Understanding Synchronous FIFOs

Introduction

Synchronous FIFOs have quickly become the FIFOs of choice for new designs. This movement to synchronous FIFOs from their asynchronous predecessors is due mainly to speed and ease of operation. However, there are also many other advantages which these devices bring such as synchronous flags, programmable almost empty and almost full flags, depth expansion, and retransmit. Synchronous FIFOs are easier to use at high speeds since they can be operated by free running clocks. Asynchronous FIFOs required read and write pulses to be generated as data is moved through the part, and generating these pulses is difficult to do at high speed. Synchronous FIFOs can be used just as their asynchronous counterparts by tying the read and write strobes to the RCLK and WCLK lines respectively. This makes migration to synchronous FIFOs very easy, even for designers who are mostly familiar with asynchronous FIFOs.

Scope

Due to the large number of synchronous FIFOs available from Cypress, this application note will not discuss features of individual part numbers, but rather discuss the general operation of and the features available with these devices. The data sheets should be referenced for a specific device to determine which of the features discussed below are supported by that device.

Applications for Synchronous FIFOs

A FIFO's largest benefit is its ability to pass data between two data buses that are asynchronous from each other. This includes buses running at different rates as well as buses that are running at the same rate, but whose clock is generated from different sources. Although two crystals may be labeled with the same value, small variations in the crystals cause them to oscillate at slightly different rates.

Often when data is passed between two boards, each board is operating from a different crystal. Incoming data must be synchronized to the local clock before it is usable. By passing that data through a FIFO, the synchronization is done automatically. This synchronization is possible since the FIFOs are built from dual ported memory cells. Dual ported memory cells allow unconstrained simultaneous access through both ports without any timing restrictions.

In many board to board communication schemes, error checking is done to insure proper transmission. Synchronous FIFOs have a retransmit feature which allows the board which is sending the data to re-send or "retransmit" the data when an error occurs.

Another popular use for FIFOs is interprocessor communication. Often processors run at different bus rates, so passing data through a FIFO allows each processor to burst data into and out of the FIFO at their maximum speeds.

FIFOs have no address lines, which saves pin count and therefore board space. Because of this, FIFOs are often used to buffer sequential data such as video or voice. Telecommunication and datacommunication information possesses this sequential ordering as well. Often FIFOs are used on the front end of each network port to synchronize incoming network packets.

Synchronous FIFO Architecture

The basic building blocks of a synchronous FIFO are the: memory array, flag logic, and expansion logic. The memory array is built from dual ported memory cells. These cells allow simultaneous access between both ports of the memory, the write port and the read port. This simultaneous access gives the FIFO its inherent synchronization property. There are no restrictions regarding timing or phase between accesses of the two ports. This means simply, that while one port is writing to the memory at one rate, the other port can be reading at another rate totally independent of one another. This also allows for optimization of speed at which data is written to and read from the memory array. Synchronous FIFOs allow transfers of up to 100 MHz.

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 *Figure 1*.

The flag logic simply compares the value in each of the two address pointers. If the difference between those two values 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 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 part.

Finally, expansion logic is used to create logically deeper FIFOs, by cascading multiple parts in depth expansion. In normal "non depth cascading" operation, each of the address pointers will wrap back to zero when they reach their maximum value. In 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. Once the token is passed away, 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 is handling read operations, and only one is handling write operations. When the token returns the address pointer is reset to zero and operation resumes. See application note "Depth Expansion of Synchronous FIFOs" for a more detailed description of this mode of operation.

Reset

After power-up the FIFO must be reset. Resetting the part sets the read and write address pointers to zero, clears the

