VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Logic Design with HDL (CO1025)

Assignment

FIFO - First in first out

Advisor:	Assoc. Prof. Dr. Pham Quoc Cuong
Students:	Tran Nhat Tan - 1852732
	Nguyen Huu Tra Giang - 2153315
	Ly Minh Quang - 2153722
	Phan Van Minh Tien - 2153888

HO CHI MINH CITY, MAY 2022

Contents

1 Introduction

A FIFO is a special type of buffer. The name FIFO stands for first in first out and means that the data written into the buffer first comes out of it first. There are other kinds of buffers like the LIFO (last in first out), often called a stack memory, and the shared memory. The choice of a buffer architecture depends on the application to be solved.

FIFOs can be implemented with software or hardware. The choice between a software and a hardware solution depends on the application and the features desired. When requirements change, a software FIFO easily can be adapted to them by modifying its program, while a hardware FIFO may demand a new board layout. Software is more flexible than hardware. The advantage of the hardware FIFOs shows in their speed.

Thus, we decided to choose this topic in order to see how a FIFO work in the digital system.

2 Theory base

2.1 What is FIFO?

In computing and in systems theory, FIFO an acronym for first in, first out (the first in is the first out) is a method for organizing the manipulation of a data structure (often, specifically a data buffer) where the oldest (first) entry, or "head" of the queue, is processed first.

Such processing is analogous to servicing people in a queue area on a first-come, first-served (FCFS) basis, i.e. in the same sequence in which they arrive at the queue's tail.

FCFS is also the jargon term for the FIFO operating system scheduling algorithm, which gives every process central processing unit (CPU) time in the order in which it is demanded.[1] FIFO's opposite is LIFO, last-in-first-out, where the youngest entry or "top of the stack" is processed first.[2] A priority queue is neither FIFO or LIFO but may adopt similar behaviour temporarily or by default. Queueing theory encompasses these methods for processing data structures, as well as interactions between strict-FIFO queues.
Depending on the application, a FIFO could be implemented as a hardware shift register, or using different memory structures, typically a circular buffer or a kind of list. For information on the abstract data structure, see Queue (data structure). Most software implementations of a FIFO queue are not thread safe and require a locking mechanism to verify the data structure chain is being manipulated by only one thread at a time.

FIFOs are commonly used in electronic circuits for buffering and flow control between hardware and software. In its hardware form, a FIFO primarily consists of a set of read and write pointers, storage and control logic. Storage may be static random access memory (SRAM), flip-flops, latches or any other suitable form of storage. For FIFOs of non-trivial size, a dual-port SRAM is usually used, where one port is dedicated to writing and the other to reading.

The first known FIFO implemented in electronics was by Peter Alfke in 1969 at Fairchild Semiconductor. Alfke was later a director at Xilinx.

2.2 FIFO types

Every memory in which the data word that is written in first also comes out first when the memory is read is a first-in first-out memory. Figure 1 illustrates the data flow in a FIFO. There are three kinds of FIFO:

- Shift register – FIFO with an invariable number of stored data words and, thus, the necessary synchronism between the read and the write operations because a data word must be read every time one is written
- Exclusive read/write FIFO – FIFO with a variable number of stored data words and, because of the internal structure, the necessary synchronism between the read and the write operations
- Concurrent read/write FIFO – FIFO with a variable number of stored data words and possible asynchronism between the read and the write operation

2.2.1 Shift register

The shift register is not usually referred to as a FIFO, although it is first-in first-out in nature. Consequently, this application report focuses exclusively on FIFOs that handle variable-length data.

2.2.2 Exclusive read/write FIFO

In exclusive read/write FIFOs, the writing of data is not independent of how the data are read. There are timing relationships between the write clock and the read clock. For instance, overlapping of the read and the write clocks could be prohibited. To permit use of such FIFOs between two systems that work asynchronously to one another, an external circuit is required for synchronization. But this synchronization circuit usually considerably reduces the data rate.

