Introduction

In every item of digital equipment there is exchange of data between printed circuit boards (PCBs). Intermediate storage or buffering always is necessary when data arrive at the receiving PCB at a high rate or in batches, but are processed slowly or irregularly. Buffers of this kind also can be observed in everyday life (for example, a queue of customers at the checkout point in a supermarket or cars backed up at traffic lights). The checkout point in a supermarket works slowly and constantly, while the number of customers coming to it is very irregular. If many customers want to pay at the same time, a queue forms, which works by the principle of first come, first served. The backup at traffic lights is caused by the sporadic arrival of the cars, the traffic lights allowing them to pass through only in batches.

In electronic systems, buffers of this kind also are advisable for interfaces between components that work at different speeds or irregularly. Otherwise, the slowest component determines the operating speed of all other components involved in data transfer. In a compact-disk player, for instance, the speed of rotation of the disk determines the data rate. To make the reproduced sound fluctuations independent of the speed, the data rate of the A/D converter is controlled by a quartz crystal. The different data rates are compensated by buffering. In this way, the sound fluctuations are largely independent of the speed at which disks rotate.

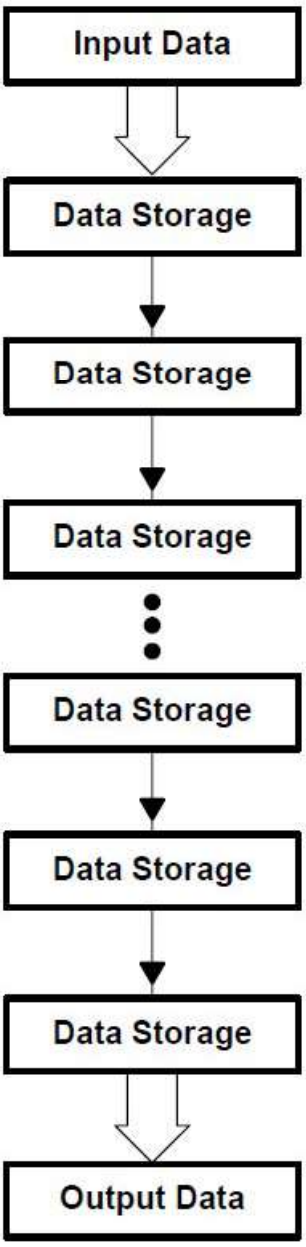
A FIFO is a special type of buffer. The name FIFO stands for first in first out and means that the data written into the buffer first comes out of it first. There are other kinds of buffers like the LIFO (last in first out), often called a stack memory, and the shared memory. The choice of a buffer architecture depends on the application to be solved. FIFOs can be implemented with software or hardware. The choice between a software and a hardware solution depends on the application and the features desired. When requirements change, a software FIFO easily can be adapted to them by modifying its program, while a hardware FIFO may demand a new board layout. Software is more flexible than hardware. The advantage of the hardware FIFOs shows in their speed.

>FIFO Types :-

Every memory in which the data word that is written in first also comes out first when the memory is read is a first-in first-out memory. Figure 1 illustrates the data flow in a FIFO.

There are three kinds of FIFO:

- 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.
- 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.
- Concurrent read/write FIFO – FIFO with a variable number of stored data words and possible asynchronism between the read and the write operation.



First-In First-Out Data Flow

The shift register is not usually referred to as a FIFO, although it is first-in first-out in nature. Consequently, this application report focuses exclusively on FIFOs that handle variable-length data.

Two electronic systems always are connected to the input and output of a FIFO: one that writes and one that reads. 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. 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.

The first FIFO designs to appear on the market were exclusive read/write because these were easier to implement. 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:

- Synchronous FIFOs
- Asynchronous FIFOs

Synchronous FIFOs are the ideal choice for high-performance systems due to high operating speed. Synchronous FIFOs also offer many other advantages that improve system performance and reduce complexity. These include status flags: synchronous flags, half-full, programmable almost-empty and almost-full flags. These FIFOs also include features such as, width expansion, depth expansion, and retransmit. Synchronous FIFOs are easier to use at high speeds because they use free-running clocks to time internal operations whereas asynchronous FIFOs require read and write pulses to be generated without an external clock reference.