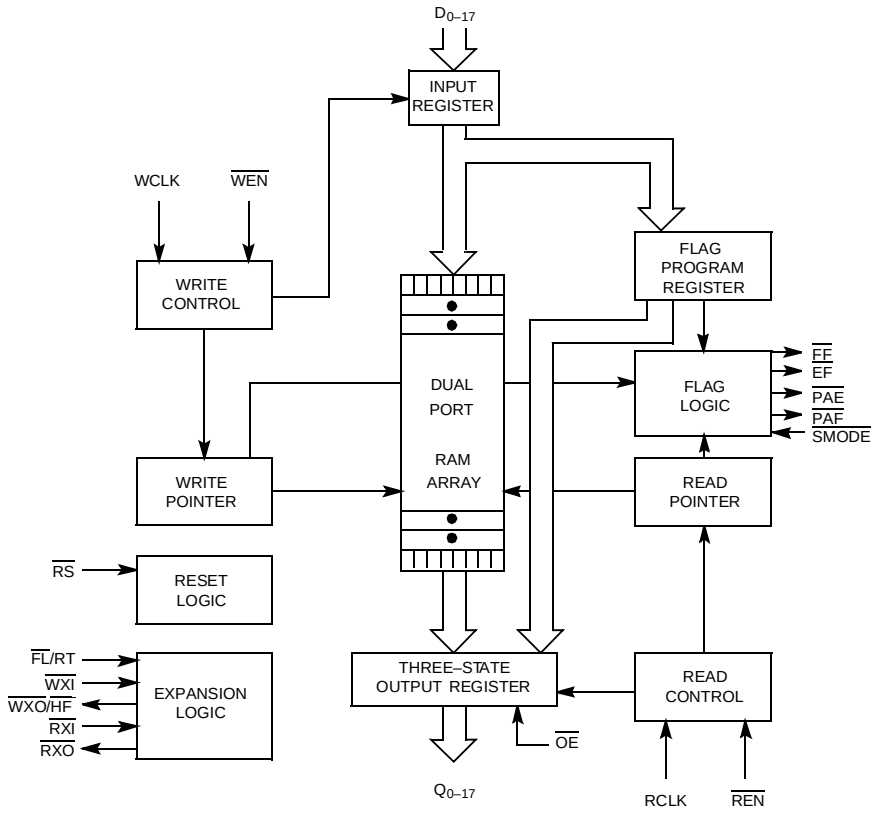

Figure 1. Logic Block Diagram—Synchronous FIFO Architecture (CY7C42x5)

output data register, and sets the status flags to represent an empty device. Resetting the device is done by asserting the $\overline{RS}$ pin LOW. Synchronous FIFOs do not 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 holds it LOW for a minimum time to allow V_{CC} and all clocks to stabilize.

During $\overline{RS}$ assertion, no read or write operations may be attempted to the part. This can be done by deasserting the read and write enables ($\overline{REN}$, $\overline{WEN}$), or by gating the 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 edge of $\overline{RS}$.

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

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 number of words in the FIFO. These flags are generated by comparing the values in the read and write address pointers. These flags should be used by control logic to determine if read or write operations are to be performed to 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). By synchronizing the flag to the respective clock, there is no need to synchronize it externally for control logic. Most often the logic that writes to a FIFO must ensure that the FIFO is not full before writing. Likewise the read control logic will examine the empty flag ($\overline{EF}$) before reading from the FIFO. External synchronization of these flags would require feeding the flag through two registers. This delays the visibility of the flag assertion to the control logic by two clocks and makes the flag much less usable.

The programmable almost empty ($\overline{PAE}$) and programmable almost full ($\overline{PAF}$) flags are synchronous on some devices such as the CY7C42x1/42x2 FIFOs. Like the empty and full flags the $\overline{PAE}$ and $\overline{PAF}$ flags are synchronized to their respective clocks. The $\overline{PAE}$ flag is synchronized to RCLK and the $\overline{PAF}$ flag is synchronized to WCLK. Other FIFOs such as the CY7C42x5 allow for both synchronous and asynchronous operation.

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

Flag Update Cycle

Since the empty and full flags are synchronous, they require a rising edge on their respective clock to update them to their

most current value. Internally the flag is generated in the flag logic and then fed to a register which synchronizes it.

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 which monitors it.

For example, lets say we have an empty FIFO. After you write one word into the part it is no longer empty. Since the empty flag ($\overline{EF}$) is synchronized to the read clock, the flag will not be asserted until the part receives a RCLK rising edge. No read operations are performed to the device until after the $\overline{EF}$ is deasserted (the first RCLK rising edge updates the flags and the second RCLK rising edge reads the first word out). This dead cycle insures that the assertion and deassertion of the empty flag ($\overline{EF}$) will always be at least one cycle long. Under asynchronous conditions of the RCLK and WCLK (very common for FIFO applications), the flag assertion could be infinitely small without this dead cycle. See *Figure 2*.

