FIFO Architecture, Functions, and Applications

SCAA042A
November 1999

Texas Instruments

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Contents

Title *Page*

Abstract **1**

Introduction **1**

FIFO Types **2**
Exclusive Read/Write FIFOs 3
Concurrent Read/Write FIFOs 3
Metastability of Synchronizing Circuits 3
Asynchronous FIFOs 6
Synchronous FIFOs 9

FIFO Architectures **11**
Fall-Through FIFOs 11
Architecture 11
Advantages and Drawbacks 13
FIFOs With Static Memory 13
Architecture 13
Advantages and Drawbacks 14

FIFOs From TI **14**
Features 14
Data Outputs With Latches 14
Synchronization of Flags 15
Edges of Outputs 16
Extending Word Width 16
Extending Memory Depth 17

Application Examples **18**
Asynchronous Operation of Exclusive Read/Write FIFOs 18
Connection of Peripherals to Processors 20
Block Transfer of Data 23
Programmable Delay 24
Collecting Data Before an Event 24
Collecting Data Before and After an Event 26

Summary **29**

Acknowledgment **29**

List of Illustrations

Figure	*Title*	*Page*
1	First-In First-Out Data Flow	2
2	Synchronization of External Signal	3
3	Timing Diagram for the Metastable State	4
4	Block Diagram of Two-Level Synchronization	4
5	Timing Diagram for Two-Level Synchronization	5
6	Signals of FIFO With Single-Level Synchronization (Recorded for 15 Hours Under Worst-Case Conditions)	5
7	Signals of TI SN74ACT7807 FIFO With Three-Level Synchronization (Recorded for 15 Hours Under Worst-Case Conditions)	6
8	Connections of an Asynchronous FIFO	6
9	Timing Diagram for Asynchronous FIFO of Length 4	7
10	Asynchronism When Resetting $\overline{\text{FULL}}$ Signal	8
11	Connections of a Synchronous FIFO	9
12	Timing Diagram for a Synchronous FIFO of Length 4	10
13	Circuitry of 4×5 Fall-Through FIFO	11
14	Timing Diagram of 4×5 Fall-Through FIFO in Figure 13	12
15	Circular FIFO With Two Pointers	13
16	Block Diagram of FIFO With Static Memory	14
17	Waveforms on FIFO Outputs	14
18	Signals in Synchronous FIFO SN74ACT7881 With Multilevel Synchronization of Status Outputs	15
19	Extending Word Width of Asynchronous FIFOs	16
20	Extending Word Width of Synchronous FIFOs	17
21	Extending Memory Depth of Synchronous FIFOs	17
22	Timing Conditions for WRITE CLOCK and READ CLOCK	18
23	Synchronizing Circuit for Generating WRITE CLOCK and READ CLOCK Signals	18
24	Timing of Signals in Synchronizing Circuit	19
25	Connection of Unidirectional Peripheral With the SN74ACT7881 FIFO	20
26	Connection of Bidirectional Peripheral With the SN74ACT2235 FIFO	21
27	Connection of Bidirectional Peripheral With DMA and SN74ACT2235 FIFO	22
28	Video Signal With Picture Information and Porch	22
29	Digitizing and Compressing a Video Signal With the TMS320C30 Signal Processor	23
30	Block Transfer of Data With Synchronous SN74ACT7881 FIFO	23
31	Programmable Digital Delay With the SN74ACT7881	24
32	Timing of Signals in Programmable Digital Delay With the SN74ACT7881	24
33	Collecting Data Before an Event With the SN74ACT7881	25
34	Timing of Device Shown in Figure 33	25
35	Collecting Data Before and After an Event With the SN74ACT7881	26
36	Timing of Signals in the SN74ACT7881 During Initialization and Start of Data Capture	27
37	Timing of Signals in the SN74ACT7881 at End of Data Capture and Start of Readout	28

Abstract

First-in first-out memories (FIFOs) have progressed from fairly simple logic functions to high-speed buffers incorporating large blocks of SRAM. This application report takes a detailed look at the evolution of FIFO device functionality and at the architecture and applications of FIFO devices from Texas Instruments (TI™). The first part presents the different functions of FIFOs and the resulting types that are found. The second part deals with current FIFO architectures and the different ways in which they work. Finally, some application examples are given to illustrate the use of FIFOs from the TI product spectrum.

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. A data rate of 3.6 gigabits per second is specified for a Texas Instruments (TI™) SN74ABT7819 FIFO.

This application report takes a detailed look at TI FIFO devices. The first part presents the different functions of FIFOs and the resulting types that are found. The second part deals with current FIFO architectures and the different ways in which they work. Finally, some application examples are given to illustrate the use of FIFOs available from TI.

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

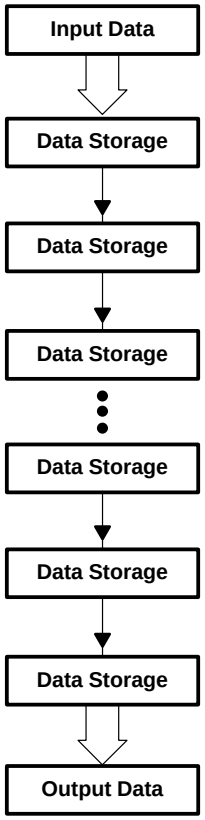

Figure 1. First-In First-Out Data Flow

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

Metastability of Synchronizing Circuits

In digital engineering, there is the constantly recurring problem of synchronizing two systems that work at different frequencies. Concurrent read/write FIFOs can also handle the data exchange between two systems of different frequencies, so internal synchronizing circuits are called for. This section is a brief introduction to the problems that are involved in synchronization.

The problem of synchronizing an external signal with a local clock generator normally is solved by using a flip-flop (see Figure 2), but this means violating the setup and hold times stated in the data sheets for the devices. As a result, the flip-flop can go into a metastable state.

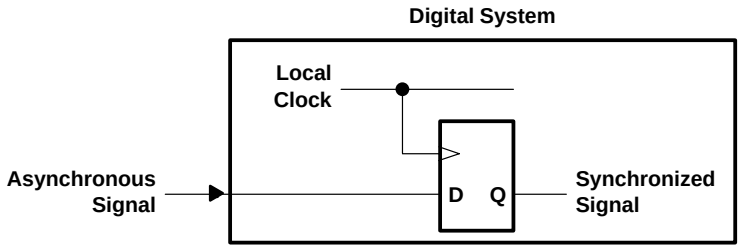

Figure 2. Synchronization of External Signal

With a D-type flip-flop, the setup or hold times must be maintained. This means that for a short time before the clock edge (setup time) and for a short time afterward (hold time) the level on the D input must not change to ensure that the function of the flip-flop is executed correctly. If these conditions are not maintained, the flip-flop can become metastable. In an RS flip-flop, metastable states are possible if the reset and set inputs change from the active to the inactive state at the same time. In both cases, the flip-flop adopts an undefined and unstable, or metastable, state (see Figure 3). No defined state can be ensured on the Q outputs. After a time, the flip-flop goes into one of the two stable states, but it is impossible to predict which.

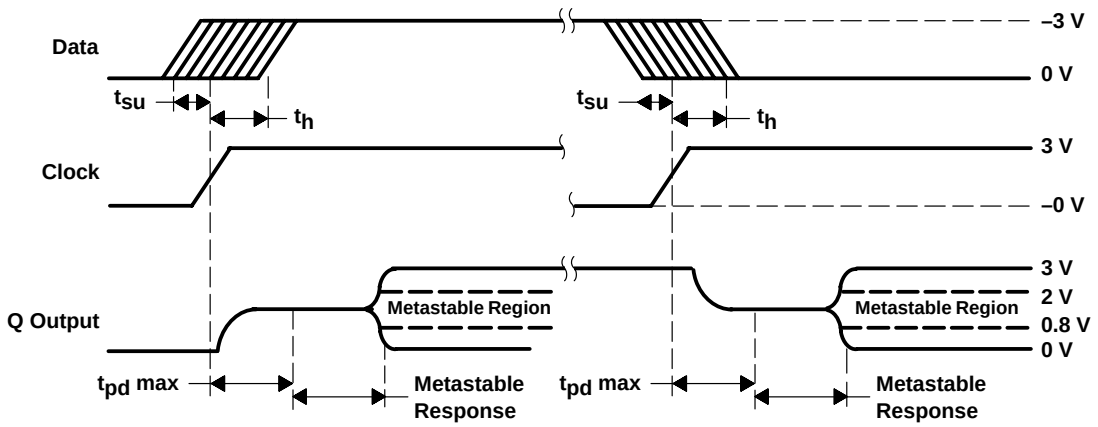

Figure 3. Timing Diagram for the Metastable State

These operating conditions for flip-flops can be maintained easily in synchronous circuits. But, with asynchronous circuits and in synchronizing circuits, violations of the operating conditions for D and RS flip-flops are unavoidable. Concurrent read/write FIFOs that are driven by two systems working asynchronously to one another must perform internal synchronization of the asynchronous external signals.

For physical reasons, there are no ideal flip-flops with a setup and hold time of zero, so there can be no synchronizing circuit that works faultlessly. The quality of a synchronizing circuit is indicated by its mean time between failures (MTBF). This is calculated from the frequency of the asynchronous signal (f_{in}), the clock frequency of the synchronizing circuit (f_{clk}), and the length of the critical time window (t_d).

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

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

$$\text{MTBF} = \frac{1}{1\ \text{kHz} \times 1\ \text{MHz} \times 30\ \text{ps}} = 33.3\ \text{s} \quad (2)$$

If a flip-flop is used to synchronize two signals, you can no longer expect the maximum delays stated in the data sheets. Therefore, for reliable operation of a system, you must know how long it is necessary to wait after the clock pulse until data can be evaluated.

MTBF can be improved appreciably by multilevel synchronization. Figure 4 illustrates a two-level synchronizing circuit, and the timing of the signals is shown in Figure 5. The second flip-flop can only go into a metastable state if the first flip-flop is already metastable. This metastability can considerably increase the delay of the first flip-flop. But if the period of the clock signal is longer than the sum of the increased delay plus the setup time of the second flip-flop, the second flip-flop can never go into a metastable state.

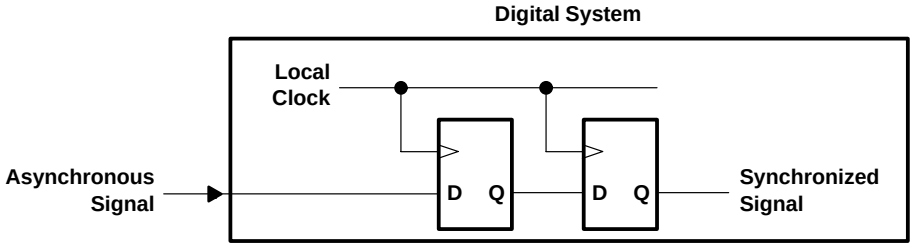

Figure 4. Block Diagram of Two-Level Synchronization

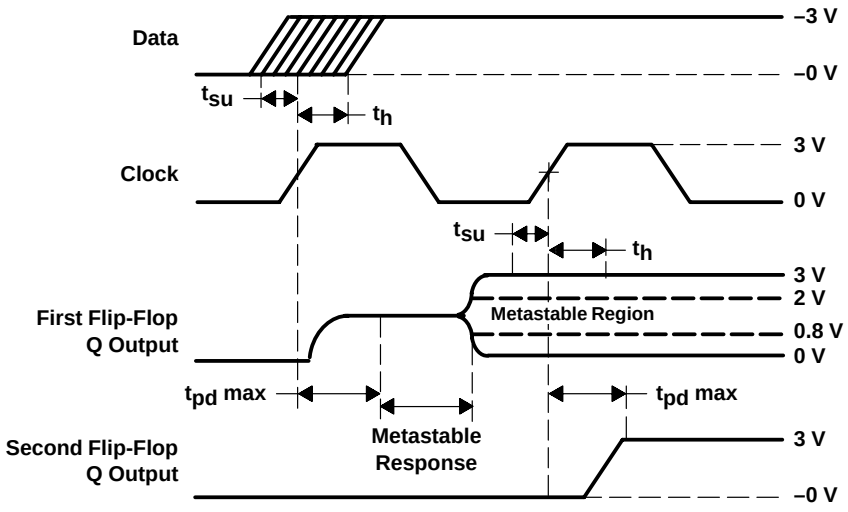

Figure 5. Timing Diagram for Two-Level Synchronization

To measure metastability, conventional concurrent read/write FIFOs were operated in the metastable region, and the READ CLOCK input signal and $\overline{\text{EMPTY}}$ output signal were recorded for a period of 15 hours using a storage oscilloscope.

The signal patterns for single-level synchronization are shown in Figure 6. It is easy to see that the synchronizing flip-flop sometimes decides for a 1 level and sometimes for a 0 level on the first clock edge. In some cases, the decision is obviously difficult for the flip-flop because it takes much more time than normal. After the second clock edge, the output is stable again in every case.