2.2.3 Concurrent read/write FIFO

In concurrent read/write FIFOs, there is no dependence between the writing and reading of data. Simultaneous writing and reading are possible in overlapping fashion or successively. This means that two systems with different frequencies can be connected to the FIFO. The designer need not worry about synchronizing the two systems because this is taken care of in the FIFO. Concurrent read/write FIFOs, depending on the control signals for writing and reading, fall into two groups:

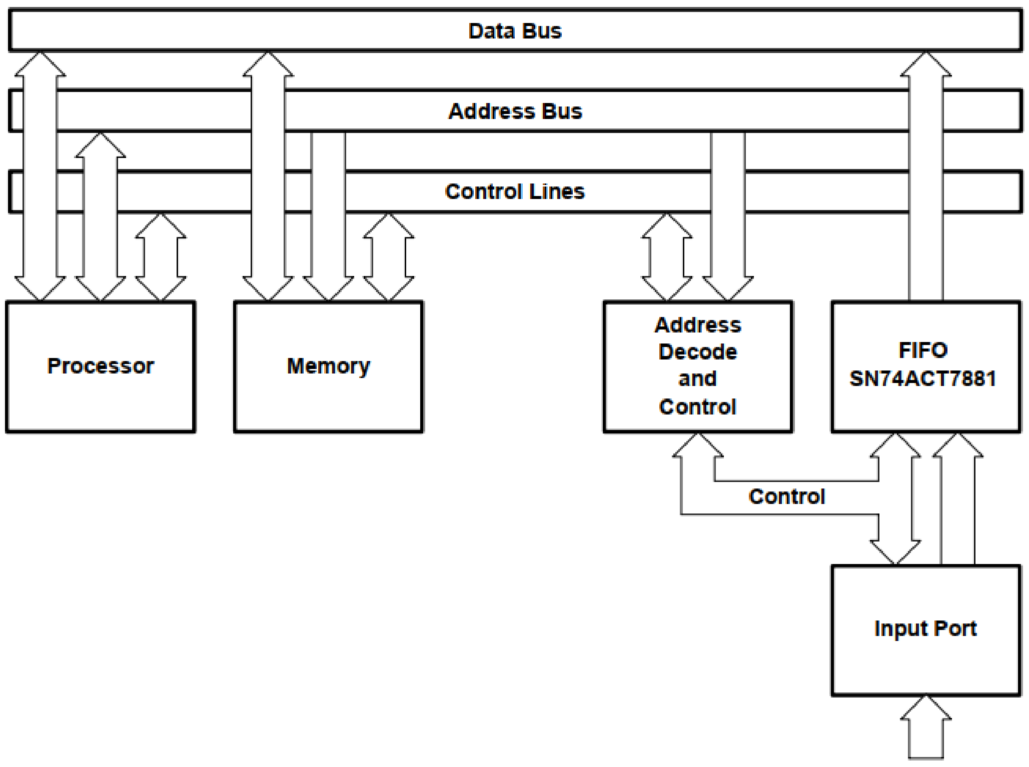
- Synchronous FIFOs
- Asynchronous FIFOs

This assignment only focus on synchronous FIFO

2.3 Application of FIFO

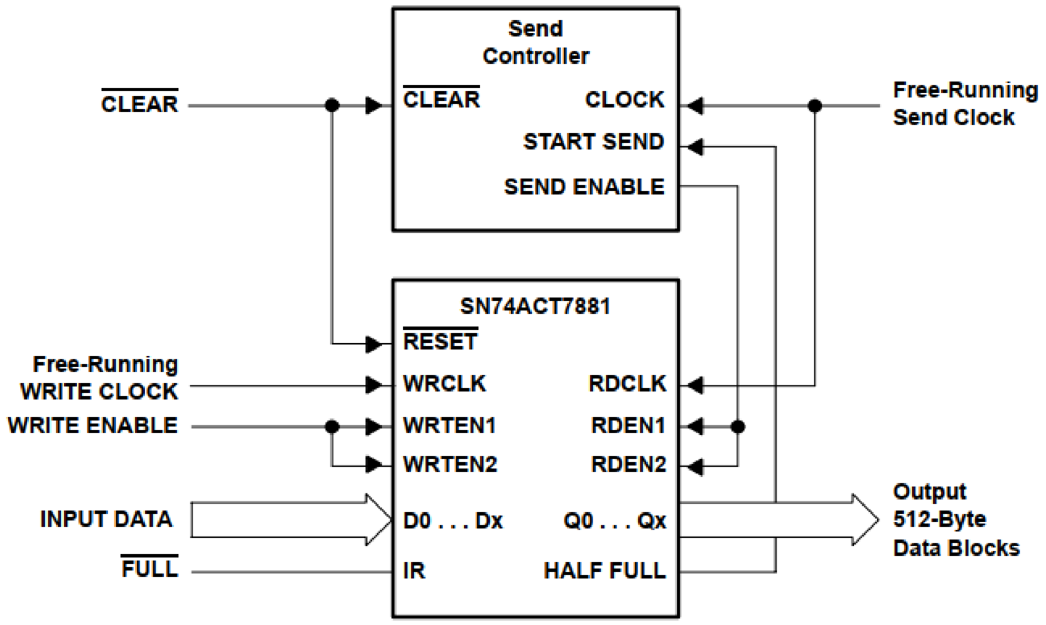
2.3.1 Connection of Peripherals to Processors

