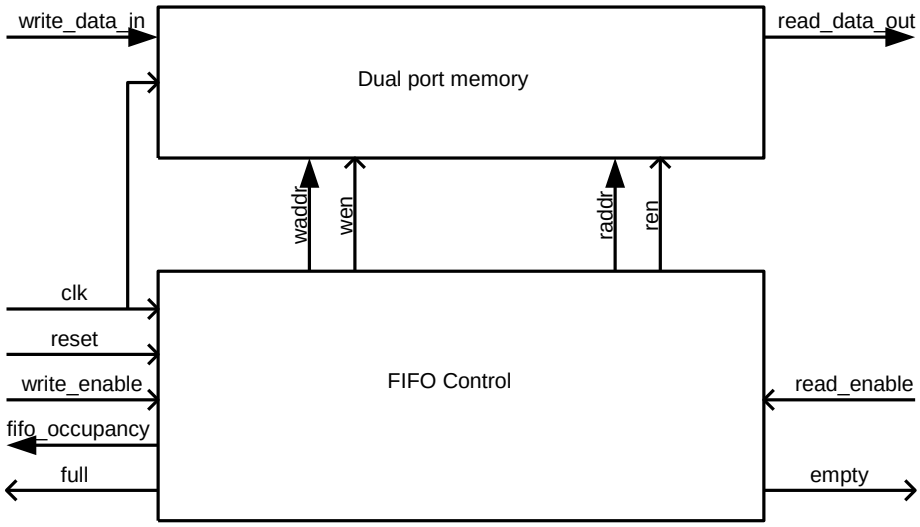
EXERCISE 2 (Asynchronous FIFO)

The purpose of this exercise is to design an asynchronous FIFO for communication between different clock domains. First, synchronous FIFO design is reviewed followed by an introduction to asynchronous FIFO design.

Synchronous FIFO

A Block diagram of a synchronous FIFO is shown below. It is synchronous because only one clock signal exists, and this clock signal is used for both writing and reading.

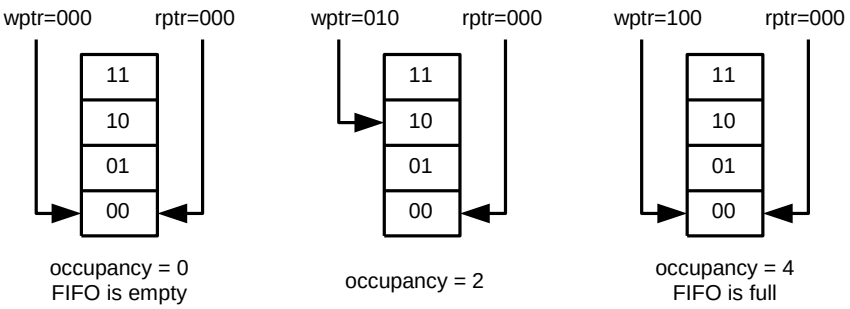


We assume that the Dual port memory has 2^n locations, i.e. the write and read addresses have n bits: waddr[n-1:0],raddr[n-1:0]. Internally in the FIFO control module, pointers are used to determine memory addresses, full status, empty status and FIFO occupancy. As we will see in the following, it is convenient to have (n+1) pointer bits. Thus, the write and read addresses are determined from the n least significant pointerbits:
waddr[n-1:0] = wptr[n-1:0],
raddr[n-1:0] = rptr[n-1:0].

Full/empty flags.
The write pointer is incremented for each write operation. The most significant bit (wptr[n]) indicates if the write address has wrapped around an even or uneven number of times. Ditto for the read pointer.
If (waddr=raddr) the FIFO is either full or empty. The additional pointer bit can be utilized to distinguish between the two situations: if (wptr[n]=rptr[n]) the FIFO is empty, and if (wptr[n]!=rptr[n]) the FIFO is full.

