NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY – FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT 1

SUBJECT: LOGIC DESIGN WITH HDL

TEACHER: NGUYỄN THẾ BÌNH

Topics: FIFO – FIRST IN FIRST OUT

Name:	Student's ID
Phạm Anh Tài	2350023
Phạm Ngọc Huy	2352404
Trần Đình Duy Khương	2352638
Lục Tấn Phúc	2352935

Contents

List of Illustrations

Introduction

FIFO principle 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. The backup at traffic lights is caused by the sporadic arrival of the cars, the traffic lights allowing them to pass through only in batches.

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.

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. A data rate of 3.6 gigabits per second is specified for a Texas Instruments (TI□) SN74ABT7819 FIFO.

This report takes a detailed look at FIFO devices. The first part presents the different functions of FIFOs and the resulting types that are found. The second part deals with current FIFO architectures and the different ways in which they work. Finally, some application examples are given to illustrate the use of FIFOs available from TI.

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

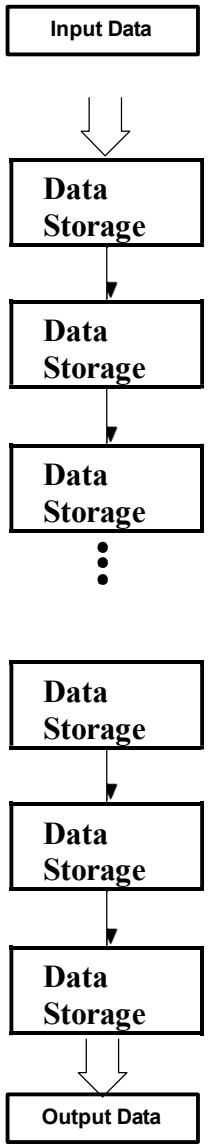


Figure 1. First-In First-Out Data Flow

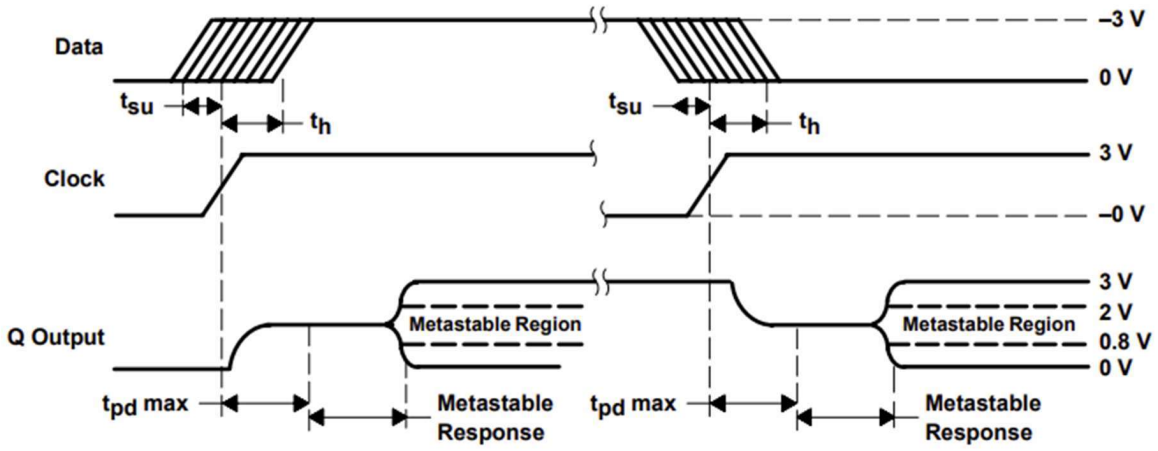


Figure 3. Timing Diagram for the Metastable State

Operating conditions for flip-flops can be maintained easily in synchronous circuits but with asynchronous and in synchronizing circuits, violations of the operating conditions for D flip-flops are unavoidable. Concurrent read/write FIFOs that are driven by two systems working asynchronously to one another must perform internal synchronization of the asynchronous external signals.