Modern processors are often considerably faster than peripherals that are connected to them. FIFOs can be used so that the processing speed of a processor need not be reduced when it exchanges data with a peripheral. If the peripheral is sometimes faster than the processor, a FIFO can again be used to resolve the problem. Different variations of circuitry are possible, depending on the particular problem.



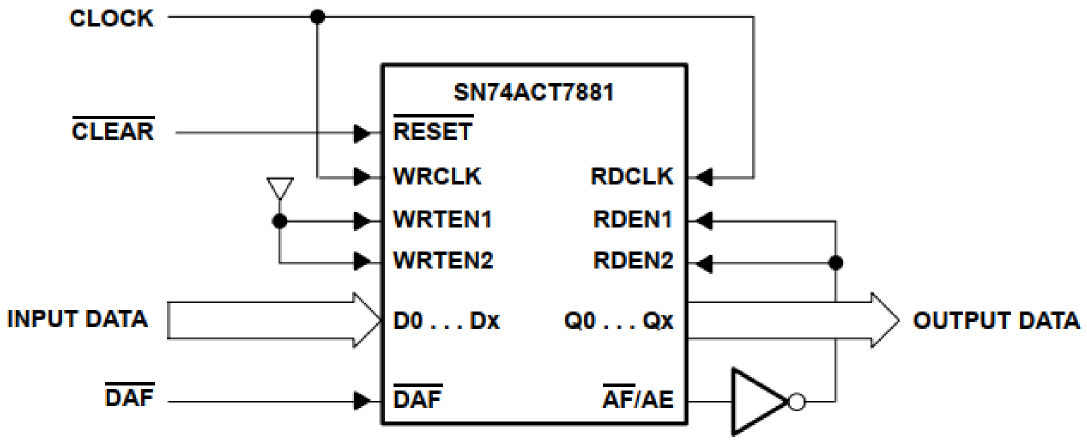
2.3.2 Block Transfer of Data

Data are often split into blocks and transmitted on data lines. Computer networks and digital telephone-switching installations are examples of this application. If such a conversion into blocks has to be made at very high speed, it is possible only with appropriate hardware, not through software



2.3.3 Programmable Delay

With FIFOs, it is possible to implement a programmable, digital delay line with minimum effort. Because of the programmable AF/AE (ALMOST FULL/ALMOST EMPTY) flag of the SN74ACT7881, only one inverter in addition to the FIFO is necessary



3 FIFO Implementation

3.1 Verilog code

Procedure to implement FIFO:

- Create a normal memory in Verilog.
- When the data and push signal is given, write to the memory starting from first address.
- When pop signal is given, read from the memory from the first address.
- When FIFO becomes empty, assert empty and if it becomes full, assert full signal.

The idea is that we use an array for output data and use two pointers called **read_pointer** and **write_pointer**. There are two cases:

- If the last operation was push and the pointers become equal, then the FIFO is full.
- If the last operation was pop and the pointers become equal, then the FIFO is empty.

