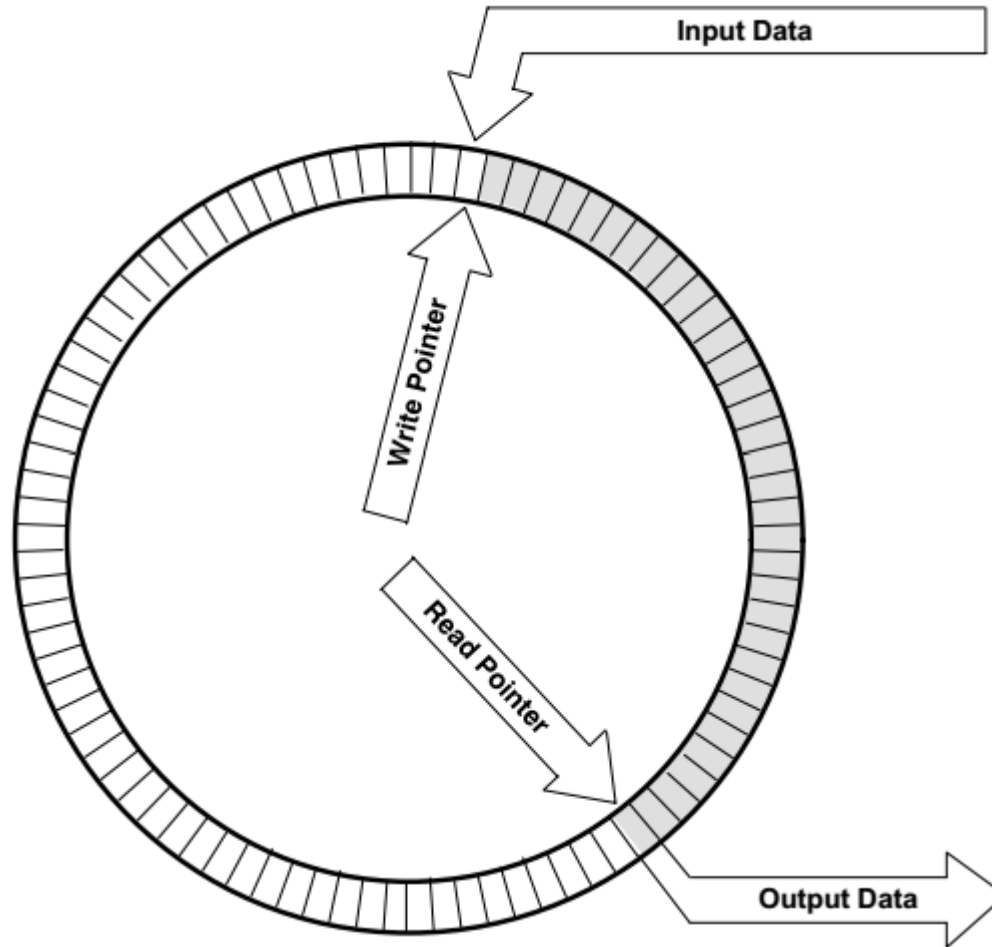
FIFO w/ Static Memory

- To counter the **disadvantage** of a **long** fall-through time in **long** FIFOs,
 - the architecture should no longer shift the data words through all memory locations.
- **The problem** is solved by a **circular memory** with **two pointers**.
 - In a circular FIFO concept, the memory address of the **incoming data** is in the write pointer.
 - The address of the first **data word** in the FIFO that is to be read out is in the read pointer.
- After reset, both pointers indicate the same memory location.
 - After each write operation,
 - the write pointer is set to the next memory location.
 - The reading of a **data word** sets the read pointer to the next **data word** that is to be read out.
- The read pointer constantly follows the write pointer.
 - When the read pointer reaches the write pointer, the FIFO is **empty**.
 - If the write pointer catches up with the read pointer, the FIFO is **full**.