Synchronous FIFO Architecture

The basic building blocks of a synchronous FIFO are: memory array, flag logic, and expansion logic. Figure 1 shows the logic block diagram of a synchronous FIFO. The memory array is built from dual-port memory cells. These cells allow simultaneous access between the write port and the read port. This simultaneous access gives the FIFO its inherent synchronization property. There are no timing or phase restrictions between accesses of the two ports. This means that while one port writes to the memory at one rate, the other port can read at another rate, independent of one another. This also enables optimization of the speed at which data is written to and read from the memory array. Cypress offers the synchronous FIFO CY7C42x5 in x9 & CY7C42x5 in x18 bit width. Both provide a high speed of 66 MHz and 100 MHz operation respectively.

Data is steered into and out of the memory array by two pointers, a read address pointer and write address pointer. After each operation, the respective pointer is incremented to allow access to the next address sequentially in the array. See the tutorial on synchronous FIFOs for more information.

The flag logic compares the value in each of the two address pointers. If the difference between the two pointers is zero, the FIFO is empty and the empty flag is asserted. If the difference between the two values is equal to the depth of the part, the FIFO is full and the full flag is asserted. Other flags, such as half-full, programmable almost-empty and programmable almost-full flags, are generated by the same means. The programmable flags are generated by comparing the values programmed in an offset register with the number of words in the FIFO.

Finally, expansion logic is used to create logically deeper FIFOs, by cascading multiple parts in depth expansion. In the normal “non-depth cascading” operation, each of the address pointers wraps back to zero when it reaches its maximum value. In the depth expansion mode, when an address pointer reaches its maximum value, a pulse is driven to an expansion pin, which passes a token to another FIFO. After the token is passed, the address pointer does not increment until the token returns. Essentially, the responsibility for handling the write or read operation is passed to another device. At any given time, only one FIFO in a depth expansion configuration handles read operations and only one handles write operations. When the token returns, the address pointer is reset to zero and the operation resumes.

