AN1042

Understanding Synchronous FIFOs

Author: Cypress

Associated Part Family: CY7C42x5 / CY7C42x1

AN1042 gives a brief introduction of the features and functionalities provided by synchronous FIFOs. The application note also discusses width and depth expansion of synchronous FIFOs.

1 Introduction

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.

2 Scope

This application note gives an overview of the architecture of synchronous FIFOs and discusses key features, usage guidelines, and typical applications.

This application note does not discuss features of individual Cypress sync FIFO devices but provides a general overview. For information on individual devices, review the associated device datasheet on the Cypress website (www.cypress.com).

3 Synchronous FIFO Architecture

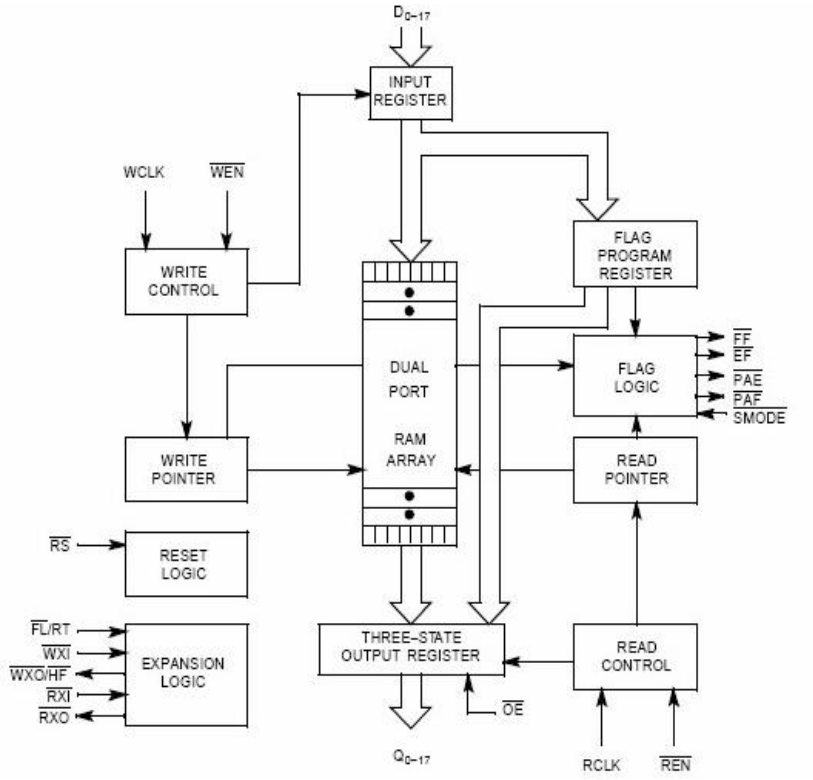
The basic building blocks of a synchronous FIFO are: memory array, flag logic, and expansion logic. Figure 1 shows the logic block diagram of a synchronous FIFO. The memory array is built from dual-port memory cells. These cells allow simultaneous access between the write port and the read port. This 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 (CY7C42x5)



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 $\overline{RS}$ pin LOW. Synchronous FIFOs require a falling edge on $\overline{RS}$. This allows devices, such as processor supervisory chips, to drive $\overline{RS}$ directly. These devices assert reset as V_{CC} ramps and hold it LOW for a minimum time to allow V_{CC} and all clocks to stabilize.

During $\overline{RS}$ assertion, read or write operations should not be attempted to the part. This can be done by deasserting the read and write enables ($\overline{REN}$, $\overline{WEN}$), 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 $\overline{RS}$.

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 ($\overline{EF}$, $\overline{PAE}$, $\overline{HF}$, $\overline{PAF}$, $\overline{FF}$) 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.

The empty flag ($\overline{EF}$) and full flag ($\overline{FF}$) are synchronous flags, meaning they are synchronized to their respective clocks. The empty flag ($\overline{EF}$) is synchronized to the read clock (RCLK) and the full flag ($\overline{FF}$) is synchronized to the write clock (WCLK). Synchronizing the flag to the respective clock eliminates the need for external synchronization. Most often, the logic that writes to a FIFO must ensure that the FIFO is not full before writing. Similarly, the read control logic examines the empty flag ($\overline{EF}$) before reading from the FIFO. The programmable almost-empty ($\overline{PAE}$) and programmable almost-full ($\overline{PAF}$) flags are synchronous on the CY7C42x1 FIFOs. The $\overline{PAE}$ flag is synchronized to RCLK and the $\overline{PAF}$ flag is synchronized with WCLK. Other FIFOs, such as the CY7C42x5, permit synchronous and asynchronous operation of the programmable flags using the $\overline{SMODE}$ control signal. For more information on programming $\overline{PAE}$ and $\overline{PAF}$ flags and flag operation, refer to the device datasheets on the Cypress website (www.cypress.com).

