APPLICATION NOTE
AN-60

Designing With The IDT SyncFIFO™: The Architecture of The Future

By J. Scott Gardner

INTRODUCTION

The use of First-In-First-Out (FIFO) buffers to pass information between digital circuits with differing data rates has been a standard practice in interface design. The IDT synchronous FIFO is a new architecture designed to support high-speed systems.

THE EVOLUTION OF FIFO ARCHITECTURES

The IDT SyncFIFO can be viewed as the third generation architecture in FIFO design. The initial FIFO architecture (illustrated in Figure 1) used an architecture based on a register array indexed by special control logic which sequenced a pointer within the array. Prior to the introduction of register-based FIFOs, designers used shift registers to buffer data between systems. More general than the shift register approach, the register-based FIFO architecture is also limited in depth, due to the number of transistors needed to build each flip-flop storage element.

The second-generation FIFO introduced very large buffers based on a static memory array. The RAM-Based FIFO is shown in Figure 2. The internal RAM array is actually a dual-ported memory addressed by the use of internal pointers. These internal pointers determine which address of the RAM will provide the data during a FIFO READ or store data during a FIFO WRITE.

With the availability of large FIFOs with buffer memory as large as 4Kbytes, the need for memory management accentuated the need for external flags. These flags allow the user to monitor the amount of data in the FIFO. Most second- generation FIFOs have been enhanced to provide flags indicating a variety of FIFO conditions. Newer FIFOs, including the SyncFIFO, allow flags to be programmed to a selectable depth. Figure 3 is a block diagram of the Enhanced Asynchronous FIFOs.