FIFO w/ Static Memory (2): Block Diagram

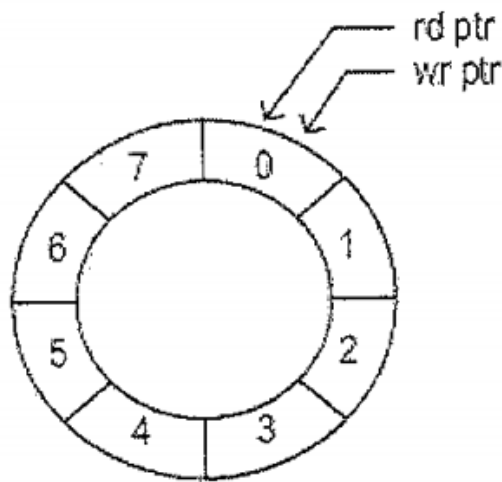


FIFO w/ Static Memory (3): Circular FIFO w/ 2 Pointers

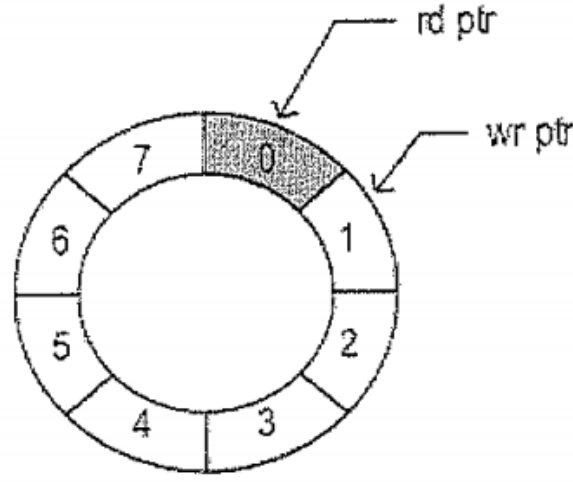


FIFO w/ Static Memory (3): Circular Queuing

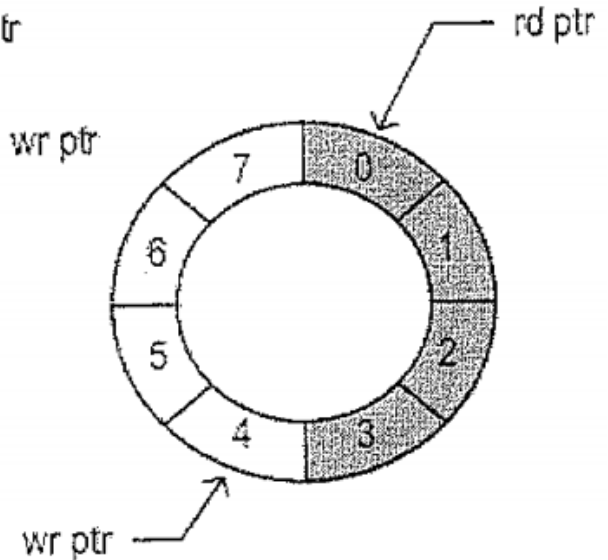
- Use 2 pointers and one memory/storage/queue
 - Write pointer: point to the empty slot before the head of the queue
 - Read pointer: point to the tail of the queue



(a). initial (empty)