The half full flag ($\overline{HF}$) is asynchronous because it is not determined whether this flag will be used by the read and write control logic.

3.3 Operation as an Asynchronous FIFO

Often, users do not want to tie free-running clocks to the RCLK and WCLK pins of synchronous FIFOs, but rather pulse these clocks when data is intended to be moved into or out of the device. This is a legal operating mode for synchronous FIFOs but does require special considerations. To use a synchronous FIFO as an asynchronous one, simply pulse the RCLK and WCLK pins of the synchronous FIFO. Each rising edge of WCLK will write data into the device and each rising edge of RCLK will read data out. Read and write enables ($\overline{REN}$, $\overline{WEN}$) can be driven LOW during all phases of operation with the possible exception of reset and retransmit. Ensure that there are no reads or writes during reset and retransmit operations. This means either driving the $\overline{REN}$ and $\overline{WEN}$ HIGH or by gating the RCLK or WCLK LOW. Gating the RCLK and WCLK HIGH generates the internal clock pulses (only when its respective enable is asserted) and is not allowed.

The other concern with this mode of operation is the flag update cycle. As no free-running clocks are provided to the device, the synchronous flags are not always automatically updated. Consider an empty FIFO that then receives a number of write operations. The FIFO is no longer empty, but the $\overline{EF}$ is still asserted because there is no “flag update cycle”. To the user, it looks as if two read cycles are needed to read the first word from the FIFO: the first is the flag update cycle and the second performs the first read.

After the flag is updated, data will be read from the FIFO on each RCLK rising edge. Take care to add this extra read cycle each time the $\overline{EF}$ is asserted. Similarly, the $\overline{FF}$ requires a flag update cycle at the full boundary. Each time the FIFO is full and a read is performed, two WCLK rising edges are needed to write the next piece of data.

3.3.1 Flag Update Cycle

As the empty and full flags are synchronous, they require a rising edge on their respective clocks to update them to their most current value.

Under boundary conditions (full or empty) there is a dead cycle known as the “flag update cycle”. This dead cycle ensures that the full and empty flags are asserted for at least one full clock cycle, and can therefore be seen by the control logic that monitors it.

For example, assume we have an empty FIFO. As shown in Figure 2, a write is performed to the part on clock cycle 1 of WCLK. The part is no longer empty. Because the empty flag ($\overline{EF}$) is synchronized to the read clock, the flag will not be deasserted until the part receives a RCLK rising edge. Read operations to the device are prevented until $\overline{EF}$ is deasserted. The first RCLK rising edge (after the first write is complete – clock cycle 2 of RCLK) updates the flags and the second RCLK rising edge (clock cycle 3) reads out the first word. The initial RCLK cycle is referred to as a dead cycle or flag update cycle. This additional cycle ensures that the assertion and deassertion of the empty flag ($\overline{EF}$) will always be at least on cycle long.

