**FIFO**

A FIFO or Queue is an array of memory commonly used in hardware to transfer transfer data between two circuits with different clocks. There are many other use of FIFO also. FIFO uses a dual port memory and there will be two pointers to point read and write addresses. Here is a generalized block diagram of FIFO.

A screenshot of a cell phone

Description automatically generated

Generally fifos are implemented using rotating pointers. We can call write and read pointers of a FIFO as head and tail of data area. Initially read and write pointers of the FIFO will point to the same location. In a fifo with n locations, after writing data to 'n'th location, write pointer points to 0th location.

Here is an example to explain how FIFO uses the memory. This is a fifo of length 8, WP and RP are the locations where write pointer and read pointer points. Shaded area in the diagram is filled with data.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| One data pushed to fifo: | | | | | | | |
| RP | WP |  |  |  |  |  |  |
| 5 more data pushed to fifo | | | | | | | |
| RP |  |  |  |  |  | WP |  |
| 3 data popped from fifo | | | | | | | |
|  |  |  | RP |  |  | WP |  |
| 3 more data pushed to fifo | | | | | | | |
|  | WP |  | RP |  |  |  |  |
| 2 more data pushed to fifo: fifo full | | | | | | | |
|  |  |  | WP, RP |  |  |  |  |
| 6 data popped from fifo | | | | | | | |
|  | RP |  | WP |  |  |  |  |

A FIFO may be synchronous or asynchronous. There is no clock in asynchronous fifo. In synchronous fifo, there may be 1 or 2 clocks since some FIFOs have separate clocks for read and write. Asynchronous fifos are not used commonly now a days because synchronous FIFOs have improved interface timing.

When ever FIFO counter becomes zero or BUF\_LENGTH, empty or full flags will beset.

fifo\_counter is incremented if write takes place and buffer is not full and will be decremented id read takes place and buffer is not empty. If both read and write takes place, counter will remain the same.

rd\_ptr and wr\_ptr are read and write pointers. Since we selected the bits in these registers same as address width of buffer, when buffer overflows, values will overflow and become 0.

Following testbench can be used to test the fifo code. Some push and pop are made to test normal full and empty conditions. We can observe in the output that we popped the values in the order we pushed. Simultaneous push and pop is tested using fork-join once.

Tasks 'push' and 'pop' are used to write and read data from the queue. After placing the data and address in the appropriate port, task waits for a clock to come. Then data is read from the port. In the output, you can notice that it displays 'cannot push' message when data pushed is BUF\_SIZE and displays 'cannot pop' when all the data are popped.

Here is the output of displayed:

# Pushed 1  
# Pushed 2  
# -------------------------------Poped 1  
# Pushed 10  
# Pushed 20  
# Pushed 30  
# Pushed 40  
# Pushed 50  
# Pushed 60  
# Pushed 70  
# ---Cannot push: Buffer Full---  
# ---Cannot push: Buffer Full---  
# ---Cannot push: Buffer Full---  
# ---Cannot push: Buffer Full---  
# ---Cannot push: Buffer Full---  
# ---Cannot push: Buffer Full---  
# -------------------------------Poped 2  
# Pushed 2  
# -------------------------------Poped 10  
# -------------------------------Poped 20  
# -------------------------------Poped 30  
# -------------------------------Poped 40  
# Pushed 140  
# -------------------------------Poped 50  
# Pushed 50  
# -------------------------------Poped 60  
# -------------------------------Poped 70  
# -------------------------------Poped 2  
# -------------------------------Poped 140  
# -------------------------------Poped 50  
# ---Cannot Pop: Buffer Empty---  
# ---Cannot Pop: Buffer Empty---  
# ---Cannot Pop: Buffer Empty---  
# ---Cannot Pop: Buffer Empty---  
# ---Cannot Pop: Buffer Empty---  
# ---Cannot Pop: Buffer Empty---  
# Pushed 5  
# -------------------------------Poped 5