For applications that use free running clocks, this “dead” cycle or “flag update cycle” is transparent and of no concern. The read control logic will not assert the read enable ($\overline{REN}$) until the empty flag ($\overline{EF}$) deasserts. The first rising edge of RCLK after $\overline{REN}$ is asserted will read the first word from the FIFO.

For applications which do not use free running clocks. The RCLK will have to transition from LOW to HIGH twice to read out the first word. Once for the “flag update cycle”, and the second to read out the first word.

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

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 physical location zero. Note in order 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’s an example. Let’s say you have a 1K deep packet of data that you want to send 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. The 1K words of data are then written to the FIFO. Once writing to the FIFO begins and the $\overline{EF}$ deasserts the serial transceiver device can begin reading from the FIFO.

Empty Flag Timing

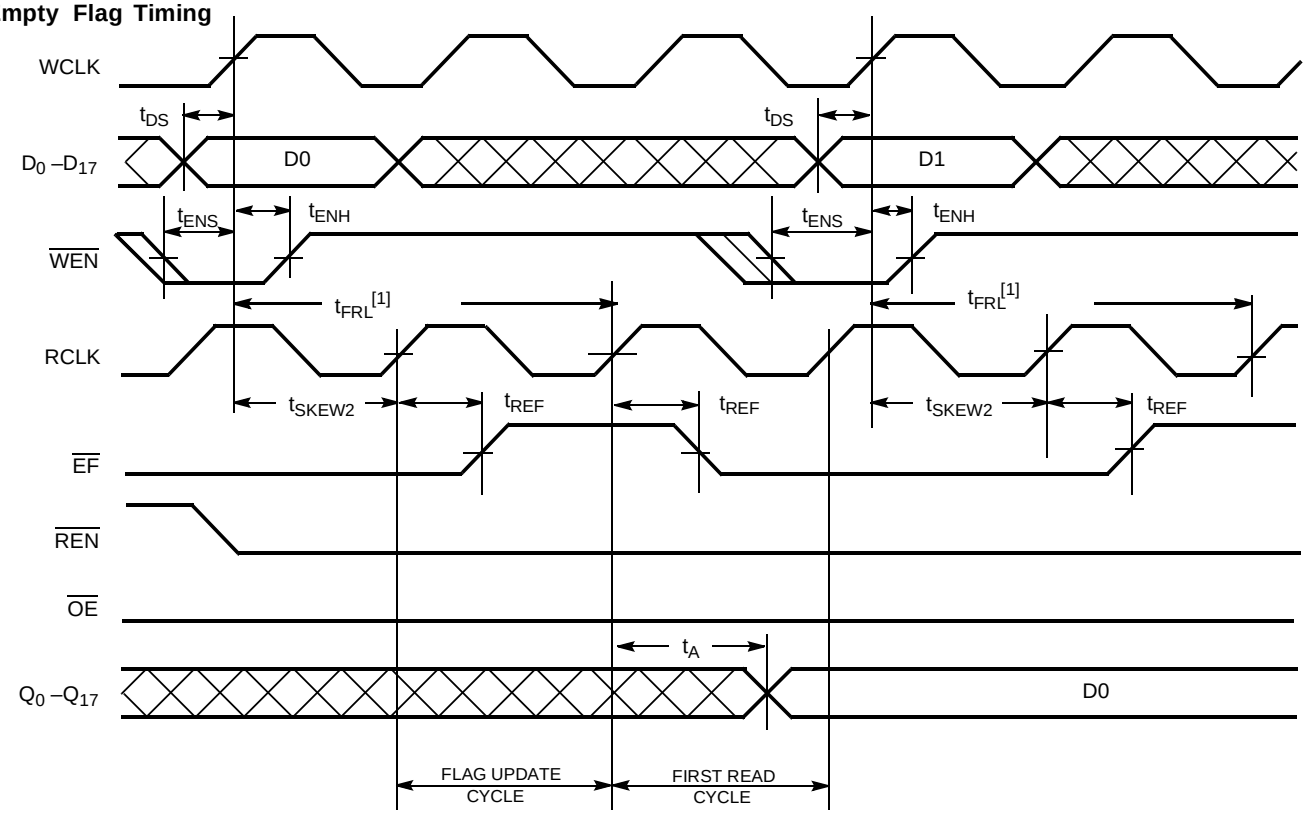

Figure 2. Flag Update Cycle

Note:

1. When $t_{SKEW2} \geq$ 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).

As data is read from the FIFO the read address pointer increments until it reaches location 1024 and the FIFO becomes empty. Note, 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.

