Generic FIFO Buffer and Simulation

Submitted by ~

Keshav Singh (2K19/EE/134)

Submitted to ~

Garima Solanki Ma’am

Index

Introduction

In every item of digital equipment there is exchange of data between printed circuit boards (PCBs). Intermediate storage or buffering always is necessary when data arrive at the receiving PCB at a high rate or in batches, but are processed slowly or irregularly.

- Buffers of this kind also can be observed in everyday life (for example, a queue of customers at the checkout point in a supermarket or cars backed up at traffic lights).
- The checkout point in a supermarket works slowly and constantly, while the number of customers coming to it is very irregular. If many customers want to pay at the same time, a queue forms, which works by the principle of first come, first served.

Why FIFO ?

In electronic systems, buffers of this kind also are advisable for interfaces between components that work at different speeds or irregularly. Otherwise, the slowest component determines the operating speed of all other components involved in data transfer.

In a compact-disk player, for instance, the speed of rotation of the disk determines the data rate. To make the reproduced sound fluctuations independent of the speed, the data rate of the A/D converter is controlled by a quartz crystal. The different data rates are compensated by buffering. In this way, the sound fluctuations are largely independent of the speed at which disks rotate.

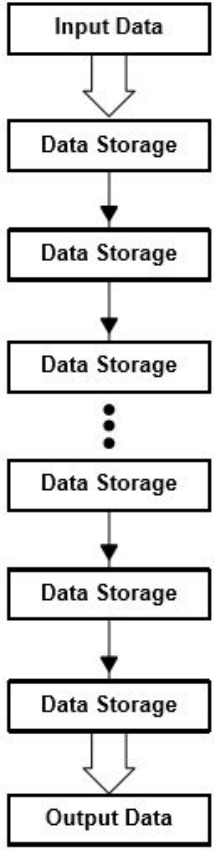
Types of FIFO

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 on the right sides 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**



Shift Registers

- A shift register is a type of digital circuit using a cascade of flip flops where the output of one flip-flop is connected to the input of the next. They share a single clock signal, which causes the data stored in the system to shift from one location to the next.
- The shift register is not usually referred to as a FIFO, although it is first-in first-out in nature.
- The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock

Exclusive read/write FIFO

If certain timing conditions must be maintained between the writing and the reading systems, we speak of exclusive read/write FIFOs because the two systems must be synchronized.

The first FIFO designs to appear on the market were exclusive read/write because these were easier to implement.

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.

Concurrent Read/Write FIFO

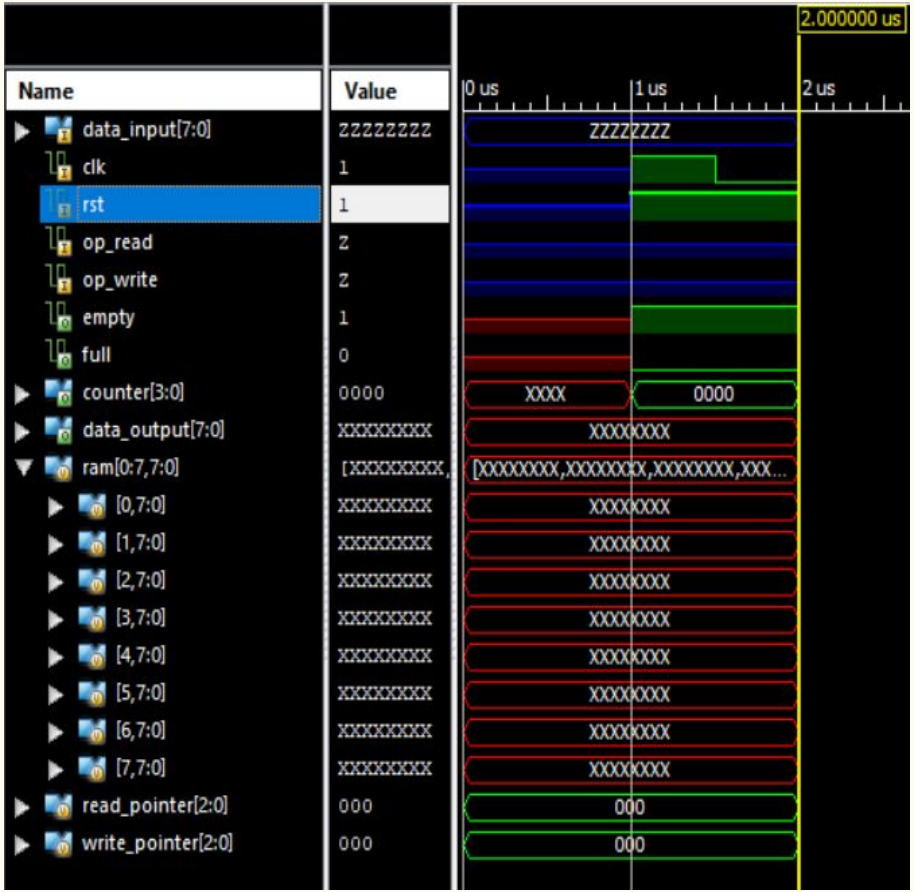
If there are no timing restrictions in how the systems are driven, meaning that the writing system and the reading system can work out of synchronism, the FIFO is called concurrent read/write.