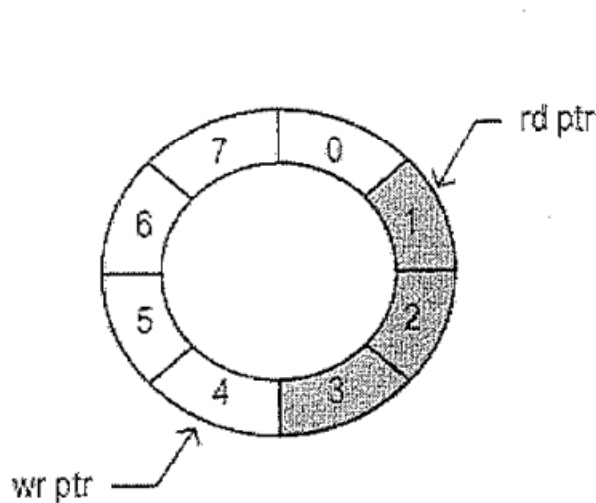
(b). after a write



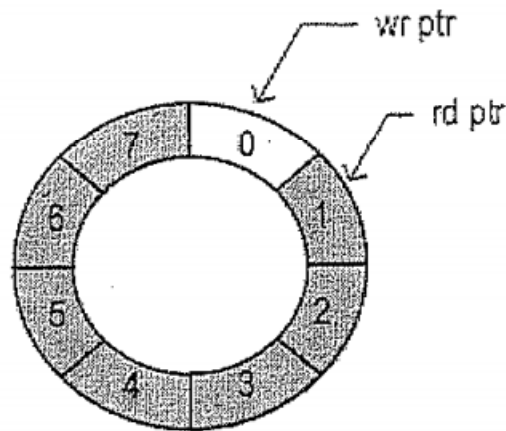
(c). 3 more writes

FIFO w/ Static Memory (4): Circular Queuing

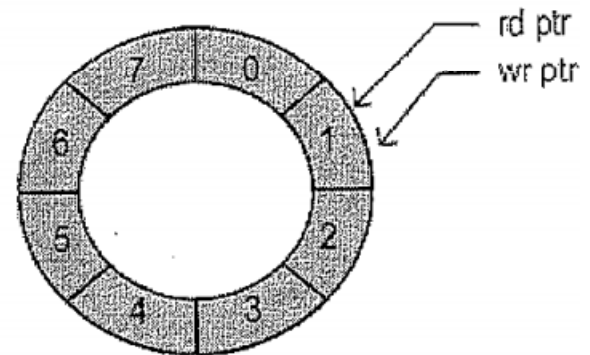
- Use 2 pointers and one memory/storage/queue
 - Write pointer: point to the empty slot before the head of the queue
 - Read pointer: point to the tail of the queue



(d). after a read



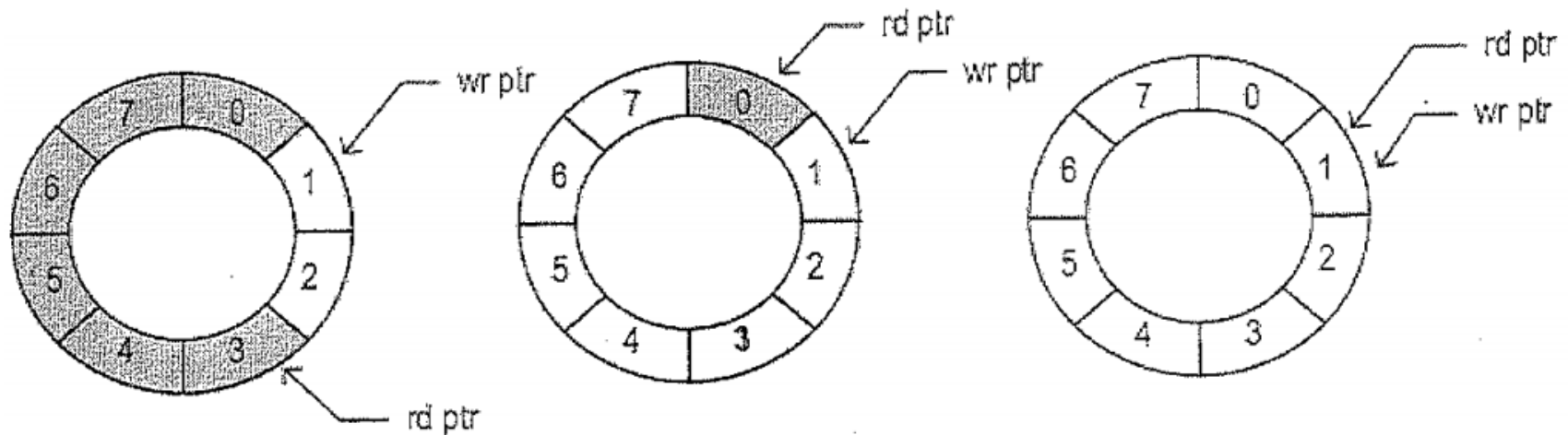
(e). 4 more writes



(f). 1 more write (full)

FIFO w/ Static Memory (5): Circular Queuing

- Use 2 pointers and one memory/storage/queue
 - Write pointer: point to the empty slot before the head of the queue
 - Read pointer: point to the tail of the queue



Thank You