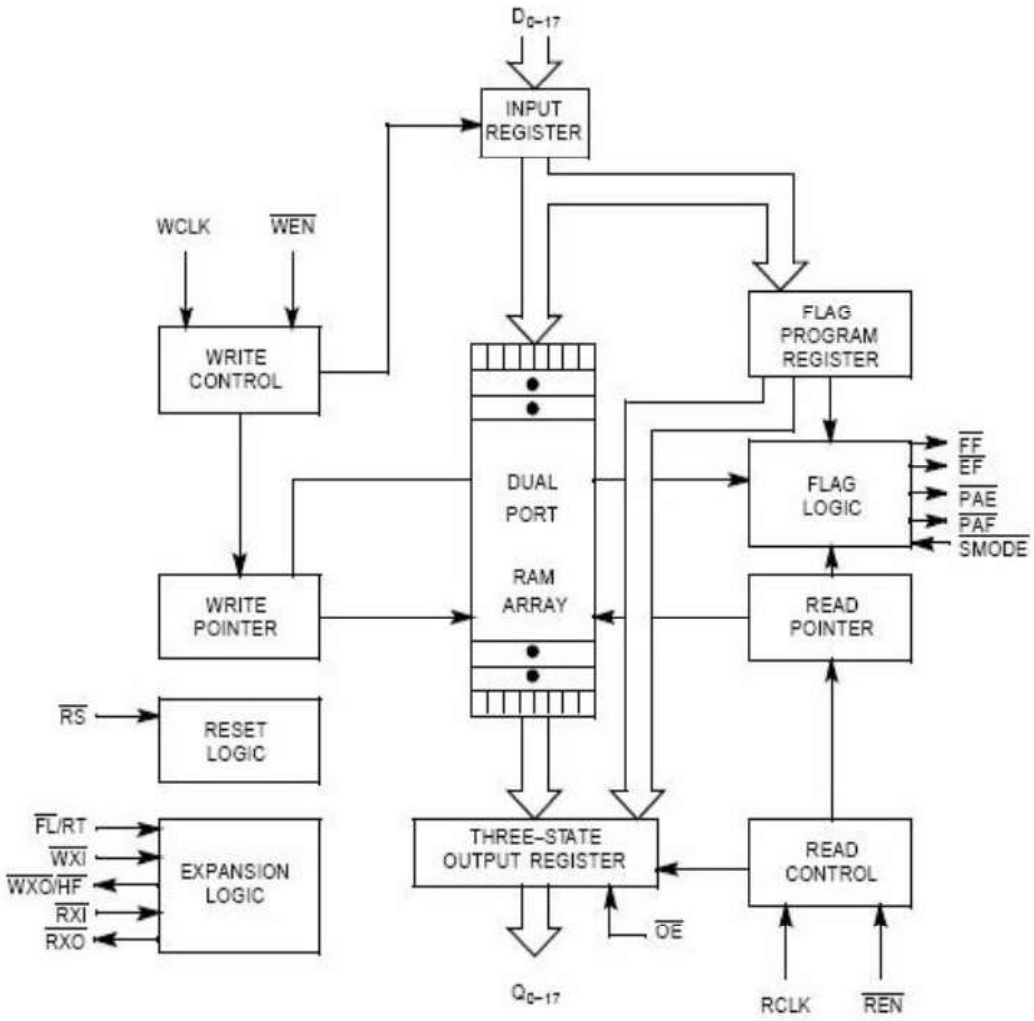


Figure 1. Logic Block Diagram – Synchronous FIFO Architecture .

3.1 Reset

After power-up, the FIFO must be reset. Resetting the part sets the read and write address pointers to zero, clears the output data register, and sets the status flags to represent an empty device. The device is reset by asserting the RS pin LOW. Synchronous FIFOs require a falling edge on RS. This allows devices, such as processor supervisory chips, to drive RS directly. These devices assert reset as Vcc ramps and hold it LOW for a minimum time to allow Vcc and all clocks to stabilize.

During RS assertion, read or write operations should not be attempted to the part. This can be done by deasserting the read and write enables (REN, WEN), or by gating both RCLK and WCLK to a low state. Write and read operations must also be disabled until the reset recovery time expires t_{RSR} after the deassertion (rising) edge of RS.

Note :- Reset is an asynchronous operation and does not require transitions of WCLK and RCLK to complete.