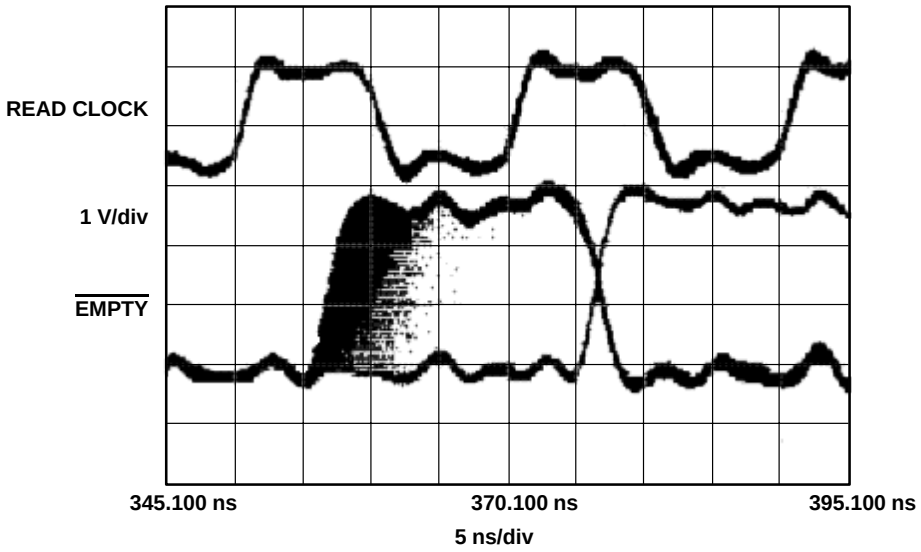

Figure 6. Signals of FIFO With Single-Level Synchronization (Recorded for 15 Hours Under Worst-Case Conditions)

The measurement illustrated in Figure 7 was performed on a TI SN74ACT7807 FIFO. This FIFO features three-level synchronization. Here too, the decision for a high level on the $\overline{\text{EMPTY}}$ output sometimes comes one clock cycle later, but the metastable response, with the much longer time to decide, can no longer be observed.

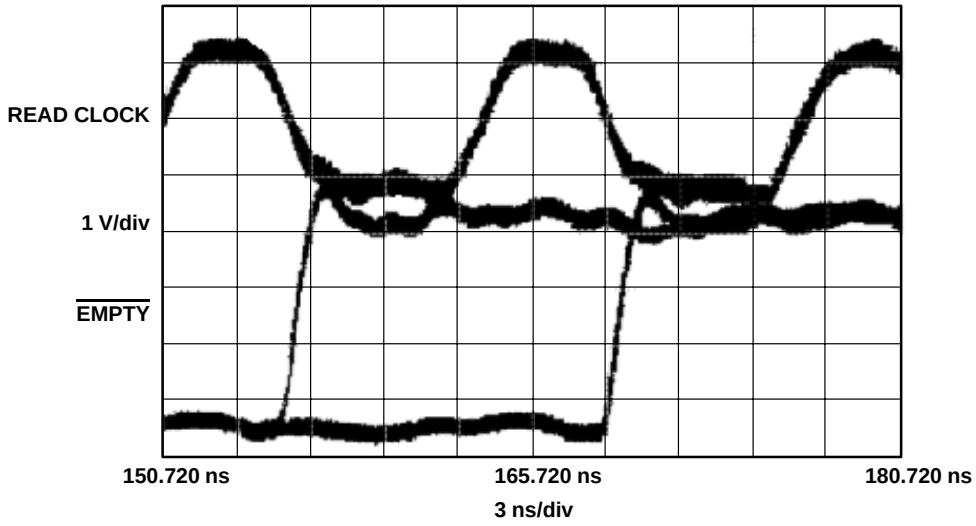

Figure 7. Signals of TI SN74ACT7807 FIFO With Three-Level Synchronization (Recorded for 15 Hours Under Worst-Case Conditions)

Asynchronous FIFOs

The control signals of an asynchronous FIFO correspond most closely to human intuition and were, in the past, the only kind of FIFO driving. The block diagram in Figure 8 shows the control lines of an asynchronous FIFO, and Figure 9 illustrates the typical timing on these lines in a read and write operation.

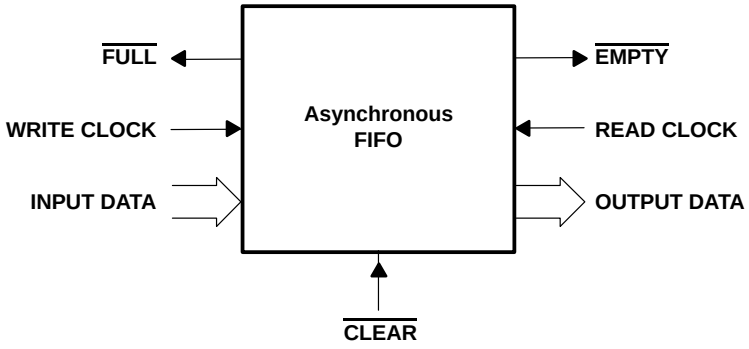

Figure 8. Connections of an Asynchronous FIFO

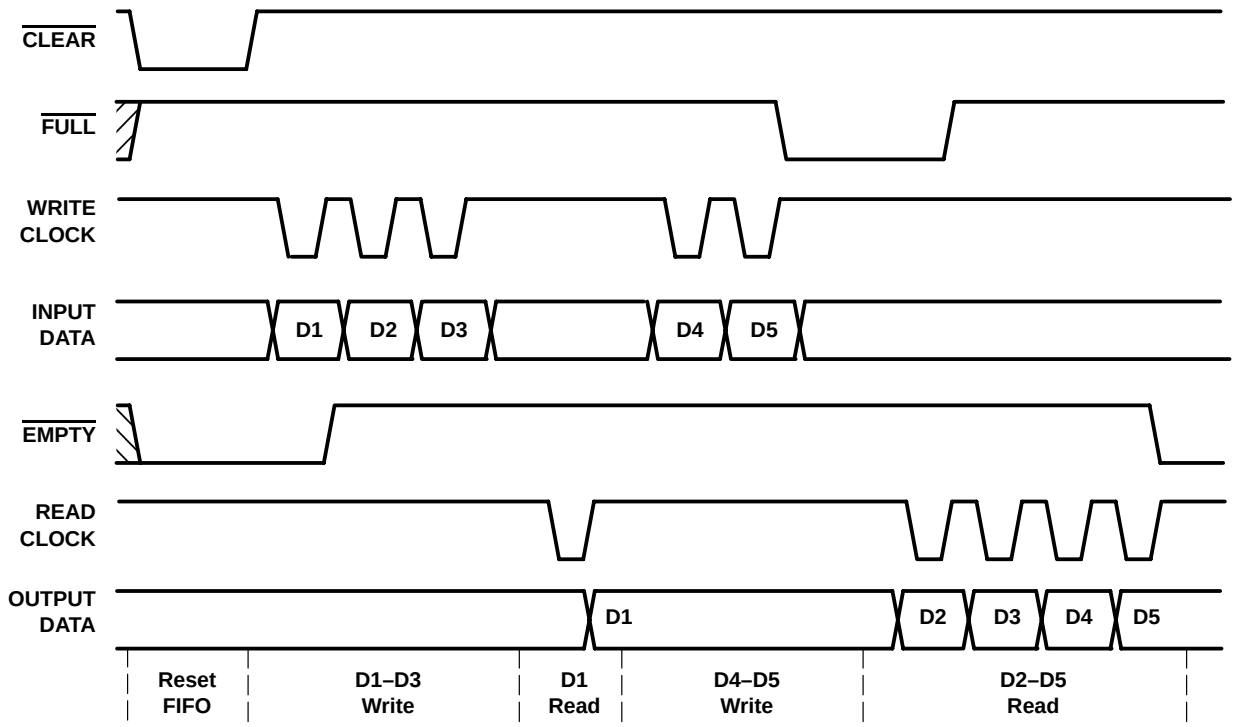

Figure 9. Timing Diagram for Asynchronous FIFO of Length 4

The control lines WRITE CLOCK and $\overline{\text{FULL}}$ are used to write data. 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 $\overline{\text{FULL}}$ status line. If free space is indicated, the data word is applied to the data inputs and written into the FIFO by a clock edge on the WRITE CLOCK input.

In analogous fashion, the control lines READ CLOCK and $\overline{\text{EMPTY}}$ are used to read data. In this case, the $\overline{\text{EMPTY}}$ status output has to be queried before reading, because data can be read out only if it is stored in the FIFO. Then, a clock edge is applied to the READ CLOCK input, causing the first word in the data queue to appear on the data output.

The timing diagram in Figure 9 shows the resetting 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 for transfer of the data. Once the first data word has been written into the FIFO, the $\overline{\text{EMPTY}}$ signal changes from low level to high level. 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 does not alter the status signals. With the writing of another two data words, the FIFO is full. This is indicated by the $\overline{\text{FULL}}$ signal. Finally, the four data words D2 through D5 remaining in the FIFO are read out. Thus, the FIFO is empty again, so the $\overline{\text{EMPTY}}$ status line shows this by low level.

The disadvantage of a FIFO of this kind is that the status signals cannot be fully synchronized with the read and write clock. An example of this is shown in Figure 10.

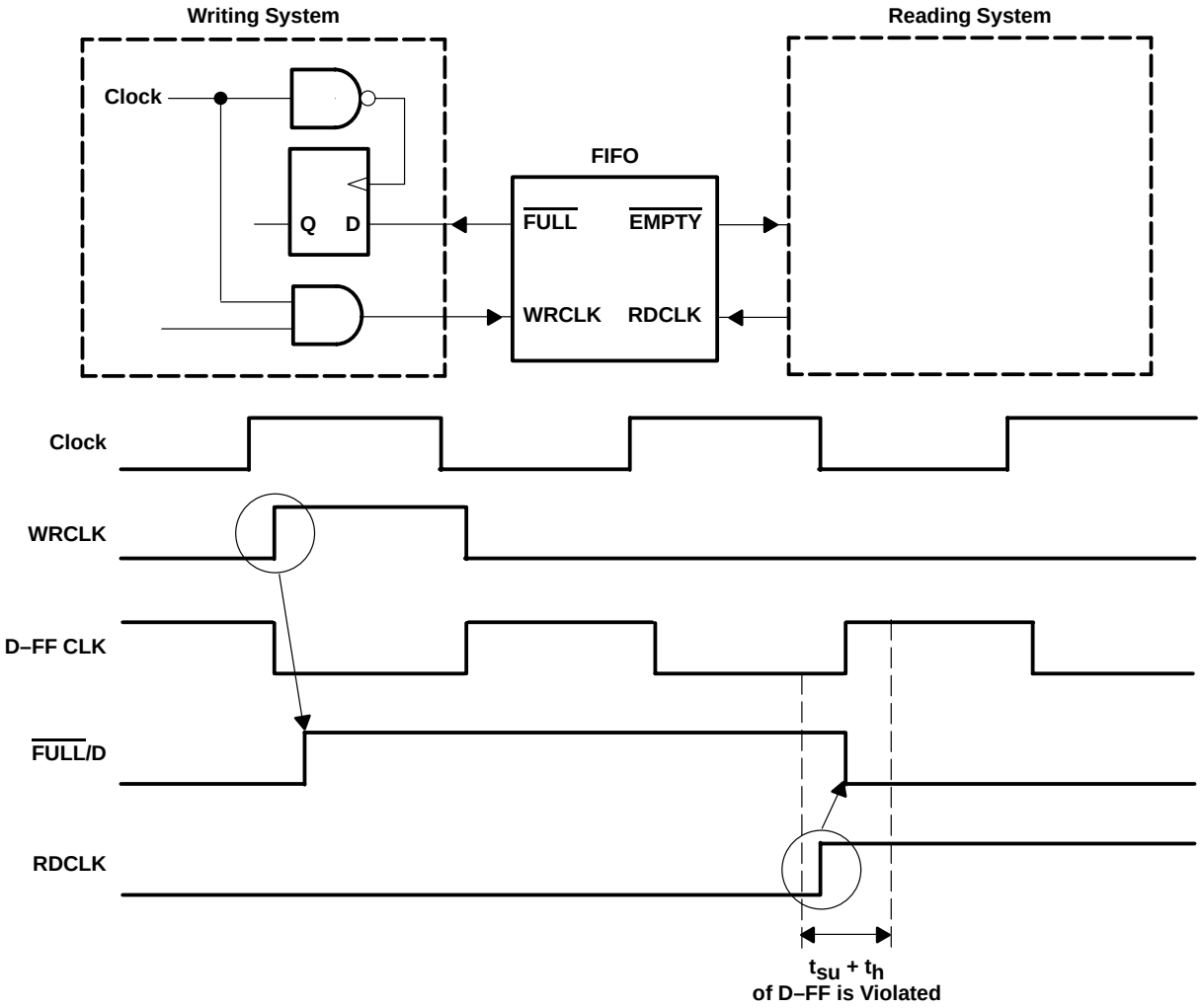

Figure 10. Asynchronism When Resetting $\overline{\text{FULL}}$ Signal

If there is space in the FIFO for only one data word, the next write cycle sets the $\overline{\text{FULL}}$ signal. Then, the writing system queries the $\overline{\text{FULL}}$ signal with the aid of its D flip-flop and waits until there is again space in the FIFO. When a data word is read, READ CLOCK resets the $\overline{\text{FULL}}$ status line. This reset is synchronous with the reading system but asynchronous to the writing system. In the worst case, the setup or hold time of the flip-flop in the writing system is violated. This flip-flop goes into a metastable state, the results of which were discussed previously.

The problem described above also occurs with the $\overline{\text{EMPTY}}$ status signal. $\overline{\text{EMPTY}}$ should be synchronous with the reading system, but it is reset by the writing system. So, the resetting of $\overline{\text{EMPTY}}$ is inevitably asynchronous to the reading system.

This asynchronism is a result of the system, and synchronization is not possible within the asynchronous FIFO. If synchronization becomes necessary, the designer must provide it externally. Nevertheless, there are wide-ranging application possibilities for asynchronous 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, 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 $\overline{\text{FULL}}$ 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 $\overline{\text{EMPTY}}$ signal with READ CLOCK.

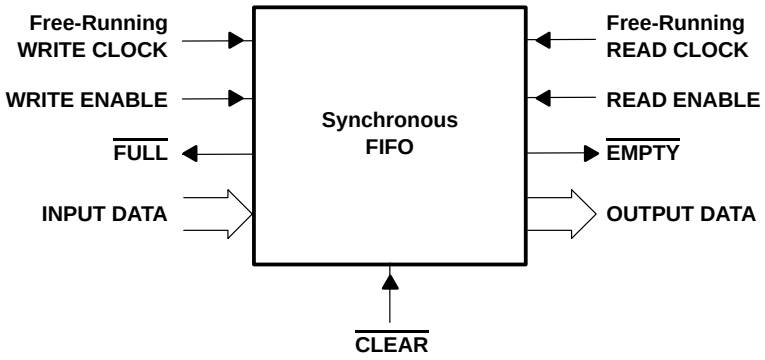

Figure 11. Connections of a Synchronous FIFO

Thus, synchronous FIFOs are integrated easily into common processor architectures, offering complete synchronism of the $\overline{\text{FULL}}$ and $\overline{\text{EMPTY}}$ status signals with the particular free-running clock.