Nearly all present FIFOs are concurrent read/write because so many applications call for concurrent read/write versions. Concurrent read/write FIFOs can be used in synchronous systems without any difficulty.

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

Simulation



The simulations presented here are performed using the Xilinx ISE software, and the source code in later slides.

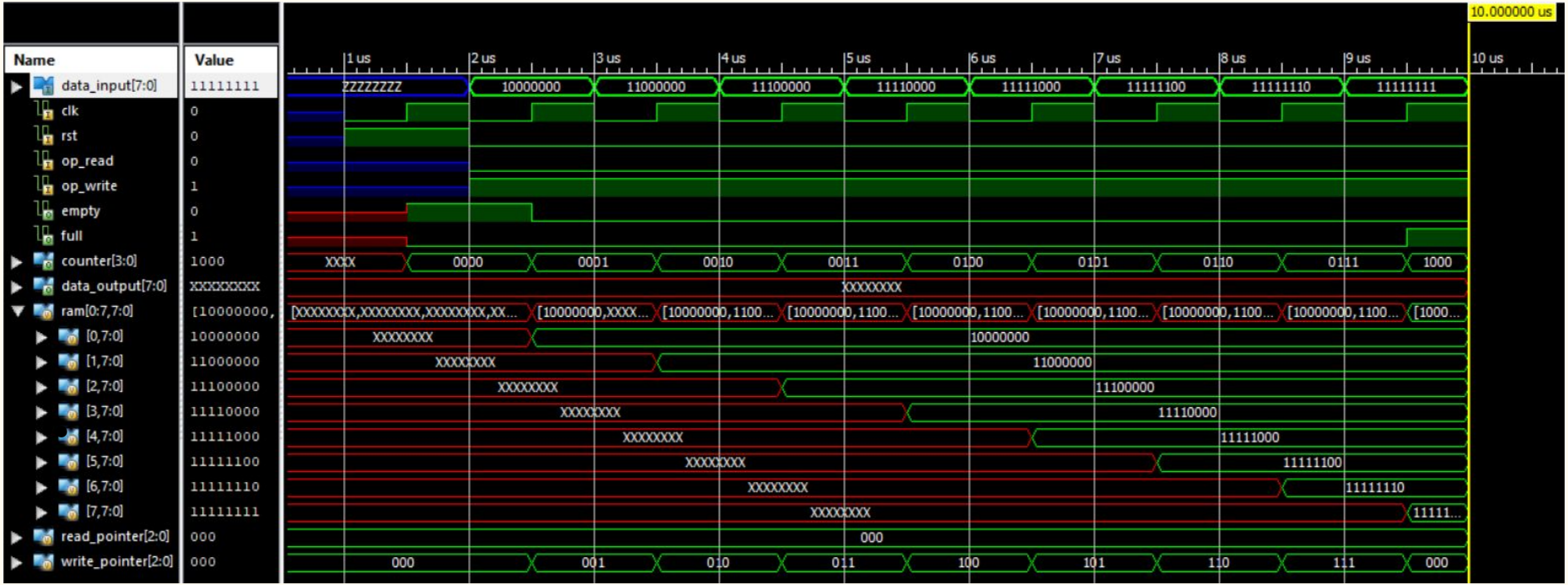
Initially, only the read and write pointers are assigned zero value.