For physical reasons, there are no ideal flip-flops with a setup and hold time of zero, so there can be no synchronizing circuit that works faultlessly. The quality of a synchronizing circuit is indicated by its mean time between failures (MTBF).

MTBF can be improved by multilevel synchronization. Figure 4 illustrates a two-level synchronizing circuit, and the timing of the signals is shown in Figure 5. The second flip-flop can only go into a metastable state if the first flip-flop is already metastable.

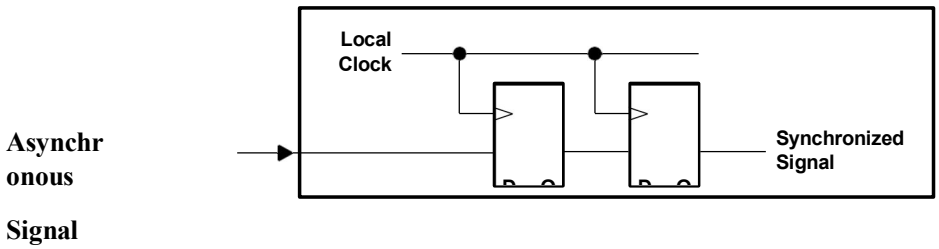


Figure 4. Block Diagram of Two-Level Synchronization

FIFO Architectures

All the kinds of FIFOs described in *FIFO Types* can be implemented in different hardware architectures. Initially, FIFOs worked by the fall-through principle. Today, FIFOs are nearly always based on an SRAM, which produced a considerable increase in the number of data words stored, despite the faster speed.

Fall-Through FIFOs

Fall-through FIFOs were the first FIFO generation. The customers queuing at the checkout point of a supermarket could be an example for this variant. The first customer goes right up to the checkout point, while all others queue behind. Once the first customer has paid and left the front of the queue, the other customers all move up one place.

There must be a status flip-flop for each data word; the effort involved in controlling the data latches is justifiable only for very short FIFOs. With long FIFOs, there is also an enormous increase in fall-through time. For the 16 x 5 fall-through FIFO SN74S225, a maximum delay of 400 ns is specified from the READ CLOCK to the FULL signal. New developments no longer use this principle.

FIFOs With Static Memory (Ring Buffer Principle)

To counter the disadvantage of a long fall-through time in long FIFOs, the architecture should no longer shift the data words through all memory locations. The problem is solved by a circular memory with two pointers.

In a circular FIFO concept, the memory address of the incoming data is in the write pointer. The address of the first data word in the FIFO that is to be read out is in the read pointer. After reset, both pointers indicate the same memory location. After each write operation, the write pointer is set to the next memory location. The reading of a data word sets the read pointer to the next data word that is to be read out. The read pointer constantly follows the write pointer. When the read pointer reaches the write pointer,

the FIFO is empty. If the write pointer catches up with the read pointer, the FIFO is full. Figure 15 illustrates the principle of a circular FIFO with two pointers.

