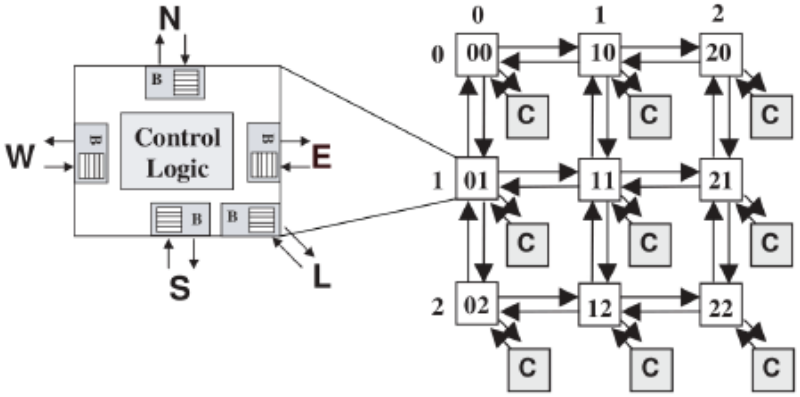
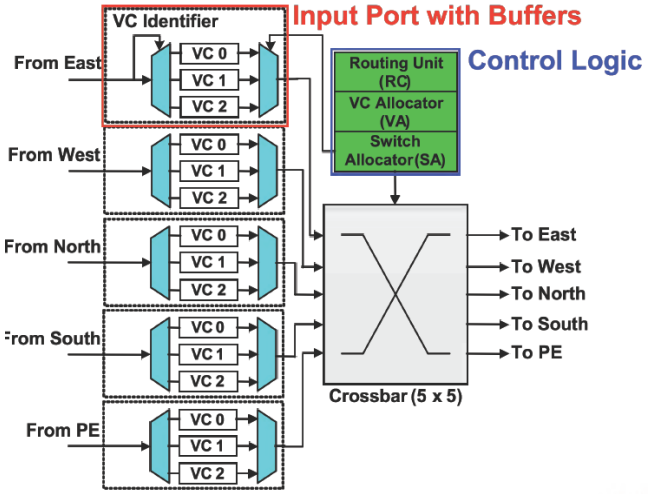
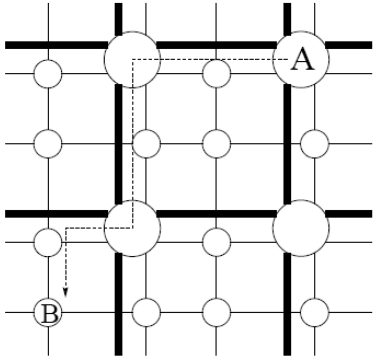
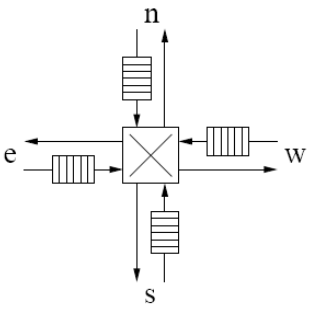
Synchronous FIFOs