First we allot a clock cycle of Time Period =1000000 and reset the buffer.

When Reset operation is set to high,

- The FIFO memory is made empty by setting the ‘empty’ bit to ‘1’ and ‘full’ bit to ‘0’.
- The counter is also set to zero.

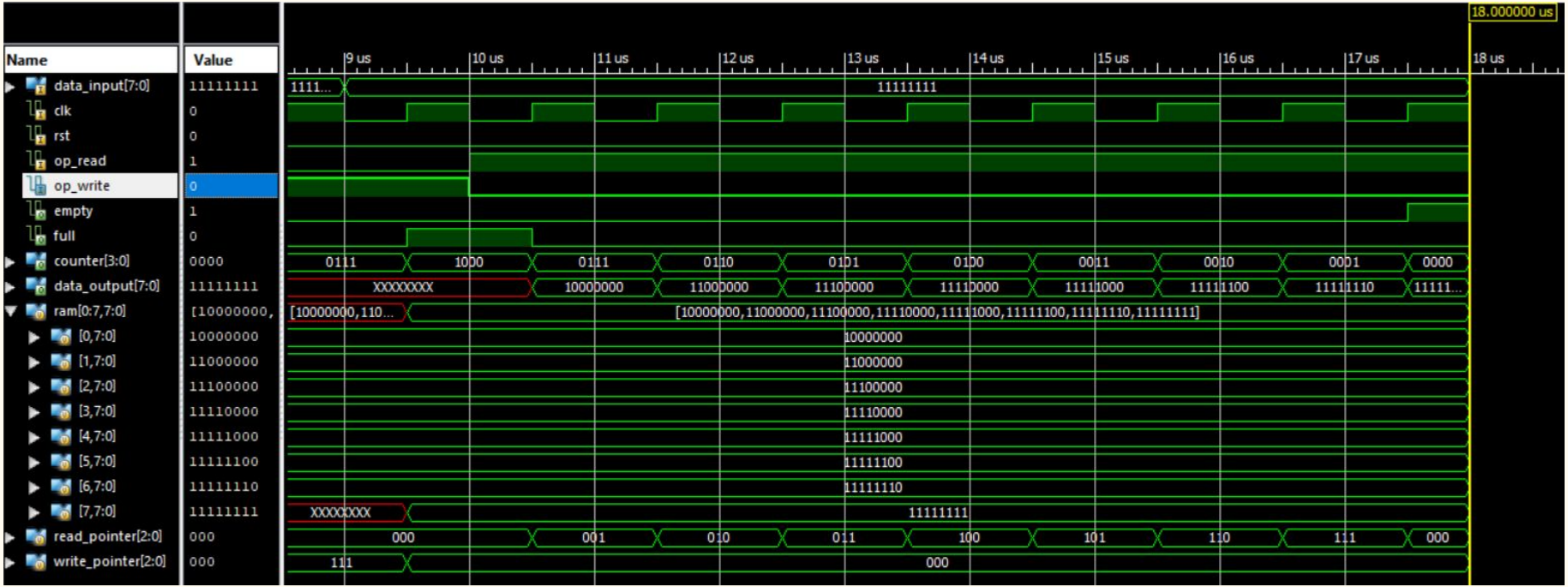
Write Operation



Some points of observation in this WRITE operation are as follows:

- After setting reset bit to 0. Write Operation is performed 8 times on FIFO memory and 8 binary(8 bit) numbers are fed to the RAM.
- As soon as the first binary number is fed into FIFO memory, the ‘empty’ variable changes to ‘0’, indicating that the previously empty memory is no longer empty anymore.
- Each write operation is performed when positive edge of clock is encountered,
- And every time a binary number is stored in the memory the ‘write_pointer’ as well as the ‘counter’ variable increases by 1.
- Counter variable shows the present occupied state of the memory i.e. how much data is stored in the FIFO memory.
- After feeding the 8th binary number, ‘full’ bit is changed to ‘1’ and the ‘write_pointer’ also resets to ‘0’.
- Any data fed into the memory past this point will not be fed into the FIFO memory and will be lost.

