L6 - The Asynchronous FIFO

In an asynchronous FIFO, the read pointer (counter) is controlled by the clock signal from the receiving subsystem and the write pointer (counter) is controlled by the clock signal from the sending subsystem. Since the operation of these counters only involves the clock signal from its own domain, the counters impose no synchronization problem. The difficulty comes from the **full** and **empty** status signals.

A synchronous FIFO with an n-bit address space (i.e., 2^n words), it is constructed as follows:

- Use two (n + 1)-bit binary counters as the pointers, one for the read pointer and one for the write pointer.
- Use two n-bit binary counters (which are the n LSBs of the (n + 1)-bit counters) as the read and write addresses to access the designated element of the memory array.
- Compare the two (n + 1)-bit counters to obtain full and empty status.

To use this scheme in an **asynchronous** environment, we must ensure that the comparison circuit can generate the full and empty status signals that are synchronized with their respective clock domains. To accomplish this, we must revise the design as follows:

- Add a synchronizer in the comparison circuit to synchronize the pointer from the other clock domain
- Replace (n + 1)-bit and n-bit binary counters with the (n + 1)-bit and n-bit Gray counters, to avoid the skew effect when multiple bits change in the count word

Note that the MSb of the Gray counter is the same as the MSb of the binary counter, and thus it can be used to distinguish whether the FIFO is empty or full.
As in the binary counter-based implementation, we need two (n + 1)-bit Gray counters as the pointers and two n-bit Gray counters as the addresses.

In order to get the Gray counter, it can be converted from a binary counter. Let b_n, b_{n-2},... , b_0 the bits representing the binary numbers, where b_0 is the LSb and b_n is the MSb, and let a_n,... , a_0 be the bits representing the gray code of the binary numbers, where a_0 is the LSb and a_n is the MSb. Boolean expression for conversion of binary to gray code for (n+1)-bit is:

$$a_i = \begin{cases} b_i, \text{if } i = n \\ b_i \text{ XOR } b_{i-1}, \text{otherwise} \end{cases}.$$

It is straightforward to obtain the n-bit binary counting patterns since they are the same as the n LSbs of the (n + 1)-bit binary counter. It is more difficult for the Gray counter. For example, the counting patterns of three LSbs of the 4-bit Gray counter and the counting patterns of a 3-bit Gray counter are different in the bottom half.

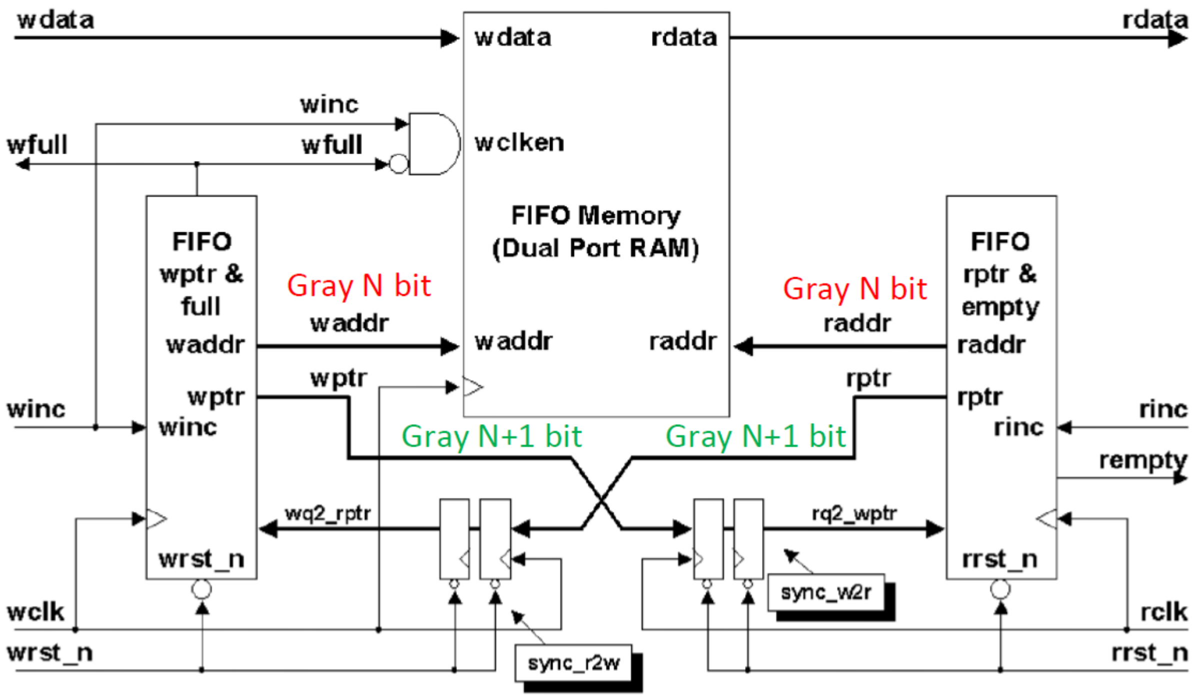
An (n + 1)-bit binary counter can be obtained from Gray counter values as follows, where a_n, a_{n-1},... , a_0 are the bits of an (n + 1)-bit Gray counter, and b_n, b_{n-1},... , b_0 be the bits of an (n + 1)-bit binary counter:

$$b_i = \begin{cases} a_i \text{ if } i = n \\ a_{i+1} \text{ XOR } a_i \text{ if } i = n-1 \\ b_{i+1} \text{ XOR } a_i, \text{otherwise} \end{cases}.$$

However, there is no need to construct a separate n-bit Gray counter from scratch. Closer observation shows that the (n - 1) LSbs of the (n + 1)-bit Gray counter and n-bit Gray counter are identical, and the MSb of the n-bit Gray counter can be obtained by performing an xor operation on the two MSbs of the (n + 1)-bit Gray counter, i.e. if a_n, a_{n-1},... , a_0 are the bits of an (n + 1)-bit Gray counter, and b_{n-1}, b_{n-2},... , b_0 be the bits of an n-bit Gray counter, we can derive the n-bit counting pattern by using:

$$b_i = \begin{cases} a_{i+1} \text{ XOR } a_i \text{ if } i = n-1 \\ a_i, \text{otherwise} \end{cases}.$$

In this lab you have to implement the logic interfacing with a memory array of an asynchronous FIFO implemented according to the following diagram:


wdata
rdata
wdata
rdata
winc
wfull
wfull
wclken
FIFO Memory (Dual Port RAM)
FIFO wptr & full
FIFO rptr & empty
Gray N bit
Gray N bit
waddr
waddr
waddr
raddr
raddr
raddr
wptr
wptr
rptr
rptr
winc
winc
rinc
rinc
rempty
Gray N+1 bit
Gray N+1 bit
wq2_rptr
rq2_wptr
wrst_n
rrst_n
sync_w2r
sync_r2w
wclk
wrst_n
rclk
rrst_n