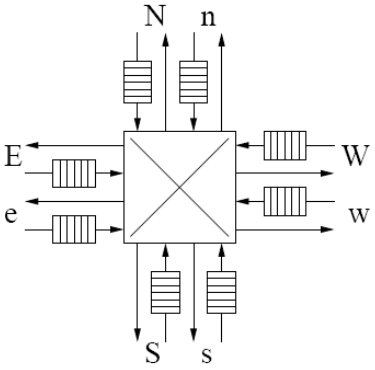
FIFO is Everywhere



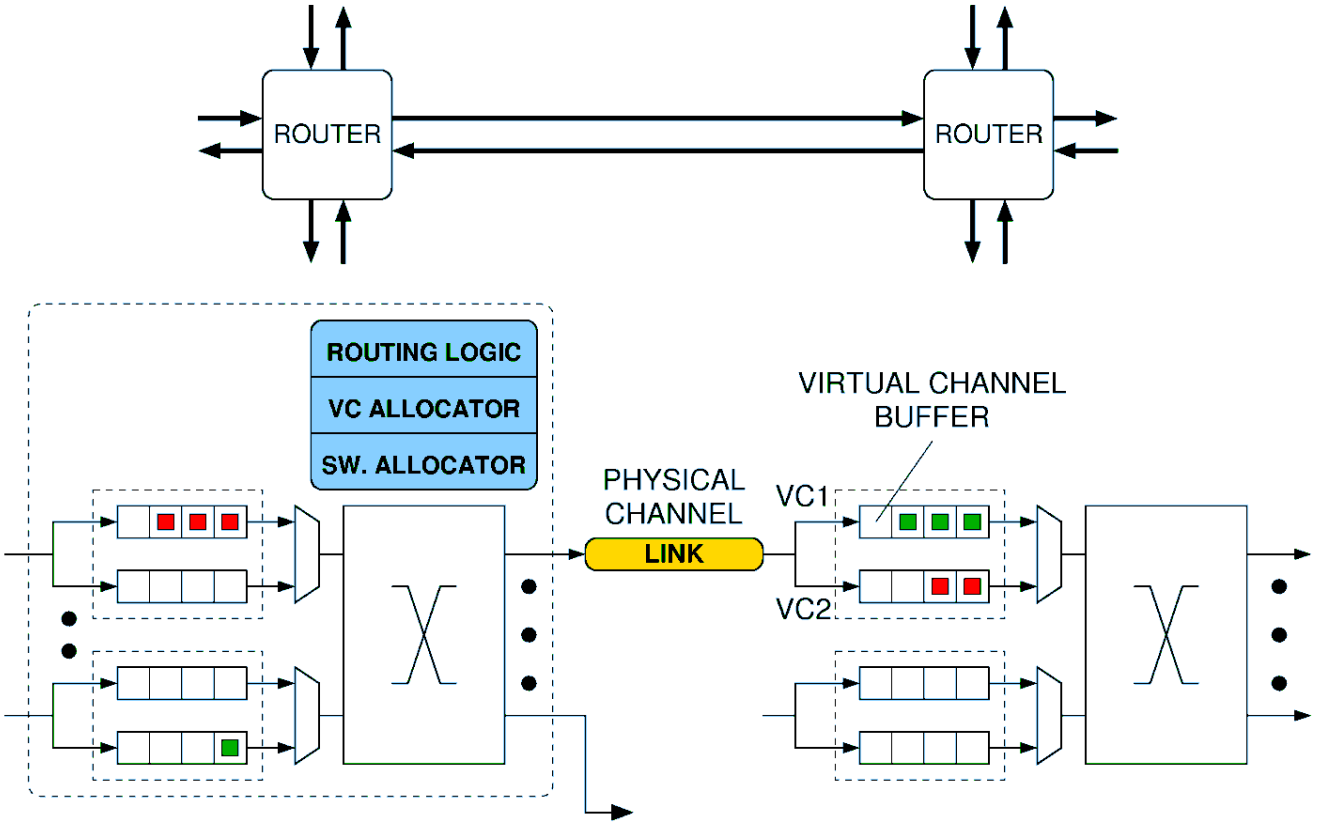




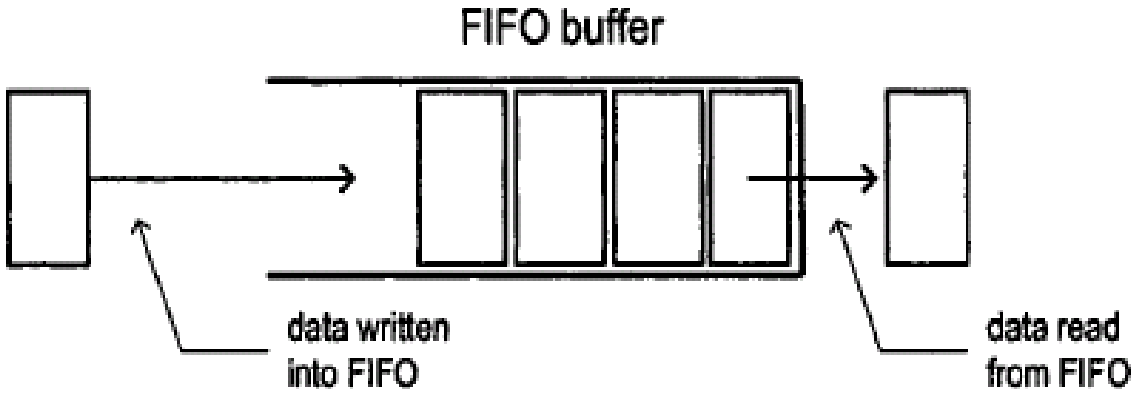




Virtual-Channel Flow Control

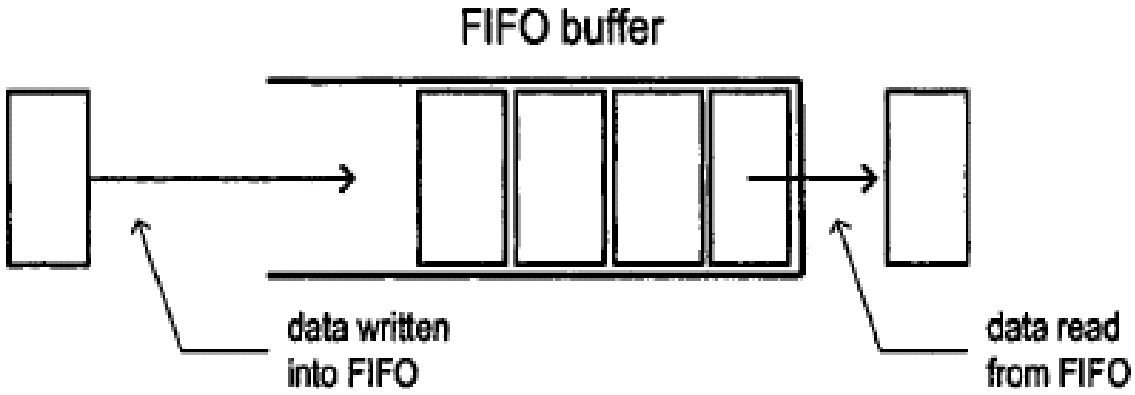


A Synchronous FIFO Buffer



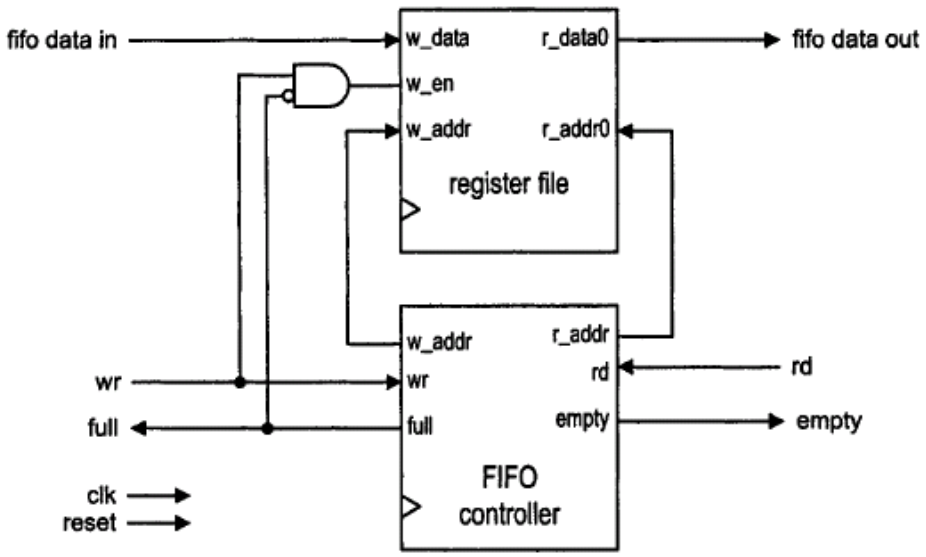
- A first-in-first-out (FIFO) buffer acts as "elastic" storage between two subsystems.
- One subsystem stores (i.e., writes) data into the buffer, and
  - the other subsystem retrieves (i.e., reads) data from the buffer and removes it from the buffer.

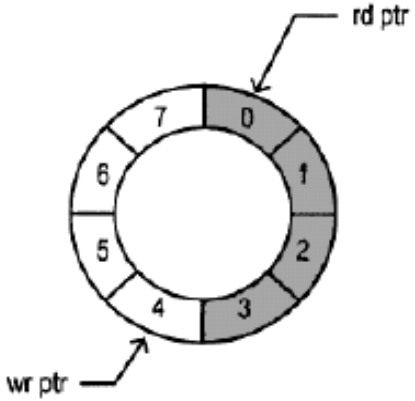
A Synchronous FIFO Buffer



- The order of data retrieval is same as the order of data being stored, and
  - thus the buffer is known as a first-in-first-out buffer.