The most important item to remember is that the $\overline{RT}$ pin does nothing more than reset the read address pointer. The FIFO has no knowledge of where certain packets of your data are stored in the device. It is the responsibility of the user to 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.

System Design Concerns

Width Expansion

Width expansion is used to create FIFOs with wider data paths. Two x18 FIFOs can be width expanded to create a x36 FIFO, etc. Read, write, and retransmit operations are the same for FIFOs in width expansion, and in fact the FIFO has no knowledge that it is being used in this mode.

To expand multiple FIFOs in width, the flags must be combined to create “composite flags”. This is done by 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 3*.

Understanding how the FIFOs can get out of synchronization is somewhat confusing. The root of this problem lies with the asynchronous relationship of the clocks, RCLK and WCLK. The explanation of this phenomenon is best described with an example. Lets say we have two FIFOs width expanded that each have one word in them (one location from empty). Due to the asynchronous phase relationship between the read and write clocks it is highly possible that the FIFO 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 causing it to become not empty. 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: no flag update cycle is needed since the FIFO never became empty. Width expanded FIFOs can become out of synchronization due to the simple fact that they may respond to this above scenario slightly differently. 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. Since 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. Control logic which drives the $\overline{REN}$ and $\overline{WEN}$ should deassert these enables in the event that one of the composite flags becomes asserted.

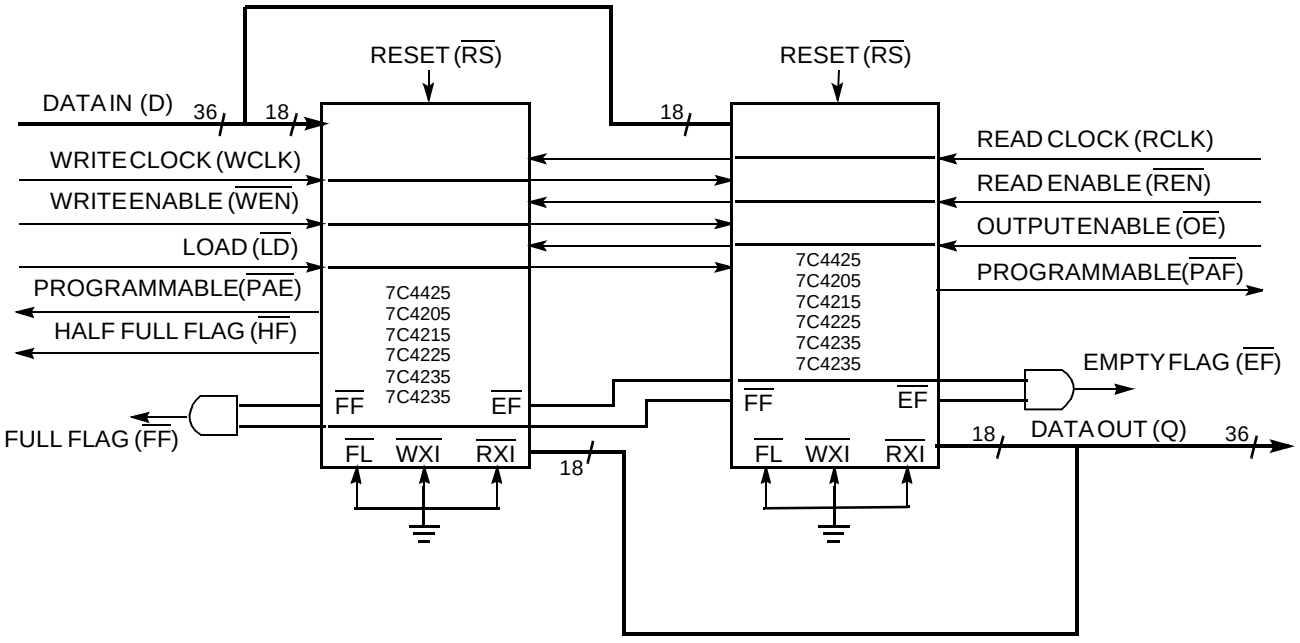

Figure 3. Width Expansion of Synchronous FIFOs (CY7C42x5)

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 $\overline{WEN}$, $\overline{REN}$, RCLK, 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. Likewise when the read address pointer reaches it's maximum value, it passes a token via the $\overline{RXO}$, to the $\overline{RXI}$ pin of the next device. See *Figure 4* for a diagram of depth expansion configuration of the CY7C42x5 FIFOs.