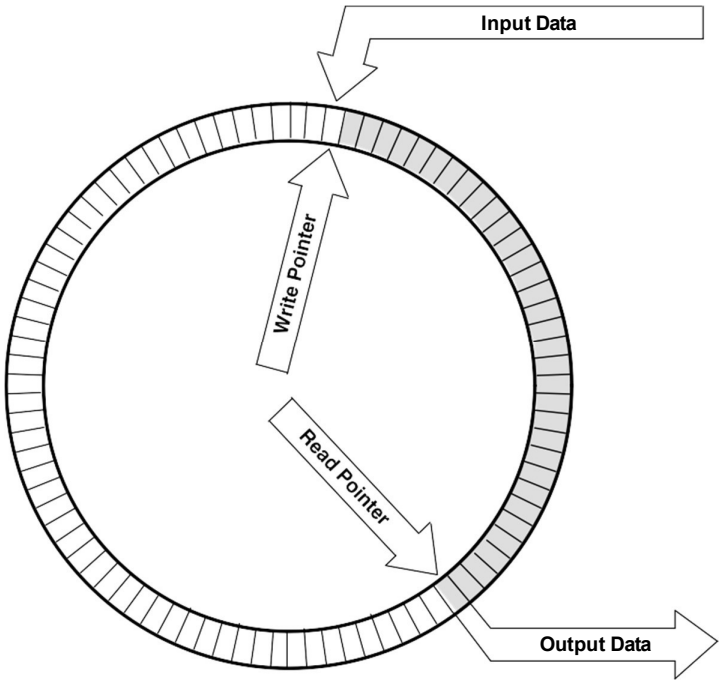


Figure 15. Circular FIFO With Two Pointers

Architecture

In the implementation of a circular memory, a dual-port SRAM is used for data storage. The pointers take the form of binary counters, which generate the memory addresses of the SRAM. It is an advantage if the number of memory locations is 2n because a pointer can be implemented as an n-bit binary counter whose carry can be ignored.

Read addresses are generated by the READ POINTER and write addresses by the WRITE POINTER. The write-control and read-control blocks control operation during write and read access. The FULL and EMPTY status signals are generated by separate flag logic. Some FIFOs also offer the status signals HALF FULL, ALMOST FULL, and ALMOST EMPTY. A FIFO with static memory also has to be initialized before it is used to set the two pointers and the status logic to defined initial states.

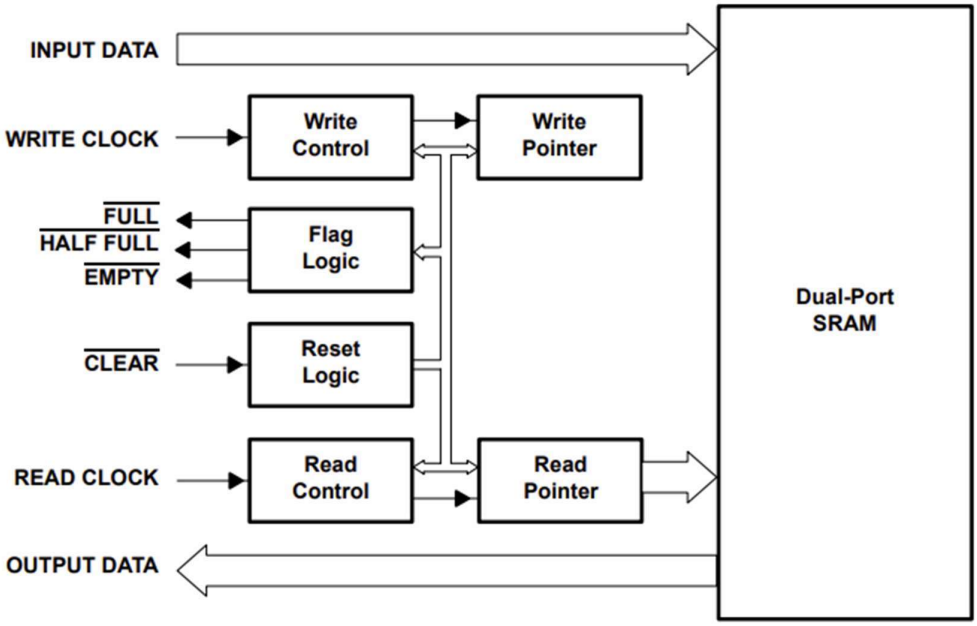
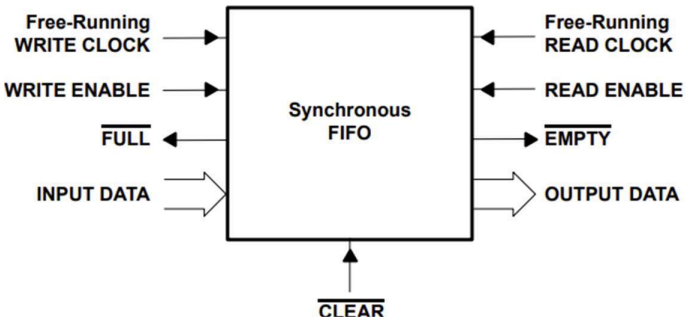


Figure 16. Block Diagram of FIFO With Static Memory

Synchronous FIFOs

Synchronous FIFOs are controlled based on methods of control proven in processor systems. Every digital processor system works synchronized with a system-wide clock signal. This system timing continues to run even if no actions are being executed.

