Assignment Homework 6

Due date: Check Canvas

Student Version – "Clock Domain Crossing and Synthesis"

Please team up with a fellow student and start working on the following tasks (100 pts total). In addition, you *must* consider the following:

- Vivado creates a default template including a comment block with:
  - "Company" – insert team name/number
  - "Engineer" – put team member names here
  - "Design Name" – based on task description (*do not leave empty*)
  - "Project Name" – Assignment 6
  - "Description" – based on task description (*do not leave empty*)
- To prepare your assignment for submission through Canvas, you must use (in Vivado): `File > Project > Archive`
- Preparing your HDL project through this process is especially important when working with IP cores such as DSPs and BRAMs!
- You work as a team. Please alternate between one student creating the testbench, the other student creating the implementation, or other aspects of the implementation.

We will start this homework assignment together through Zoom as a lab. Please try to follow along and ask questions if you get stuck somewhere. Your homework assignment includes all tasks mentioned here (even those, that you manage to complete during our joint lab time).

1 Task

Modern ICs integrate multiple functional blocks that need to exchange data while operating at different clock frequencies. This introduces the need for implementing proper clock domain crossing (CDC) techniques to safely pass data across. One such popular technique for passing multi-bit signals across clock domains is an Asynchronous FIFO.

An Async FIFO, is a shared memory/register buffer where data is inserted from the write clock domain and data is removed from the read clock domain using a pair of read and write pointers. The write pointer always points to the next word to be written, while the read pointer points to the current FIFO word to be read; therefore, on reset, both pointers are set to zero. On a FIFO-write

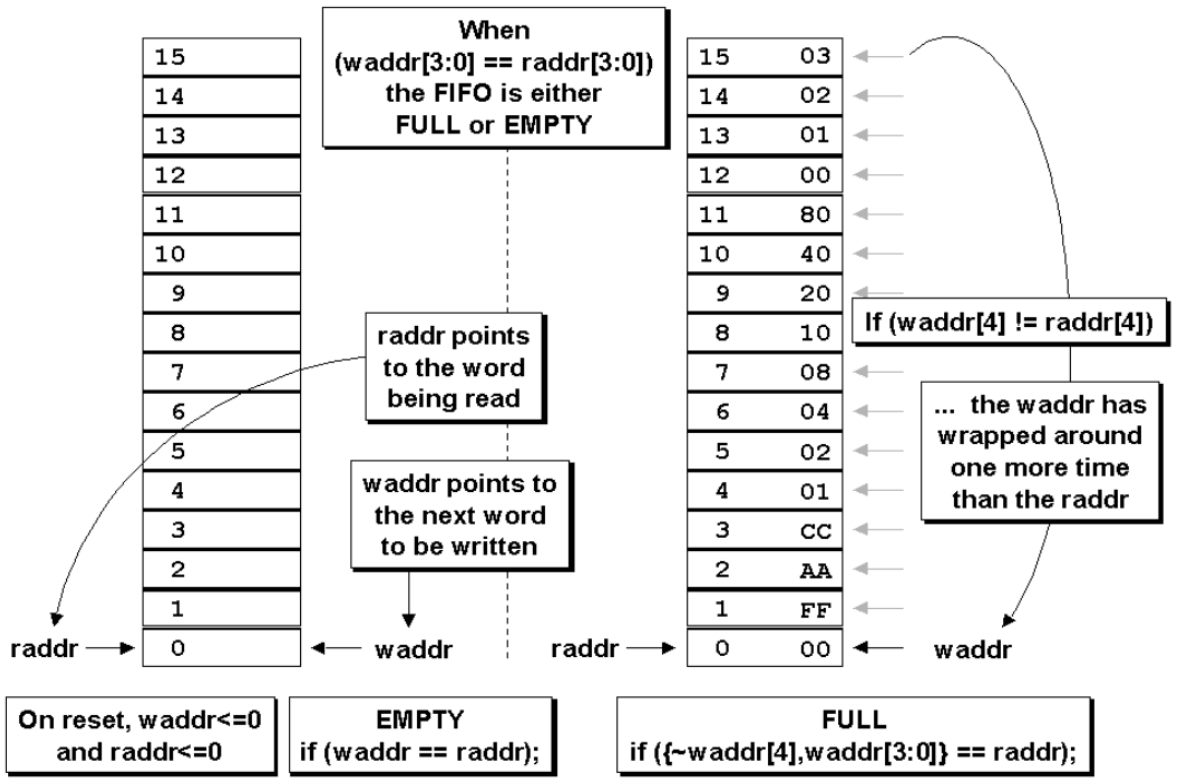
When (waddr[3:0] == raddr[3:0]) the FIFO is either FULL or EMPTY
raddr points to the word being read
waddr points to the next word to be written
If (waddr[4] != raddr[4])
... the waddr has wrapped around one more time than the raddr
raddr
waddr
15 03
14 02
13 01
12 00
11 80
10 40
9 20
8 10
7 08
6 04
5 02
4 01
3 CC
2 AA
1 FF
0 00
On reset, waddr<=0 and raddr<=0
EMPTY if (waddr == raddr);
FULL if ({~waddr[4],waddr[3:0]} == raddr);