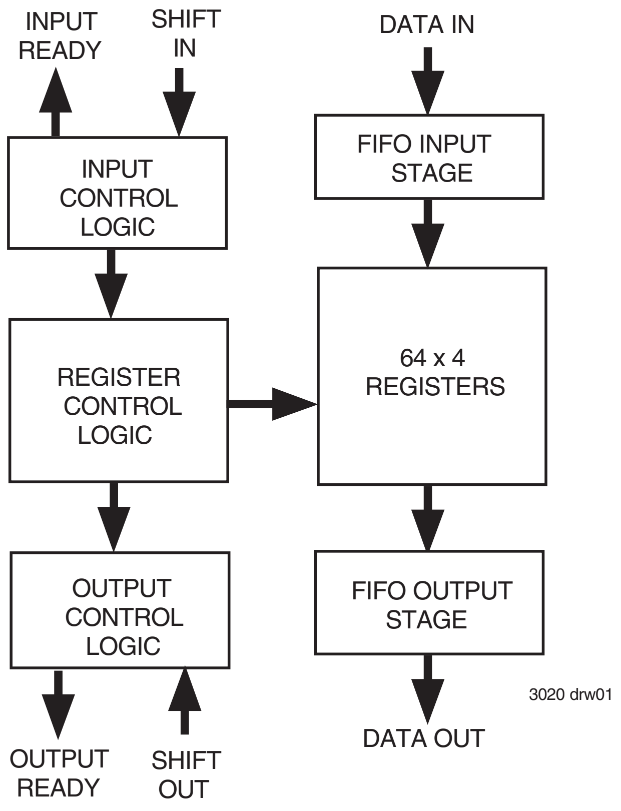


Figure 1. Register-Based FIFO Architecture-First Generation

FEBRUARY 1990

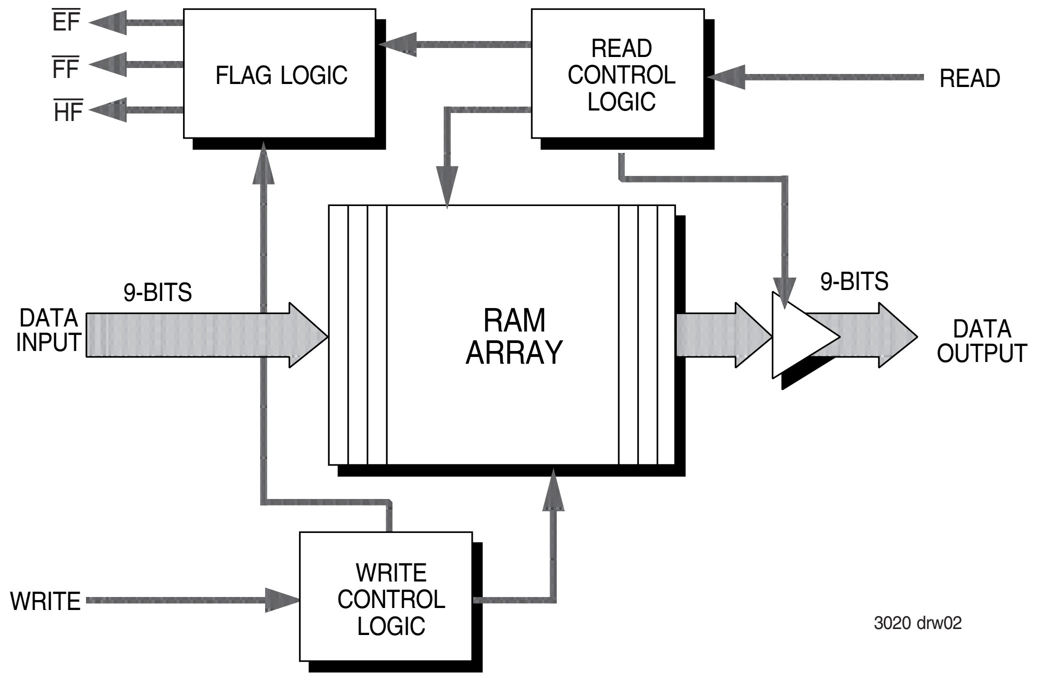


Figure 2. RAM-Based FIFO Architecture-Second Generation

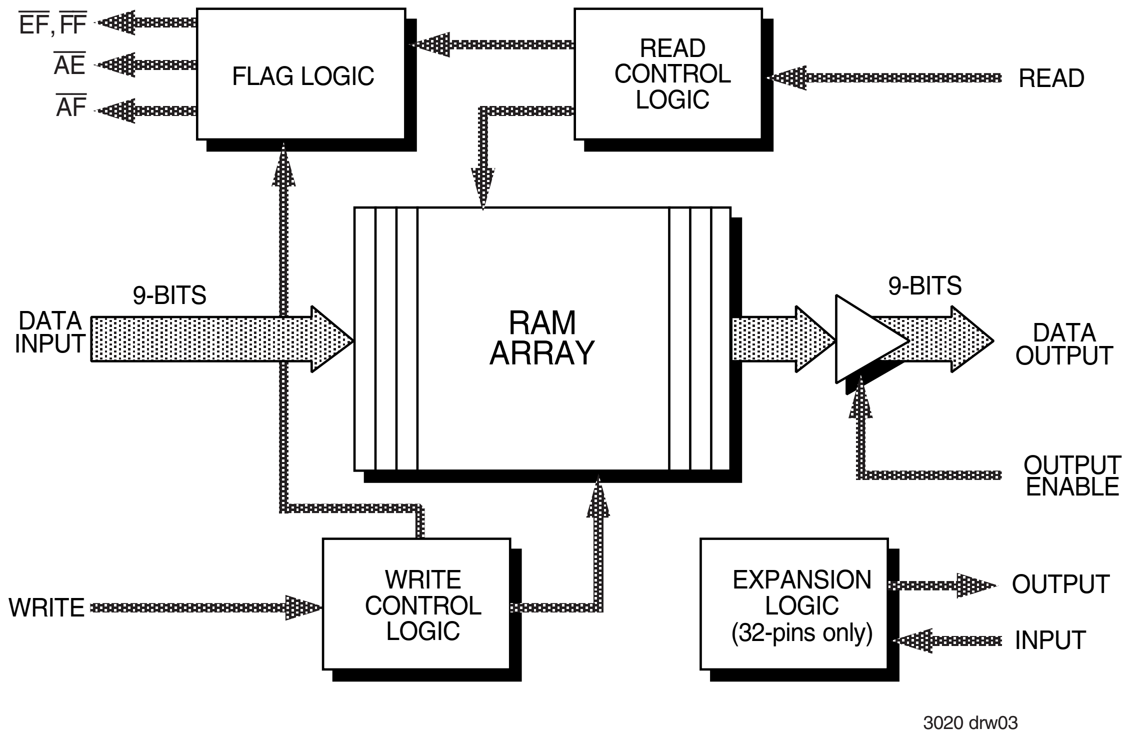


Figure 3. Enhanced Asynchronous FIFO

THE SyncFIFO ARCHITECTURE: THE ARCHITECTURE OF THE FUTURE

The SyncFIFO improves on the RAM-based FIFO architecture by adding input and output registers in the data path. These registers are controlled by independent external clocks, allowing data operations to be synchronized with the clock edges. Many system designers are designing high-speed systems using a synchronous approach, since the complexity of the control circuitry increases with speed in an asynchronous system. In a synchronous system, the amount of control logic is minimal and does not change as the system clock frequency is increased. As system clock rates approach 25MHz, it becomes more economical to use a synchronous design. (See Figure 4.)

ADVANTAGES OVER THE ASYNCHRONOUS FIFO

The concept of the synchronous FIFO is not new. Many synchronous designs requiring that data transfers occur on a clock edge use external registers with an asynchronous FIFO. The READ and WRITE control signals for the FIFO must be generated by special control logic, but the device accessing the FIFO through the registers sees the FIFO as a synchronous device. Figure 5 shows the asynchronous FIFO used in a synchronous application. The primary limitation of this approach is the performance degradation due to the long data set-up time needed by the FIFO. The performance loss is evident in operations requiring consecutive accesses, since the data set-up time must be taken into account when determining the maximum cycle time.

Using an asynchronous FIFO in a high-speed system can also require special design techniques for generating the FIFO control signals. As the cycle time is decreased in higher speed systems, the width of the READ or WRITE pulse becomes very small. The minimum pulse width for asynchronous FIFOs has been reduced to less than 20ns on faster devices. Generating accurately timed control pulses can require additional circuitry of greater complexity. Very narrow control pulses must be generated by using pulse shaping logic based on a system clock. It is sometimes difficult to generate properly timed narrow control pulses, since the timing margins become so small.

Figure 6 illustrates the simplicity of the IDT SyncFIFO interface. Passing data through the IDT SyncFIFO is based on a clock edge with a data set-up time of only 5ns and a data hold time of 1ns. A FIFO of this type allows clock rates of 50MHz. No external pulse shaping logic is required; the only control pulses required are the free-running system clock and a simple enable signal.

In systems using pipelining, the SyncFIFO can be used as a pipeline stage without external registers, as the registers that would normally be added externally for pipelining are included in the SyncFIFO. The use of pipelining can lead to even faster aggregate data rates.

REDUCED SENSITIVITY TO GLITCHES

Another problem faced by the designer of high-speed systems using fast asynchronous FIFOs is the sensitivity to glitches. A FIFO capable of responding to fast READ or WRITE pulses may recognize noise-induced glitches as valid control pulses. The minimum pulse widths are specified as worst-case values, and a designer must be careful to consider all operating conditions. A glitch which doesn't affect system operation during lab tests may become a problem at cold temperatures or higher supply voltages. Careful board design techniques or additional circuitry may be required in fast systems to reduce glitches on the READ and WRITE lines. By comparison, the SyncFIFO only recognizes READ and WRITE accesses during the transition of the clock signals, insuring increased noise immunity in the system.