Figure 12 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 $\overline{\text{FULL}}$ line only changes its level synchronously with WRITE CLOCK, even if the change is produced by the reading of a data word. Likewise, the $\overline{\text{EMPTY}}$ 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.

All TI synchronous FIFOs feature multilevel synchronization of the status lines as described previously.

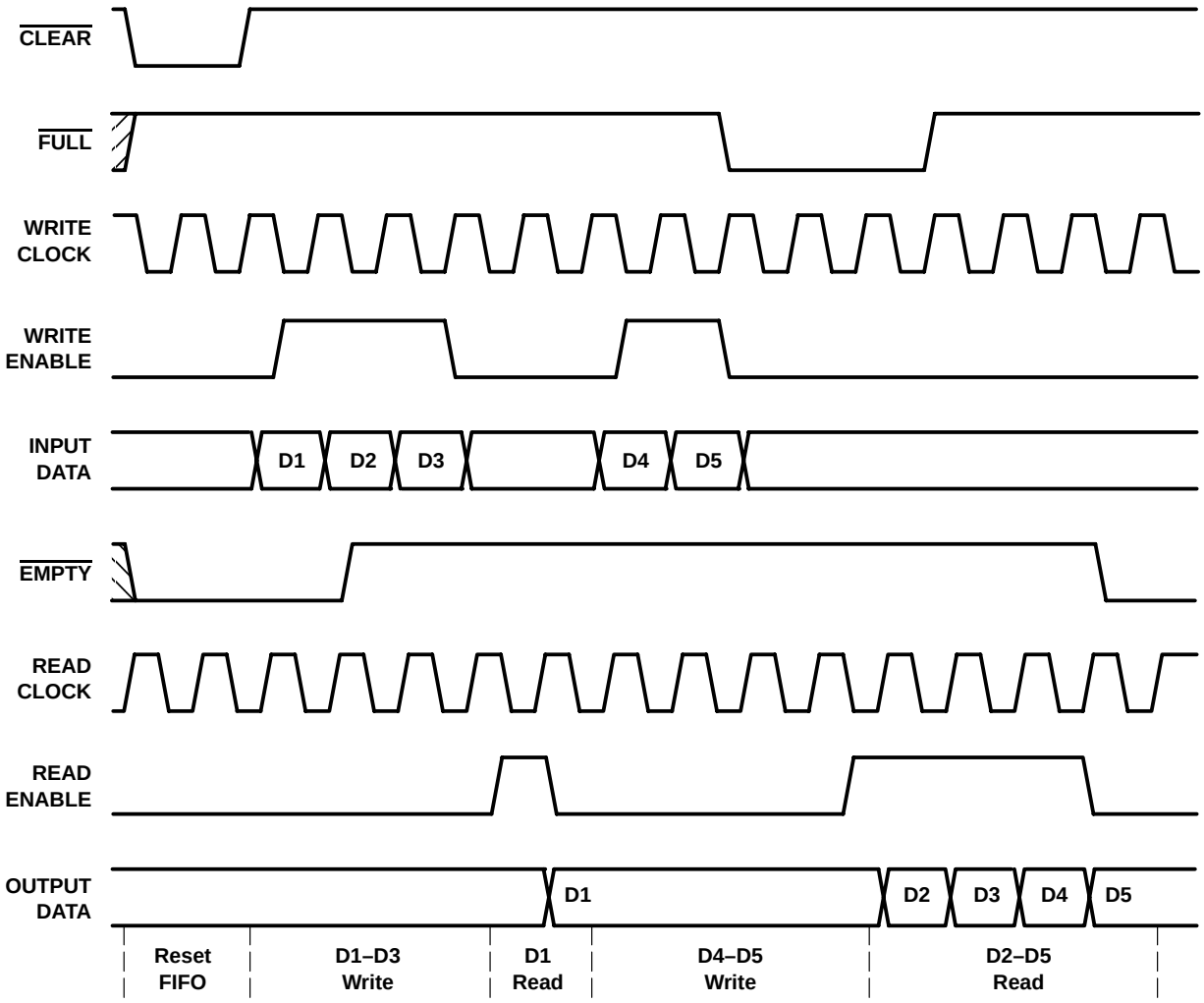

Figure 12. Timing Diagram for a Synchronous FIFO of Length 4

FIFO Architectures

All the kinds of FIFOs described in *FIFO Types* can be implemented in different hardware architectures. The architecture of conventional FIFOs has constantly been developed. Initially, FIFOs worked by the fall-through principle. Today, FIFOs are nearly always based on an SRAM, which produced a considerable increase in the number of data words stored, despite the faster speed. All possible hardware architectures also are found in software FIFOs.

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.

Architecture

Figure 13 shows one possible design for a fall-through FIFO as implemented, for example, in the SN74S225. In the top half of the illustration, there is a row of latches for storing the data words. The clock generator in the bottom half controls data transfer, the shifting of data, and the data output of the chain of latches. When a new word is written into the FIFO, it "falls through" the entire row of latches and is stored at the last free location. The reading out of a word causes the remaining words to be shifted one position in the direction of the output. A fall-through FIFO, like any other FIFO, has to be reset before it is used so that the clock generator is in a defined initial state.

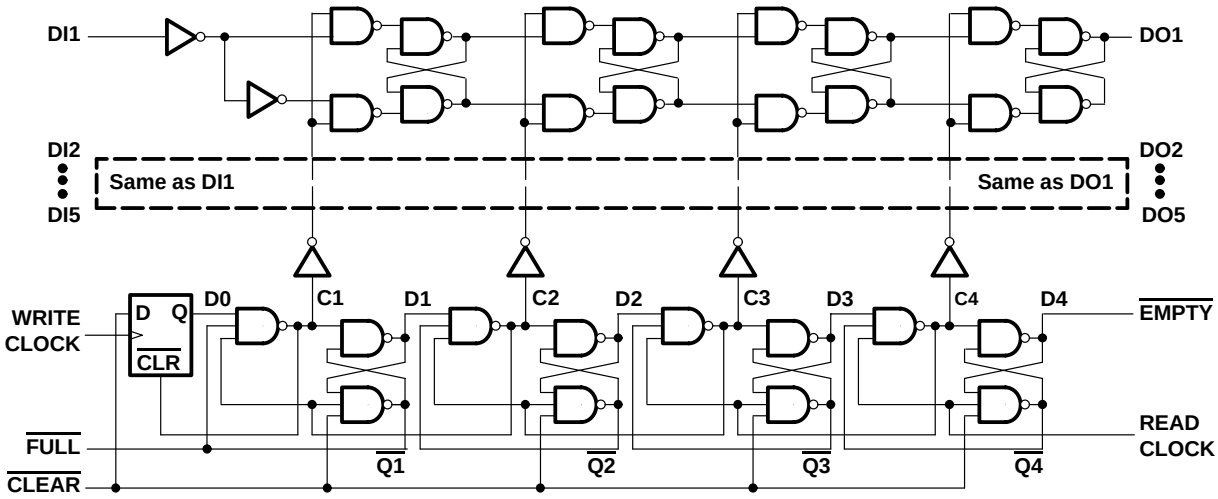

Figure 13. Circuitry of 4 × 5 Fall-Through FIFO

The internal states produced by this clock generator when shifting in a word are shown in Figure 14. The clock pulses are generated entirely by the internal gate propagation delays.

In Figure 14, it is possible to see how the clock generator, when writing in a word, produces a clock pulse at C1, C2, C3, and C4, in turn. With these clock pulses, the data word is shifted through the latches to the last one that is free. The clock generator also produces the necessary status information for each data word, specifying whether a data latch already contains a valid data word or is empty.

The time required for a data word to be shifted from the input to the output is called the fall-through time.

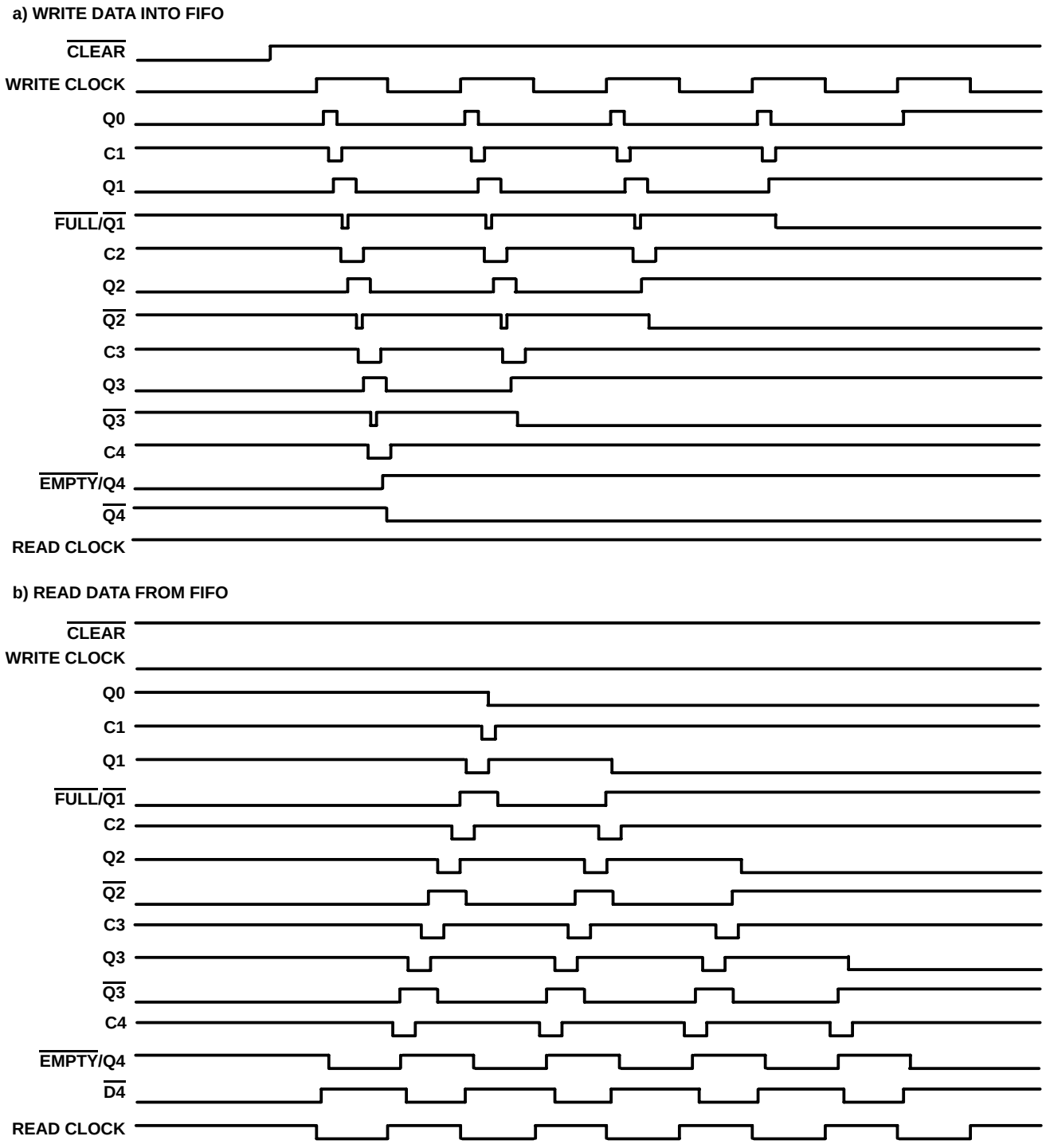

Figure 14. Timing Diagram of 4 × 5 Fall-Through FIFO in Figure 13

Advantages and Drawbacks

There must be a status flip-flop for each data word; therefore, the effort involved in controlling the data latches is justifiable only for very short FIFOs. With long FIFOs, there is also an enormous increase in fall-through time. For the 16×5 fall-through FIFO SN74S225, a maximum delay of 400 ns is specified from the READ CLOCK to the $\overline{\text{FULL}}$ signal. A 1024×18 FIFO, such as SN74ACT7881, requires a multiple of this time with fall-through architecture. The existence of fall-through architecture has a historical basis. New developments no longer use this principle.

FIFOs With 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. Figure 15 illustrates the principle of a circular FIFO with two pointers.

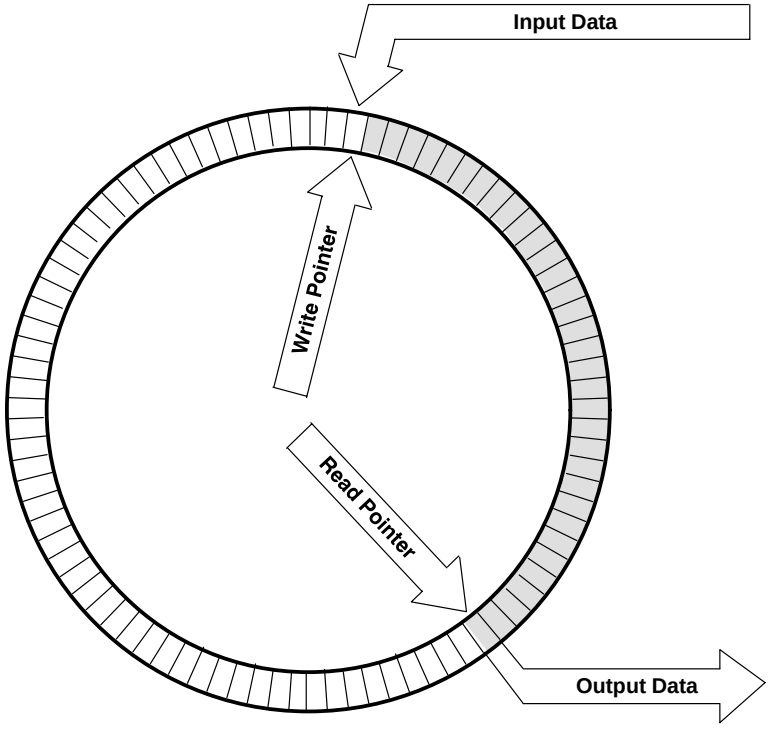

Figure 15. Circular FIFO With Two Pointers

Architecture

In the hardware implementation of a circular memory, a dual-port SRAM is used for data storage. The pointers take the form of binary counters, which generate the memory addresses of the SRAM. It is an advantage if the number of memory locations is 2n because a pointer can be implemented as an n-bit binary counter whose carry can be ignored.