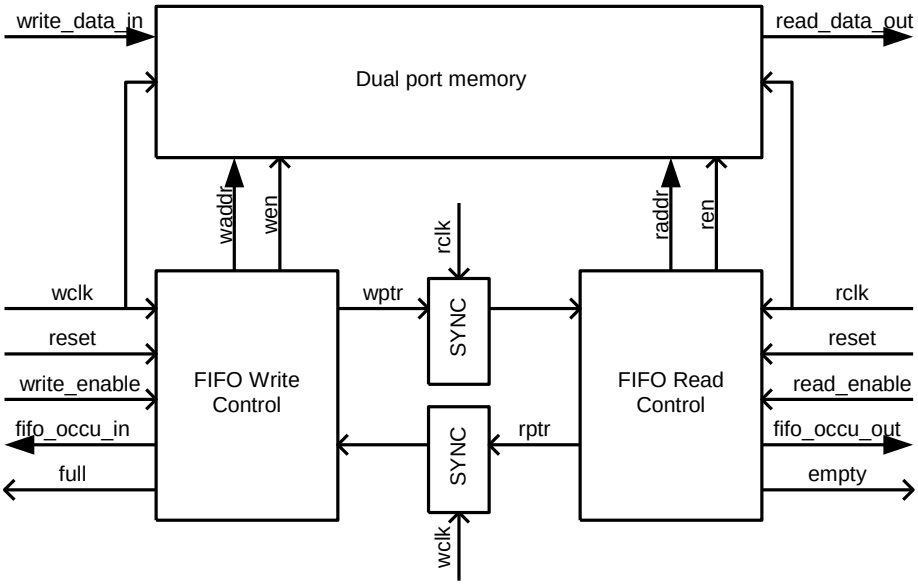
FIFO occupancy.
The FIFO occupancy can be calculated from the pointer values. In this case, the most significant pointer bits are not necessarily utilized to perform the calculation:
(wptr[n]=rptr[n]) : fifo_size = wptr[n-1:0] - rptr[n-1:0] = wptr-rptr,
(wptr[n]!=rptr[n]) : fifo_size = 2^n – (rptr[n-1:0] - wptr[n-1:0])) = wptr-rptr

The relationship between FIFO pointers and occupancy is illustrated below with three examples.



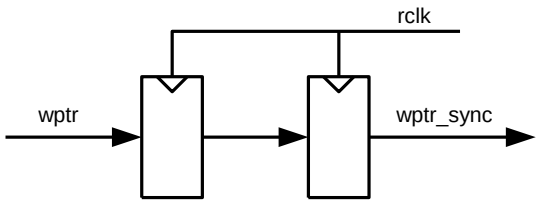
Asynchronous FIFO

Asynchronous FIFOs are commonly used to transfer data between two clock domains. A block diagram is shown below. In order to determine full/empty flags and FIFO size, the read pointer (rptr) must be synchronized to the write domain, and the write pointer (wptr) must be synchronized to the read-domain.



TASK 1

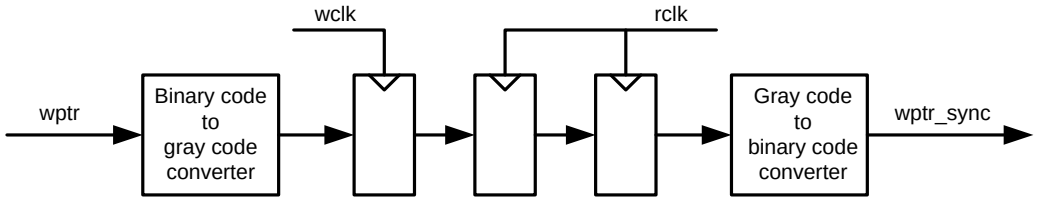
Typically, synchronization is achieved by a chain of flip-flops to resolve metastability, as shown below:



Explain why this circuit is insufficient for use in the asynchronous fifo design. Would it be beneficial to add more flip-flops in the chain?

TASK 2

Correct synchronization is obtained with the circuit below. Explain why. Specify logical equations for the binary to gray code converter and for the gray to binary code converter.



The synchronization circuit introduces a delay. Is this a problem?

TASK 3

Implement the Asynchronous FIFO. The entity declaration is given below. The tool can generate the dual-port memory. Note that the FIFO has **16 memory locations**.

```
entity async_fifo is
  port (
    reset          : in std_logic;
    wclk           : in std_logic;
    rclk           : in std_logic;
    write_enable   : in std_logic;
    read_enable    : in std_logic;
    fifo_occu_in   : out std_logic_vector(4 downto 0);
    fifo_occu_out  : out std_logic_vector(4 downto 0);
    write_data_in  : in std_logic_vector(7 downto 0);
    read_data_out  : out std_logic_vector(7 downto 0)
  );
end async_fifo;
```

Synthesize and place & route the code and note the clock frequency (device: Stratix IV EP4SGX230KF40C2). Verify the design in a testbench.

Report

The report must contain:

- Answers to questions in task 1 & task 2
- VHDL code
- Test results from simulation
- Synthesis report, Place & route report