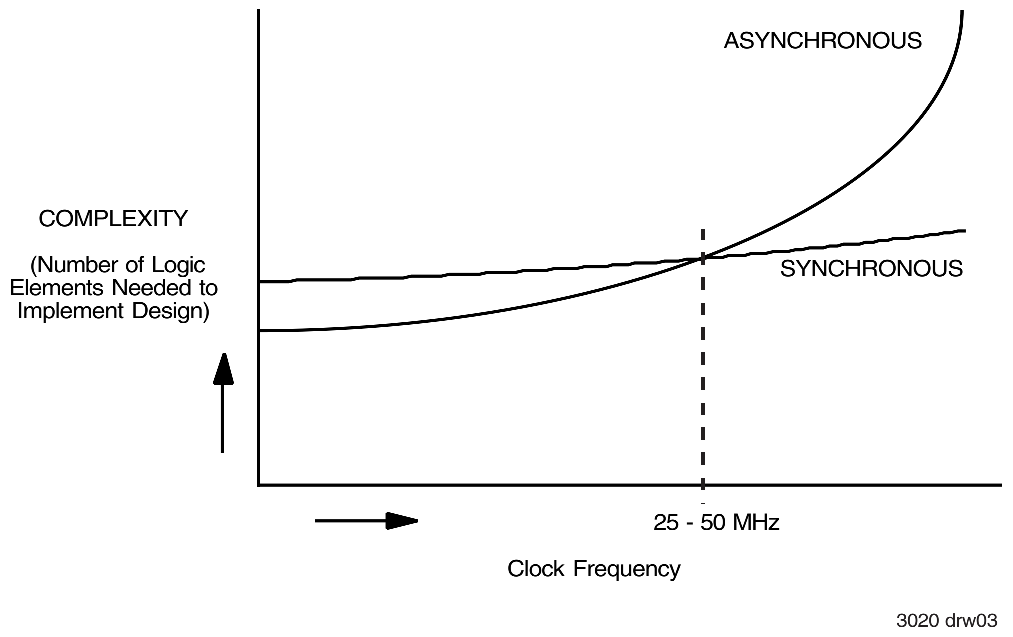


Figure 4. Increasing Clock Frequency Necessitates Synchronous Designs

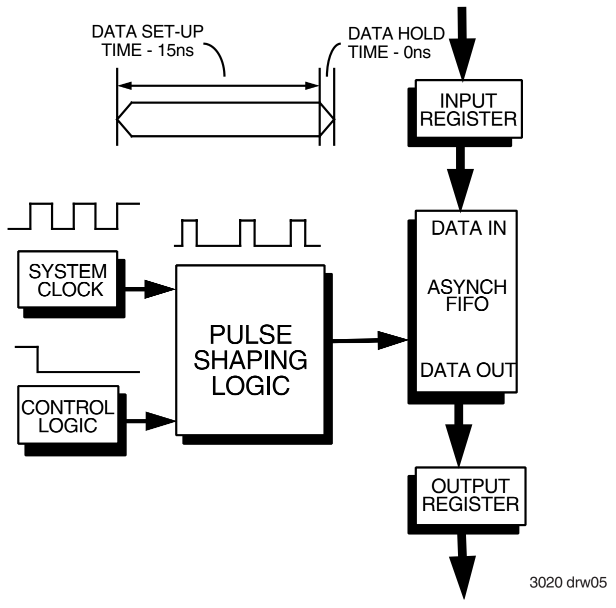


Figure 5. Typical Asynchronous Design

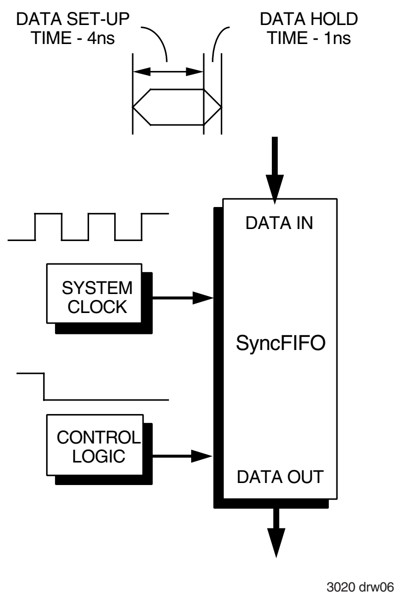


Figure 6. Typical Synchronous Design

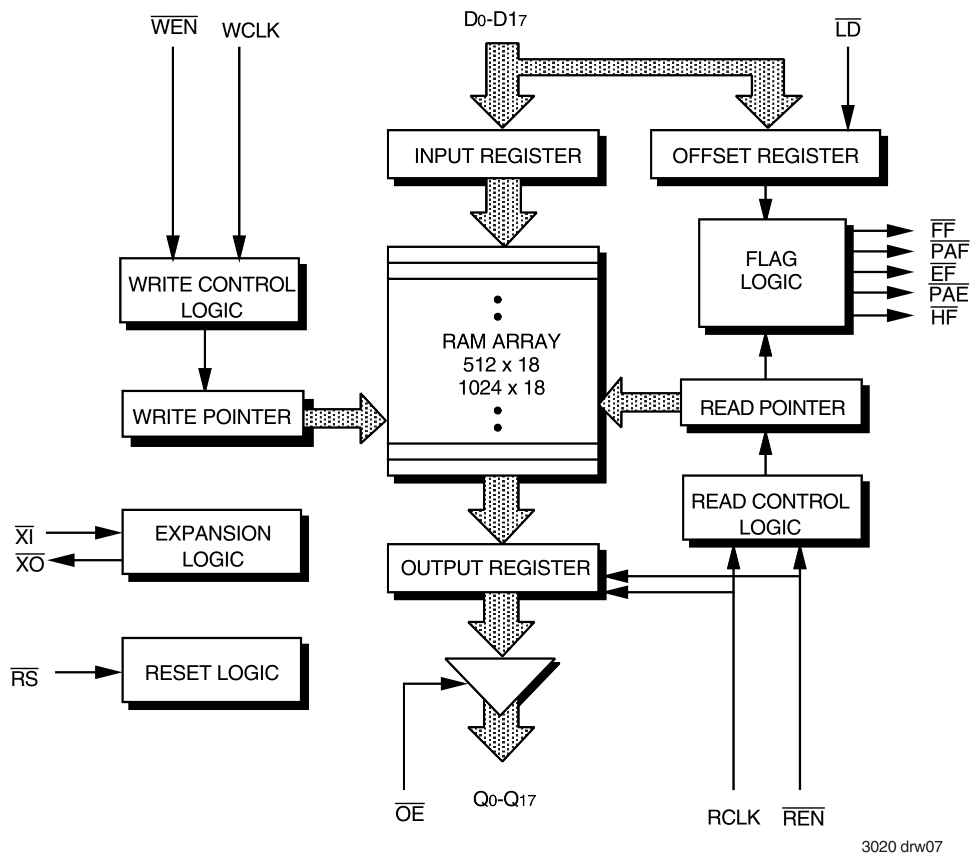


Figure 7. Block Diagram of the IDT SyncFIFO

FEATURES AND OPERATION OF THE SyncFIFO

Many of the features of IDT's SyncFIFOs are similar to the features of IDT's asynchronous FIFOs. Numerous Technical Notes and Application Notes have been published by IDT to assist the designer using asynchronous FIFOs; therefore, only the new features of the SyncFIFO will be covered in depth in this document.

The functional block diagram of the IDT SyncFIFO in stand-alone mode is shown in Figure 7. The first devices from IDT are the IDT72215 with a 512 x 18 memory array and the IDT72225 with a 1024 x 18 memory array. These devices allow for very fast throughput with read or write cycle times as fast as 20ns.

Many other sizes and word widths will be provided in subsequent devices. The speed of the IDT SyncFIFO is specified by the maximum clock rate. Both SyncFIFOs operate up to 50MHz. The synchronous nature of the architecture will allow the clock rate to increase in later products.

WIDTH EXPANSION

As in previous FIFOs, width and depth expansion are easily accomplished. Expanding the width of the FIFO is very straightforward. Basically, data passes in parallel through multiple devices. The input control signals are connected on all devices, and the status flags may be read from any device. Any word width may be attained which is a multiple of the device word size. Figure 8 shows how a 36-bit word is implemented using two 72215s or 72225s.

DEPTH EXPANSION

To expand in depth, a daisy-chain technique is used. The first FIFO in the chain is the master (designated by tying $\overline{FL}$ to ground). The remaining FIFOs in the chain are slave components (designated by tying $\overline{FL}$ to VCC).

The master device is the device which controls all the flags and must always be the first device. The flags are ignored from the other devices. In the depth expansion mode, the Half-Full Flag ($\overline{HF}$) is not available, since this pin is shared with the Write Expansion Out ($\overline{WXO}$) signal.

To control how data is passed from one device to the other, Expansion In ($\overline{XI}$) and Expansion Out ($\overline{XO}$) signals are provided to control the transfer of data. In single device mode the $\overline{XI}$ lines are tied to ground. For multiple devices, the $\overline{XI}$ and $\overline{XO}$ lines are tied together between each device. Figure 9 is an example of the SyncFIFO used in depth expansion mode.

This example shows how three devices can be chained together to provide a deeper FIFO interface. Using three 72215s, the total depth is 1536 words of 18-bit data. This daisy-chain technique can be used to achieve depths up to 32,768 words by adding more devices.