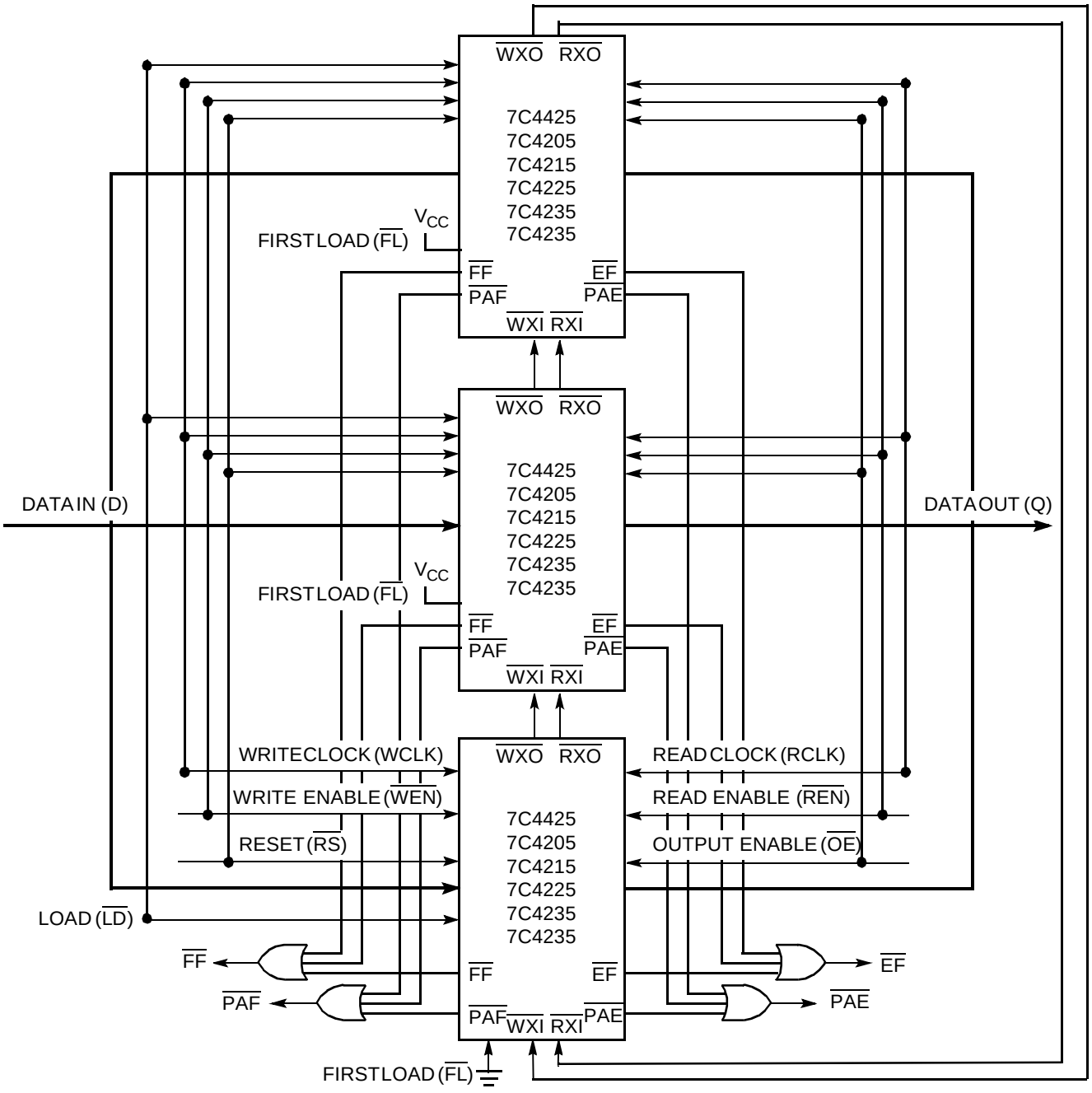

Figure 4. Depth Expansion of Synchronous FIFOs (CY7C42x5)

Composite empty and full flags must be generated for proper operation. Composite flags are generated by 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. Likewise all FIFOs must be empty for the composite empty flag to assert. The PAE, PAF, and HF flags are essentially useless in depth expansion. 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 little information about the overall state of the depth expanded FIFO buffer.