Figure 1: Full and Empty FIFO flags

operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented. Similarly, when the FIFO is empty, the read pointer is pointing to an invalid data (the FIFO is empty, and the empty flag is asserted). Once the first word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the first memory word that the read pointer is still addressing is immediately driven onto the output port.

The FIFO is empty when the read and write pointers are equal, which happens upon reset, when both pointers are set to 0, or when the read pointer catches up to the write pointer. However, the pointers can also be equal when the write pointer has wrapped around one more time and the LSBs value caught up to the read pointer. This is an undesirable situation that can be addressed by making the pointers 1 bit wider than the FIFO depth. In which case, when the write pointer increments past the final FIFO address, the MSB is set, and can be used to properly detect this scenario, as seen in Figure 1.

Synchronizing a binary count value from one clock domain to another is problematic, because multiple signals synchronized to one clock will experience skew that might cause each signal to be sampled at slightly different times in the second clock domain. One efficient multi-bit CDC strategy that we can use is to encode the read and write pointers using Gray codes.

Following the diagram in Figure 2, design and implement an asynchronous FIFO, as described above, using Gray-encoded write and read pointers synchronized in the opposite clock domain before generating the full or empty status flags.

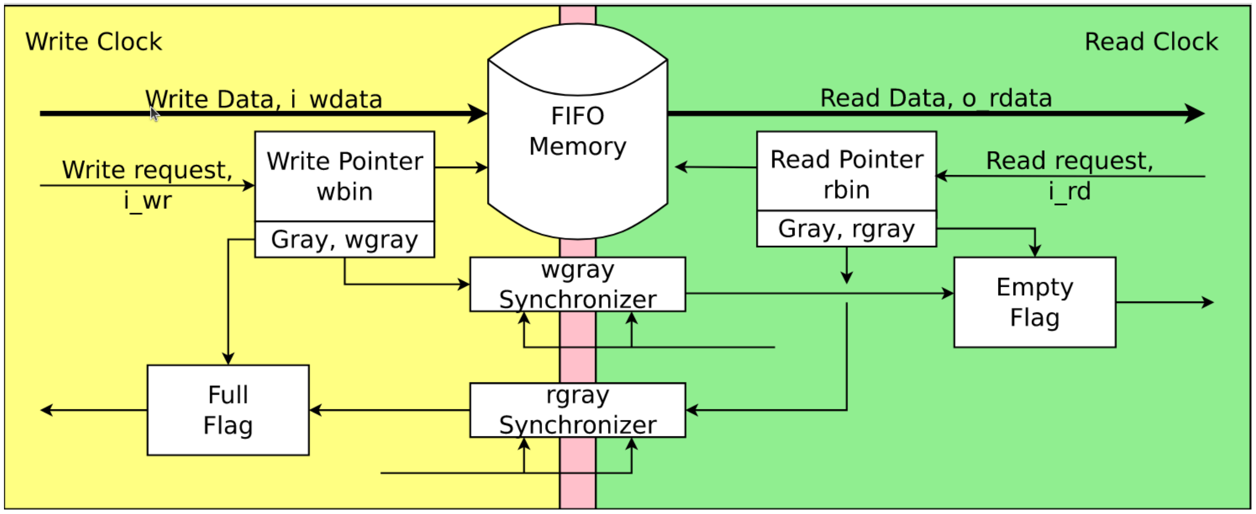

Figure 2: Read and write pointers crossing clock domains need to be synchronized in the new clock domain

2 Output

Create a verilog module called asyncfifo.v and a testbench asyncfifo_tb.v. Asyncfifo.v design file should contain the Asynchronous FIFO sythesizable RTL design. The testbench should implement 3 tests:

1. Verify for correct Binary to Gray code conversion
2. Check that the FIFO full status flag is asserted when FIFO becomes full
3. Check that the FIFO full empty flag is asserted when FIFO becomes empty

3 Hints

1. FIFO Memory, for this purpose, can simply be an array of memory elements, ex: reg [DATA_WIDTH-1:0] memory [(1 « ADDR_WIDTH) - 1 : 0], where DATA_WIDTH and ADDR_WIDTH are verilog module parameters for the FIFO data bit width and FIFO depth. A FIFO depth of 16 (ADDR_WIDTH = 4) should be enough.