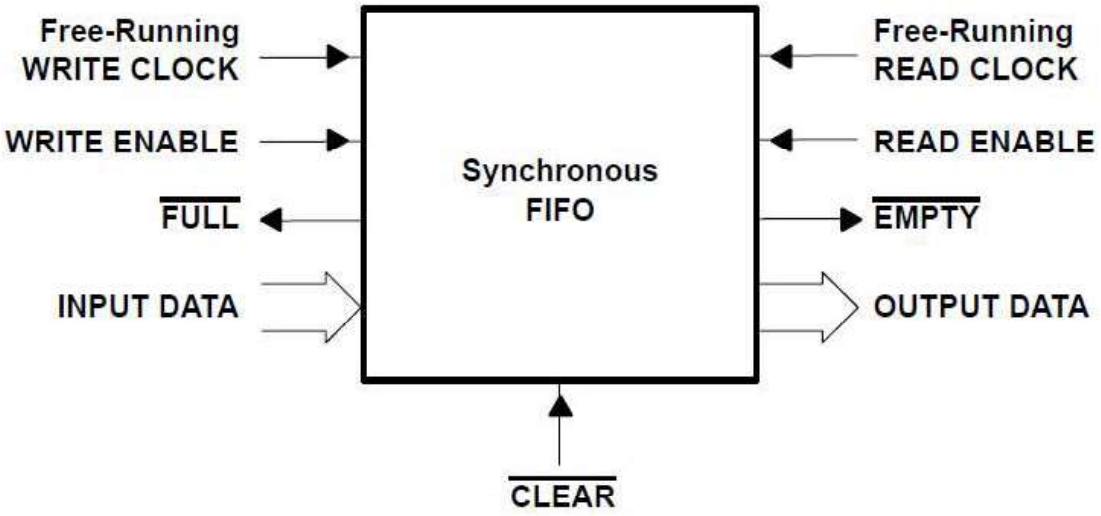
3.2 Status Flags

Status flags, such as the empty flag, programmable almost-empty flag, half-full flag, programmable almost-full flag, and full flag (EF, PAE, HF, PAF, FF) are used to determine the FIFO status. These flags are generated by comparing the values in the read and write address pointers. External control logic should use these flags to determine whether read or write operations can be performed on the FIFO. The flag logic in the FIFO also inhibits reading from an empty FIFO and writing to a full FIFO. When reading an empty FIFO, the outputs will always show that last valid data read from the device. Writes to a full FIFO are discarded.

Synchronous FIFOs

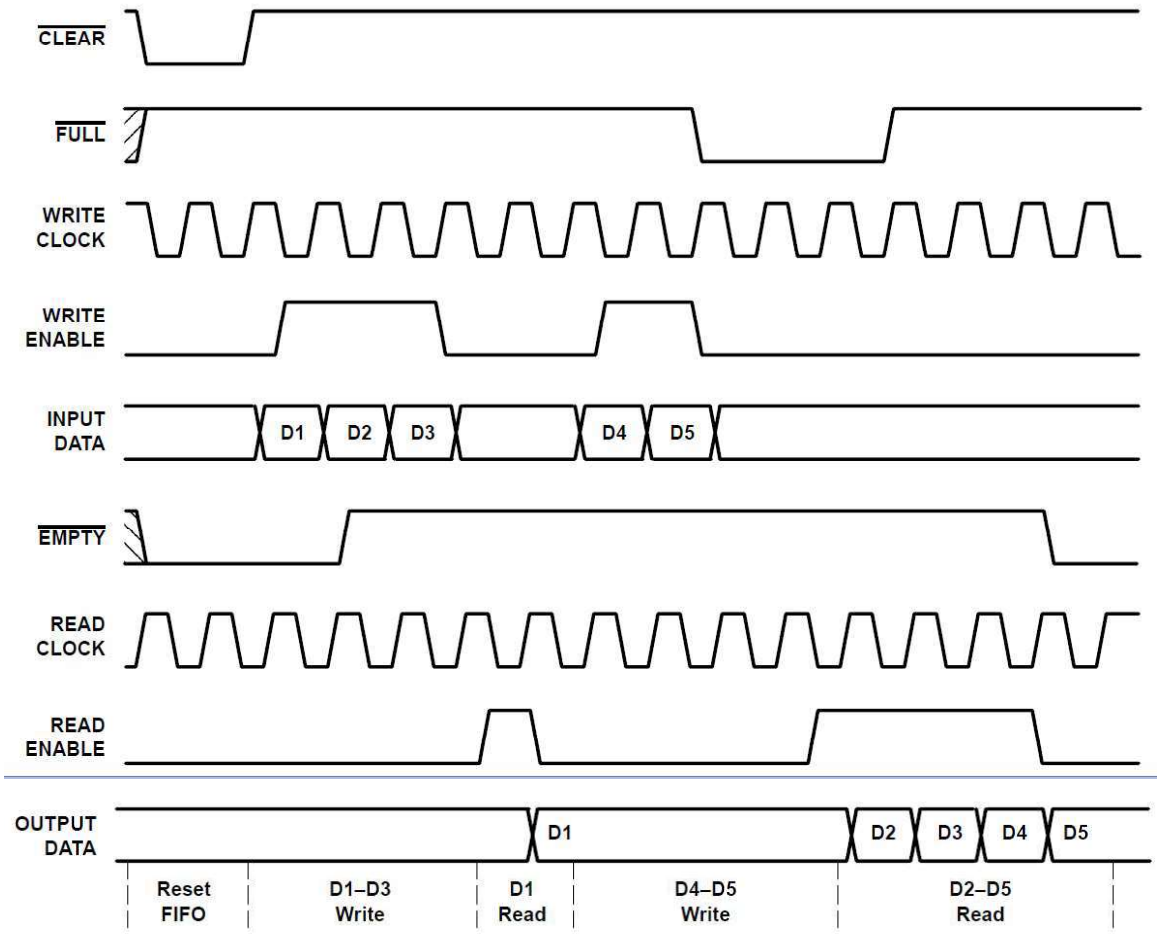
Synchronous FIFOs are controlled based on methods of control proven in processor systems. Every digital