Some FIFOs such as the CY7C4282/92 use multiplexed expansion pins. This approach uses a single XI and XO connection where the first pulse driven is expected to be the write token and the second is the read token. The token passing approach is essentially identical except that two less signals are needed per FIFO.

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. This approach is described along with a more thorough description of the token passing approach in another application note entitled “Depth Expansion of Synchronous FIFOs.” Check the cypress web page at www.cypress.com or your Cypress CD ROM for a copy of this application note.

Operation as an Asynchronous FIFO

Often users will 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 perfectly 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 (REN, WEN) can be driven LOW during all phases of operation with the possible exception of reset and retransmit. Remember, reset and retransmit both require that no reads or writes occur during these operations. This means either driving the REN and WEN HIGH or by gating the RCLK or WCLK LOW. Gating the RCLK and WCLK HIGH will cause internal clock pulses to be generated (only when its respective enable is asserted) and is disallowed.

The other concern with this mode of operation is the flag update cycle. Since no free-running clocks are provided to the device, the synchronous flags are not always automatically updated. Consider an empty FIFO which then receives a number of write operations. The FIFO is no longer empty, but the EF is still asserted since no “flag update cycle” has occurred. To the user it will look as if two read cycles will be needed to read the first word from the FIFO, the first is the flag update cycle and the second performs the first read. Once the flag is updated, data will be read from the FIFO on each RCLK rising edge. Care must be taken to add this extra read cycle each time the EF is asserted. Likewise the FF requires a flag update cycle at the full boundary. Essentially each time the FIFO is full and a read is performed, two WCLK rising edges are needed to write the next piece of data.

Decoupling

Decoupling capacitors are used to provide instantaneous current required by CMOS devices. In general it is good practice to have one decoupling cap per V_{CC} pin of the device. The value of the decoupling cap needed varies with the speed at which the FIFO is run. Capacitor values should be chosen that have resonant frequencies just below the speed at which the FIFO operates. In most cases values such as .1 μF or .01 μF will be sufficient. The smaller the capacitance value the higher the resonant frequency.

Clock Termination

As with any high-speed digital device clock, signal integrity is crucial. Care should be taken to drive the FIFO with properly terminated clocks to prevent double registering or malfunction of the FIFO. The type of termination used will vary by system due to trace width, trace length, and dielectric constants of board materials, etc. Two common types of termination that have been used are series and thevenin termination. Series termination requires a 20 to 50 Ohm series resistor placed at the clock source. Thevenin termination involves a pull-up and pull-down resistor at the end of the clock trace. The thevenin equivalence of the two resistors should equal the trace impedance. The values of the resistors should also DC bias the clock trace to about 3V. A common thevenin termination is 75 ohms to V_{CC} and 150 Ohms to GND. The thevenin equivalence of these two values is 50 Ohms, which is a common trace impedance.

All clock traces should be daisy chained from one device to another so that no “dangling” traces exist. Such “dangling” traces cause reflections that can distort clock signals.

For fine tuning clock termination values or to experiment with different termination types, IBIS model simulations can be performed. IBIS models of most Cypress FIFOs and clock devices are available for download from the Cypress web site at www.cypress.com. IBIS models are used by an increasing number of simulation platforms. In many cases the models can be used with the layout package which will include trace widths, lengths, and even geometry related issues such as crosstalk or impedance mismatches from vias.

Summary

The Cypress line of Synchronous FIFOs prove to be a very versatile and efficient method of buffering and synchronizing data. These FIFOs can be cascaded together in width or depth to create FIFO buffers not available in single devices. Synchronous FIFOs support high-speed 100-MHz operation with extremely low first word latency. These devices can be used in synchronous mode by tying the clock input’s to free running clocks for optimum speed, or used as asynchronous FIFOs by pulsing the clock inputs for easy integration to designs which expect asynchronous FIFOs. Finally features such as retransmit and synchronous flags make the devices more versatile and easier to use.