The total depth of the configuration is programmed into the master device using the depth register. The total number of FIFOs in the configuration is loaded into the 5-bit register on the third Write Clock (WCLK) while the Load ($\overline{LD}$) pin and the Write Enable ($\overline{WEN}$) are held Low. Figure 10 shows all the possible values for the depth register of the 72215 and the 72225.

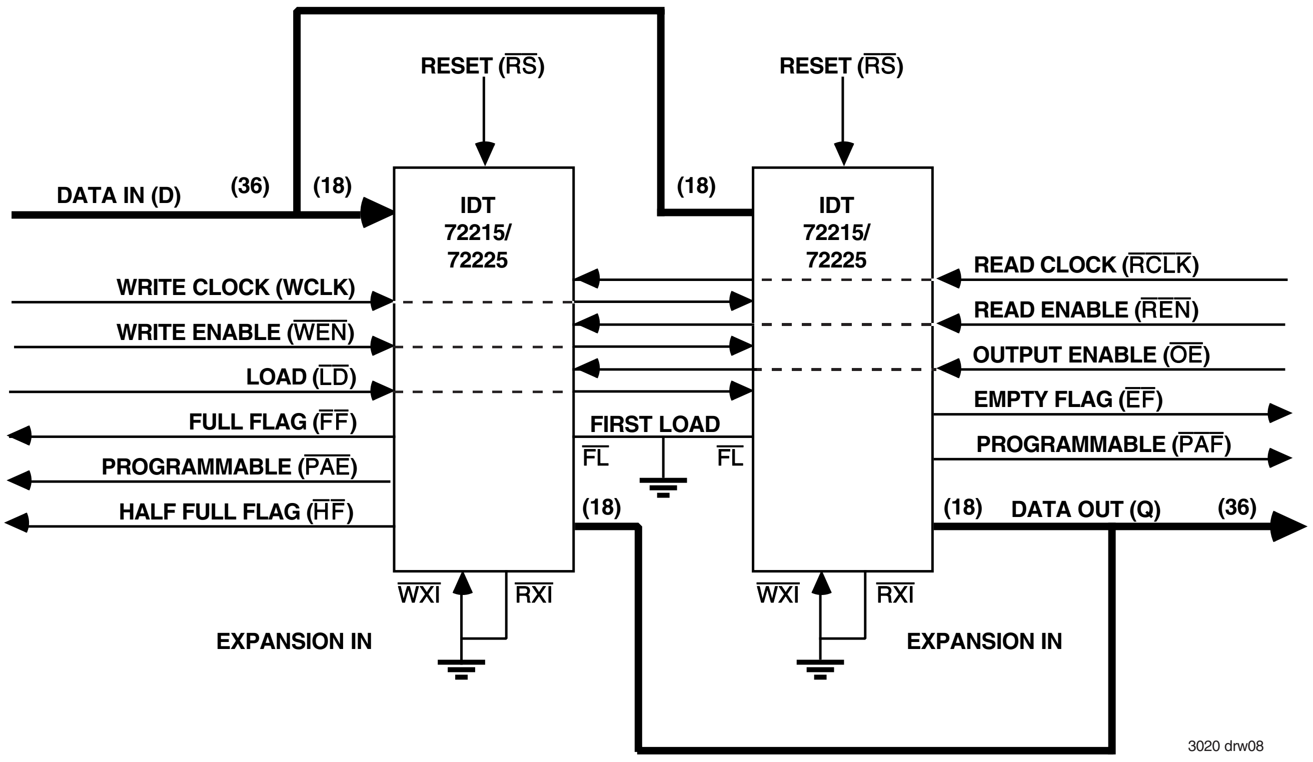


Figure 8. Width Expansion of the IDT SyncFIFO

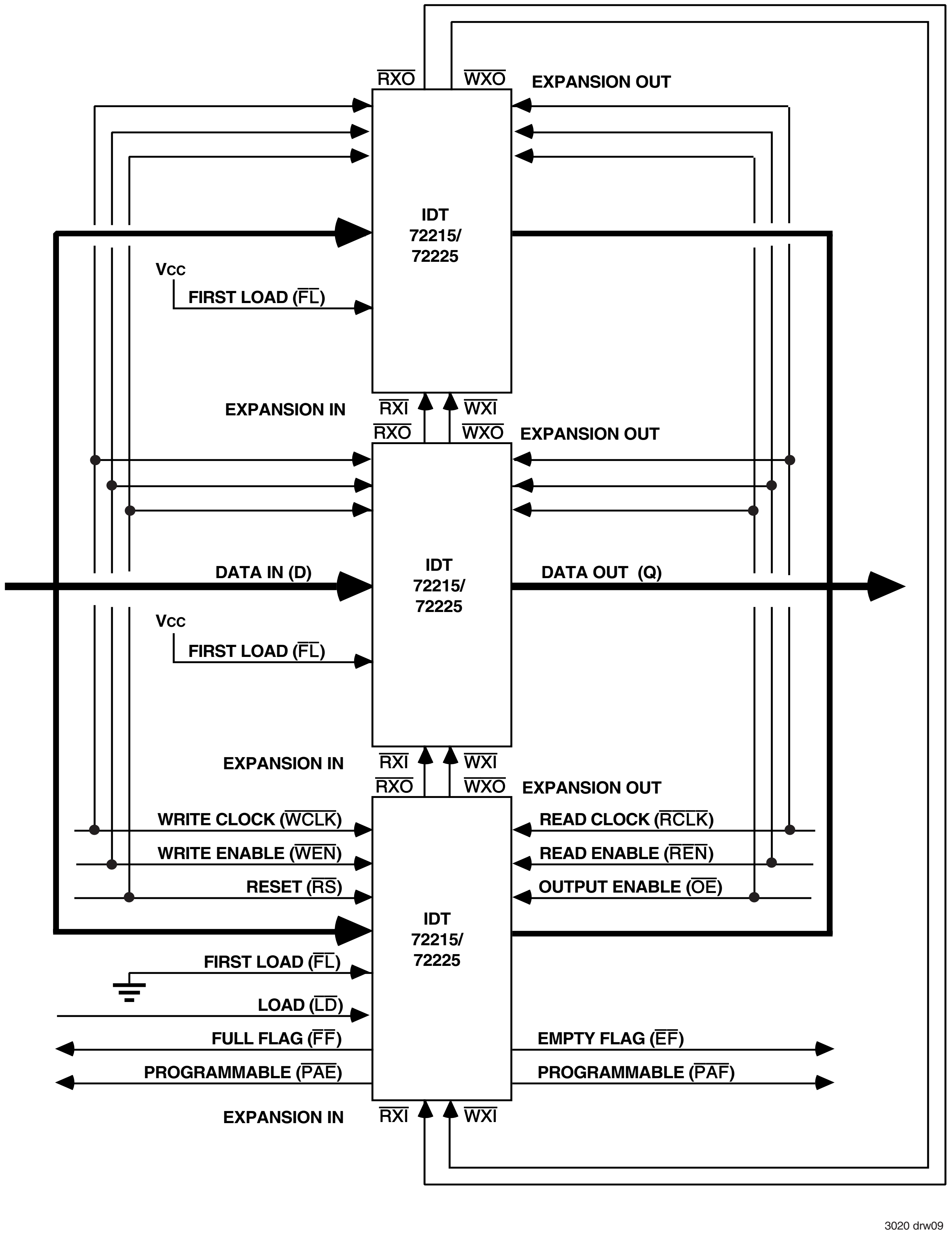


Figure 9. Depth Expansion of the IDT SyncFIFO

IDT72215		IDT72225	
DATA LOADED IN DEPTH REGISTER	TOTAL DEPTH IN EXPANSION CONFIGURATION	DATA LOADED IN DEPTH REGISTER	TOTAL DEPTH IN EXPANSION CONFIGURATION
0 (DEFAULT)	512	0 (DEFAULT)	1024
1	512	1	1024
2	1024	2	2048
3	1536	3	3072
4	2048	4	4096
5	2560	5	5120
6	3072	6	6144
⋮	⋮	⋮	⋮

3020 drw10

Figure 10. Depth Register Programming

$\overline{\text{LD}}$	$\overline{\text{WEN}}$	WCLK	SELECTION
0	0	↑	EMPTY OFFSET → FULL OFFSET → DEPTH REGISTER
0	1	↑	NO OPERATION
1	0	↑	WRITE INTO FIFO
1	1	↑	NO OPERATION

3020 drw11

Figure 10. Depth Register Programming

INDEPENDENT READ AND WRITE CLOCKS

Although the SyncFIFO handles data synchronously, the clocks are independent on each side of the FIFO. These clocks could even be free-running system clocks with different frequencies. The IDT SyncFIFO handles the transfer of data between the two systems, simplifying the timing issues normally associated with a system of this type. It is, however, possible to use the same clock for both sides of the FIFO.

USING FLAGS FROM THE IDT SyncFIFO

The flags available on the IDT72215 and IDT72225 are the Empty Flag ($\overline{\text{EF}}$), the Full Flag ($\overline{\text{FF}}$), the Half-full Flag ($\overline{\text{HF}}$) and two programmable flags. The Programmable Almost Empty ($\overline{\text{PAE}}$) and the Programmable Almost Full ($\overline{\text{PAF}}$) flags can be set to any location by the user.

These flags differ from the flags available on the latest asynchronous FIFOs in that they are updated on clock transitions. The $\overline{\text{EF}}$ is tied to RCLK and the $\overline{\text{FF}}$ is tied to WCLK. The three other flags are updated by either clock, depending on their current state (see data sheet). The impact of this difference will be discussed in the next section.