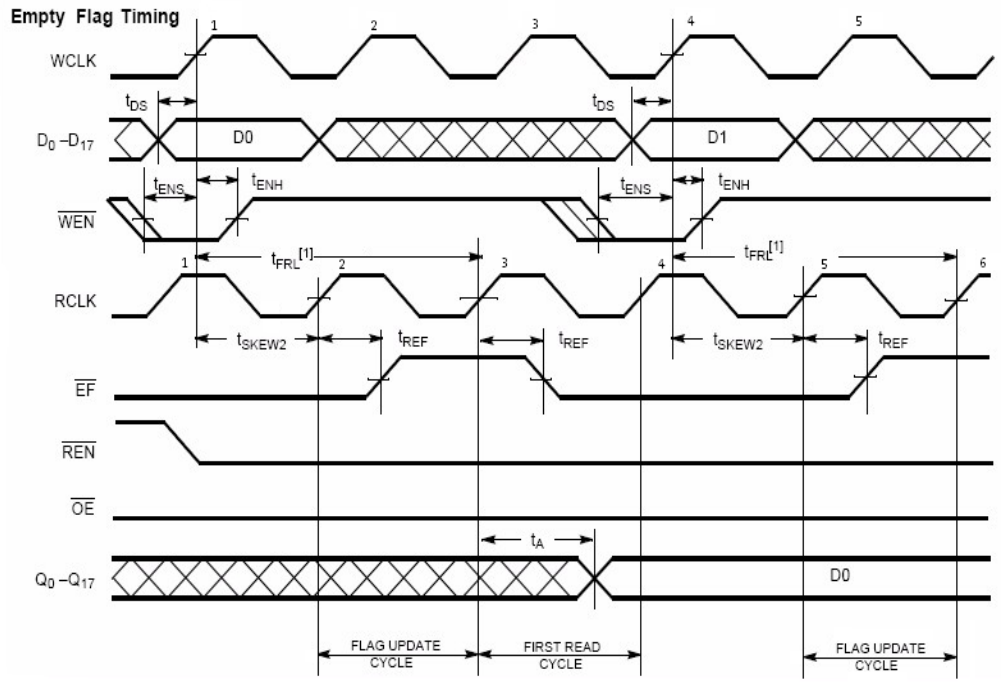
Under asynchronous conditions of RCLK and WCLK (very common for FIFO applications), the flag assertion can be infinitely small without this dead cycle.

For applications that used free-running clocks, this “dead” cycle or “flag update” cycle is transparent and they are of no concern. The read control logic will ignore the read enable ($\overline{REN}$) until the empty flag ($\overline{EF}$) is deasserted. Hence, data does not appear on the data lines even though the control signals are set up for a read on RCLK cycles 2 and 4. The first rising edge of RCLK, after $\overline{EF}$ is deasserted, will read the first word from the FIFO (provided $\overline{REN}$ is asserted – RCLK clock cycle 6).

For applications that do not use free-running clocks, the RCLK must transition from LOW to HIGH twice to read out the first word - once for the “flag update cycle” and the second edge to read out the first word.

Note The flag update cycle occurs on both the empty and full boundaries. There are no “flag update cycles” associated with the $\overline{PAE}$ and $\overline{PAF}$ flags.

Figure 2. Flag Update Cycle[1]



4 Retransmit

The retransmit feature is used to reread a block of data from the FIFO that was previously read. This feature is commonly used in serial communications interfaces. If an error occurs during transmission of data, the packet can be retransmitted from the FIFO and consequently resent through the serial media.

The retransmit feature is accessed through pulsing of the retransmit ($\overline{RT}$) pin of the FIFO. By driving the $\overline{RT}$ pin LOW, the read address pointer of the FIFO is set to the physical location, zero. Figure 3 shows the retransmit operation. Note that for the retransmit feature to operate correctly, the FIFO must first be reset before data is written to the FIFO that might be retransmitted.

Here is an example. Let us say you want to send a 1-K deep packet of data to another board. The data can be written to a FIFO and passed to a serial transceiver, which sends the data through a serial media. The FIFO is first reset, setting the read and write address pointers in the FIFO to location zero. 1-K data words are written to the FIFO. $\overline{EF}$ is deasserted and the serial transceiver device begins reading from the FIFO.

As data is read from the FIFO, the read address pointer increments until it reaches location 1024 and the FIFO becomes empty. Note that although the data has been read from the FIFO, the data is not erased from the FIFO. If a problem occurs at any point during the read process, the $\overline{RT}$ pin can be pulsed setting the read address pointer back to location zero, and the packet of data can be resent to its destination. This process can be repeated indefinitely.

