FIFO POINTERS

Counters as FIFO Pointers

Two types of counters are used as FIFO pointers- binary counters and Grey counters. Each of these methods has merits and demerits. Synchronization advantages and pitfalls between the read and write clock domain is the decisive factor in choosing right counter design as pointers.

Asynchronous FIFO Pointers Using Binary Counters

Binary counter is natural counter and hence easy to design and implement. This counter works very well in addressing FIFO. In our case we have total 16 memory locations in the FIFO. Hence to address these $16=2^4$ locations we need 4 bit counter.(actually we need to design 5 bit counter….why…? we will see later !) binary code patterns are not unidistance. Number of bits changing from one count to another count can be more than one. For example bits changing from 0001 to 0010 are 2. In worst case all 4 bits can change simultaneously like 0111 to 1000. Then changing bits (nothing but pointers) has to be synchronized with the other clock domains to generate empty and full condition. If the synchronization clock edge becomes active in between the transition of binary bits, say, 0111 to 1000, then metastability can occur with any of the four bits or with all the bits. This metastable state can be resolved to any four-bit count value prediction of which is almost impossible. The pointer value synchronized with other clock domain may become entirely different than intended. This is the biggest drawback of using binary counters as FIFO pointers. One way to counter this problem is to use holding register for synchronization. This uses handshaking signals to communicate between synchronizer and the clock domain. Binary count values from original clock domain is sampled and held in holding register and a “ready” signal is passed to other clock domain. Upon the receipt of “ready” signal other clock domain receives the count values and sends back a synchronized acknowledgement. Original clock domain resamples another count value.

Asynchronous FIFO Pointers Using Gray Counters

Gray numbers are unidistance numbers i.e. to say that unlike binary numbers only one bit changes from one count to another count. Gray pointers too have problem of metastability while synchronizing with other clock domains but it is minimized by the fact of one bit change. Metastability condition on one bit causes +/- 1 count error that is better compared to +/-8 count error in binary pointers. Because of this minimized error gray counters are generally used as FIFO pointers.

Generation of Empty and Full Pointers

Careful observation of 4 bit gray counter values reveal that first half of the numbers (i.e. 0 to 7) are mirror image of the second half of the number (i.e. from 8 to 15) except for MSB. Recall the wrap around condition of full pointer after complete write operation of FIFO. Since read pointer is still in first location, after the wrap around of full pointer they both become equal asserting false empty condition. To avoid this one extra bit is added to both pointers. Thus, instead of using a 4-bit counter use 5 bit counter. Out of this 5 bit only 4 bits are used to address 16 memory locations, while the MSB is used to detect the wrap around conditions and pointer comparisons. Thus when write pointer increases over the final FIFO address the unused bit (MSB bit) toggles and resets remaining bits to zero. For read pointer also same treatment is given. Thus if MSBs of both pointers are same it means that both pointers have wrapped else write pointer has wrapped one more time than read pointer. With this technique if including MSB both pointers are same then it is an FIFO empty condition. If the MSBs different and remaining all bits are equal then it is FIFO full condition.

Considering the case wherein all FIFO locations except last one are written and then same number of locations is read. Now both pointers are pointing towards 15th location i.e. last location of the FIFO. On the next write operation 5 bit gray counter will be incremented and the count is 1_1000 (=16). Remember that earlier count was 0_1000 (=15). Hence when counter is incremented, only MSB (extra bit) changed and remaining address pointer bits remained as it is. Write pointer and read pointer both are pointing to same address location because MSB is not used for addressing FIFO; it is to test full and empty condition. Since write pointer extra bit has set and read pointer extra bit has reset, this condition will be assumed as FIFO full condition but in reality this is NOT at all FIFO full condition. In addition to this problem data will be overwritten on the location 1_1000. Hence necessary condition to generate full pointer is that both MSB (extra bit) and next to MSB bit of write pointer and synchronized read pointer must be different and remaining bits must be equal.

Dual n Bit Gray Code Counter-First Architecture

This architecture solves the problem which is identified in the previous section. A dual n bit gray code counter generates both (n-1) gray code (used to address the memory location) and n bit gray code sequence, nth bit being used for pointer comparison to detect empty and full condition. The block diagram of the dual n-bit gray counter is shown in Figure (4a).

Instead of using a customized gray counter solution a general method of gray code generation is presented here. Gray code and binary codes are related by equations:

$g_n = b_n$, $g_i = b_i$ XOR b_{i+1}, for all $i \neq n$ eq(1)

$b_n = g_n, b_i = g_i$ XOR b_{i+1}, for all $i \neq n$ eq(2)

where g▤▤☜refers to gray code and b✳refers to binary code

Same equations have been implemented in the block diagram Figure (1.4b). Convert the gray code value to binary using eq(1). Add one (i.e. increment the count by one) to this binary value and convert it back to gray by implementing eq(2).

To avoid underflow or overflow design should have precautionary circuit such that when full pointer or empty pointer is asserted, counter should not be incremented any more. This is accomplished by a OR and AND gate with “not empty” and “not full” inputs. When full pointer is asserted there should not be a write operation to avoid overflow. Hence there is a necessity of a circuit which disables write_enable signal. NAND or AND can fulfill this requirement. Same way we can add status flag for “overflow” and “underflow” error indication which could only be cleared by a reset signal.