As with the asynchronous FIFOs, the flags operate based on the value of the internal pointers. Two separate pointers are maintained, a read pointer and a write pointer. When the last word is read from the FIFO, the read pointer equals the write pointer, and $\overline{\text{EF}}$ is asserted. When the write pointer reaches the last location in the FIFO and data is written, then $\overline{\text{FF}}$ is asserted. Likewise, the half-full flag is asserted whenever the difference between the Read pointer and Write pointer is ≥half the size of the FIFO RAM array.

The programmable flags use offset values which are programmed into internal registers. For the $\overline{\text{PAE}}$ flag, the signal will be asserted when the Read pointer is n locations less than the Write pointer. As an example, suppose the FIFO is being used as a 175-word frame buffer. It is necessary to notify the processor when a frame has been received, but data may continue to arrive in the buffer. The Empty Offset Register would be programmed with the value 175. Using this value, the $\overline{\text{PAE}}$ flag would go High after 175 writes to the FIFO. The processor could receive this signal as an interrupt to read out the 175-word data block.

The $\overline{\text{PAF}}$ flag can be used to signal the processor after (FULL - m) writes to the FIFO. For instance, a certain system might take some time to respond to a signal from the FIFO and begin clocking out data. To notify the processor 6 write clocks in advance of the FIFO becoming full, a value of 6 is written into the Full Offset Register. This allows the user to optimize his system for the depth of the FIFO.

To write a value into one of the offset registers, $\overline{\text{WEN}}$ and $\overline{\text{LD}}$ are set Low. The data on the input data bus is written into the Empty Offset Register on the first Low-to-High transition of WCLK. On the second Low-to-High transition of WCLK, the Full Offset Register is written with the data inputs. The third clock transition programs the depth register. Figure 11 shows the manner in which the internal registers are programmed.

DESIGN CONSIDERATIONS

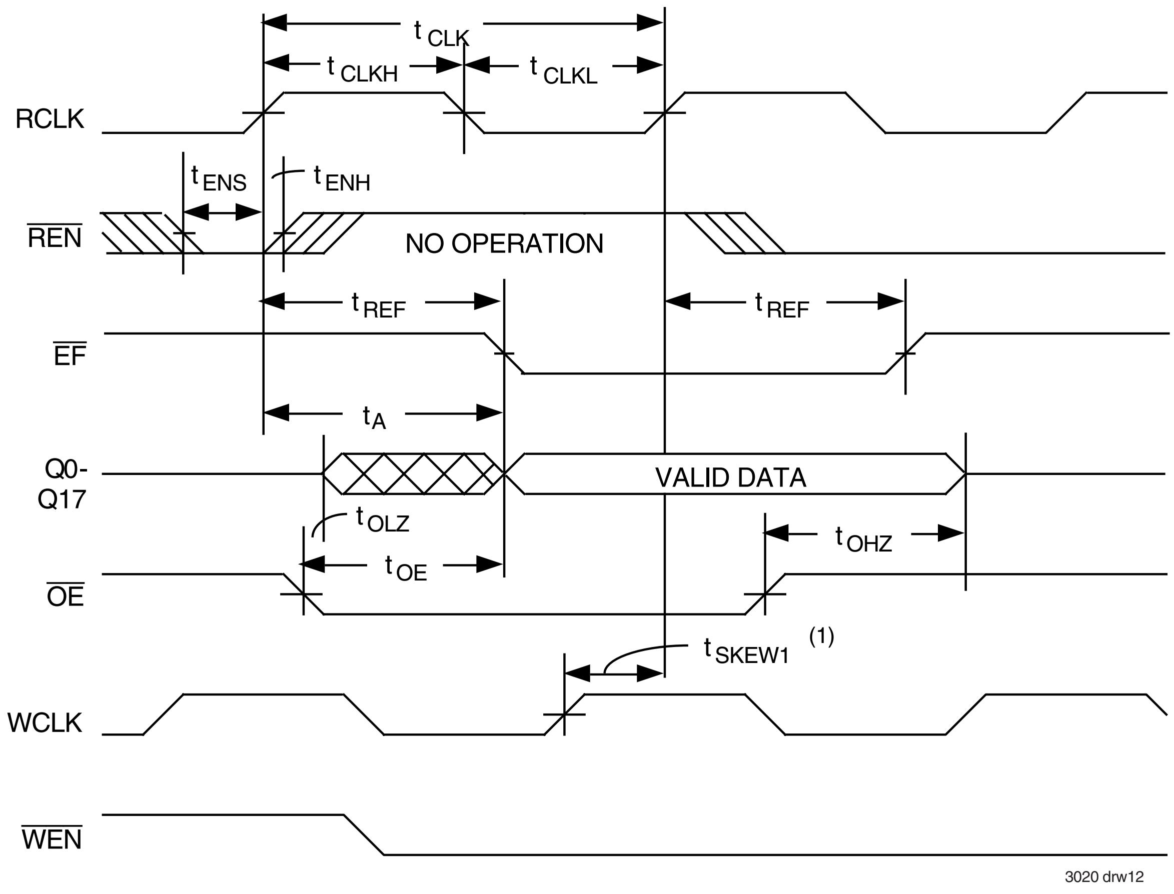
The simplicity of the interface to the SyncFIFO makes it an ideal candidate for new designs, especially when higher throughput is required. If the design is made synchronous at the outset, later speed improvements to a system do not require a redesign of the FIFO control logic.