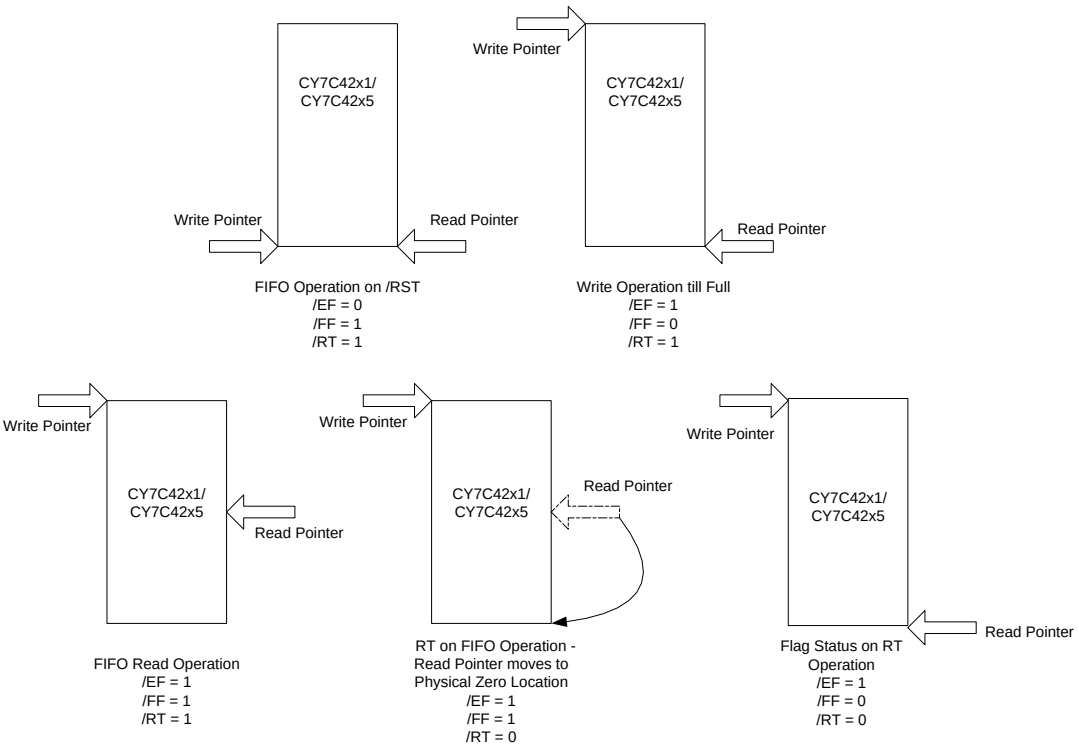
[1] When t_{SKEW2} > minimum specification, t_{FRL} (maximum) = t_{CLK} + t_{SKEW2}. When t_{SKEW2} < minimum specification, t_{FRL} (maximum) = either (2 × t_{CLK} + t_{SKEW2}) or (t_{CLK} + t_{SKEW2}). The Latency Timing applies only at the Empty Boundary ($\overline{EF}$ = LOW).

Note that the $\overline{RT}$ pin does nothing more than reset the read address pointer to location zero. The FIFO does not know where certain packets of your data are stored in the device. You must reset the device before a packet is written that may need retransmitting.

At any time during normal FIFO operation, or any time after the $\overline{RT}$ pin has been deasserted, the FIFO can perform both read and write transactions normally. No read or write operations may be performed during assertion of $\overline{RT}$.

Note Retransmit is an asynchronous operation and does not require transition on the RCLK or WCLK to operate.

Figure 3. FIFO Operation on Retransmit



5 Expansion Configurations

5.1 Width Expansion

Width expansion is used to create FIFOs with wider data paths. Two ×18 FIFOs can be width-expanded to create a ×36 FIFO, and so on. Read, write, and retransmit operations are the same for FIFOs in width expansion, and, in fact, the FIFO has no knowledge that it is used in this mode.

To expand multiple FIFOs in width, the flags must be combined to create “composite flags”. This is done by externally ANDing flags between each of the FIFOs.

Composite flags must be generated for both the empty and full flags. By combining the flags, this insures that the FIFOs stay synchronized (each contain the same number of words.) See Figure 4.