Figure 11 shows all the signal lines of a synchronous FIFO. It requires a free-running clock from the writing system and another from the reading system. Writing is controlled by the WRITE ENABLE input synchronous with WRITE CLOCK. The FULL status line can be synchronized entirely with WRITE CLOCK by the free-running clock. In an analogous manner, data words are read out by a low level on the READ ENABLE input synchronous with READ CLOCK. Here, too, the free-running clock permits 100 percent synchronization of the EMPTY signal with READ CLOCK.



Figu **Figure 9. Connections of Synchronous FIFO**

Thus, synchronous FIFOs are integrated easily into common processor architectures, offering complete synchronism of the FULL and EMPTY status signals with the particular free-running clock.

Figure 12 shows the typical waveform in a synchronous FIFO. The writing of new data into the FIFO is initialized by a low level on the WRITE ENABLE line. The data are written into the FIFO with the next rising edge of WRITE CLOCK. In analogous fashion, the READ ENABLE line controls the reading out of data synchronous with READ CLOCK.

The FULL line only changes its level synchronously with WRITE CLOCK. Likewise, the EMPTY signal is synchronized with READ CLOCK. A synchronous FIFO is the only concurrent read/write FIFO in which the status signals are synchronized with the driving logic.

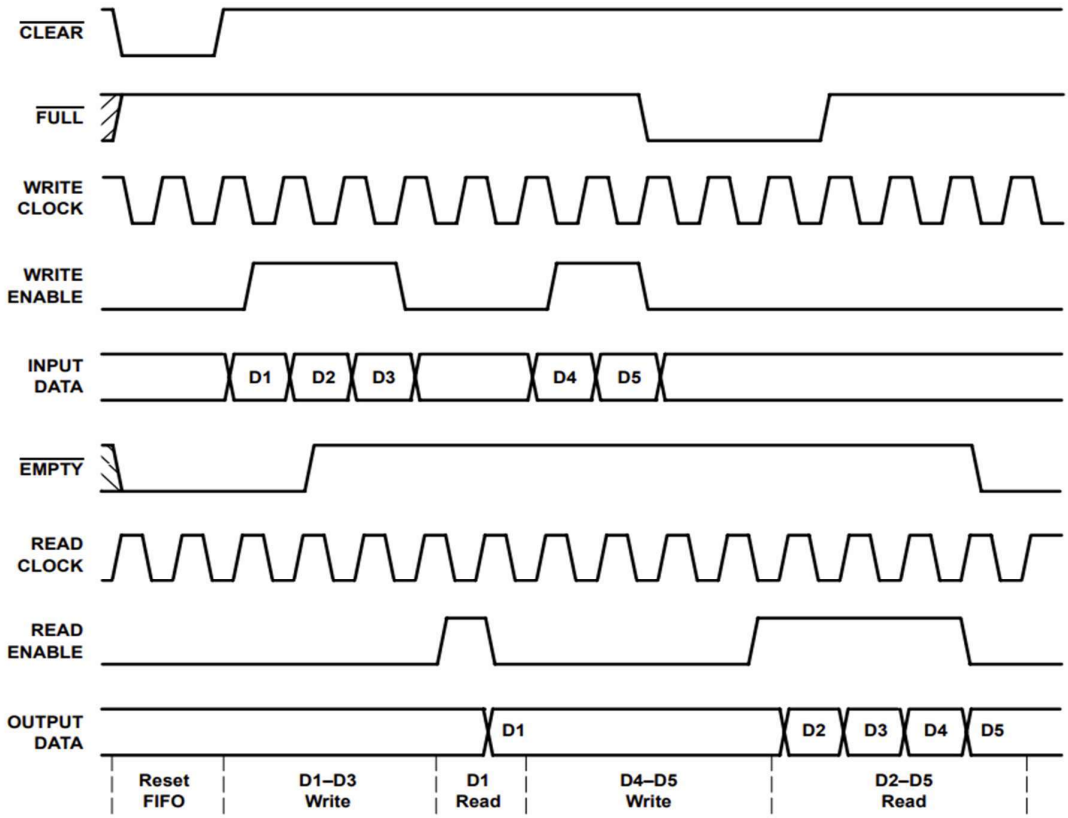


