Frequently Asked Questions about FIFOs and Dual Ports

Scope

The following questions are frequently asked by customers who are evaluating and using Cypress FIFOs and Dual Ports. These answers will serve as an introduction for each topic. Separate application notes cover these topics in more complete detail.

Frequently Asked Questions about FIFOs

1. How does retransmit work?

Retransmit allows the user to re-read a block of information that was previously read from the FIFO. This feature is commonly used in serial transmission applications. If the serial data was corrupted during transmission, retransmit allows an easy way to resend that data packet. FIFOs have also been used to hold instruction code for processors. Once the processor has read the last instruction it can pulse the retransmit line on the FIFO to re-execute the same code. Retransmit also allows FIFOs to act as data pattern generators.

To use the retransmit feature effectively the FIFO must first be reset. By resetting the device the read and write pointers are moved to location zero. The device is empty and ready to receive data. Once a packet of data is written into the device the write pointer sits at the top of the data packet. Once the packet is read the read pointer moves to the top of the packet as well. Pulsing the Retransmit pin (RT*) simply moves the read pointer back to location zero. The flags will be updated to reflect the number of words in the device and the packet is now ready to be re-read. Care should be taken to ensure that reads and writes are not performed to the device during the pulsing of the RT* signal. Refer to the data sheet for a complete description of requirements to perform a valid retransmit.

2. How do I reset the device?

After power-up the FIFO must be given a reset pulse which empties the device and sets the flags to represent the empty state. Reads and writes must not be performed during the reset pulse except with some devices when programming the $\overline{\text{PAE}}$ and $\overline{\text{PAF}}$ flags. Giving the device a reset pulse includes both an assertion and deassertion edge to the reset pin.

3. What is the advantage of Synchronous $\overline{\text{PAE}}$ and $\overline{\text{PAF}}$ flags?

By tying the $\overline{\text{SMODE}}$ pin to GND on the CY7C42x5 FIFOs the $\overline{\text{PAE}}$ and $\overline{\text{PAF}}$ flags will be synchronous. The $\overline{\text{PAE}}$ flag is synchronized to the read clock, and $\overline{\text{PAF}}$ is synchronized to the write clock. When the flags are synchronous there is no need to synchronize them externally (double or triple registering the flag to avoid metastability concerns.) By avoiding external synchronization a system can respond much faster to the flag assertion. Synchronized flags are also guaranteed to be asserted for a minimum of 1 clock cycle. This ensures that the flag transition is visible to a synchronous environment.

The CY7C42x1 FIFOs have synchronous $\overline{\text{PAE}}$ and $\overline{\text{PAF}}$ flags at all times.

4. What is a flag update cycle with respect to a Synchronous or Clocked FIFO?

A flag update cycle (otherwise known as boundary latency cycle) refers to the clock cycle which updates the synchronous flags at a boundary. When starting with an empty FIFO it essentially takes two read clocks to read the first word from the device. The first read clock rising edge updates the Empty Flag (assuming a write has been performed). This update cycle occurs whether the read clock is enabled or not. The second read clock rising edge (enabled) will read the first word from the device. Without asserting the read enable ($\overline{\text{REN}}$) a read will not be performed. Similarly, when the device is full, a write clock is needed to update the full flag ($\overline{\text{FF}}$). In this case it takes two write cycles to write into a device which just became not full.

This type of flag operation is necessary to ensure that the empty and full flags will be valid and usable for a minimum of one clock cycle. This architecture eliminates short flag pulses characteristic of an asynchronous FIFO.

5. Are there any concerns with width expansion?

Many applications require that multiple FIFOs are used in parallel to generate a wider data path. Such applications should use external logic to generate composite empty and full flags. When using FIFOs that have active LOW flags, ANDing the EMPTY and full flags of each device will ensure that the FIFOs do not get out of sync. See the application note “Understanding Synchronous FIFOs” for a full description of which circumstances can allow the devices to get out of sync. All other operations to the FIFOs will occur just as a single device.

6. Are there any concerns with depth expansion?

Many FIFOs have expansion logic which allows cascading of multiple devices to create a logically deeper FIFO. Such cascading does not slow overall FIFO operation, but does limit the usability of the $\overline{PAE}$, $\overline{PAF}$, and $\overline{HF}$ flags. Such flags are generated by subtracting the value of the read and write pointers in the FIFO. Each device will generate flags which accurately represent the state of that device. However, any one individual flag can not be used to determine the overall state of the system. (For instance: One device may be just over half full and the other device may be close to empty. The two FIFOs which make up the system are less than half full.)

7. What is Minimum Pulse Width Violation with respect to Asynchronous FIFOs?

When using an Asynchronous FIFO, users must not attempt to write to a full device or read from an empty device (only when performing simultaneous reads and writes.) When an asynchronous FIFO is full the device disables the $\overline{W}$ input. It does this by internally asserting the $\overline{W}$ signal to the inactive state. If a read pulse occurred during an attempted write to a full device the following occurs:

1.The read operation brings the device to a state of full –1

2.Write operations are then enabled

Internally after the $\overline{W}$ signal is enabled, the $\overline{W}$ pulse which was given to the device is truncated (since part of the pulse was disabled). The width of the truncated pulse depends on the phase relationship between the $\overline{R}$ pulse and the $\overline{W}$ pulse. Under certain phase conditions the effective pulse width is shorter than allowed. The phenomenon is therefore called “minimum pulse width violation” Such “runt” pulses can irritate flag state machines and cause them to reflect incorrect values.

Likewise the same phenomenon can occur at the empty boundary.

To avoid minimum pulse width violation do not attempt to write to a full device or read from an empty device in applications which allow simultaneous read and write operations.

Frequently Asked Questions about Dual Ports

1. What is the difference between a Master and a Slave device?

When multiple dual ports are used in width expansion, typically one device is a master and the rest of the devices are slaves. A master device performs arbitration between ports. A slave device does not have an on chip arbiter and must be told which port is $\overline{BUSY}$ in the case of an address collision. Typically the $\overline{BUSY}$ signals are connected between the master and slave devices. This procedure ensures that no two devices will arbitrate differently.

$\overline{BUSY}$ can be driven directly to a slave device in cases where the user wants to perform arbitration externally.

2. What is the difference between a CY7Bxxx device and a CY7Cxxx device?

The CY7B designator is used to describe a BiCMOS dual port. The CY7C designator is used to describe a CMOS device. Some Dual Ports were originally built using the BiCMOS process to achieve faster access times. More recent CMOS devices are just as fast and have replaced the BiCMOS parts. CMOS dual ports with the same part number extensions are functionally identical to their BiCMOS predecessors (i.e., a CY7C135 is functionally identical to a CY7B135). CMOS dual ports typically consume much less power than BiCMOS dual ports.

3. What can I use the Semaphore latches for?

Semaphore latches are used to “reserve” certain portions of memory space to a particular port. If one port requires the use of a particular port it writes a '0' to a semaphore latch which represents that address space. Once written, that port will read the same latch to determine if it gained access. If that port reads a '0' the attempt was successful and that port now has access. A '1' represents a failed attempt. To effectively utilize the semaphores, both ports must use the semaphores in a friendly fashion. The dual port does not “enforce” the semaphores active state since the part does not know which portion of memory is being allocated. Each port must monitor the semaphores to make them effective.

4. What can I use interrupts for?

One port can generate an interrupt to the other port by writing to a designated address. Once the receiving port has serviced that interrupt it reads the same address to clear the interrupt. Such interrupt passing is useful in notifying one port that a block of data has been written and is ready for use.