Figure 16 shows a block diagram of a FIFO with static memory. Read addresses are generated by the READ POINTER and write addresses by the WRITE POINTER. The write-control and read-control blocks control operation during write and read access. The $\overline{\text{FULL}}$ and $\overline{\text{EMPTY}}$ status signals are generated by separate flag logic. Some FIFOs also offer the status signals HALF FULL, ALMOST FULL, and ALMOST EMPTY. A FIFO with static memory also has to be initialized before it is used to set the two pointers and the status logic to defined initial states.

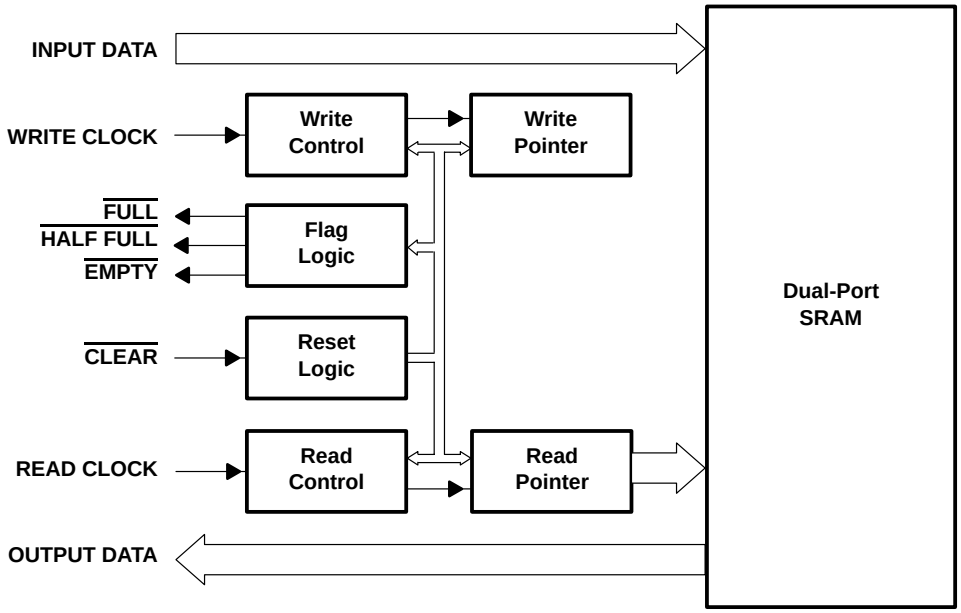

Figure 16. Block Diagram of FIFO With Static Memory

Advantages and Drawbacks

Unlike a fall-through FIFO, the fall-through time of a FIFO with static memory is independent of its length. Therefore, it is possible to create fast FIFOs with a length of several thousand words. The effort that goes into the circuitry of the control logic does not increase to any significant degree with length because only the two pointers and parts of the flag logic have to be expanded. Today's new developments use only the architecture of the FIFO with static memory.

FIFOs From TI

TI produces asynchronous FIFOs (e.g., ACT2235) and synchronous FIFOs (e.g., ACT7881). All of the circuits offered are concurrent read/write FIFOs with static memory.

Features

Data Outputs With Latches

The data outputs of TI FIFOs are buffered in a latch (see Figure 17), so the output signal is always valid. The outputs can be placed in the high-impedance state by a separate input signal (OE).

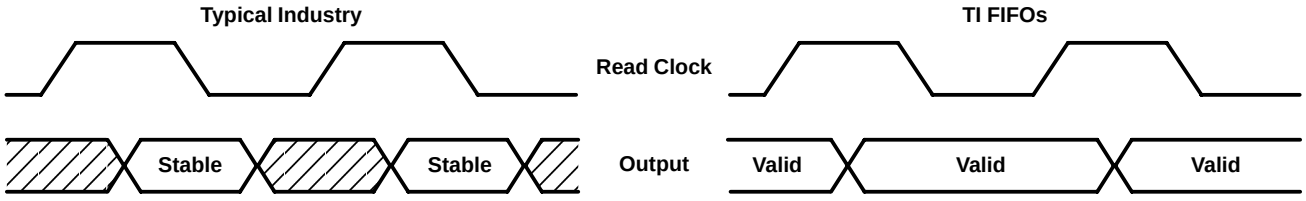

Figure 17. Waveforms on FIFO Outputs

Synchronization of Flags

In TI FIFOs, the flags ALMOST EMPTY, HALF FULL, and ALMOST FULL often are implemented in addition to the $\overline{\text{FULL}}$ and $\overline{\text{EMPTY}}$ status outputs. The conditions for ALMOST EMPTY and ALMOST FULL are usually programmable. The flags of the TI synchronous FIFOs bear the following designations:

$\overline{\text{EMPTY}}$ = OR (OUTPUT READY)

$\overline{\text{FULL}}$ = IR (INPUT READY)

As shown in the *Asynchronous FIFOs* section, the status outputs of asynchronous FIFOs cannot be synchronized entirely because of factors in the system. The resetting of the status outputs is always triggered by signals that are asynchronous to the status signal:

- WRITE CLOCK signal resets the $\overline{\text{EMPTY}}$ status output, but $\overline{\text{EMPTY}}$ should be synchronous with READ CLOCK.
- READ CLOCK signal resets the $\overline{\text{FULL}}$ status output, but $\overline{\text{FULL}}$ should be synchronous with WRITE CLOCK.

The status outputs of all TI synchronous FIFOs go through multilevel synchronization when the flags are reset, as described in the *Metastability of Synchronizing Circuits*. Thus, the resetting of the status outputs is delayed by a few clock cycles. Figure 18 shows this taking the OR (OUTPUT READY) flag of SN74ACT7881 as an example. However, in most cases, this delay is insignificant, because it does not affect the data rate either in writing or reading. The status outputs are set without a delay because the status outputs are triggered by a synchronous signal.

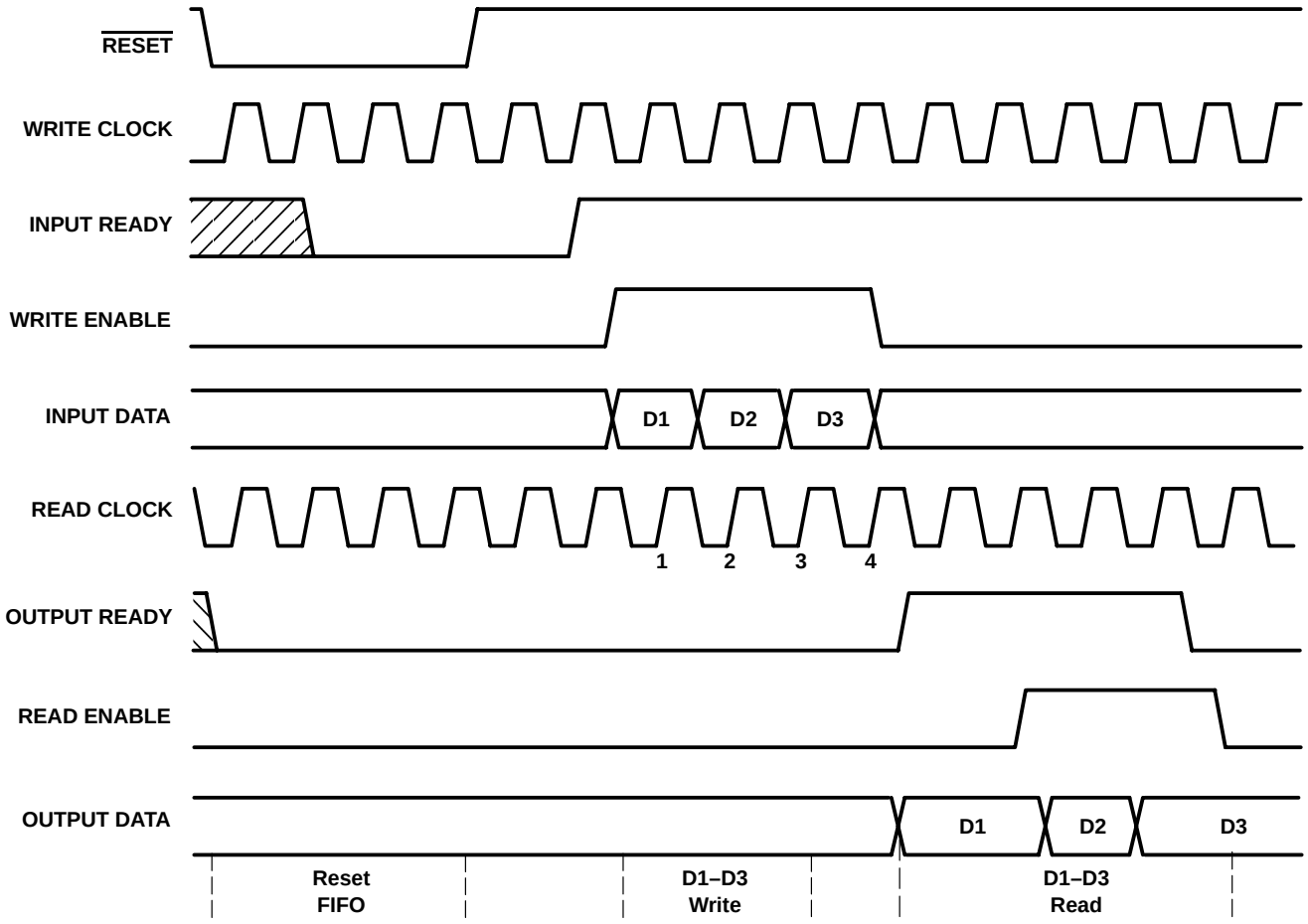

Figure 18. Signals in Synchronous FIFO SN74ACT7881 With Multilevel Synchronization of Status Outputs

Edges of Outputs

The patented output edge control (OEC™) circuit simultaneously limits the current across V_{CC}, GND, and the outputs, reducing noise caused by switching operations. This offers the best possible protection against noise on waveforms and against alteration of memory contents.

Extending Word Width

All types of FIFOs are very easy to cascade in their word width, as shown in Figures 19 and 20. The control inputs of all FIFOs used are connected in parallel. In theory, for the status outputs it is sufficient to look at just one FIFO because all parallel-connected FIFOs must always show the same internal state. But, it is safer to consider the status lines of all FIFOs so that differences in delay between the FIFOs do not lead to malfunctions. This can be taken care of simply with AND or OR gates.

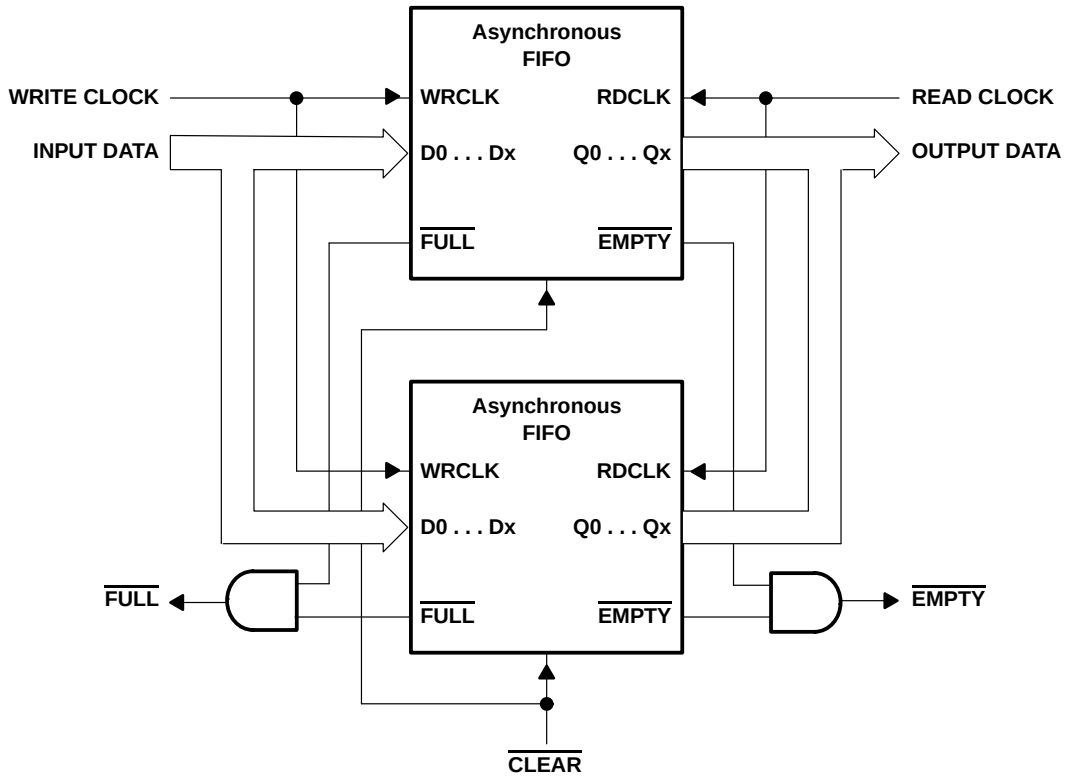

Figure 19. Extending Word Width of Asynchronous FIFOs

OEC is a trademark of Texas Instruments Incorporated.