- If two subsystems are synchronous (i.e., driven by the same clock), we need only one clock for the FIFO buffer and
  - it is known as a synchronous FIFO buffer

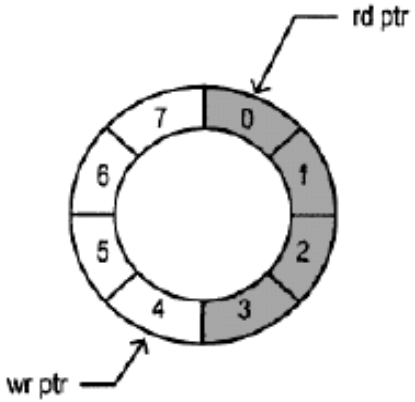
A Synchronous FIFO Buffer





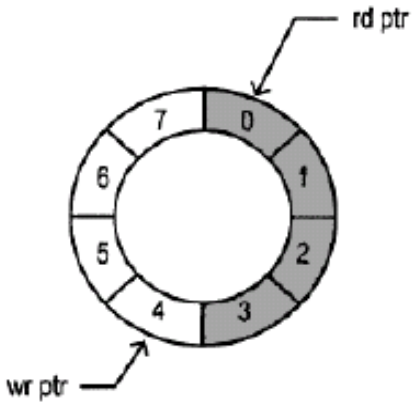
- The most common way to construct a FIFO buffer is to add a simple control circuit to a generic memory array, such as a register file or RAM.
- We can arrange the generic memory array as a circular queue and use two pointers to mark the beginning and end of the FIFO buffer.

Model for the Write Pointer



- The ***write pointer*** (labeled **wr_ptr),**
  - Points to the first empty slot in front of the buffer, if it is not full.
  - During a write operation,
    - the data is stored in this designed slot, and
    - the write pointer advances to the next slot (incremented by 1).