Read Operation



Some Observations related to READ operation:

- Read operation is performed on the already full FIFO from writing done on the previous page.
- Read Operation can show us clearly that the first binary number to be fed into the memory was also the first binary number to be fed to the data output line, and hence the name First In First Out (FIFO).
- As soon as the first binary number is fed to the output line, the ‘full’ variable changes from ‘1’ to ‘0’, indicating that the previously full memory is no longer full.
- Each read operation is performed when positive edge of clock is encountered, and every time a binary number is read from the memory and fed to the data_output line the ‘read_pointer’ variable increases by 1, whereas the ‘counter’ variable decreases by 1.
- Counter variable shows the present occupied state of the memory i.e. how much data is stored in the FIFO memory.
- After reading the 8th binary number, ‘empty’ bit is changed to ‘1’ and the ‘read_pointer’ also resets to ‘0’.
- If anymore data is tried to read from the FIFO memory, the FIFO memory will simply return the last data read.

Source Code

The code given below is written in Verilog Hardware Description Language.

```
module fifo( input [7:0] data_input, input clk, rst, op_read, op_write,
             output empty, full, output reg [3:0] counter, output reg [7:0] data_output);

reg [7:0] ram[0:7];
reg [2:0] read_pointer=0, write_pointer=0;
assign empty=(counter==0)?1:0;
assign full=(counter==8)?1:0;

//////////WRITE OPERATION//////////

always @ (posedge clk) begin: write
   if(op_write && !full)
      ram[write_pointer]<= data_input;
   else if(op_write && op_read)
      ram[write_pointer]<= data_input;
end
```

```
//////////READ OPERATION//////////

always @ (posedge clk) begin: read
   if(op_read && !empty)
      data_output<= ram[read_pointer];
   else if(op_write && op_read)
      data_output<= ram[read_pointer];
end

//////////POINTER//////////

always @ (posedge clk) begin: pointer
   if(rst) begin
      write_pointer <=0;
      read_pointer <=0;
   end else begin
      write_pointer <= ((op_write && !full) || (op_write && op_read)) ? write_pointer+1 : write_pointer;
      read_pointer <= ((op_read && !empty) || (op_write && op_read)) ? read_pointer+1 : read_pointer;
   end
   if(write_pointer==8)
      write_pointer=0;
   else if(read_pointer==8)
      read_pointer=0;
end
```

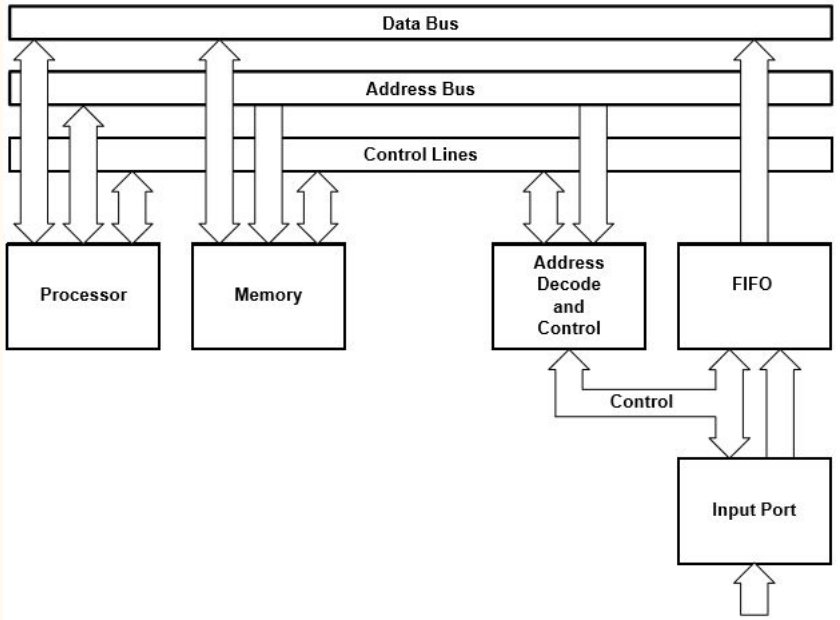
```
//////////COUNTER//////////

always @ (posedge clk) begin: count
   if(rst) counter <=0;
   else begin
      case({op_write,op_read})
         2'b00: counter <= counter;
         2'b01: counter <= (counter==0) ? 0 : counter-1;
         2'b10: counter <= (counter==8) ? 8 : counter+1;
         2'b11: counter <= counter;
         default: counter <= counter;
      endcase
   end
end

endmodule
```