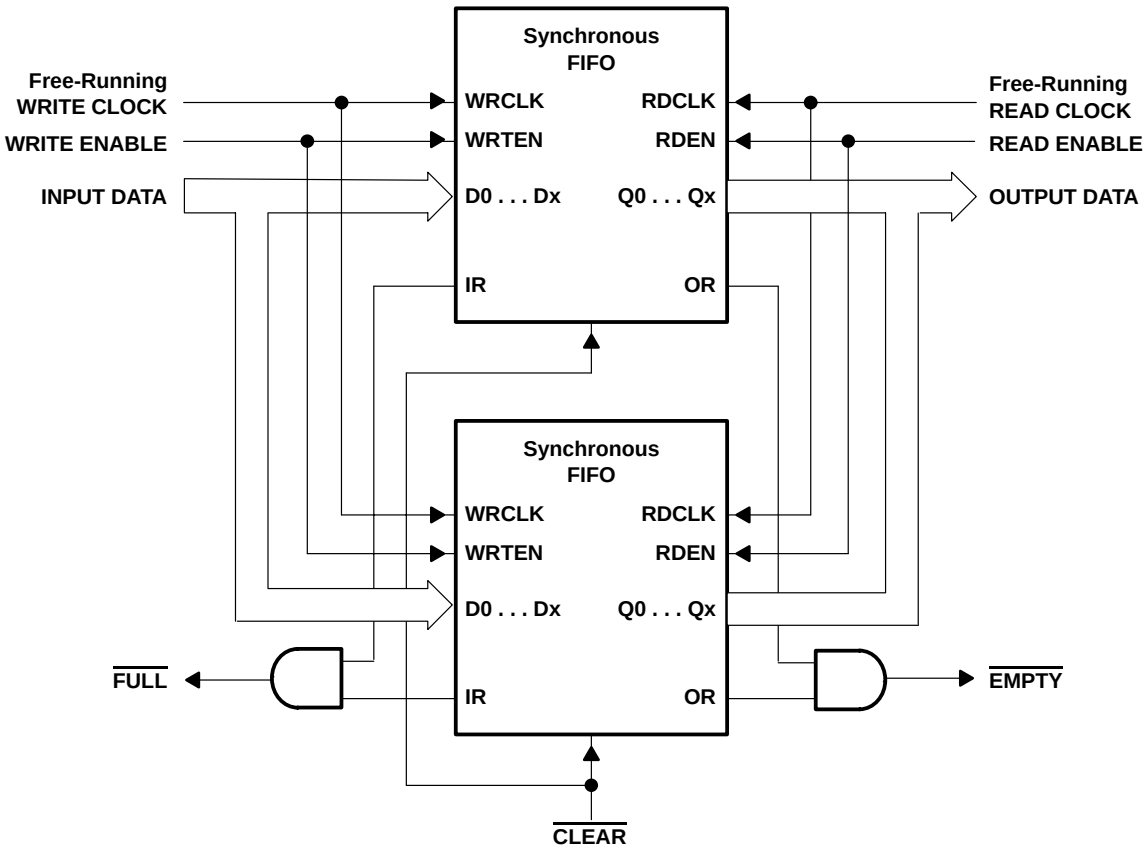

Figure 20. Extending Word Width of Synchronous FIFOs

Extending Memory Depth

The memory depth of synchronous FIFOs is easily extended. Figure 21 shows that the individual FIFOs are configured by the fall-through principle. An additional clock signal is required to transfer the data from the first FIFO to the second. This clock signal can be the WRITE CLOCK, the READ CLOCK, or another free-running clock signal, and the frequency of this clock signal determines the fall-through time. The higher the frequency of the clock signal, the faster is the fall-through time of the overall system. However, this fall-through time does not degrade the data rate when reading or writing.

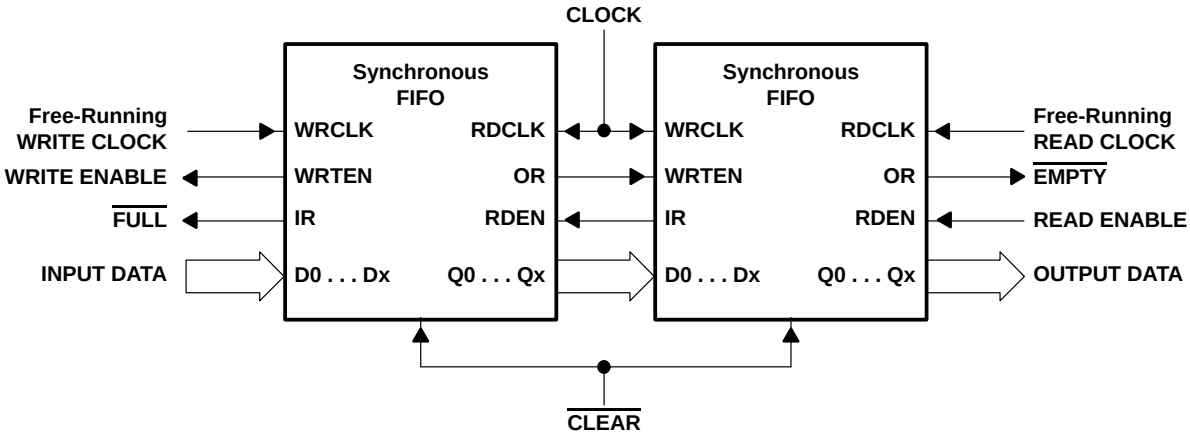

Figure 21. Extending Memory Depth of Synchronous FIFOs

Application Examples

In this section, a number of application examples illustrate the versatility and performance of TI FIFOs.

Asynchronous Operation of Exclusive Read/Write FIFOs

In use, the SN74LS224A exclusive read/write FIFO timing conditions on the write (WRITE CLOCK) and read (READ CLOCK) inputs must be maintained to ensure proper functioning of the devices. Concurrent read/write FIFOs are also offered on the market that, for their empty or full state, require timing conditions between the write and read inputs that ensure error-free operation of the $\overline{\text{FULL}}$ and $\overline{\text{EMPTY}}$ flags. Figure 22 shows the timing conditions for the two signals WRITE CLOCK and READ CLOCK in exclusive read/write FIFOs. In addition to the minimum pulse widths for the two signals (WRITE CLOCK 60 ns minimum, READ CLOCK 30 ns minimum), it is necessary to ensure that alternating write and read instructions are separated by a time window of at least 50 ns.

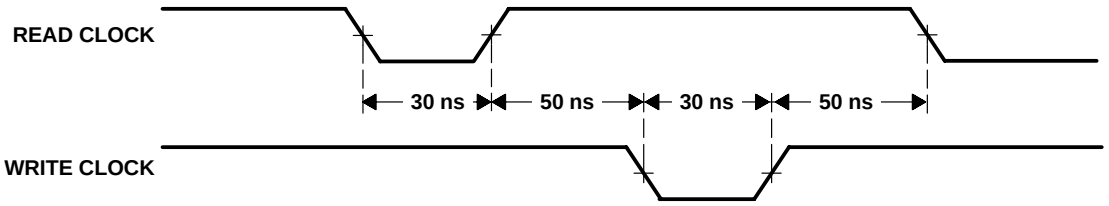

Figure 22. Timing Conditions for WRITE CLOCK and READ CLOCK

If the write and read signals originate from two sources that work asynchronously to one another, a synchronizing circuit should be used as shown in Figure 23. This ensures correct operation even if the two signals appear at the same time.

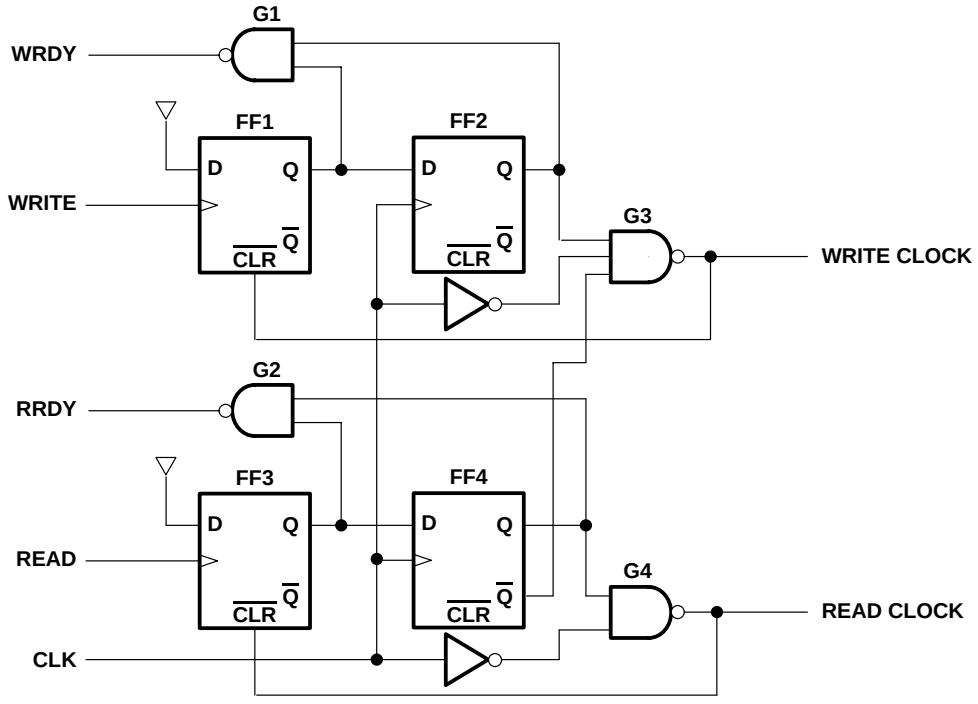

Figure 23. Synchronizing Circuit for Generating WRITE CLOCK and READ CLOCK Signals

The positive edge of the WRITE signal first sets flip-flop FF1. Flip-flop FF2 is set by the next edge of the CLK signal. If the clock signal goes low again, the WRITE CLOCK signal appears on the output of gate G3. Its length is determined by how long the clock signal is low level. The circuit for generating the READ CLOCK signal works in the same way. To avoid conflicts if the WRITE and READ signals appear at the same time, the READ function has higher priority. To produce this, the Q output of flip-flop FF4 disables gate G3 when the READ CLOCK signal is being generated, and the WRITE CLOCK signal is delayed by one clock period. There are also the WRDY and RRDY outputs, which are low level during a write or read cycle.

Figure 24 shows the timing in the circuit. In this case, the WRITE and READ signals appear simultaneously. As mentioned previously, the READ function has higher priority; first, the READ CLOCK signal is generated, then, in the next clock period, the WRITE CLOCK signal is generated.

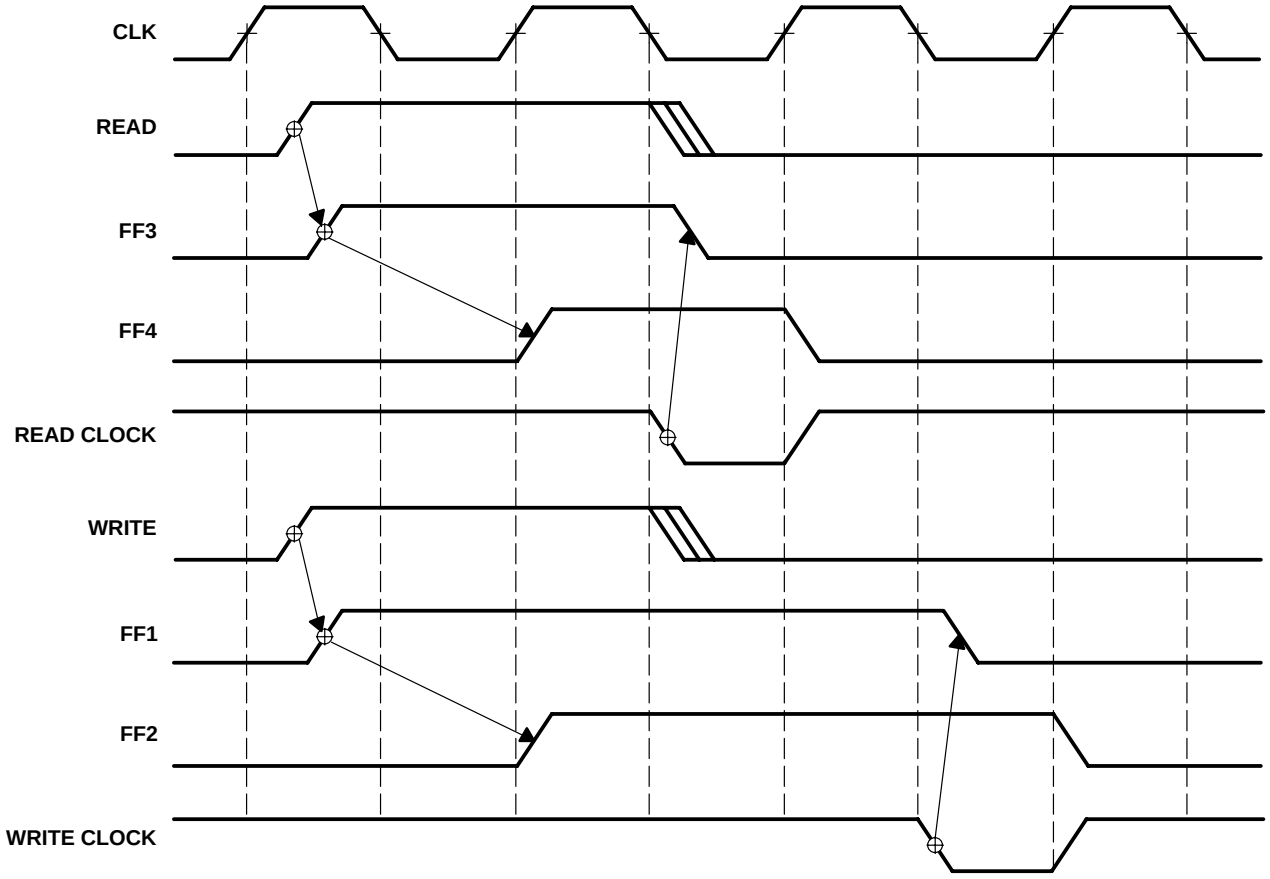

Figure 24. Timing of Signals in Synchronizing Circuit

Finally, the maximum possible frequency of the CLK signal must be determined. The WRITE CLOCK signal, according to the data sheet, should be at least 60 ns in length. Thus, the minimum time (t_{wL}) during which the clock signal must be low is defined. According to Figure 22, there must be a time window of at least 50 ns between the WRITE CLOCK and READ CLOCK signals. This corresponds to the minimum time (t_{wH}) that the clock signal must be high.