Model for the Read Pointer



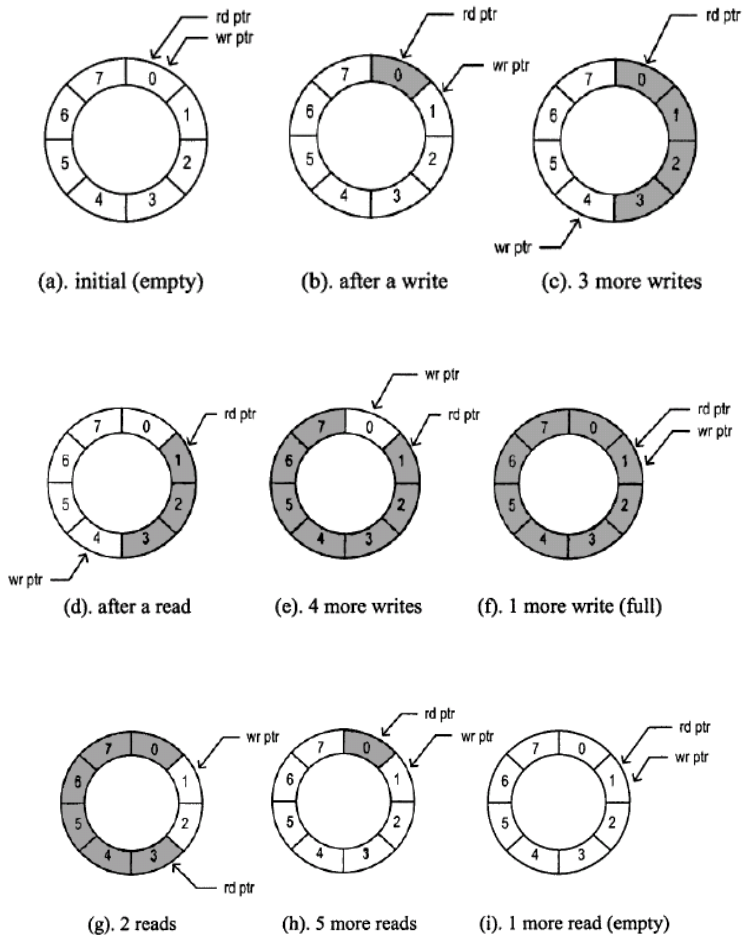
- The ***read pointer*** (labeled **rd_ptr),**
  - Points to the end of the buffer. If it is not empty, it points always to a slot containing the valid data.
  - During a read operation,
    - the data is retrieved and
    - the read pointer advances one slot, effectively releasing the slot for future write operations.

The Pointer Model

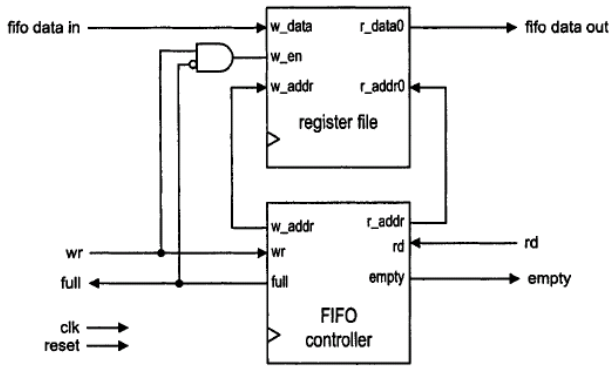
- Our Model:
  - ***write-and-increment***
  - ***read-and-increment***
- However, some FIFOs use
  - an *increment-and-write* and *increment-and-read* approach.

Write and Read Operations

- Initially, both the read and write pointers point to the 0 address.
  - Since the buffer is empty, no read operation is allowed at this time.
- After a write operation, the write pointer increments and the buffer contains one item in the 0 address.
- After a few more write operations, the write pointer continues to increase and the buffer expands accordingly.
- The read pointer advances in the same direction, and the previous slot is released.
- After several more write operations, the buffer is full, and no write operation is allowed.
- Several read operations are then performed, and the buffer eventually shrinks to 0.



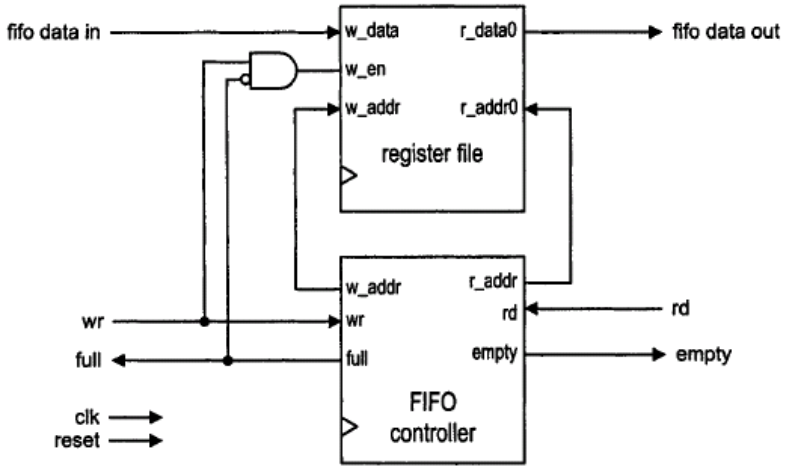
Write and Read Pointers



- At the rising edge of the clock, if the **wr** signal is asserted and **the buffer is not full**,
  - the corresponding input data will be sampled and stored into the buffer.
  - The output data from the FIFO is always available.