Applications Examples

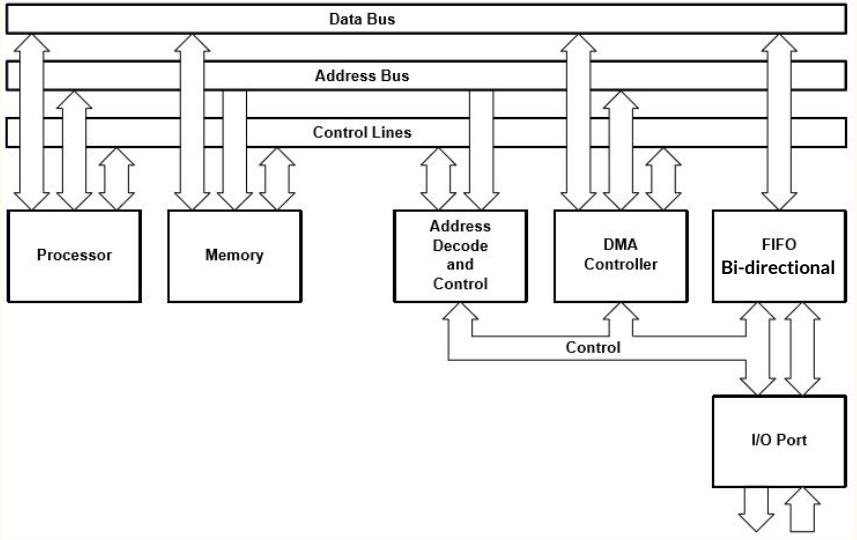
Connection of Peripherals to Processors

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.

In some cases, the processor reads input data over a unidirectional peripheral, for example, from an A/D converter. A FIFO can buffer a certain amount of input data, then use an interrupt so that the processor reads the data. This interrupt can be triggered by a HALF FULL, ALMOST FULL, or FULL flag.