One of the design considerations for the SyncFIFO concerns the manner in which the input and output registers interact with the internal RAM array. It is important to note that the data goes into the input register on the Low-to-High transition of the WCLK. During the first write to the FIFO, data is also stored into the internal RAM array on the same Low-to-High transition that loads the input register. This avoids the presence of a write latency cycle on the first Write.

To determine when data can be clocked out of the FIFO, the amount of skew between the WCLK and the following RCLK will determine if sufficient time has been allowed for the new data to be stored in the RAM. Once data has been clocked in using WCLK, the data will be available in the internal RAM for access by the read port after tSKEW1 ns (see Figures 12 and 13 for read and write cycle waveforms, and Figure 14 for the skew specifications). If this skew timing is not met, an extra cycle of latency will be required for clocking data from the FIFO.

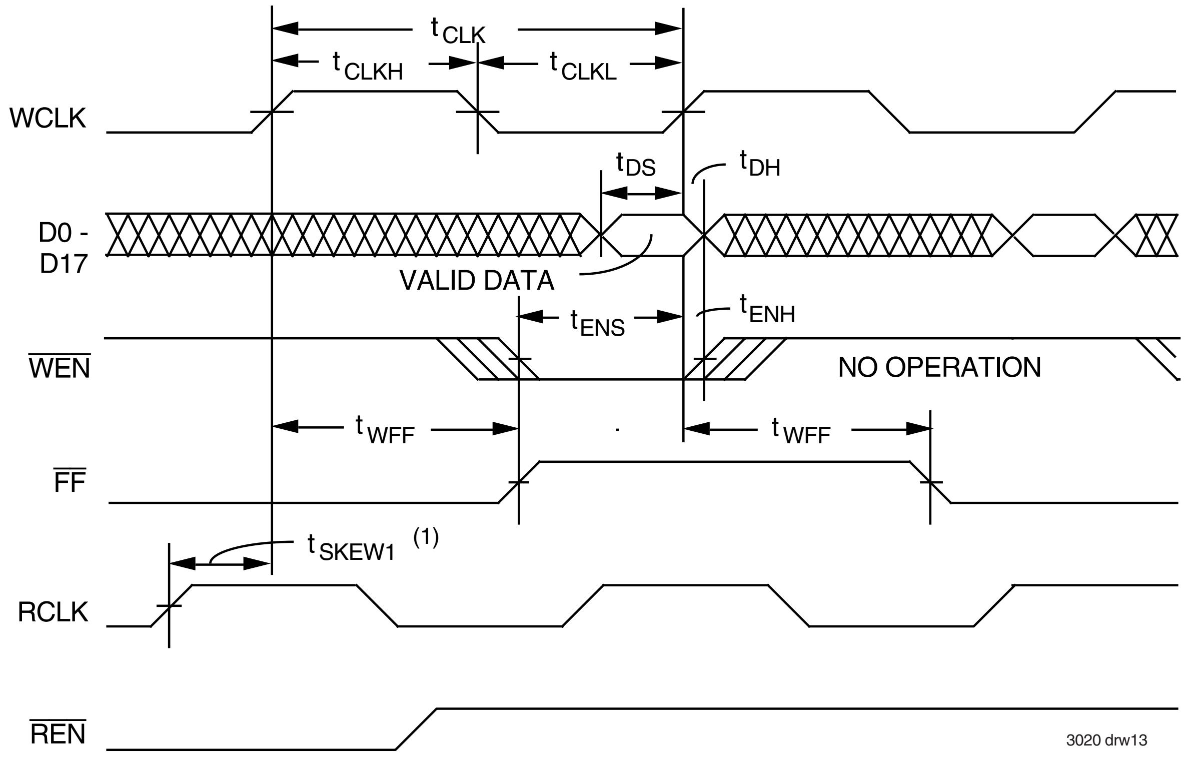
The output register of the SyncFIFO adds a full cycle of read latency before data is available on the output pins. This latency is a result of the need to allow time for the flags to be updated. External circuitry may need extra time to respond to the flag signals. This is especially true for the $\overline{EF}$. If a single word is written into an empty FIFO, the $\overline{EF}$ will become deasserted tREF ns after the following RCLK. This assumes that the RCLK occurs more than tSKEW ns after the WCLK. If the skew time is not met, an additional RCLK cycle will be required before the $\overline{EF}$ can be asserted. The data will be available on the output pins tA ns after the next RCLK.

To help clarify the timing issues, consider an example of a system which uses coincident clocks by tying the RCLK and the WCLK lines together. The $\overline{WEN}$ and $\overline{REN}$ lines are used to control the transfer of data. If a single word is written into the FIFO on the WCLK, it will take two additional RCLKs before data will appear on the output pins. The coincident RCLK obviously does not meet the skew timing requirement. The first RCLK after the coincident WCLK/RCLK will update the flags and the second RCLK will affect the data transfer to the output pins.



NOTES:
1. tSKEW1 is the minimum time between a rising WCLK edge and a rising RCLK edge for $\overline{EF}$ to change during the current clock cycle. If the time between the rising edge of WCLK and the rising edge of RCLK is less than tSKEW1, then $\overline{EF}$ may not change state until the next RCLK edge.

Figure 12. Read Cycle Timing



NOTE:
1. t_{SKEW1} is the minimum time between a rising RCLK edge and a rising WCLK edge for $\overline{FF}$ to change during the current clock cycle. If the time between the rising edge of RCLK and the rising edge of WCLK is less than t_{SKEW1}, then $\overline{FF}$ may not change state until the next WCLK edge.