- The **rd** signal might better be interpreted as a "remove" signal.
- If **rd** is asserted at the rising edge and **the buffer is not empty**,
  - the FIFO's read pointer advances one position and makes the current slot available.

Overflow and Underflow



- During FIFO operation,
  - an **overflow** occurs when the external system attempts to write new data **when the FIFO is full**
  - an **underflow** occurs when the external system attempts ***to*** read (i.e., remove) a slot **when the FIFO is empty**.

Full and Empty **Status** Signals

- To ensure correct operation,
  - a FIFO buffer must include the **full** and **empty** status signals for the two special conditions.
- In a properly designed system,
  - the external systems should check the status signals before attempting to access the **FIFO**.

Full and Empty **Status** Signals

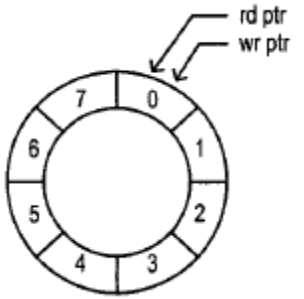
- The major components of a FIFO control circuit are **two counters**,
  - whose outputs function as write and read pointers, respectively.
- During regular operation,
  - the write **counter** advances one position when the **wr** signal is asserted at the rising edge of the clock, and
  - the read **counter** advances one position when the **re** signal is asserted.

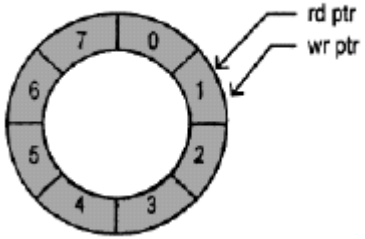
Full and Empty **Status**

- We normally prefer to add some safety precautions to ensure that data will not be written into a full buffer or removed from an empty buffer.
- Under these conditions, the counters will retain the previous values.

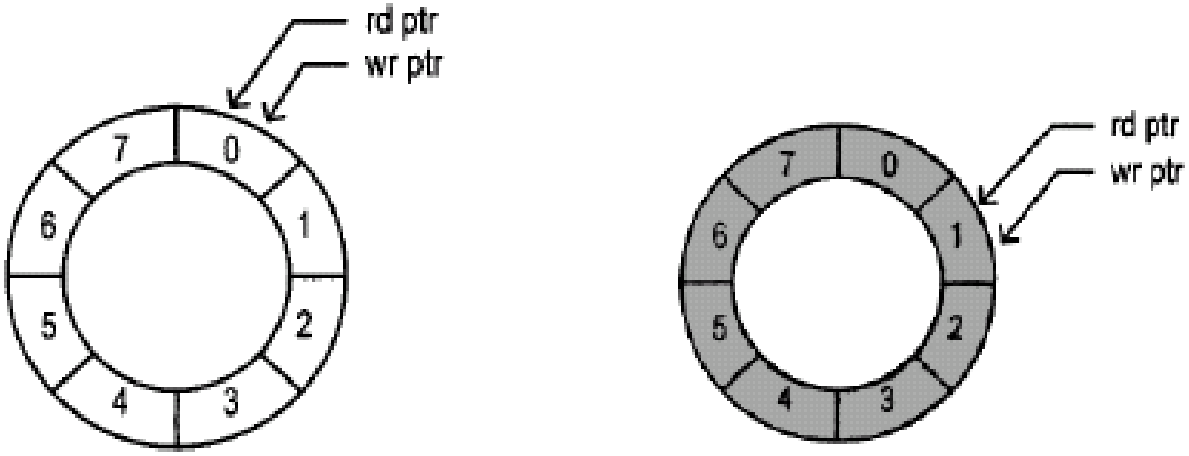
Full and Empty **Status**

- The difficult part of the control circuit is the handling of two special conditions in which the **FIFO** buffer is empty or full.
- When the FIFO buffer is empty,
  - the read pointer is the same as the write pointer.
- Unfortunately, this is also the case when the FIFO buffer is full.
  - Thus, we cannot just use read and write pointers to determine full and empty conditions.





Full and Empty **Status**

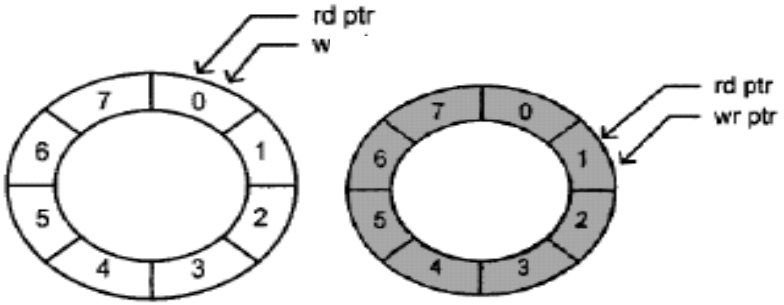


- There are several schemes to generate the status signals, and all of them involve additional circuitry and FFs.