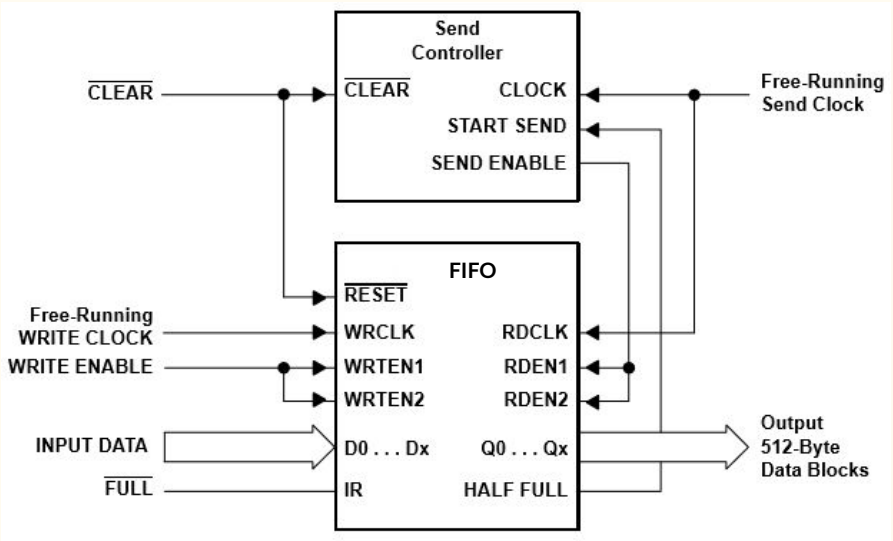


- Often the connected peripherals are bidirectional, like a parallel port, a serial port, a hard-disk controller, or the interface with a magnetic-tape drive.
- The bidirectional FIFO also can be combined with a DMA(Direct Memory Access) controller. Without a FIFO, the DMA controller normally requests from the processor control of the bus for the entire duration of data transfer.
- This requires control of the bus only during data transfer from FIFO to RAM, after which control can be returned to the processor. The data are transferred block by block from the FIFO into the RAM.
- A FIFO can be used in the same way to optimize data transfer from a processor or RAM to a peripheral.



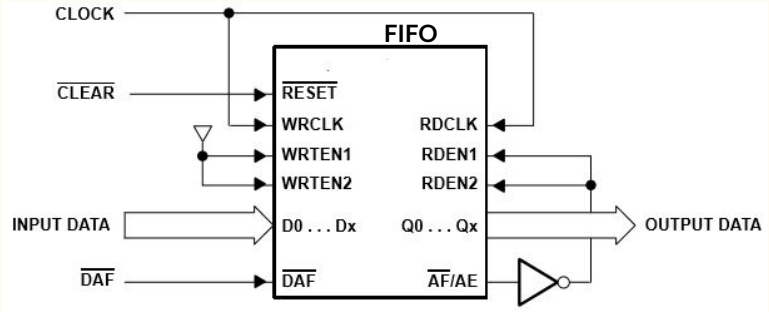
Block Transfer of Data

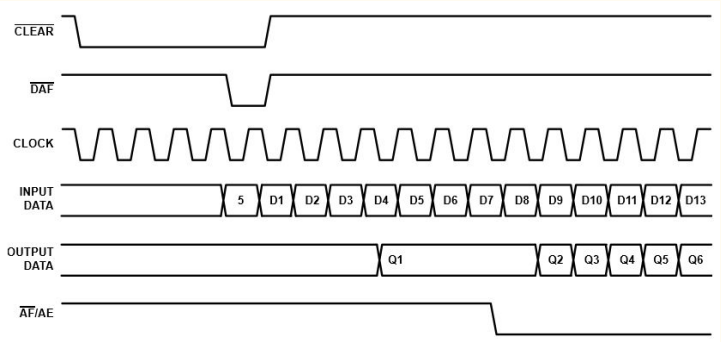
- Data are often split into blocks and transmitted on data lines. Computer networks and digital telephone-switching installations are examples of this application.
- A simple solution to this hardware problem is the use of FIFOs. A FIFO with a HALF FULL flag should be selected.
- Furthermore, memory depth should correspond to twice the size of a block. As soon as the HALF FULL flag is set, the send controller starts sending the data block. The send controller consists, in this case, of a counter and some gates and is very easy to implement with just one PAL.
- The writing of data into the FIFO can be carried on continuously and is independent of the transfer of data blocks.



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 FIFO, only one inverter in addition to the FIFO is necessary.
- A programmable delay line allows a user to programmatically specify and modify delay in a high frequency/RF signal path.
- This results in a signal delay, as measured in the time domain, or a phase shift, as measured in the frequency domain.





References

- https://www.ti.com/lit/an/scaa042a/scaa042a.pdf
- https://www.xilinx.com/support.html
- https://verilogguide.readthedocs.io/en/latest/
- Verilog HDL Basics by Intel FPGA: https://www.youtube.com/watch?v=PJGvZSlsLKs
- https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)

Acknowledgement

I would like to express my sincere thanks to Professor Garima Solanki, Department of Electrical Engineering, DTU.

This project was made possible by her guidance during theory classes, using which I was able to expose myself to new opportunities in this subject.