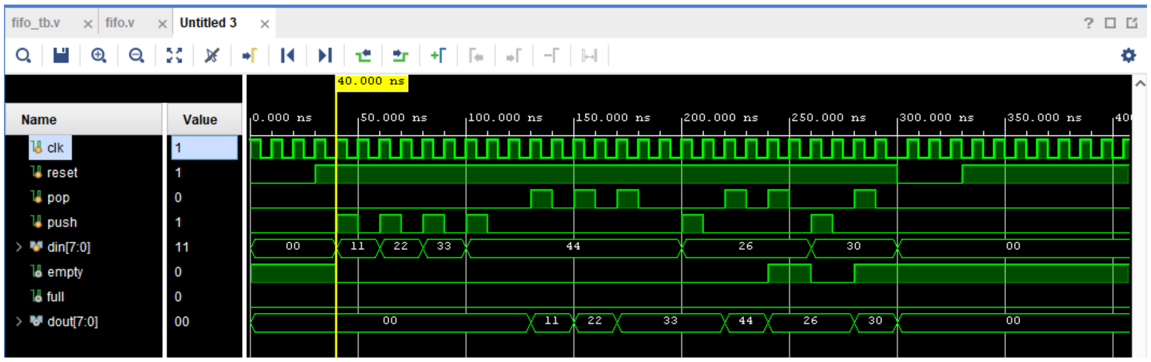
Thus, we use two variable to check if the FIFO is empty or full:

```
assign fullcheck = push && !(|(writeptr^(readptr-1'b1)));
assign emptycheck = pop && !(|(readptr^(writeptr-1'b1)));
```

***Full code:**

```
module sync_fifo(
 clk,
 reset,
 pop,
 push,
 empty,
 full,
 din,
 dout
    );

parameter PTR_WIDTH = 3;
parameter DATA_WIDTH = 8;
parameter DEPTH = 8;

input clk;
input reset;
input pop;
input push;
input [DATA_WIDTH-1:0]din;

output [DATA_WIDTH-1:0]dout;
output empty;
output full;

reg [DATA_WIDTH-1:0]fifo[DEPTH-1:0];
reg [PTR_WIDTH-1:0]readptr, next_rdptr;
reg [PTR_WIDTH-1:0]writeptr, next_wrptr;
reg [DATA_WIDTH-1:0]dout, next_dout;
reg empty, next_empty;
reg full, next_full;

assign fullcheck = push && !(|(writeptr^(readptr-1'b1)));
assign emptycheck = pop && !(|(readptr^(writeptr-1'b1)));

always @ (posedge clk) //Sequential block
begin
  if(!reset)
  begin
    dout     <= 8'd0;
    empty    <= 1'b1;
    full     <= 1'b0;
    readptr    <= 1'b0;
    writeptr    <= 1'b0;
  end
  else
  begin
    dout     <= next_dout;
    empty    <= next_empty;
    full     <= next_full;
    readptr    <= next_rdptr;
    writeptr    <= next_wrptr;
  end
end

always @ (*) //Combinational Block
begin
  next_dout    = dout;
  next_empty    = emptycheck ? 1'b1 : push ? 1'b0 : empty;
  next_full    = fullcheck ? 1'b1 : pop ? 1'b0 : full;
  next_rdptr    = readptr;
  next_wrptr    = writeptr;

  if(push)  //write
  begin
      fifo[writeptr] = din;
      next_wrptr   = writeptr+1;
  end
  else if(pop)  //read
  begin
      next_dout    = fifo[readptr];
      next_rdptr    = readptr+1;
  end
  else
  begin
      next_dout   = dout;
      next_rdptr = readptr;
      next_wrptr = writeptr;
  end
end
endmodule
```

3.2 Testbench



- From 0 to 40ns, the FIFO is empty (empty signal is high and full signal is low).
- At 40ns, push signal is hight and the value 11 is pushed to the FIFO, empty signal becomes low
- At 60ns, 80ns and 100ns, 22, 33 and 44 are pushed to the FIFO
- At 130ns, 150ns and 170ns the pop signal is high and 11, 22, 33 was popped from the FIFO, dout get the respective value, only 44 remains in the FIFO
- At 200ns, 26 is pushed to the FIFO, pop signal is high and the FIFO has 44 and 26
- At 220ns and 240ns, 26 and 44 are popped from the FIFO and the FIFO become empty (empty signal is high)
- At 260ns, 30 is pushed to the FIFO, the empty signal becomes low
- At 280ns, 30 is popped from the FIFO and the empty signal becomes high

4 Conclusion

Because of their versatile possibilities of use, FIFOs present solutions to many different applications.

There are different kinds of FIFOs according to their functions. Especially attractive features are offered by asynchronous FIFOs and synchronous FIFOs.

The implementation of FIFO in hardware is different from those of software