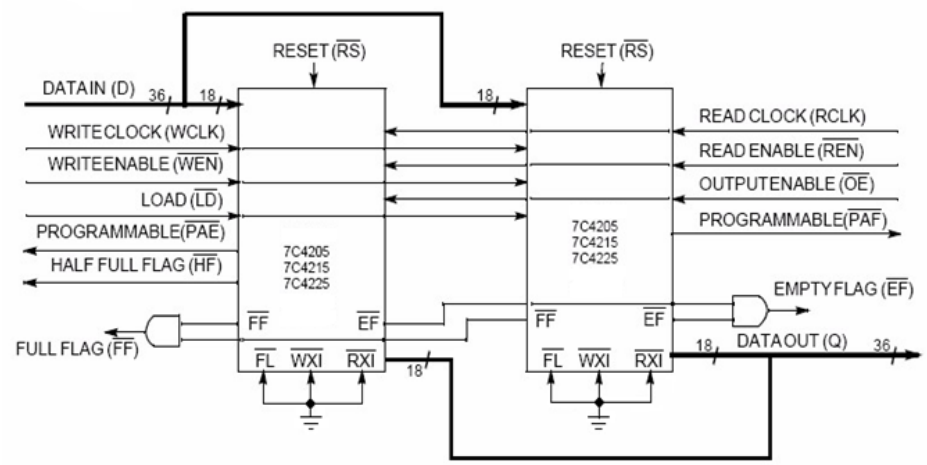
We must first understand how the width-expanded FIFOs can lose their synchronization. The root of this problem lies with the asynchronous relationship of the clocks, RCLK and WCLK. Consider an example where two FIFOs are width-expanded and each FIFO has one word in it (one location from empty). Due to the asynchronous phase relationship between the read and write clocks, it is possible that the FIFOs will receive both a read and write operation almost simultaneously. If the read operation is performed first, the part becomes empty just before a word is written to it. At this point, the device is waiting for a flag update cycle and one read cycle is essentially lost. In the other case, if the write operation occurs just before the read, the device has two words in it for a quick instance and then returns to having one word. Note that no flag update cycle is needed because the FIFO never became empty.

The width-expanded FIFOs may respond differently to this scenario and, therefore, may lose synchronization. Small clock skew and even device process variations can cause one FIFO to see the read operation first while others may see the write first. As one or more of the devices require a flag update cycle the FIFOs are now out of synchronization.

By combining the empty flags of the FIFOs to create a composite empty flag, the read enable ($\overline{REN}$) can be deasserted in the event that any of the FIFOs become empty. As long as the $\overline{REN}$ is deasserted for at least one clock cycle, all empty FIFOs get the flag update cycle they require and the FIFOs stay synchronized.

Note the same idea applies to the full flag at the full boundary. The control logic that drives $\overline{REN}$ and $\overline{WEN}$ should deassert these enables in the event that one of the composite flags becomes asserted.

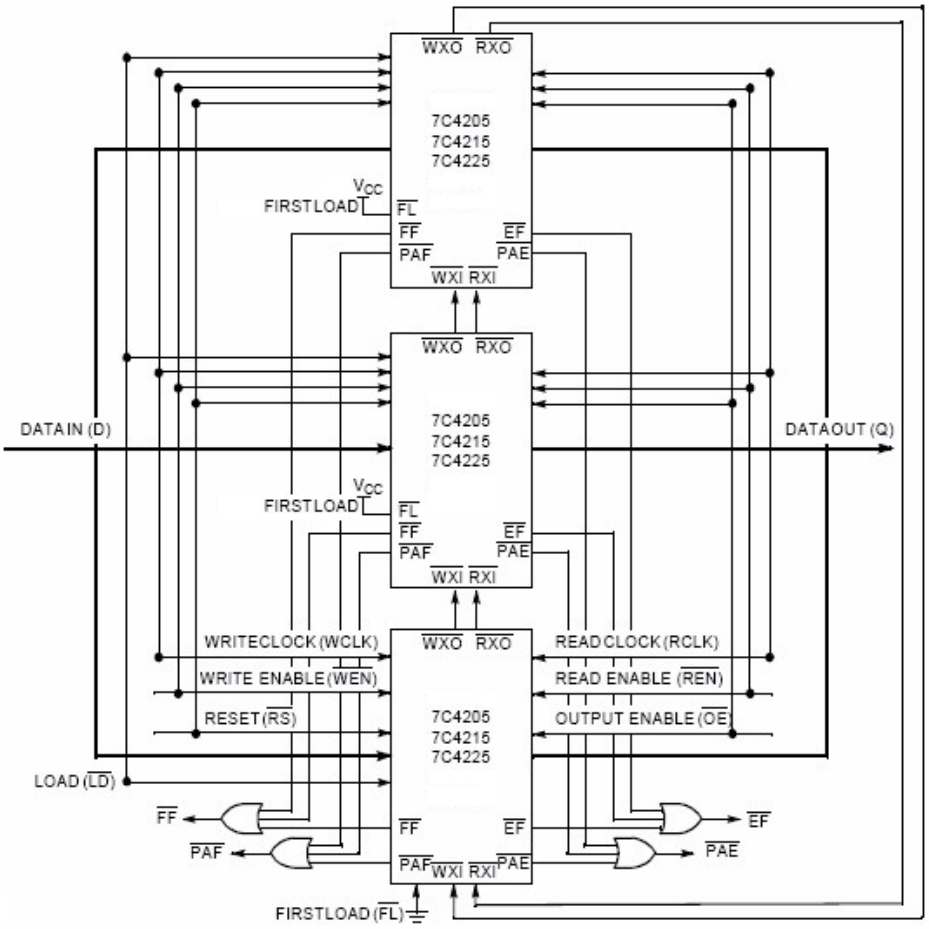
Figure 4. Width Expansion of Synchronous FIFOs (CY7C42x5)