Figure 13. Write Cycle Timing

SYMBOL	PARAMETER	COM'L 72215 L20 72225 L20 MIN.	MAX.	COM'L & MIL. 72215 L25 72225 L25 MIN.	MAX.	MIL. 72215 L30 72225 L30 MIN.	MAX.	COM'L & MIL. 72215 L50 72225 L50 MIN.	MAX.	UNIT
t_{SKEW1}	Skew time between Read Clock & Write Clock for Empty Flag & Full Flag	14	—	16	—	18	—	20	—	ns

3020 drw14

Figure 14. Skew Specifications

EXAMPLES OF SyncFIFO DESIGNS

The application areas of the SyncFIFO are not different from applications of asynchronous FIFOs. Figure 15 shows an application of the SyncFIFO in a graphics system. An asynchronous FIFO could have been used for this application, but the speeds typically required by graphics systems would have made the implementation difficult. The data sent to the graphics rasterizer usually occurs as blocks of data, and repetitive writes to the FIFO require very fast cycle times.

Figure 16 shows how two SyncFIFOs can be used as a high-speed bidirectional interface between two 16-bit microprocessors. As in the previous example, an asynchronous FIFO might be sufficient for slow systems. The steady advance in processor speeds has led to the need for the SyncFIFO and its ability to handle the fast data rates.

In a microprocessor system the speed of the processor is very important, but of equal concern is the amount of bus bandwidth available for communicating with external devices. It is important that bus operations be accomplished as efficiently as possible. In the multiprocessing example, processor A usually needs to pass a block of data to processor B. The sooner the block of data is written into or read from the FIFO, the sooner the processor can return to its own processing tasks. The SyncFIFO is able to accommodate a block transfer rate of 50MHz or 100Mbytes/sec, including parity. Width expansion would allow for 400Mbytes/sec transfer rate in a 64-bit system. The SyncFIFO architecture will allow for the bandwidth requirements of even faster systems in the future.

USE IN ASYNCHRONOUS SYSTEMS

A SyncFIFO can be used in an asynchronous system; however, care must be taken in observing the timing considerations. Of primary concern is the minimum setup time for the $\overline{\text{REN}}$ or $\overline{\text{WEN}}$ signals. This is the time before the data can be clocked in or out of the FIFO registers. For instance, if the $\overline{\text{MemWR}}$ pulse in an asynchronous system is to be used to generate the WCLK pulse, care must be taken to insure that $\overline{\text{WEN}}$ occurs at least 8ns before the rising edge of WCLK (for a 20ns device). One method of generating the proper timing is to use a system clock to synchronize the control signal, thus insuring proper setup times.

In a simple asynchronous interface, the $\overline{\text{WEN}}$ pulse can be generated by the chip select pulse. The chip select pulse is generated by the FIFO address decoder. The $\overline{\text{MemWR}}$ signal can be used to drive the WCLK line. The data lines must be stable at least 5ns before the rising edge of the $\overline{\text{MemWR}}$ pulse.

To clock data asynchronously from the FIFO, the chip select line can be used to drive the $\overline{\text{REN}}$ line. The $\overline{\text{MemRD}}$ pulse can be inverted to provide a properly timed RCLK. The $\overline{\text{REN}}$ signal must meet the 8ns setup requirements (for a 20ns device). The extra latency cycle for reading data may be taken into account by the device reading the FIFO. The $\overline{\text{EF}}$ will go Low when the last word is stored in the output register.

It is possible to use the $\overline{\text{MemRD}}$ signal directly, but data won't be available until 12ns after the end of the $\overline{\text{MemRD}}$ pulse (worst case). The access time for the FIFO would have to include the width of the $\overline{\text{MemRD}}$ pulse. If the $\overline{\text{MemRD}}$ is inverted using a 7.5ns PAL, the asynchronous read is accomplished in about 20ns