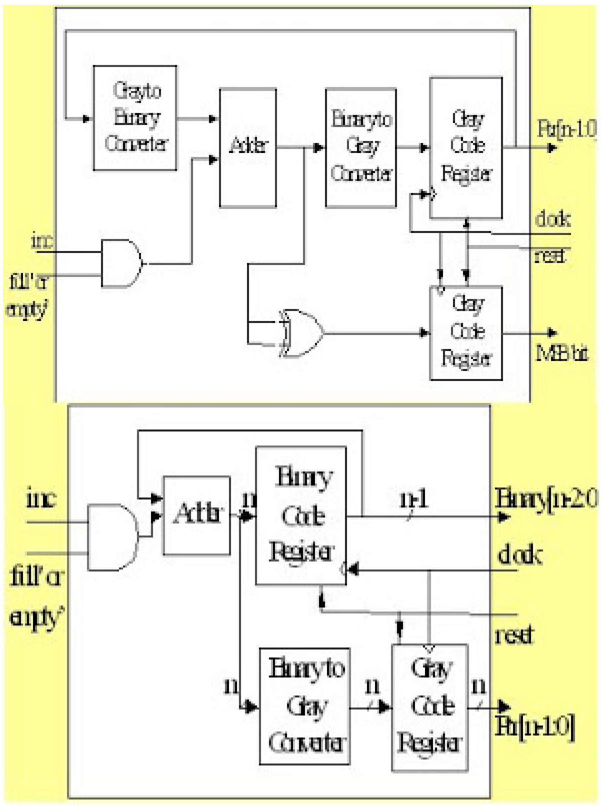

Figure (4a and 4b) Dual n Bit Gray Counter-Architecture 1 and 2 [2]

Dual n Bit Gray Code Counter-Second Architecture

In this architecture, shown in Figure (4b), both binary and gray counters are used. Binary code is used to address the memory locations of FIFO and gray code is used for synchronizing with opposite clock domain. Two registers are used to register binary code and gray code. The advantage of ease of pointer comparison in binary codes is the key factor to choose binary pointer for FIFO memory addressing. Binary code is not suitable for synchronizing with opposite clock as it has multiple bit changes from one count to another count. For this purpose gray codes are used, as these are less prone to errors.

Almost Full and Almost Empty

When write pointer reaches very close to full condition an almost full condition can be generated. When read pointer approaches empty condition (i.e. read pointer closes to catch the write pointer) then an almost empty can be generated. This almost full and almost empty condition can be generated for programmed difference between write pointer and read pointer. Thus when the programmed difference value is reached corresponding pointer flag becomes active. Almost full and almost empty condition helps the pointer comparison logic circuitry to get "ready" for the full and empty condition detection. For a fixed value of difference design of almost full and almost empty is easy-combination of NOT, AND or OR gate can do the job

Going to Full and Going to Empty

In this architecture FIFO memory location is logically divided into four quadrants. Two MSBs of two pointers are used to decode these quadrants. Possible bit combinations of MSBs are 00, 01, 10 and 11. If the write pointer is one quadrant behind the read pointer then this indicates FIFO 'going to full' condition. FIFO can be considered 'going to empty' if the above-mentioned condition is reverse.

Pessimistic Full and Empty

Delay in synchronization pointers may cause wrong reporting of full and empty condition. Full pointer may become active even if FIFO is not full or empty pointer may become active when FIFO is not yet empty- such conditions are called as 'pessimistic reporting'. We know full pointer is generated when write pointer catches up the synchronized read pointer. When read pointer increments, one data has been read from FIFO. Detection of this status by write pointer logic takes minimum two clock cycles due to the presence of synchronizers. Hence even if there is a memory location free which can be written, write pointer logic does not allow writing the FIFO till two clock cycle delay is elapsed and it detects the read pointer status. Same is true with read pointer logic also. This pessimistic report doesn't harm FIFO data.

Binary Counter Vs Gray Counter

Here is the trade off between binary counter and gray counter as pointers:

- Binary pointers pose multibit synchronization problems. Gray counter minimizes this problem.
- A gray counter designed for any mod number other than 2^n, n being number of bits, does not remain as gray code. Hence we must design a mod 2^n gray counter. This implies that FIFO memory location must also be 2^n. But binary counters can be designed to have any mod number and hence FIFO memory locations can also be any arbitrary number.
- Since binary arithmetic is natural, it is easy to calculate and implement almost empty and almost full with binary numbers.
- The sampling technique using holding register and handshaking control is advantageous in passing any arbitrary multibit values or pointers. But in gray counters arbitrary value is not possible, they either increment or decrement.
- Since gray counter has to be designed for mod (2^n), FIFO depth (maximum) must also be power of 2. But in binary any depth is permitted.
- Usage of binary pointer introduces latency of minimum 2 clock cycles in synchronization.

FIFO Depth

Size of the FIFO basically refers to the amount of data available at a given time. In asynchronous FIFO this depends on both read and write clock domain frequencies and number of data written and read (data rate). Data rate can vary depending on the two clock domain operation and requirement (and of course frequency!). The worst case condition is the maximum data rate difference between read and write clock. This can happen when data rate of writing operation is maximum and for read operation data rate is minimum.

Let $f_{write\ -->}$be the frequency of write clock domain

Fread ▤▤☜be the frequency of read clock domain

Bmax ▤▤☜burst of data written or maximum number of data bytes can be written

Bwrite▤▤☜number of bytes that is written per clock cycle

Bread▤▤☜number of bytes that is read per clock cycle

Then FIFO size can be given by,

Fsize=Bmax- [Bmax.fread.Bread/fwrite.Bwrite]

If number of bytes read or written per clock cycle is one then we have,

Fsize=Bmax-[Bmax.fread/fwrite]

Taking an example,

fwrite=10 MHz

Fread=2.5 MHz

Let Bmax=2 then Fsize=2-[(2*2.5)/10] =2-0.5=1.5~2

If Bmax=5, then Fsize=5-[(5*2.5)/10]=5-1.25=3.75~4.

Source : http://asic-soc.blogspot.in/2007/11/fifo-pointers.html