The READ and WRITE signals are asynchronous to the CLK signal, so the setup and hold timing conditions of flip-flops FF2 and FF4 are not maintained. But if flip-flops of the SN74ALS series are used, the time (twH) is sufficient to prevent metastable states that can occur with these flip-flops (MTBF > 1000 years).

Therefore, if t_{wL} and t_{wH} are 60 ns each, the maximum clock frequency becomes $1/(t_{wL} + t_{wH}) = 8$ MHz. This, in turn, means that the maximum write frequency and the maximum read frequency are 4 MHz.

Connection of Peripherals to Processors

Modern processors are often considerably faster than peripherals that are connected to them. FIFOs can be used so that the processing speed of a processor need not be reduced when it exchanges data with a peripheral. If the peripheral is sometimes faster than the processor, a FIFO can again be used to resolve the problem. Different variations of circuitry are possible, depending on the particular problem.

In some cases, the processor reads input data over a unidirectional peripheral, for example, from an A/D converter (see Figure 25). A FIFO can buffer a certain amount of input data, then use an interrupt so that the processor reads the data. This interrupt can be triggered by a HALF FULL, ALMOST FULL, or $\overline{\text{FULL}}$ flag.

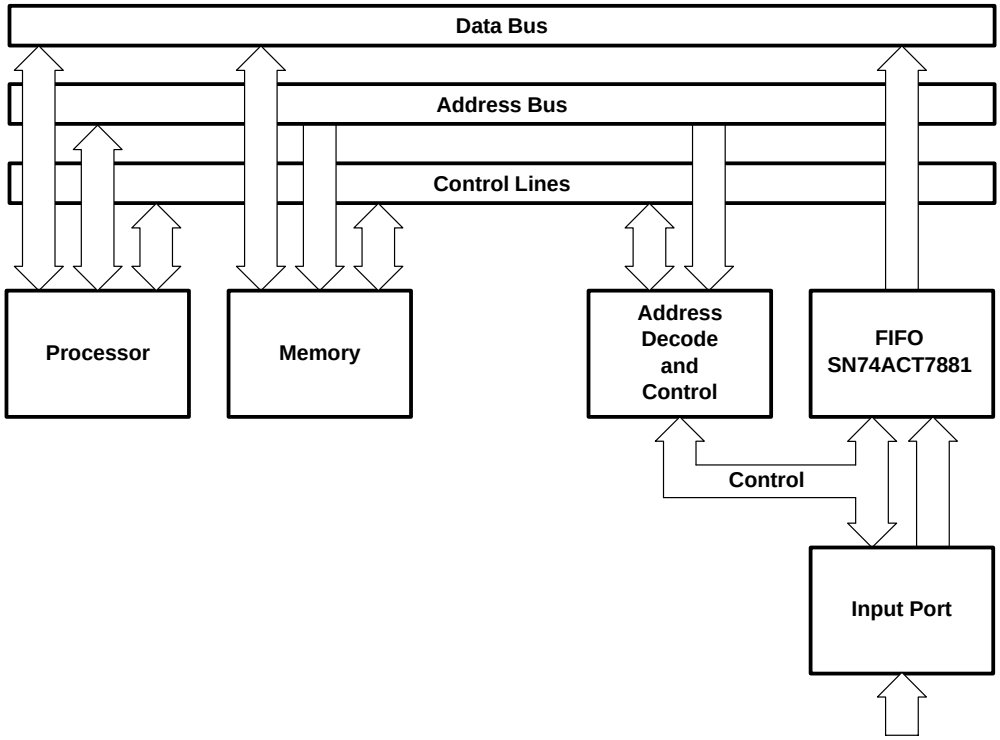

Figure 25. Connection of Unidirectional Peripheral With the SN74ACT7881 FIFO

Often the connected peripherals are bidirectional, like a parallel port, a serial port, a hard-disk controller, or the interface with a magnetic-tape drive. In these cases, there is the possibility of using a bidirectional FIFO like SN74ACT2235. Two independent FIFOs are implemented in this device, each with a word width of 9 bits and memory depth of 1024 words. Figure 26 shows a block diagram for such an application.

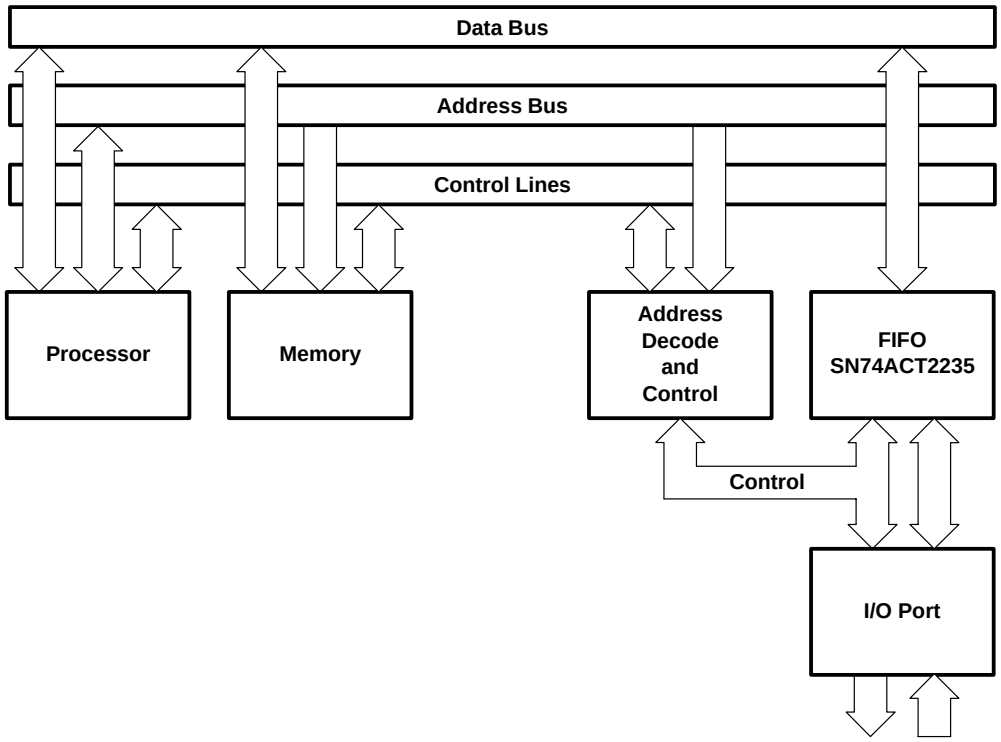

Figure 26. Connection of Bidirectional Peripheral With the SN74ACT2235 FIFO

The SN74ACT2235 also can be combined with a DMA controller. Without a FIFO, the DMA controller normally requests from the processor control of the bus for the entire duration of data transfer. The processor must wait for the end of the transfer and cannot work in the meantime. A FIFO (see Figure 27) permits block transfer of the data. The FIFO collects the incoming data. When it has stored enough data, the FIFO uses the HALF FULL, ALMOST FULL, or $\overline{\text{FULL}}$ flag to request data transfer to RAM of the DMA controller. So, the latter requires control of the bus only during data transfer from FIFO to RAM, after which control can be returned to the processor. The data are transferred block by block from the FIFO into the RAM.

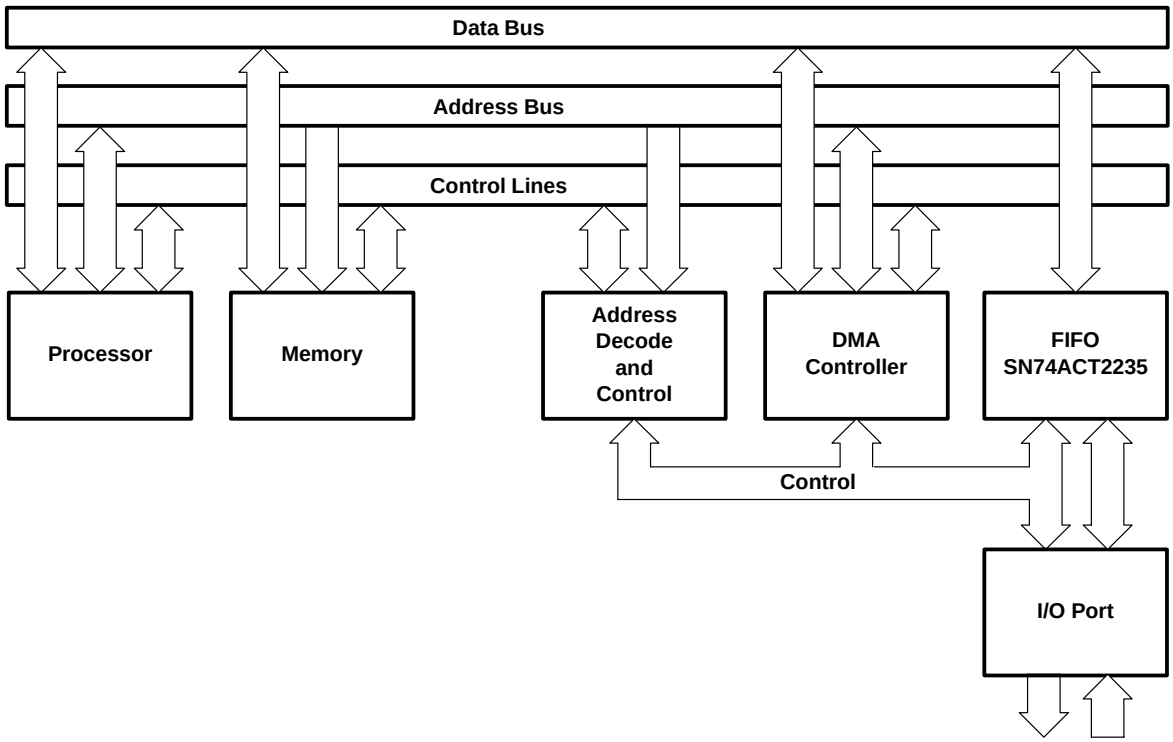

Figure 27. Connection of Bidirectional Peripheral With DMA and SN74ACT2235 FIFO

A FIFO can be used in the same way to optimize data transfer from a processor or RAM to a peripheral.

As a practical example of the connection of a fast peripheral to a processor, consider the compression of a digitized video signal with the TMS320C30 signal processor. The video signal is digitized by an A/D converter (ADC), then the picture information contained in the video signal is compressed by the signal processor. In addition to picture information, a video signal (see Figure 28) includes pulses for horizontal and vertical synchronization, as well as the porch. But only the pure picture information is of interest for further processing by the TMS320C30 signal processor.

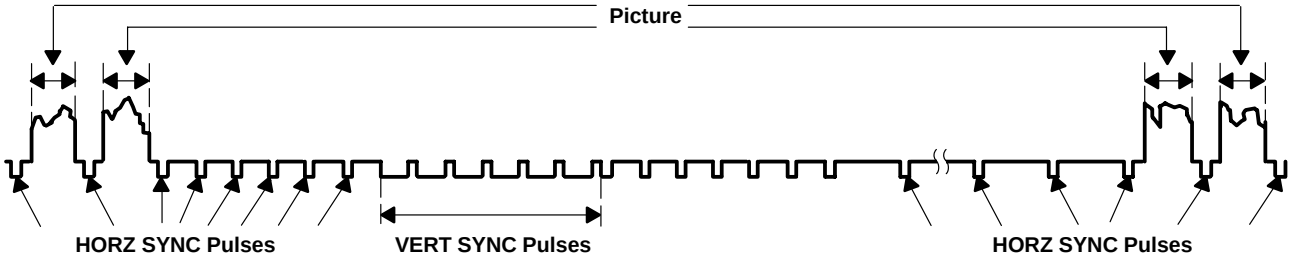

Figure 28. Video Signal With Picture Information and Porch

The data rate of the digitized picture information is too high for direct acceptance by the TMS320C30. But, because the picture information accounts for only part of the video signal, the time of one scanning line is quite sufficient for receiving the data. If a FIFO is used to buffer the data, the computing time that is gained can even be made use of to compress the data.

The circuit shown in Figure 29 first digitizes the video signal with the aid of the ADC. The clock generator produces the timing for the ADC from the video signal. The cycle generator transfers the clock signal to the FIFO only for digitized picture data, not for data originating from the porch or synchronization. Once there are enough data buffered in the FIFO, the HALF FULL flag starts data transfer by means of the direct memory access (DMA) controller.