FIFO Control with Binary Counters

- The first method is
  - to use the binary counters for the read and write pointers and increase their sizes by 1 bit.
  - to determine the full or empty condition by comparing the MSBs of the two pointers.

FIFO Control with Binary Counters



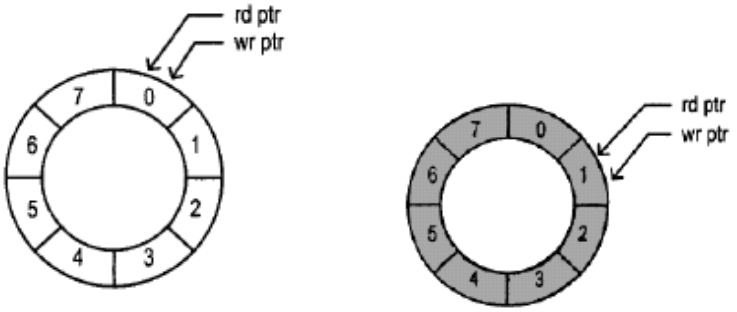
- Consider a **FIFO** with 3-bit address (i.e., 2^3 words).
- Two 4-bit counters will be used for the read and write pointers.

Representative sequence of FIFO operations

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Control with Binary Counters

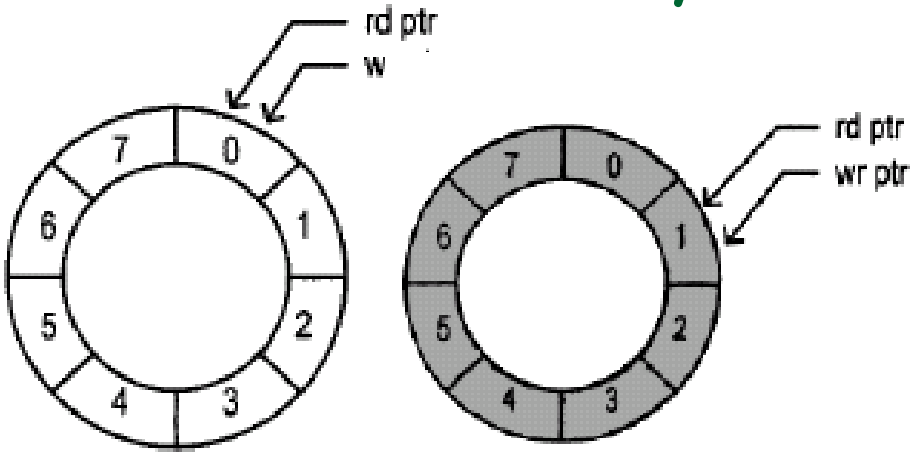
- The three LSBs of the read and write pointers are used
  - as addresses to **access the FIFO** and wrap around after eight increments.
  - tracing the addresses of the FIFO.
- The three LSBs are equal when the FIFO is empty or full.



Representative sequence of FIFO operations

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Control with Binary Counters

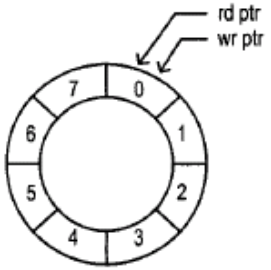


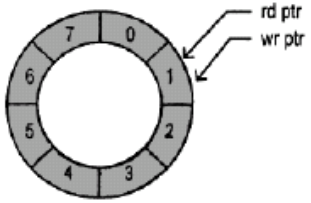
- The entire 4 bits remember the history of the advancement.
- The MSBs of the read and write pointers can be used to distinguish
  - the first encounter
  - from the second encounter

Representative sequence of FIFO operations

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Control with Binary Counters





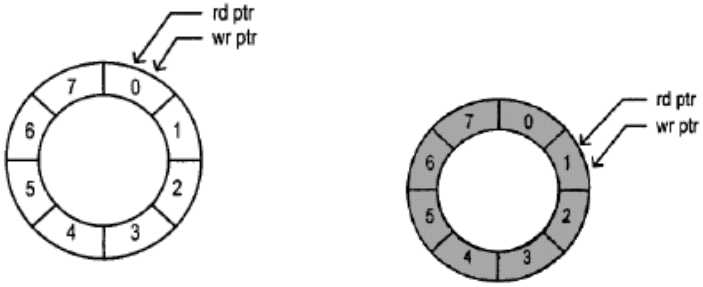
- The MSBs of the read and write pointers are
  - the same when the FIFO is empty.
  - the opposite values when the FIFO is full.

- After eight write operations, the MSB of the write pointers **flips** and becomes **the opposite of** the MSB of the read pointer.