Figure 12. Timing Diagram for a Synchronous FIFO of Length 4

Advantages and Drawbacks

Unlike a fall-through FIFO, the fall-through time of a FIFO with static memory is independent of its length. Therefore, it is faster than with a length of several thousand words. The effort that goes into the circuitry of the control logic does not increase to any significant degree with length because only the two pointers and parts of the flag logic have to be expanded. Today's new developments use only the architecture of the FIFO with static memory.

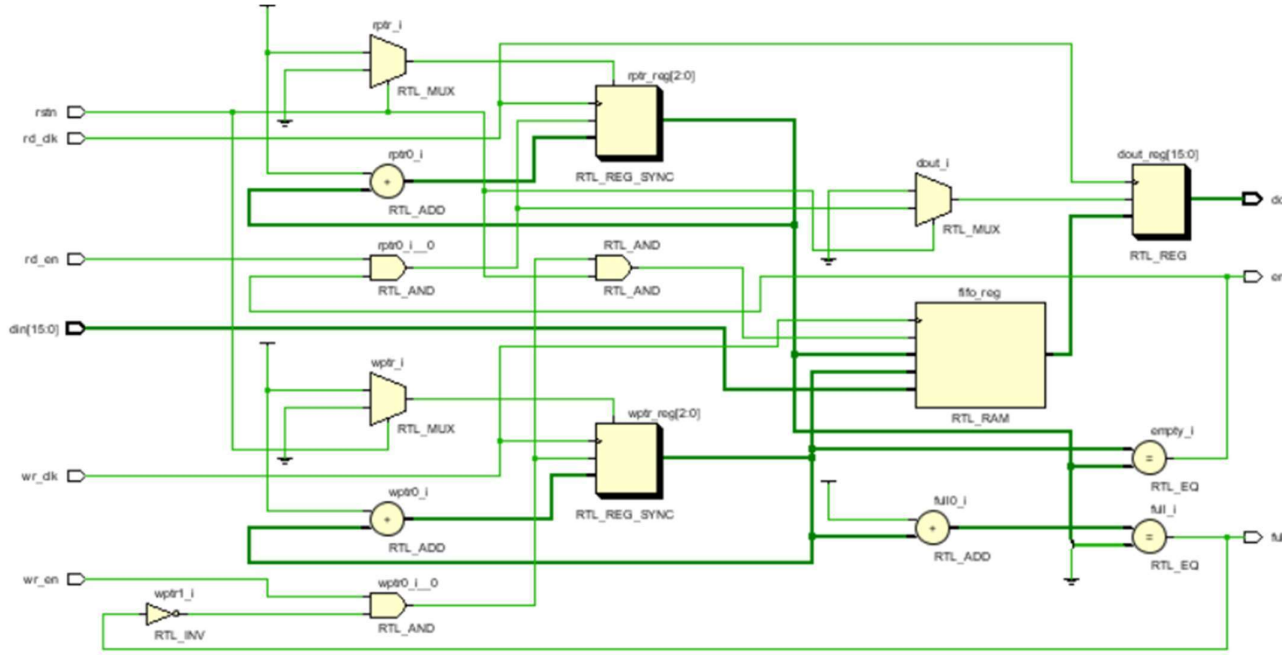
Verilog Code Implementation

```
module Synchro_FIFO #(parameter DEPTH=8, DWIDTH=16)(
input rstn,
      wr_clk,
      rd_clk,
      wr_en,
      rd_en,
  input      [DWIDTH - 1:0] din,
  output reg [DWIDTH - 1:0] dout,
  output            empty, full
  );

  reg [$clog2(DEPTH)-1:0]  wptr;
  reg [$clog2(DEPTH)-1:0]  rptr;
  reg [DWIDTH:0]    fifo [0:DEPTH - 1];

always @(posedge wr_clk) begin
    if (!rstn) begin
    wptr <= 0;
    end
    else begin
    if(wr_en & !full) begin
    fifo[wptr] <= din;
    wptr <= wptr + 1;
            end
        end
    end

  always @(posedge rd_clk) begin
  if(!rstn) begin
  rptr <= 0;
        end
   else begin
      if(rd_en & !empty) begin
                dout <= fifo[rptr];
                rptr <= rptr + 1 ;
              end
        end
    end

    assign full = (wptr+1) == rptr;
    assign empty = wptr == rptr;

endmodule
```

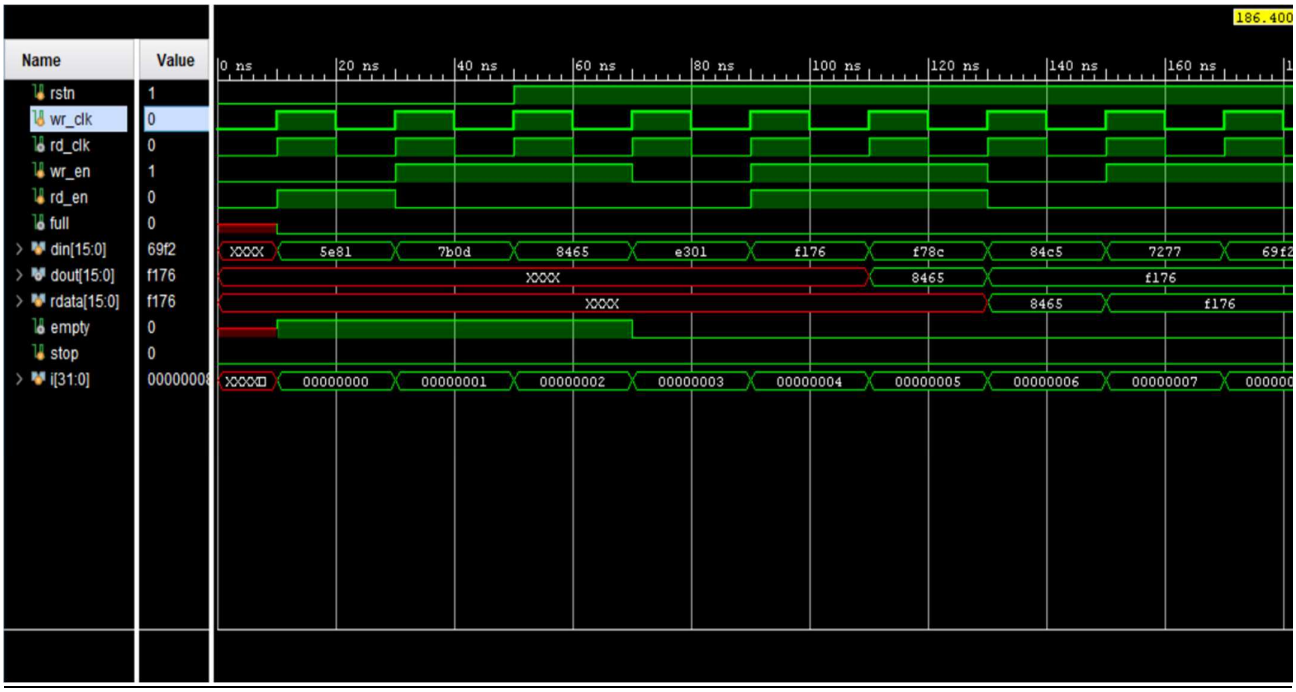
Testbench:

```
module fifo_tb;
reg rstn, wr_clk;
wire rd_clk;
reg wr_en, rd_en;
wire full;
reg[15:0] din;
wire [15:0] dout;
reg[15:0] rdata;
wire empty;
reg stop;
integer i;
Synchro_FIFO uut (.rstn(rstn), .wr_en(wr_en),
 .rd_en(rd_en), .wr_clk(wr_clk),
 .rd_clk(rd_clk), .din(din), .dout(dout),
 .empty(empty) , .full(full)
    );
    always #10 wr_clk = ~wr_clk;
    assign rd_clk = wr_clk;

    initial begin
    wr_clk <= 0;
    rstn <= 0;
    wr_en <= 0;
    rd_en <= 0;
    stop <= 0;
    #50 rstn <= 1;

  end

  initial begin
  @(posedge wr_clk);
  for( i = 0 ; i< 50; i=i+1) begin
  while (full) begin
  @(posedge wr_clk);
  $display("[%0t] FIFO is full, wait for reads to happen", $time);
         end

  wr_en <= $random;
  din <= $random;

  $display("[%0t] wr_clk i=%0d wr_en=%0d din=0x%0h ", $time, i ,wr_en, din);

  @(posedge wr_clk);
     end
  stop = 1;
  end

  initial begin
  @(posedge rd_clk);

  while(!stop) begin
      while(empty) begin
          rd_en <= 0;
          $display ("[%0t] FIFO is empty, wait for writes happen" , $time);
     @(posedge rd_clk);
  end;

  rd_en <= $random;
  @(posedge rd_clk);
  rdata <= dout;
  $display ("[%0t] rd_clk rd_en=%0d rdata=0x%0h ", $time, rd_en, rdata);
  end

  #500 $finish;
 end
endmodule
```

