PROJECT

Design of Asynchronous FIFO

Ujwal Vikas Uttarwar (21307R021)

Introduction: -

An asynchronous FIFO refers to a FIFO design where data values are written to a FIFO buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other. Asynchronous FIFOs are used to safely pass data from one clock domain to another clock domain.

Attempting to synchronize multiple changing signals from one clock domain into a new clock domain and ensuring that all changing signals are synchronized to the same clock cycle in the new clock domain has been shown to be problematic. FIFOs are used in designs to safely pass multi-bit data words from one clock domain to another. Data words are placed into a FIFO buffer memory array by control signals in one clock domain, and the data words are removed from another port of the same FIFO buffer memory array by control signals from a second clock domain.

The difficulty associated with doing FIFO design is related to generating the FIFO pointers and finding a reliable way to determine full and empty status on the FIFO.

Synchronous FIFO Pointers: -

A FIFO where writes to, and reads from the FIFO buffer are conducted in the same clock domain. For synchronous FIFO design, one implementation counts the number of writes to, and reads from the FIFO buffer to increment, decrement or hold the current fill value of the FIFO buffer. The FIFO is full when the FIFO counter reaches a predetermined full value and the FIFO is empty when the FIFO counter is zero.

Unfortunately, for asynchronous FIFO design, the increment-decrement FIFO fill counter cannot be used, because two different and asynchronous clocks would be required to control the counter. To determine full and empty status for an asynchronous FIFO design, the write and read pointers will have to be compared.

Asynchronous FIFO Pointers: -

In order to understand FIFO design, one needs to understand how the FIFO pointers work. The write pointer always points to the next word to be written. Therefore, on reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.

Similarly, the read pointer always points to the current FIFO word to be read. Again, on reset, both pointers are reset to zero, the FIFO is empty and the read pointer is pointing to invalid data (because the FIFO is empty and the empty flag is asserted). As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the read pointer that is still addressing the contents of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic.
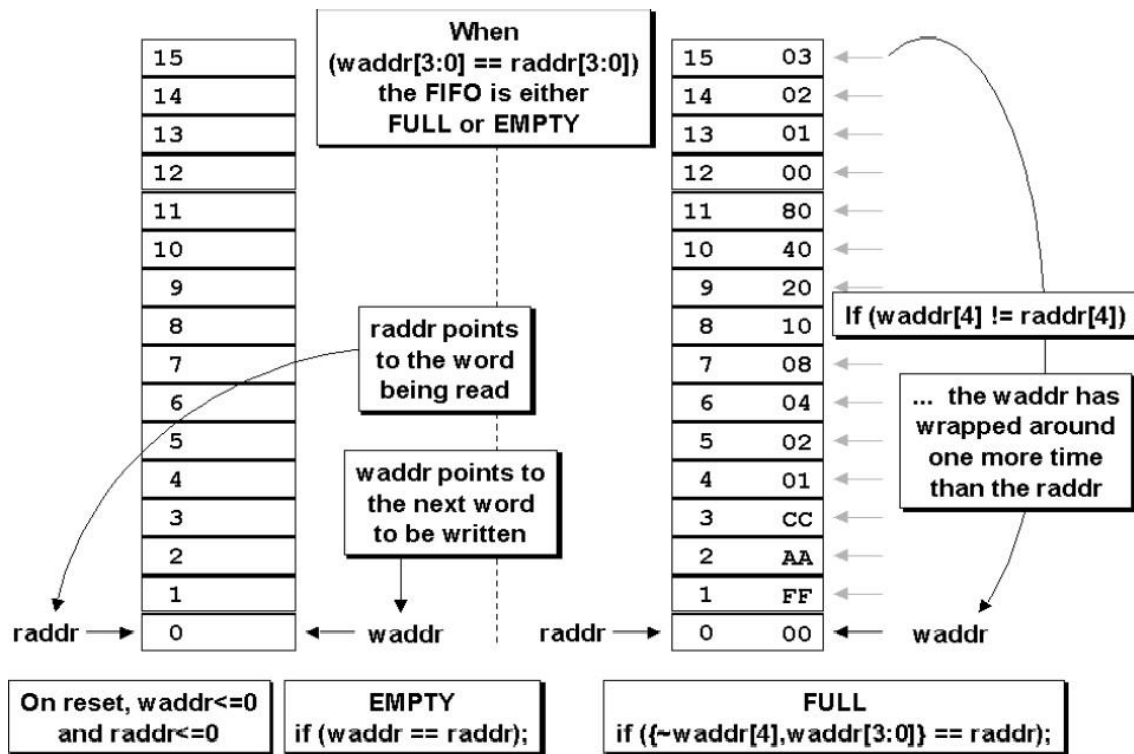
FIFO Full and Empty Condition: -

The FIFO is empty when the read and write pointers are both equal. This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, having read the last word from the FIFO.