Representative sequence of FIFO operations

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Control with Binary Counters



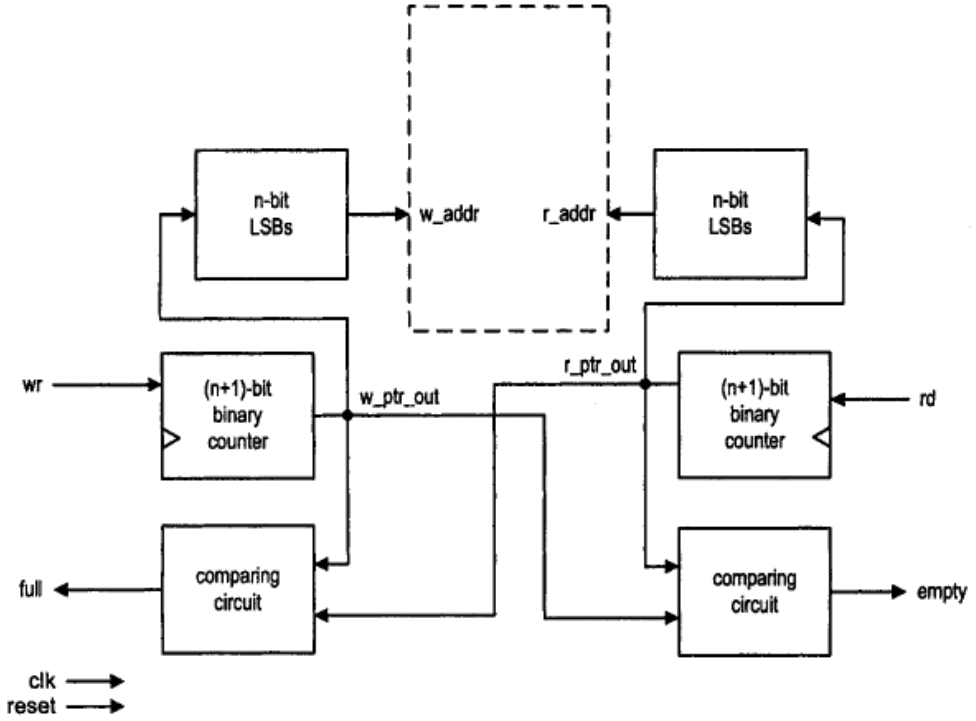
- After eight read operations,
  - the MSB of the read pointer flips and becomes identical to the MSB of the write pointer,
  - the FIFO is empty again.

Representative sequence of FIFO operations

Write pointer	Read pointer	Operation	Status
0 000	0 000	initialization	empty
0 111	0 000	after 7 writes	
1 000	0 000	after 1 write	full
1 000	0 100	after 4 reads	
1 100	0 100	after 4 writes	full
1 100	1 011	after 7 reads	
1 100	1 100	after 1 read	empty
0 011	1 100	after 7 writes	
0 100	1 100	after 1 write	full
0 100	0 100	after 8 reads	empty

FIFO Control with Binary Counters

- A detailed block diagram of this scheme:



FIFO Control with Status FFs

- An alternative design
  - to keep track the state of the empty and full conditions

- This scheme
  - does not require augmented counters
  - but needs two extra **FFs** to record the empty and full statuses.

FIFO Control with Status FFs

- During system initialization,
  - the full status FF is set to '0' and
  - the empty status FF is set to '1'.

- After initialization,
  - the **wr** and **rd** signals are examined at the rising edge of the clock
  - the pointers and the **FFs** are modified according to the rules.

Operations of Status FFs

- **wr** and **rd** are "00":
  - Since no operation is specified, pointers and FFs remain in the previous state.

- **wr** and **rd** are "11":
  - Write and read operations are performed simultaneously.
  - Since the net size of the buffer remains the same,
    - the empty and full conditions will not change.
  - Both pointers advance one position.

Operations of Status FFs

- **wr** and **rd** are "10": only a write operation is performed.

- Make sure that the buffer is not full.
  - If that is the case, the write pointer advances one position and the empty status FF should be deasserted.
- The advancement may make the buffer full.
  - This condition happens if the next value of the write pointer is equal to the current value of the read pointer
  - If this condition is true, the full status FF will be set to '1' accordingly.

Operations of Status FFs

- **wr** and **rd** are "01": only a read operation is performed.

- Make sure that the buffer is not empty.
  - If that is the case, the read pointer advances one position and the full status FF should be deasserted.
- The advancement may make the buffer empty.
  - This condition happens if the next value of the read pointer is equal to the current value of the write pointer.
  - If this condition is true, the empty status **FF** will be set to '1' accordingly.