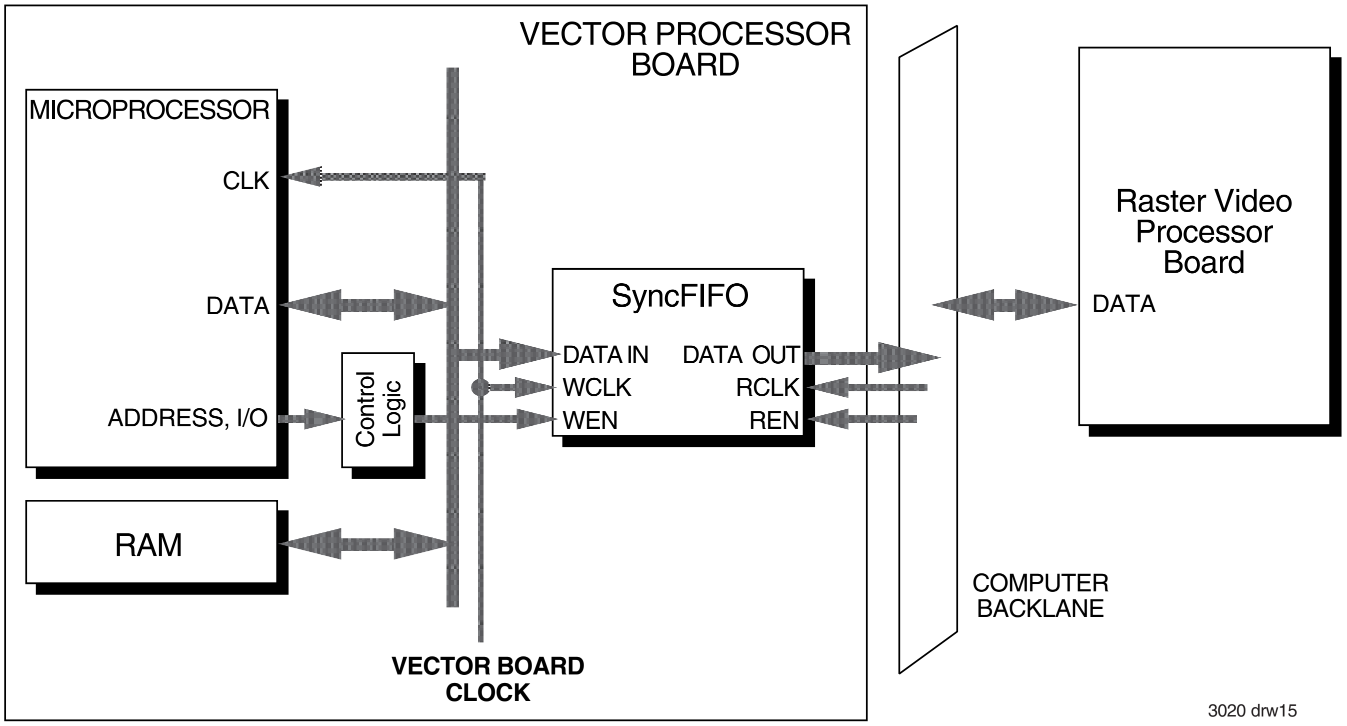


Figure 15. A Graphics System Using the SyncFIFO

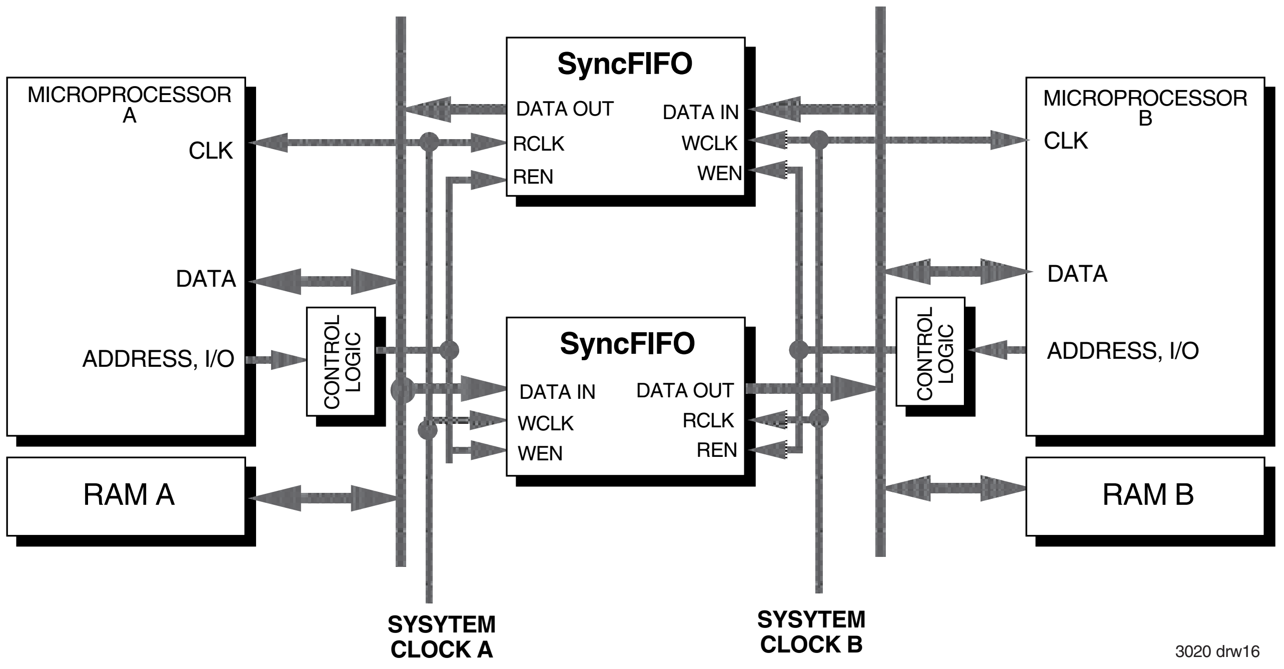


Figure 16. A Multiprocessing System Using the SyncFIFO

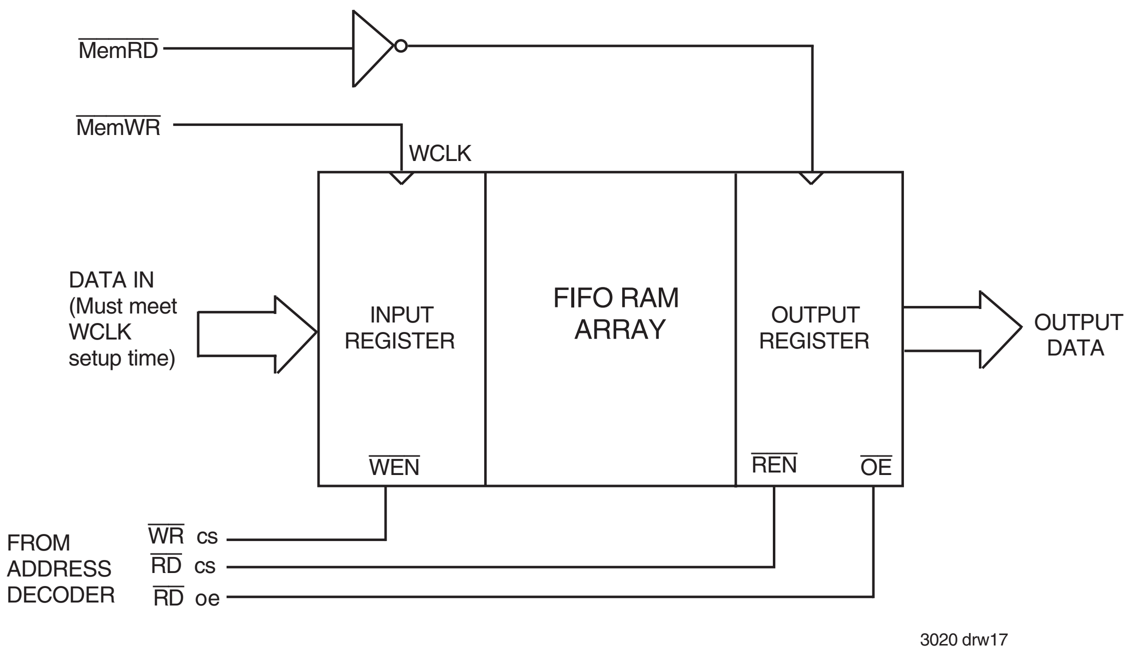


Figure 17. Using a SyncFIFO in an Asynchronous System

Figure 17 shows an example of the SyncFIFO used in a standard asynchronous system. A system of this type can be cycled much faster than previous solutions using asynchronous FIFOs. This system could also be enhanced to accommodate burst-mode data transfers available on newer processors. A simple counter could provide the burst pulse train needed to clock the data block. Also, note that the flags are synchronized with the clocks (see the next section).

HOW CLOCKS AFFECT FLAGS

Care must be taken in observing the flags in a SyncFIFO. For instance, upon reading the FIFO, the transfer of the last word from the memory array to the output register causes the assertion of the empty flag ($\overline{EF}$)—not the transfer of the last word of data out of the output register. The flags also differ from previous FIFOs in that the flags change synchronously. The flags are all updated on a clock transition.

One important consideration is that since the flags are updated synchronously, they are not updated until clocked. For instance, consider a system where the inputs are synchronous and the outputs are asynchronous. A word of data is written to the FIFO. The $\overline{EF}$ does not deassert until a clock transition on RCLK. So you could fill the FIFO without $\overline{EF}$ changing if no RCLKs are provided. One solution would be to tie any available fast clocks to RCLK. Another option would be to tie RCLK and WCLK together. (NOTE: $\overline{HF}$, $\overline{PAE}$ and $\overline{PAF}$ are asynchronous to the clocks on the 72215 and 72225. See data sheet for exact timing diagrams). So, by using the $\overline{PAE}$ instead of the $\overline{EF}$ in this case, you can signal the asynchronous side of the status of the FIFO, since this flag is updated by both clocks.

CONCLUSION

Using the SyncFIFO greatly eases the design efforts in a high-speed system. The SyncFIFO incorporates all of the enhanced features of the newest asynchronous FIFOs. These features include programmable flags, the ability to store large amounts of data, and the ability to directly drive a 3-state data bus. It is the addition of the input and output registers that makes the SyncFIFO unique. The ability to handle very fast data rates allows the IDT SyncFIFO to keep pace with the high-speed systems being designed today and the faster systems still to come. The IDT SyncFIFO truly represents the architecture of the future.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks