ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

BTL Thiết kế luận lý với HDL (TN) – CO1026

GVHD:Phạm Công Thái

FIFO – First In First Out

Nhóm 7	L01_23202
Lê Đình Anh Tài	2312994
Phạm Công Võ	2313946
Lê Thanh Phong	2312618
Trần Minh Dương	2310609
Bùi Tiến An	2310001

MỤC LỤC

I/ CƠ SỞ LÝ THUYẾT

1. Giới thiệu về FIFO:

FIFO (First In First Out) là một mạch logic cơ bản được sử dụng rộng rãi trong các hệ thống kỹ thuật số. Nó hoạt động như một bộ nhớ đệm, đảm bảo rằng dữ liệu được ghi vào nó trước sẽ được đọc ra trước. Nguyên tắc này giống như một hàng đợi, trong đó mục đầu tiên vào là mục đầu tiên rời đi. Trong Verilog, FIFO thường được sử dụng trong các thiết kế mạch số để quản lý các luồng dữ liệu giữa các phần tử của hệ thống.

2. Lý thuyết FIFO:

2.1 *Cấu trúc*:

Mạch FIFO thường được xây dựng từ các thành phần chính sau:

- Bộ nhớ (memory): Để lưu trữ dữ liệu. Đây có thể là RAM (Random Access Memory) hoặc các loại bộ nhớ khác.
- Con trỏ đầu (read pointer): Chỉ vào vị trí trong bộ nhớ nơi dữ liệu sẽ được đọc ra.
- Con trỏ cuối (Write pointer): Chỉ vào vị trí trong bộ nhớ nơi dữ liệu sẽ được ghi vào.
- Mạch điều khiển (logic control): Quản lý và điều phối việc đọc và ghi dữ liệu, đồng thời đảm bảo rằng các con trỏ hoạt động đúng cách và không ghi đè hoặc đọc dữ liệu khi bộ nhớ trống.

2.2 *Nguyên lý hoạt động*:

Việc triển khai FIFO thường dựa trên nguyên tắc bộ đệm vòng. Trong bộ đệm vòng, dữ liệu được lưu trữ trong bộ đệm có kích thước cố định và khi bộ đệm đầy, dữ liệu mới sẽ ghi đè lên dữ liệu cũ nhất. Điều này tạo ra một vòng lưu trữ dữ liệu liên tục, lý tưởng cho các ứng dụng cần luồng dữ liệu liên tục.

2.3 *Các loại FIFO*:

Mạch FIFO (First-In, First-Out) có nhiều loại khác nhau, được phân loại dựa trên cách chúng được triển khai và sử dụng trong các ứng dụng cụ thể. Dưới đây là các loại mạch FIFO phổ biến:

1. Synchronous FIFO

Synchronous FIFO hoạt động dựa trên một xung nhịp (clock) duy nhất. Cả hoạt động đọc và ghi đều được đồng bộ với xung nhịp này. Điều này làm cho việc thiết kế và sử dụng FIFO đơn giản hơn, vì tất cả các hoạt động đều dựa trên cùng một xung nhịp.

Đặc điểm:

- Đơn giản trong thiết kế vì chỉ cần một xung nhịp.
- Thích hợp cho các ứng dụng mà tốc độ xung nhịp không quá cao.

2. Asynchronous FIFO

Asynchronous FIFO sử dụng hai xung nhịp riêng biệt cho hoạt động đọc và ghi. Điều này cho phép dữ liệu được ghi vào FIFO từ một xung nhịp và đọc ra từ một xung nhịp khác, làm cho loại FIFO này phù hợp cho các hệ thống có các phần khác nhau hoạt động ở các tốc độ xung nhịp khác nhau.

Đặc điểm:

- Phức tạp hơn trong thiết kế do cần phải đồng bộ hóa giữa hai xung nhịp.
- Thích hợp cho các hệ thống có nhiều miền xung nhịp (multi-clock domain).

3. Circular Buffer FIFO

Circular Buffer FIFO sử dụng một bộ nhớ vòng (circular buffer) để lưu trữ dữ liệu. Khi con trỏ đọc hoặc con trỏ ghi đạt đến cuối bộ nhớ, chúng sẽ quay trở lại đầu bộ nhớ.

Đặc điểm:

- Hiệu quả trong việc sử dụng bộ nhớ.
- Phù hợp cho các ứng dụng cần quản lý luồng dữ liệu liên tục.

4. Shift Register FIFO

Shift Register FIFO sử dụng các thanh ghi dịch (shift register) để quản lý dữ liệu. Dữ liệu được dịch chuyển từng bước từ đầu vào đến đầu ra khi có yêu cầu đọc và ghi.

Đặc điểm:

- Thiết kế đơn giản và dễ hiểu.
- Thường được sử dụng cho các ứng dụng tốc độ cao nhưng dung lượng lưu trữ thấp.

5. Blocking FIFO

Blocking FIFO ngừng hoạt động ghi khi đầy và ngừng hoạt động đọc khi trống. Điều này đảm bảo rằng dữ liệu không bị ghi đè hoặc đọc nhầm.

Đặc điểm:

- Đảm bảo tính toàn vẹn của dữ liệu.
- Có thể gây ra độ trễ trong hệ thống khi các hoạt động bị chặn.

6. Non-Blocking FIFO

Non-Blocking FIFO tiếp tục hoạt động ghi ngay cả khi đầy, có thể ghi đè lên dữ liệu cũ, và tiếp tục đọc ngay cả khi trống, có thể đọc lại dữ liệu cũ.

Đặc điểm:

- Không bị chặn nhưng có nguy cơ mất dữ liệu.
- Thích hợp cho các ứng dụng không yêu cầu tính toàn vẹn dữ liệu cao.

7. Depth FIFO

Depth FIFO có độ sâu cố định và các con trỏ đọc/ghi được điều chỉnh theo độ sâu này.

Đặc điểm:

- Cung cấp dung lượng lưu trữ cố định.
- Dễ dàng quản lý và triển khai.

8. Width FIFO

Width FIFO hỗ trợ các độ rộng dữ liệu khác nhau. Điều này cho phép FIFO lưu trữ các từ dữ liệu có kích thước lớn hơn một byte.

Đặc điểm:

- Linh hoạt trong việc lưu trữ các từ dữ liệu lớn.
- Thích hợp cho các ứng dụng yêu cầu truyền tải dữ liệu rộng.

**Ứng dụng của các loại FIFO*:

- Synchronous FIFO: Sử dụng trong các bộ đệm dữ liệu nội bộ, nơi tất cả các phần của hệ thống hoạt động ở cùng một tốc độ.
- Asynchronous FIFO:Sử dụng trong các hệ thống truyền thông, nơi dữ liệu cần được truyền giữa các hệ thống hoạt động ở các tốc độ khác nhau.
- Circular Buffer FIFO: Sử dụng trong xử lý tín hiệu số, nơi các mẫu dữ liệu cần được quản lý liên tục.
- Shift Register FIFO: Sử dụng trong các ứng dụng tốc độ cao như các bộ truyền dữ liệu tốc độ cao.
- Blocking và Non-Blocking FIFO: Sử dụng trong các ứng dụng yêu cầu quản lý luồng dữ liệu khác nhau, tùy thuộc vào yêu cầu tính toàn vẹn dữ liệu.

Mỗi loại mạch FIFO có ưu và nhược điểm riêng, và lựa chọn loại phù hợp phụ thuộc vào yêu cầu cụ thể của ứng dụng.

2.4 *Nguyên tắc vòng đệm (ring buffer)*:

- Logic triển khai FIFO thường dựa trên nguyên tắc **bộ đệm vòng** .

- Bộ đệm vòng là cấu trúc dữ liệu hình tròn trong đó dữ liệu bao quanh từ đầu đến cuối.

- Trong FIFO, điều này có nghĩa là khi bộ đệm đầy, dữ liệu mới sẽ ghi đè lên dữ liệu cũ nhất.

=> Trong bộ đệm vòng, dữ liệu được lưu trữ trong bộ đệm có kích thước cố định và khi bộ đệm đầy, dữ liệu mới sẽ ghi đè lên dữ liệu cũ nhất. Điều này tạo ra một vòng lưu trữ dữ liệu liên tục, lý tưởng cho các ứng dụng cần luồng dữ liệu liên tục.

3. Ứng dụng:

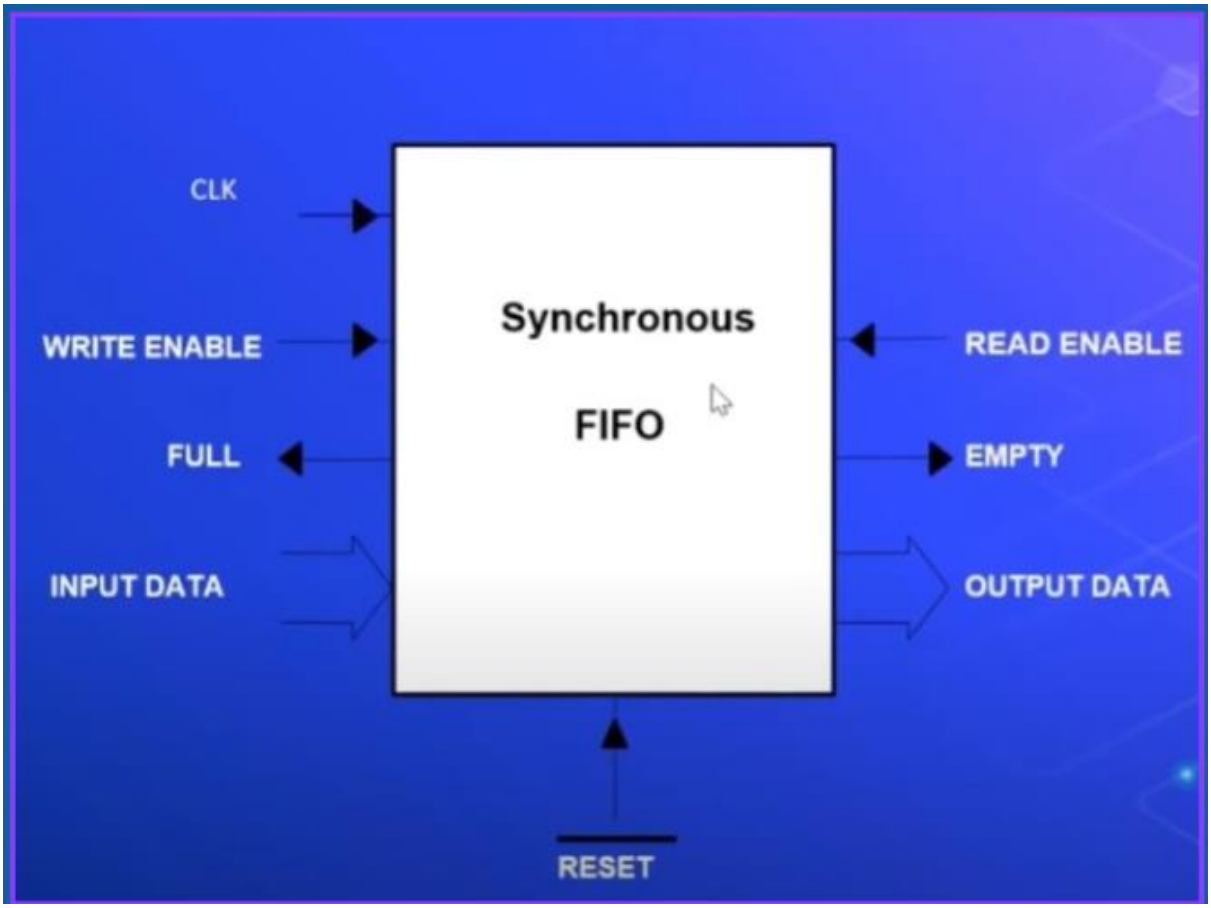
- FIFO tìm thấy các ứng dụng trong các hệ thống kỹ thuật số khác nhau, bao gồm giao diện truyền thông, bộ xử lý dữ liệu và hệ thống quản lý bộ nhớ. Chúng đặc biệt hữu ích trong các tình huống mà dữ liệu cần được lưu trữ tạm thời trước khi được xử lý hoặc truyền đi.

- Một số ứng dụng phổ biến của FIFO:

1. *Giao Tiếp Giữa Các Bộ Phận Có Tốc Độ Khác Nhau:*
    - Giao Tiếp Bộ Nhớ và CPU: FIFO giúp điều tiết luồng dữ liệu để tránh nghẽn cổ chai.
    - Truyền Thông Liên Kết Nối (Communication Links): FIFO đệm dữ liệu vào/ra giữa các thiết bị.
2. *Đồng Bộ Hóa Dữ Liệu Trong Hệ Thống:*
    - Chuyển Đổi Miền Xung Nhịp (Clock Domain Crossing): FIFO đảm bảo dữ liệu truyền chính xác và đồng bộ giữa các miền xung nhịp.
3. *Quản Lý Buffer trong Hệ Thống Mạng và Truyền Thông:*
    - Router và Switches: FIFO lưu trữ tạm thời các gói tin trước khi chuyển tiếp.
    - Bộ Đệm Dữ Liệu (Buffer): FIFO quản lý luồng dữ liệu đến và đi từ các thiết bị mạng.
4. *Hệ Thống Xử Lý Đa Nhiệm (Multitasking):*
    - Hệ Điều Hành Thời Gian Thực (RTOS): FIFO quản lý hàng đợi tác vụ, giúp điều phối các tác vụ.
5. *Ứng Dụng Trong Trò Chơi và Đồ Họa:*
    - Xử Lý Đồ Họa: FIFO lưu trữ và quản lý các lệnh đồ họa trước khi chúng được xử lý bởi GPU.
6. *Hệ Thống Âm Thanh và Video:*
    - Buffering: FIFO đệm dữ liệu âm thanh và video để đảm bảo phát lại liên tục.
7. *Thiết Kế Bộ Điều Khiển:*
    - DMA (Direct Memory Access): FIFO quản lý dữ liệu khi chuyển từ bộ nhớ đến các thiết bị ngoại vi mà không cần CPU can thiệp.
8. *Hệ Thống Ghi Nhật Ký và Giám Sát:*
    - Data Logging: FIFO ghi lại dữ liệu từ các cảm biến hoặc nguồn dữ liệu liên tục để phân tích sau này.

II/ THIẾT KẾ MẠCH FIFO

1. Sơ đồ khối FIFO:



- FIFO (First In First Out): Đây là tên của cấu trúc dữ liệu FIFO, nó là bộ nhớ đệm dữ liệu và cho phép dữ liệu được ghi vào và đọc ra theo thứ tự "vào trước, ra trước". Hình ảnh chỉ rõ rằng FIFO này là đồng bộ (Synchronous).
- CLK (Clock): Đây là xung nhịp đồng bộ, tất cả các hoạt động của FIFO sẽ được đồng bộ hóa với xung nhịp này.
- WRITE ENABLE: Đây là tín hiệu cho phép ghi dữ liệu vào FIFO. Khi tín hiệu này ở mức cao (HIGH), FIFO sẽ cho phép ghi dữ liệu vào.
- INPUT DATA: Đây là tín hiệu dữ liệu đầu vào, nó mang dữ liệu cần ghi vào FIFO.
- FULL: Đây là tín hiệu báo FIFO đã đầy. Khi FIFO đầy, tín hiệu này sẽ ở mức cao (HIGH), và FIFO sẽ không cho phép ghi dữ liệu vào nữa.
- READ ENABLE: Đây là tín hiệu cho phép đọc dữ liệu từ FIFO. Khi tín hiệu này ở mức cao (HIGH), FIFO sẽ cho phép đọc dữ liệu.

- OUTPUT DATA: Đây là tín hiệu dữ liệu đầu ra, nó mang dữ liệu được đọc từ FIFO.
- EMPTY: Đây là tín hiệu báo FIFO đã trống. Khi FIFO trống, tín hiệu này sẽ ở mức cao (HIGH), và FIFO sẽ không cho phép đọc dữ liệu.
- RESET: Đây là tín hiệu reset, nó sẽ reset FIFO về trạng thái ban đầu (trống) và các con trỏ ghi, con trỏ đọc sẽ được đặt về vị trí ban đầu.

2. Code Verilog:

```
`timescale 1ns / 1ps
module FIFO_synn (
    input clk, rst_n,
    input wr_en_i,
    input [7:0] data_i,
    input rd_en_i,
    output reg [7:0] data_o,
    output empty, full
);

parameter depth = 8;

reg [7:0] mem [0:depth-1];
reg [2:0] wr;
reg [2:0] rd_ptr;
reg [3:0] count;

assign full = (count == depth);
assign empty = (count == 0);
//WRITE
always @(posedge clk, negedge rst_n)
begin
    if (!rst_n)
        wr <= 3'd0;
    else if (wr_en_i == 1)
    begin
        mem[wr] <= data_i;
        wr <= wr + 1;
    end
end

// READ
always @(posedge clk, negedge rst_n)
begin
    if (!rst_n)
        rd_ptr <= 3'd0;
    else if (rd_en_i == 1)
    begin
        data_o <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end
// COUNT
always @(posedge clk, negedge rst_n)
begin
    if (!rst_n)
        count <= 4'd0;
    else
    begin
        case ({wr_en_i, rd_en_i})
            2'b10: count <= count + 1;
            2'b01: count <= count - 1;
            2'b11: count <= count;
            2'b00: count <= count;
        endcase
    end
end
```

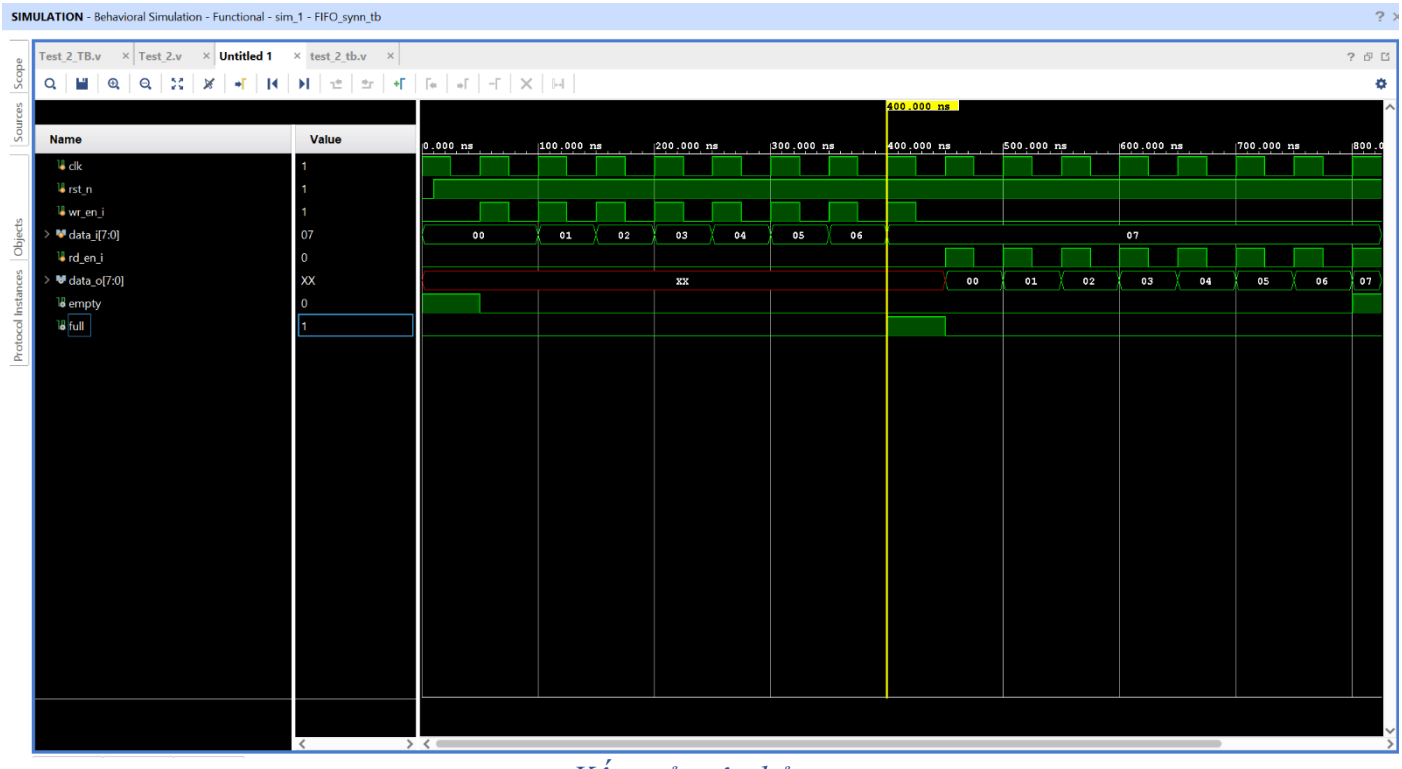
```
endmodule
```

* ***Test bench:***

```
`timescale 1ns / 1ps

module FIFO_synn_tb;

reg clk;
reg rst_n;
reg wr_en_i;
reg[7:0] data_i;
reg rd_en_i;
wire[7:0] data_o;
wire empty;
wire full;

FIFO_synn dut(
    .clk(clk),
    .rst_n(rst_n),
    .wr_en_i(wr_en_i),
    .data_i(data_i),
    .rd_en_i(rd_en_i),
    .data_o(data_o),
    .empty(empty),
    .full(full)
);

initial begin
clk = 1;
forever #25 clk = ~clk;
end

initial begin
// Initialize signals
rst_n = 0;
wr_en_i = 0;
rd_en_i = 0;
data_i = 0;
#10 rst_n = 1; // Reset FIFO trong 10ns đầu tiên

// Write 8 data elements
@(posedge clk) begin wr_en_i = 1; data_i = 0; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 1; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 2; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 3; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 4; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 5; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 6; end
#1 @(negedge clk) wr_en_i = 0;
@(posedge clk) begin wr_en_i = 1; data_i = 7; end
#1 @(negedge clk) wr_en_i = 0;

// Wait for a few cycles
#20; // Delay 20ns trước khi đọc

// Read 8 data elements
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
```

```
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;
@(posedge clk) rd_en_i = 1;
#1 @(negedge clk) rd_en_i = 0;

$finish;
end
endmodule
```

3. Mô phỏng:



Kết quả mô phỏng

***Các mô-đun bổ sung và kịch bản demo:**

Để tăng cường triển khai mạch FIFO, các mô-đun bổ sung có thể được phát triển để hỗ trợ các tính năng và chức năng khác nhau. Một số mô-đun tiềm năng bao gồm:

- Mô-đun nén/giải nén dữ liệu để lưu trữ dữ liệu hiệu quả.
- Các mô-đun phát hiện và sửa lỗi để đảm bảo tính toàn vẹn dữ liệu.
- Các mô-đun giao diện để kết nối mạch FIFO với các thiết bị hoặc hệ thống bên ngoài.

Các kịch bản demo có thể được lên kế hoạch để thể hiện khả năng của mạch FIFO, chẳng hạn như:

- Các ứng dụng truyền dữ liệu theo thời gian thực trong đó dữ liệu đến được xử lý và truyền đi không chậm trễ.

Khái niệm FIFO (First In First Out) được sử dụng rộng rãi trong điện toán và điện tử để quản lý cấu trúc dữ liệu một cách hiệu quả. Trong hệ thống FIFO, dữ liệu đầu vào đầu tiên là dữ liệu đầu tiên được xử lý hoặc xuất ra. Phương pháp này giống như phục vụ mọi người trong hàng đợi trên cơ sở ai đến trước được phục vụ trước. FIFO rất quan trọng trong các mạch điện tử để đệm và điều khiển luồng giữa các thành phần phần cứng và phần mềm.

Nó thường liên quan đến các con trỏ đọc và ghi, lưu trữ và logic điều khiển, thường sử dụng SRAM hoặc các dạng lưu trữ khác. Trong thiết bị điện tử, FIFO có thể đồng bộ hoặc không đồng bộ.

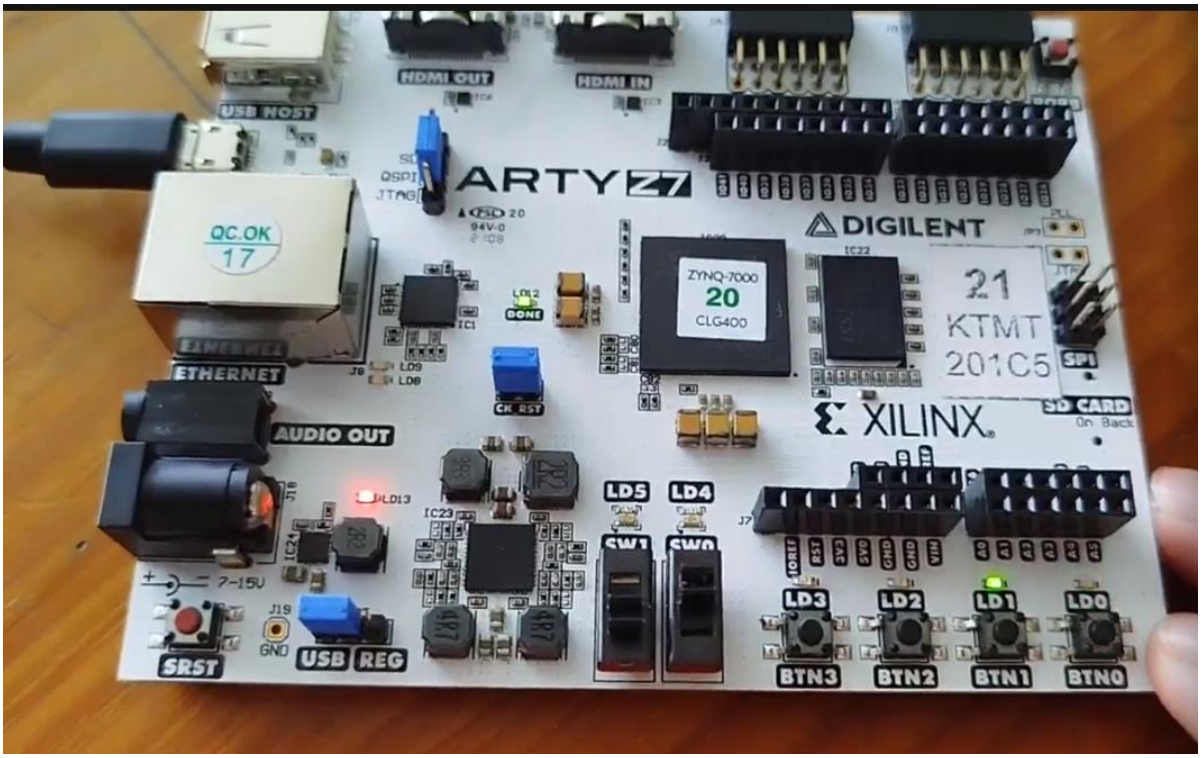
FIFO đồng bộ sử dụng cùng một đồng hồ để đọc và ghi, trong khi FIFO không đồng bộ sử dụng các đồng hồ khác nhau, có khả năng dẫn đến các vấn đề về tính di động.

Việc triển khai FIFO liên quan đến việc quản lý các biến trạng thái như đầy, trống, gần đầy và gần như trống để theo dõi trạng thái bộ đệm một cách chính xác.

Kiến trúc của FIFO bao gồm các con trỏ đọc và ghi, với con trỏ đọc tiến lên khi dữ liệu được đọc và con trỏ ghi di chuyển khi dữ liệu được ghi. FIFO có thể được triển khai dưới dạng hàng đợi tròn, đảm bảo xử lý dữ liệu hiệu quả.

Độ sâu FIFO xác định số phần tử tối đa có thể được lưu trữ trong bộ đệm.

4. Kết quả hiện thực



Kết quả hiện thực FIFO trên arty-z7-20

Link video hiện thực:

https://drive.google.com/file/d/1NpDApTZtWdUWgRfOH2vFpY9P33EAwmke/view

III/ KẾT LUẬN

FIFO là một công cụ mạnh mẽ và linh hoạt trong thiết kế hệ thống điện tử và vi mạch. Nó không chỉ giúp quản lý luồng dữ liệu hiệu quả mà còn đảm bảo tính đồng bộ và hiệu suất của hệ thống. Việc hiểu và áp dụng đúng FIFO trong các ứng dụng cụ thể là rất quan trọng để tối ưu hóa hiệu suất và độ tin cậy của hệ thống. Từ việc điều tiết dữ liệu giữa các thành phần có tốc độ khác nhau, đồng bộ hóa dữ liệu, quản lý buffer trong hệ thống mạng, đến các ứng dụng trong xử lý đa nhiệm và xử lý đồ họa, FIFO luôn là một phần không thể thiếu trong các thiết kế mạch số hiện đại.

IV/ THAM KHẢO

[1]. Thiết kế bộ nhớ FIFO dùng verilog.
https://vimach.net/threads/thiet-ke-bo-nho-dem-fifo-dung-verilog.124/
[2]. Verilog code for FIFO memory.
https://www.fpga4student.com/2017/01/verilog-code-for-fifo-memory.html