FIFO Control with Non-binary Counters

- For the previous two **FIFO** control circuit implementations,
  - the two incrementors used in the binary counters consume the most hardware resources.
- If we examine operation of the read and write pointers closely,
  - there is no need to access the register in binary sequence.
- *Any order of access is fine as long as the two pointers circulate through the identical sequence.*

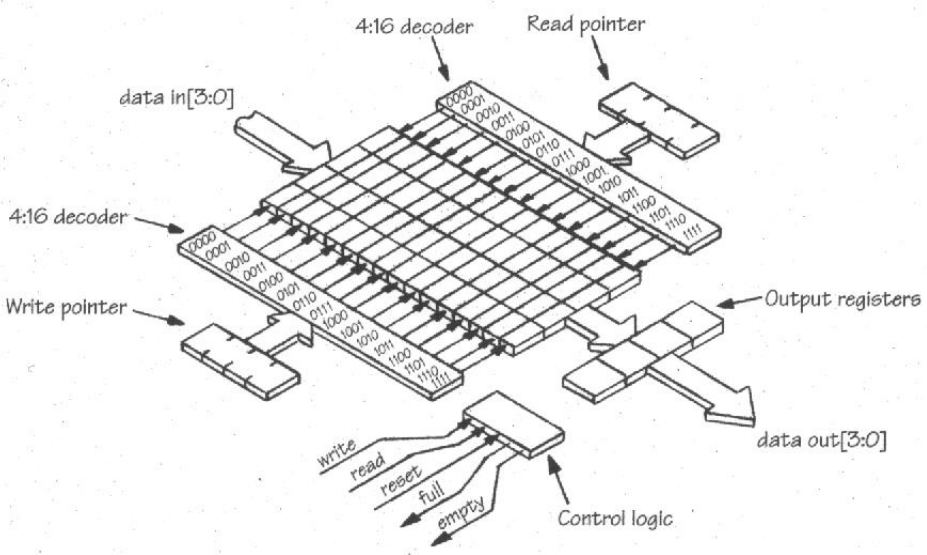
FIFO Control with Non-binary Counters

- In the first scheme, we enlarge the binary counter and use the extra MSB to determine the status.
- This approach is based on the special property of the binary counting sequence and
  - cannot easily be modified for other types of counters.

FIFO Control with Non-binary Counters

- In the second scheme,
  - the status signal relies on **the** successive value of the counter, and thus
  - this scheme can be applied to any type of counter.

FIFO Control with Non-binary Counters



- However, there is no intrinsic advantage to a **binary** sequence for this particular application.
  - Why???
- The sequence generated by a LFSR would serve equally well !

FIFO Control with Non-binary Counters

- Because of its simple next-state logic, LFSR is the best choice.
  - It replaces the incrementor of a binary counter with a few xor cells and
  - can significantly improve circuit size and performance, especially for a large FIFO address space.

FIFO Control with Non-binary Counters

- The combinational logic for
  - the 4-bit LFSR consists of a single two-input XOR gate,
  - the 4-bit binary counter requires a number of AND and OR gates.
- Additionally,
  - the LFSR's feedback only passes through a single level of logic,
  - the binary counter's feedback passes through multiple levels of logic.

FIFO Control with Non-binary Counters

- This means that the new data value is available sooner for the LFSR
  - which can therefore be clocked at a higher frequency.
- These differentiations become even more pronounced for FIFOs with more words requiring pointers with more bits.
  - Thus, LFSR'S are the obvious choice for the discerning designer of FIFOs.

FIFO Controller with a 4-bit address

- In the original code, the following two statements generate the successive values:
  - w_ptr_succ <= w_ptr_reg + 1;
  - r_ptr_succ <= r_ptr_reg + 1;
- replaced by the next-state logic of a 4-bit LFSR:
  - w_ptr_succ <=(w_ptr_reg (1) xor w_ptr_reg (0)) & w_ptr_reg (3 downto 1);
  - r_ptr_succ <=(r_ptr_rag (1) xor r_ptr_reg (0)) & r_ptr_reg (3 downto 1) ;
- revise the asynchronous reset portion of the code to initialize the counters for a non-zero value.

FIFOs

- Our Model:
  - ***write-and-increment***
  - ***read-and-increment***
- However, some FIFOs use
  - an *increment-and-write* and *increment-and-read* approach.

Accessing the Previous Values

- When using LFSR, it is required to make use of a register's previous value.
- For example, in certain FIFO implementations,
  - the full condition is detected when the **write** pointer is pointing to the location preceding the location pointed to by the **read** pointer.
  - This implies that a comparator must be used to compare the **current** value in the **write** pointer with the **previous** value in the **read** pointer.

Accessing the Previous Values

- Similarly, the **empty** condition may be detected when the **read** pointer is pointing to the location preceding the location pointed to by the **write** pointer.
- This implies that a second comparator must be used to compare the current value in the **read** pointer with the previous value in the **write** pointer.

FIFO Depth?

- Suppose that
  - 200 bytes burst is to be written at 100 MHZ, 1 byte per write clock
  - Read clock is 50 MHZ, 1 byte read per read clock.
  - What is the FIFO depth?
- Time taken to write 1 byte = (1/100) = 10 ns
  - Time taken to write 200 Bytes = 10 * 200 = 2000 ns
- Time required to read 1 byte = (1/50) = 20 ns
  - In 2000 ns, date bytes read = (2000/20) = 100 Bytes
- FIFO Depth = 200 - 100 = 100 Bytes.

References

- RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability. Pong P. Chu.