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, also often called chip-select signals, start the synchronous execution of write and read operations in the various devices, such as memories and ports.
The block diagram in Figure 11 shows all the signal lines of a synchronous FIFO. It requires a free-running clock from the writing system and another from the reading system. Writing is controlled by the WRITE ENABLE input synchronous with WRITE CLOCK. The FULL status line can be synchronized entirely with WRITE CLOCK by the free-running clock. In an analogous manner, data words are read out by a low level on the READ ENABLE input synchronous with READ CLOCK. Here, too, the free-running clock permits 100 percent synchronization of the EMPTY signal with READ CLOCK.



Connections of a synchronous FIFO

Thus, synchronous FIFOs are integrated easily into common processor architectures, offering complete synchronism of the FULL and EMPTY status signals with the particular free-running clock. Figure shows the typical waveform in a synchronous FIFO. WRITE CLOCK and READ CLOCK are free running. The writing of new data into the FIFO is initialized by a low level on the WRITE ENABLE line. The data are written into the FIFO with the next rising edge of WRITE CLOCK. In analogous fashion, the READ ENABLE line controls the reading out of data synchronous with READ CLOCK.
All status lines within the FIFO can be synchronized by the two free-running-clock signals. The FULL line only changes its level synchronously with WRITE CLOCK, even if the change is produced by the reading of a data word. Likewise, the EMPTY signal is synchronized with READ CLOCK. A synchronous FIFO is the only concurrent read/write FIFO in which the status signals are synchronized with the driving logic.



Timing Diagram for a Synchronous FIFO of Length 4

>Applications for Synchronous FIFOs :-

FIFOs are an ideal solution to the problem of moving data between a processor and peripheral device that either operate at different speeds, or use unsynchronized clock sources. For example, modern processors are faster than the peripheral devices that are connected to them. A FIFO can be used such that the processing speed need not be reduced when it exchanges data with the peripheral. If the peripheral is faster than the processor, the FIFO can again be used to resolve the problem.

In case of computer networks and digital-telephone-switching, data is split into blocks and transmitted on the datalines. The data split into blocks and transmitted on the data lines at a very high speed requires the use of FIFO for data transfer.

FIFOs find use in set-top box for HDTV/IPTV. It controls the dataflow between the main CPU, which performs MPEG encoding/decoding, in addition to audio processing and the CPLD, which is a communication processor capturing an incoming signal.

A common scenario of asymmetric bus speeds occurs in the case of data acquisition equipment where multiple boards that operate at different local bus frequencies need to exchange data with each other, or with a master controller using a higher system bus speed. FIFOs enable this operation through their ability to use two different clocks for input and output. This enables synchronization of data flow to the local bus frequency. This feature results from the use of dual-port memory cells that allow unconstrained simultaneous access from two independent ports.

FIFOs are widely used in inter-processor communication, to pass data between processors running at different speeds without incurring excessive wait state penalties. FIFOs are also used to buffer sequential data, such as video/voice and data packets in telecommunication systems.

FIFOs are differentiated from dual-port memories by the lack of addressing capability. FIFOs, true to their name, are data buffers that only support sequential access and hence, eliminate the need for external

addressing. This results in reduced complexity, pin count, and board space.