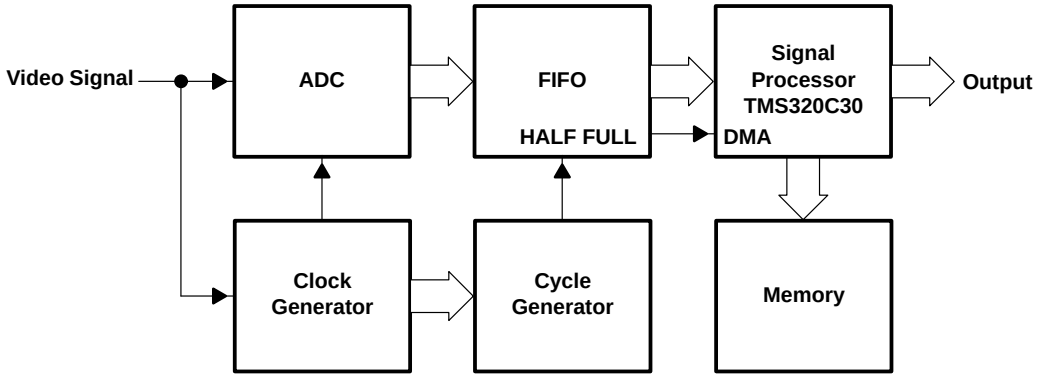

Figure 29. Digitizing and Compressing a Video Signal With the TMS320C30 Signal Processor

Block Transfer of Data

Data are often split into blocks and transmitted on data lines. Computer networks and digital telephone-switching installations are examples of this application. If such a conversion into blocks has to be made at very high speed, it is possible only with appropriate hardware, not through software.

A simple solution to this hardware problem is the use of FIFOs (see Figure 30). A FIFO with a HALF FULL flag should be selected.

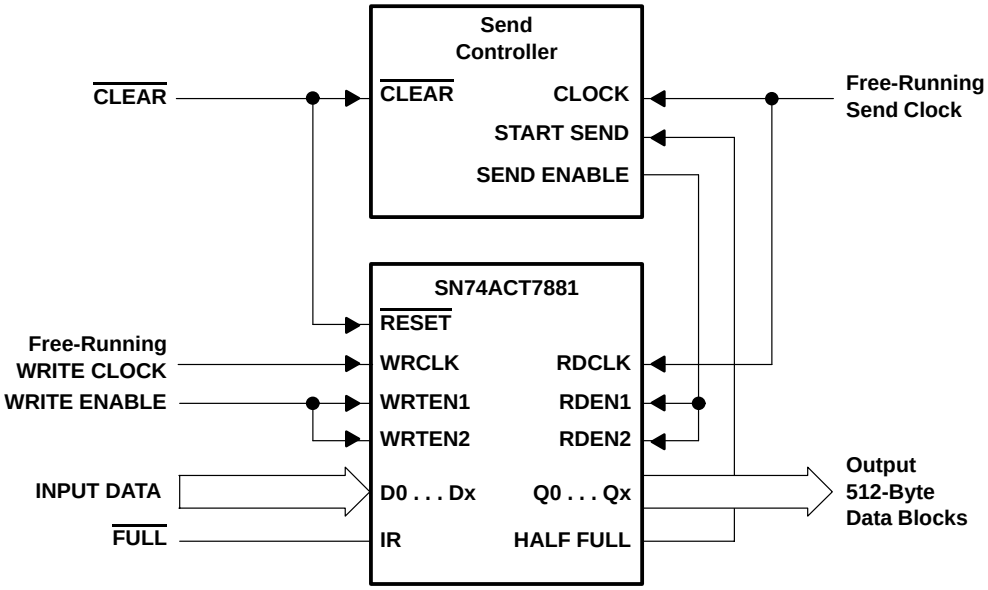

Figure 30. Block Transfer of Data With Synchronous SN74ACT7881 FIFO

Furthermore, memory depth should correspond to twice the size of a block. As soon as the HALF FULL flag is set, the send controller starts sending the data block. The send controller consists, in this case, of a counter and some gates and is very easy to implement with just one PAL. The writing of data into the FIFO can be carried on continuously and is independent of the transfer of data blocks.

Programmable Delay

With FIFOs, it is possible to implement a programmable, digital delay line with minimum effort. Because of the programmable $\overline{AF}$/AE (ALMOST FULL/ALMOST EMPTY) flag of the SN74ACT7881, only one inverter in addition to the FIFO is necessary (see Figure 31).

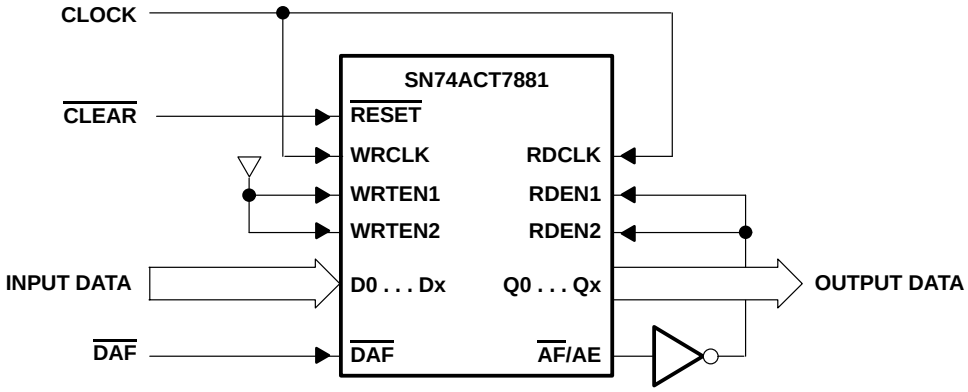

Figure 31. Programmable Digital Delay With the SN74ACT7881

First, program the $\overline{AF}$/AE flag according to the required delay of n clock cycles to the value n – 2. To obtain a 64-bit delay line, program $\overline{AF}$/AE to 62. When data words are written to the FIFO, it constantly fills. When the number of stored data words corresponds to the programmed value, the $\overline{AF}$/AE flag changes two clock cycles later to low level. The delay of two clock cycles results from the integrated, multilevel synchronization of the $\overline{AF}$/AE flag. This flag then starts, with the aid of the inverter, reading data words on the output of the FIFO. Subsequently, the FIFO works like a shift register of fixed length. The timing of the signals in this programmable digital delay is illustrated in Figure 32.

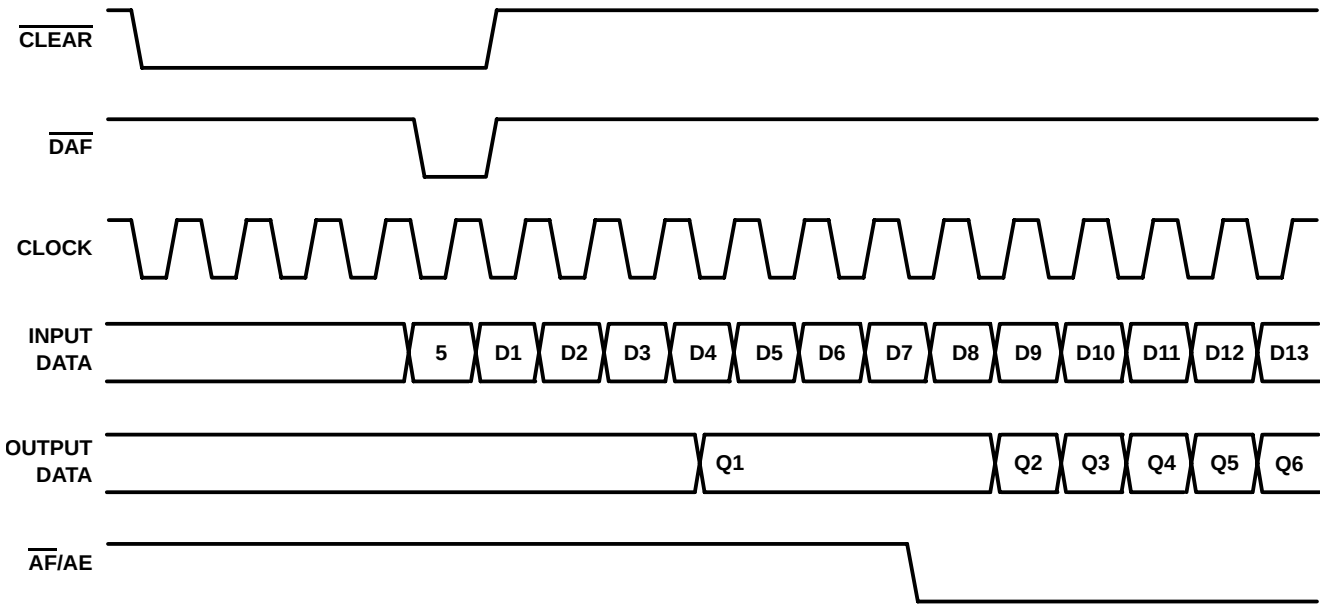

Figure 32. Timing of Signals in Programmable Digital Delay With the SN74ACT7881

Collecting Data Before an Event

Some applications require data that appear before an event to be collected. This can be solved with the aid of a circular memory whose data intake is stopped by the event. A FIFO makes an excellent basis for a device to fulfill this function. The device shown in Figure 33 is an extension of the programmable delay in Figure 31. The timing of the signals in the circuit is illustrated in Figure 34.

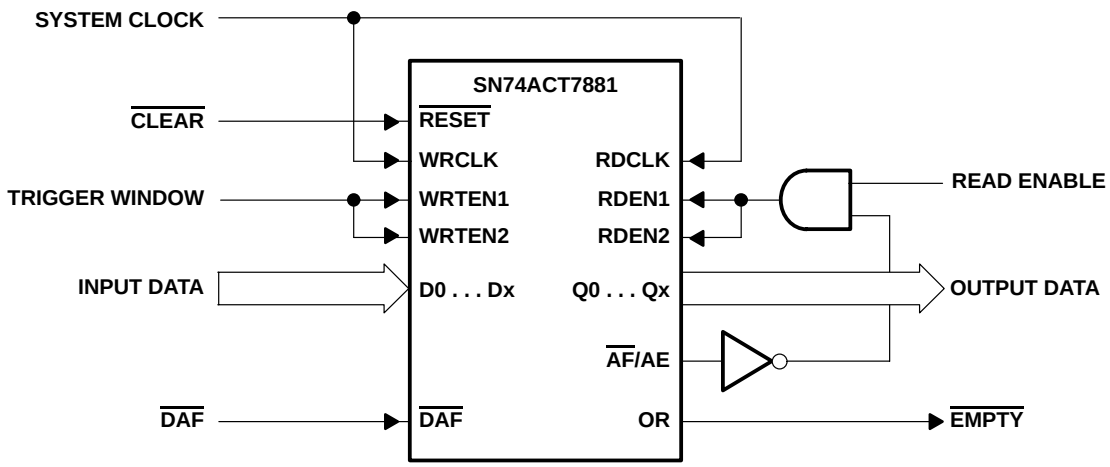

Figure 33. Collecting Data Before an Event With the SN74ACT7881

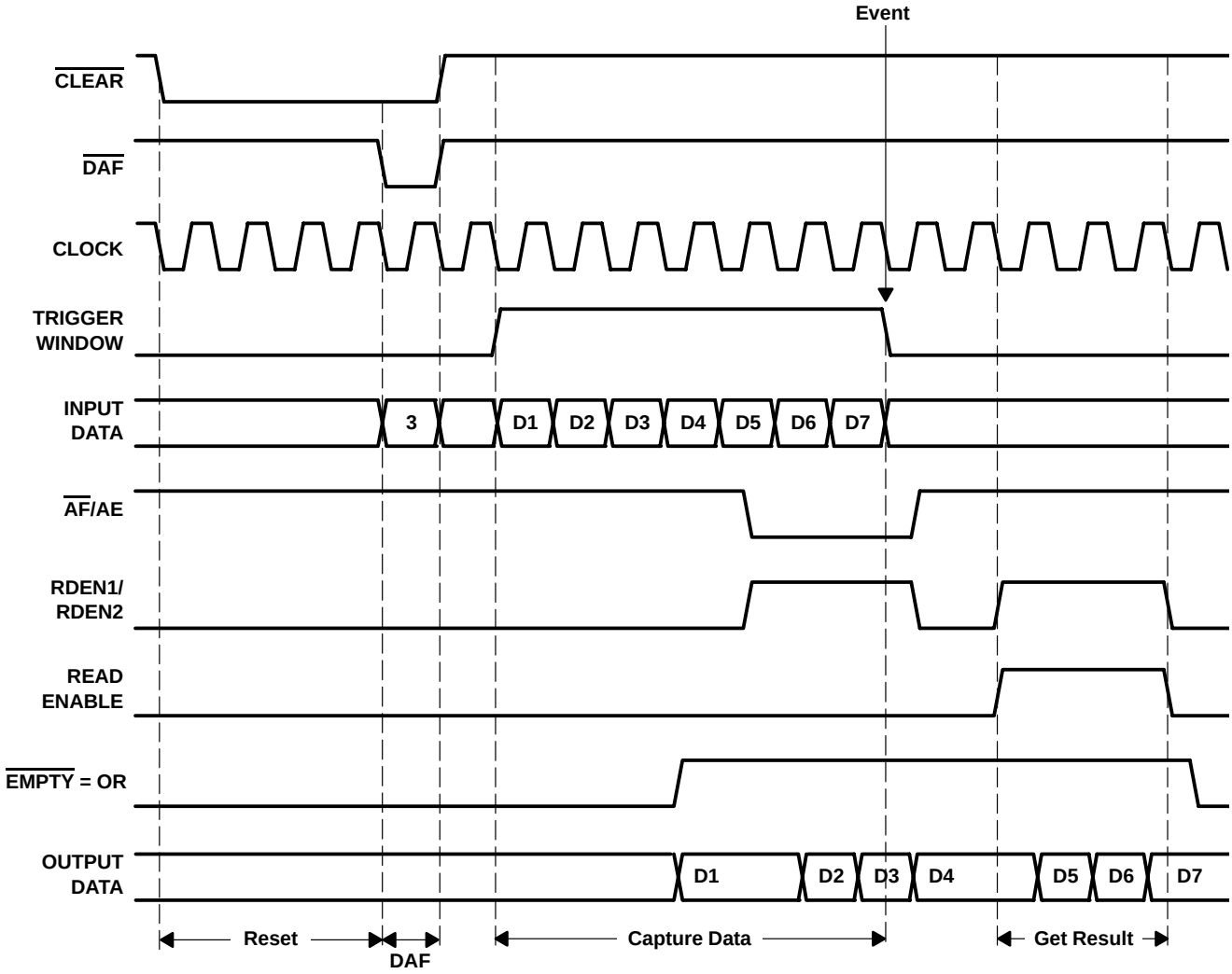

Figure 34. Timing of Device Shown in Figure 33

After the device has been reset, the $\overline{AF}$/AE flag is programmed with the aid of the $\overline{DAF}$ input signal. The programmed value n corresponds to the number of data words to be stored before the event. When the TRIGGER WINDOW input signal changes to high level, the FIFO starts to store data. Because of the integrated multilevel synchronization, the $\overline{AF}$/AE flag takes effect two clock cycles after the set value. From that time on, the FIFO begins to read out the excess data words. Thus, there always are n + 2 data words stored in the FIFO. When the event occurs and the TRIGGER WINDOW input signal changes to low level, another data word is read out, and the $\overline{AF}$/AE signal is reset. This terminates data output. The READ ENABLE input signal enables the reading out of the required data so that a data word appears on the OUTPUT DATA outputs with each rising clock edge.