5.2 Depth Expansion

Depth expansion is used to cascade multiple FIFOs to create a logically deeper FIFO buffer. Synchronous FIFOs use a token passing approach to depth expansion. Each of the FIFOs are connected in parallel (read and write data buses are shared), and control lines, such as $\overline{WEN}$, $\overline{REN}$, $\overline{RCLK}$, and $\overline{WCLK}$, are shared. One device is designated as the “first” device in the token chain by grounding the first load ($\overline{FL}$) pin. All other devices must tie $\overline{FL}$ to V_{CC}. When the write address pointer in the first device reaches its maximum value, an expansion pulse is driven on the $\overline{WXO}$ pin. The $\overline{WXO}$ pin of the first device is tied to the $\overline{WXI}$ pin of the next device, which sees the expansion pulse and takes responsibility for performing subsequent write operations. Similarly, when the read address pointer reaches its maximum value, it passes a token through the $\overline{RXO}$, to the $\overline{RXI}$ pin of the next device. See Figure 5 for a diagram of depth expansion configuration of the CY7C42x5 FIFOs.

Figure 5. Depth Expansion of Synchronous FIFOs (CY7C42x5)



Composite empty and full flags must be generated for proper operation. Composite flags are generated by externally ORing the flags of all FIFOs in the token passing chain. This ensures that all FIFOs must be full before the composite full flag asserts. Similarly, all FIFOs must be empty for the composite empty flag to assert. The $\overline{\text{PAE}}$, $\overline{\text{PAF}}$, and $\overline{\text{HF}}$ flags are not used in the depth expansion mode. Although each of these flags will be a correct indication of the number of words in a particular FIFO, generating composites of these flags provide information about the overall state of the depth expanded FIFO buffer.

Depth expansion can also be performed on FIFOs that do not have depth-expansion logic on chip. A ping-pong approach is used to alternately write and read data from multiple FIFOs. Writes are performed by rotating each transaction between devices. In a scenario where two FIFOs are depth-expanded, the first write is written to FIFO#1, the second write to FIFO#2, the third write to FIFO#1, and so on. Read operations are handled in the same manner.

6 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.

7 Summary

Synchronous FIFOs ideally suited to transfer data between systems operating at different speeds or using unsynchronized clock sources. They support high-speed operation of 100 MHz and can be used in the synchronous mode by tying the clock inputs to free-running clocks for optimum speed. They can also be used as asynchronous FIFOs by pulsing the clock inputs for easy integration to designs, which expect asynchronous FIFOs. Features, such as retransmit and synchronous flags, make the devices versatile and easier to use. These FIFOs can also be cascaded together in width or depth to create FIFO configurations, not available in single devices. Hence, Cypress's synchronous FIFOs are ideal for buffering and synchronizing data in high-performance applications.

Document History

Document Title: AN1042 - Understanding Synchronous FIFOs

Document Number: 001-19979

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1409104	ADMU	08/23/2007	Obtain spec# for note to be added to spec system. This note had no technical updates. Kindly replace existing .pdf file on cypress.com
*A	3075539	ADMU	11/02/2010	Added Abstract. Removed information about parts which are obsolete. Minor edits and updated in new template.
*B	3124837	ADMU	01/20/2011	Created hyperlinks for the application notes references in this spec.
*C	3561632	ADMU	03/26/2012	Updated template. Minor text edits. Removed Decoupling and Clock Termination sections.
*D	4197417	SMCH	11/20/2013	Updated the applications for synchronous FIFOs. Added figure 4.
*E	5536200	NILE	11/29/2016	Updated template
*F	5848814	GNKK	07/31/2017	Updated the Cypress logo and copyright notice.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709