A FIFO is full when the pointers are again equal, that is, when the write pointer has wrapped around and caught up to the read pointer. This is a problem. The FIFO is either empty or full when the pointers are equal, but which?

One design technique used to distinguish between full and empty is to add an extra bit to each pointer. When the write pointer increments past the final FIFO address, the write pointer will increment the unused MSB while setting the rest of the bits back to zero as shown in below figure (the FIFO has wrapped and toggled the pointer MSB).

The same is done with the read pointer. If the MSBs of the two pointers are different, it means that the write pointer has wrapped one more time that the read pointer. If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times.

When (waddr[3:0] == raddr[3:0]) the FIFO is either FULL or EMPTY

15		15	03	←
14		14	02	←
13		13	01	←
12		12	00	←
11		11	80	←
10		10	40	←
9		9	20	←
8		8	10	
7		7	08	←
6		6	04	←
5		5	02	←
4		4	01	←
3		3	CC	←
2		2	AA	←
1		1	FF	←
raddr → 0	← waddr	raddr → 0	00	← waddr

raddr points to the word being read

waddr points to the next word to be written

If (waddr[4] != raddr[4])

... the waddr has wrapped around one more time than the raddr

On reset, waddr<=0 and raddr<=0

EMPTY if (waddr == raddr);

FULL if ({~waddr[4],waddr[3:0]} == raddr);

Using n-bit pointers where (n-1) is the number of address bits required to access the entire FIFO memory buffer, the FIFO is empty when both pointers, including the MSBs are equal. And the FIFO is full when both pointers, except the MSBs are equal.

FIFO full occurs when the write pointer catches up to the synchronized and sampled read pointer. The synchronized and sampled read pointer might not reflect the current value of the actual read pointer but the write pointer will not try to count beyond the synchronized read pointer value. Overflow will not occur.

FIFO empty occurs when the read pointer catches up to the synchronized and sampled write pointer. The synchronized and sampled write pointer might not reflect the current value of the actual write pointer but the read pointer will not try to count beyond the synchronized write pointer value. Underflow will not occur.

The FIFO design in this project assumes that the empty flag will be generated in the read-clock domain to ensure that the empty flag is detected immediately when the FIFO buffer is empty, that is, the instant that the read pointer catches up to the write pointer (including the pointer MSBs).

Similarly, for full condition it assumes that the full flag will be generated in the write-clock domain to ensure that the full flag is detected immediately when the FIFO buffer is full, that is, the instant that the write pointer catches up to the read pointer (except for different pointer MSBs).

FIFO Empty Condition: -

The FIFO is empty when the read pointer and the synchronized write pointer are equal. The empty comparison is simple to do. Pointers that are one bit larger than needed to address the FIFO memory buffer are used. If the extra bits of both pointers (the MSBs of the pointers) are equal, the pointers have wrapped the same number of times and if the rest of the read pointer equals the synchronized write pointer, the FIFO is empty.

The Gray code write pointer must be synchronized into the read-clock domain through a pair of synchronizer registers found in the sync_w2r module. Since only one bit change at a time using a Gray code pointer, there is no problem synchronizing multi-bit transitions between clock domains. In order to efficiently register the rempty output, the synchronized write pointer is actually compared against the rgraynext (the next Gray code that will be registered into the rptr).

FIFO Full Condition: -

Since the full flag is generated in the write-clock domain by running a comparison between the write and read pointers, one safe technique for doing FIFO design requires that the read pointer be synchronized into the write clock domain before doing pointer comparison.

The full comparison is not as simple to do as the empty comparison. Pointers that are one bit larger than needed to address the FIFO memory buffer are still used for the comparison, but simply using Gray code counters with an extra bit to do the comparison is not valid to determine the full condition. The problem is that a Gray code is a symmetric code except for the MSBs.

The correct method to perform the full comparison is accomplished by synchronizing the rptr into the wclk domain and then there are three conditions that are all necessary for the FIFO to be full: -

1. The wptr and the synchronized rptr MSB's are not equal (because the wptr must have wrapped one more time than the rptr).
2. The wptr and the synchronized rptr 2nd MSB's are not equal (because an inverted 2nd MSB from one pointer must be tested against the un-inverted 2nd MSB from the other pointer, which is required if the MSB's are also inverses of each other.
3. All other wptr and synchronized rptr bits must be equal.

In order to efficiently register the wfull output, the synchronized read pointer is actually compared against the wgnext (the next Gray code that will be registered in the wptr).

Simulation Results: -