Collecting Data Before and After an Event

By extension of the device in Figure 33, data words can be collected both before and after the occurrence of an event (see Figure 35); 1024 data words are always collected. With the aid of the programmable $\overline{AF}$/AE flag, it is possible to specify the number of data words that are collected before and after the event.

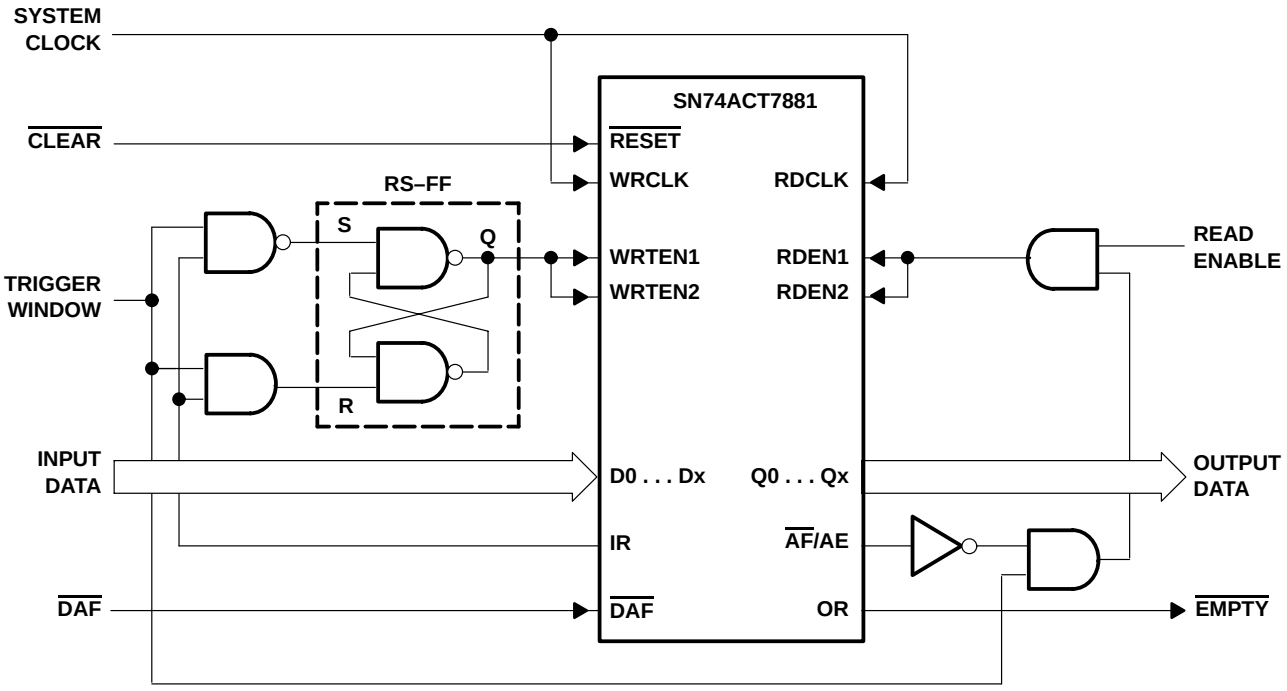

Figure 35. Collecting Data Before and After an Event With the SN74ACT7881

Until the appearance of the event, the circuit works like that in Figure 33. Here, an RS flip-flop is used to drive the WRTEN input. The TRIGGER WINDOW sets this RS flip-flop and permits data capture. When the event occurs, and the TRIGGER WINDOW signal goes to low level, data are captured until the input-ready (IR) flag is low level and the RS flip-flop is reset. Also, there is an AND gate in the signal path from the $\overline{AF}$/AE output to the RDEN1/RDEN2 input. This prevents continued reading out of data after the event. When all the data have been captured, the collected data words can be read out.

Figures 36 and 37 show the timing of the signals in this circuit with the $\overline{AF}$/AE flag programmed to n = 6. In such a case, (n + 2) = 8 data words are collected before the event and (1022 – n) = 1016 data words are collected after the event.

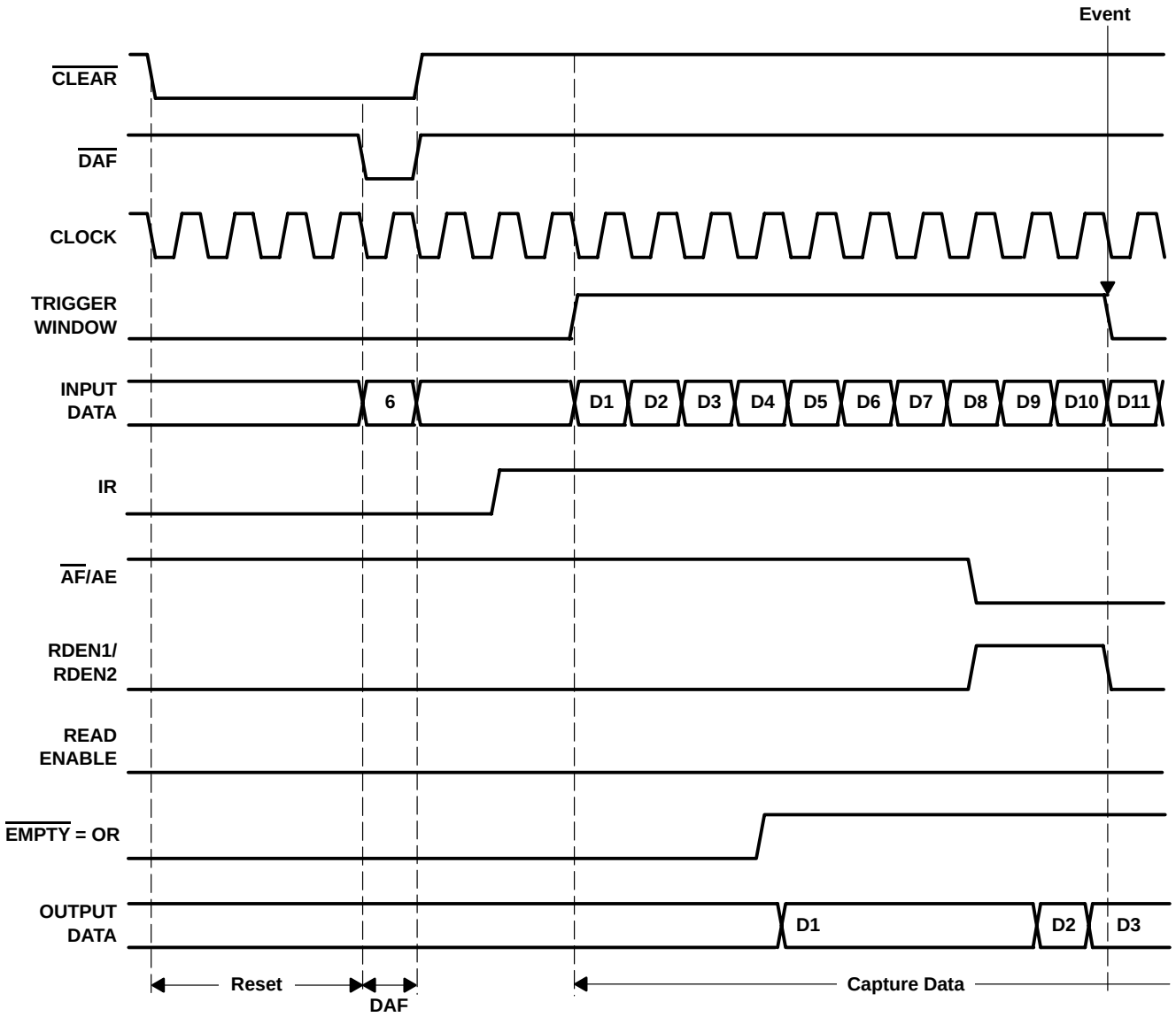

Figure 36. Timing of Signals in the SN74ACT7881 During Initialization and Start of Data Capture

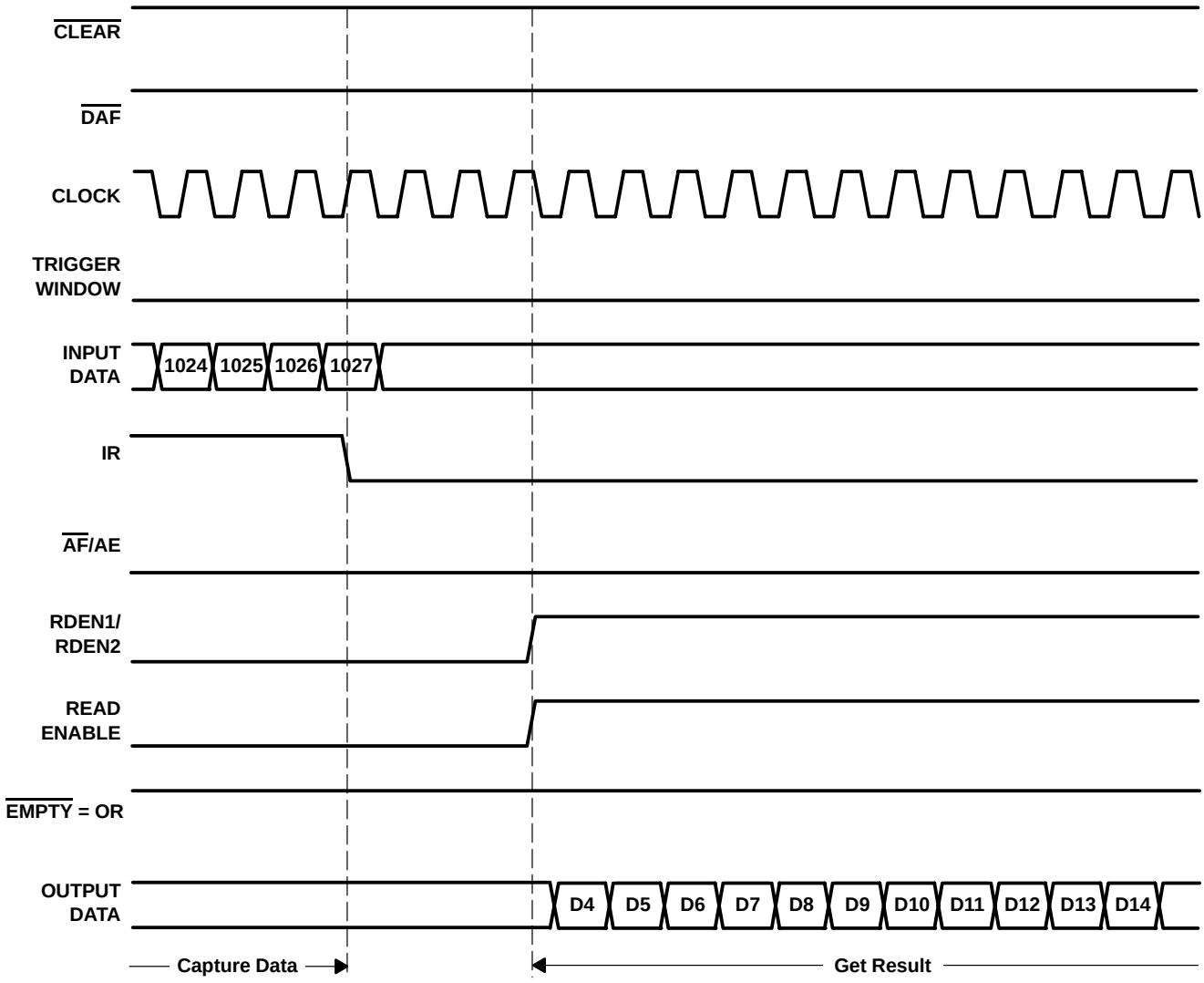

Figure 37. Timing of Signals in the SN74ACT7881 at End of Data Capture and Start of Readout

Summary

Because of their versatile possibilities of use, FIFOs present solutions to many different applications.

There are different kinds of FIFOs according to their functions. Especially attractive features are offered by asynchronous FIFOs and synchronous FIFOs.

The choice between these depends on the application for which they are intended. With asynchronous FIFOs, the flags cannot be entirely synchronized. Wherever possible, the synchronous FIFO should be given preference because it is the only FIFO that offers entirely synchronized flags. All FIFOs can be designed with different architectures. The FIFO architecture that is established firmly today is the FIFO with static memory.

Fall-through FIFOs present no advantages, but have some serious disadvantages compared to FIFOs with static memory.

TI offers only concurrent read/write FIFOs with static memory. TI FIFOs, include asynchronous and synchronous FIFOs for virtually every kind of application and every speed requirement.

Acknowledgment

The author of this application report is Peter Forstner. It was revised by Clayton Gibbs.