RTL Analysis:



Simulation:



Application Examples

In this section, a number of application examples illustrate the versatility and performance of TI FIFOs.

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. A simple solution to this hardware problem is the use of FIFOs (see Figure 30). A FIFO with a HALF FULL flag should be selected

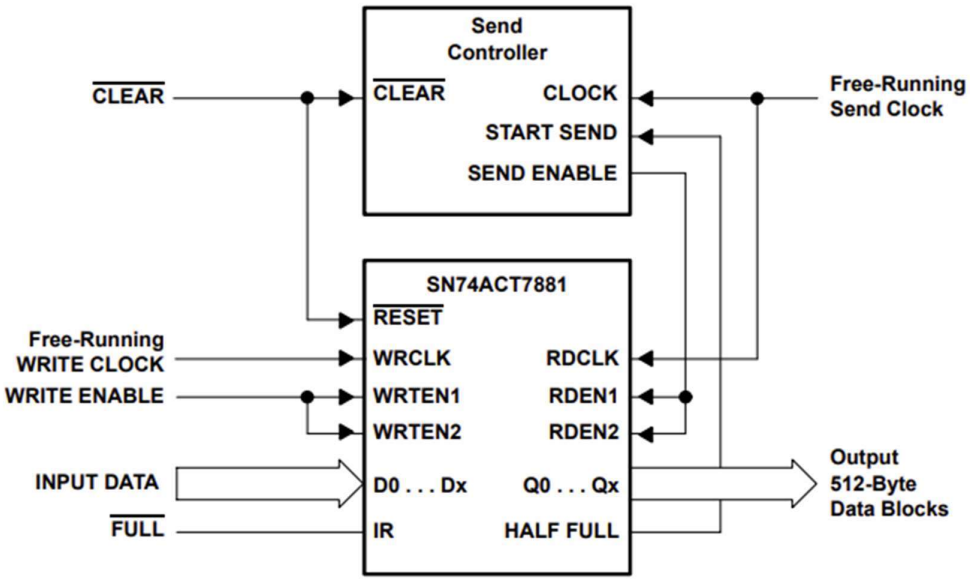


Figure 30. Block Transfer of Data With Synchronous SN74ACT7881 FIFO

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.

References

[1] *FIFO architecture, functions and applications*. Available at: https://www.ti.com/lit/an/scaa042a/scaa042a.pdf

[2] Wada, K. (2022) *Ring buffer basics*, *Embedded.com*. Available at: https://www.embedded.com/ring-buffer-basics/.