ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

-----o0o-----

BÀI TẬP LỚN

MÔN HỌC: THIẾT KẾ LUẬN LÝ

Đề tài: FIFO – FIRST IN FIRST OUT

Lớp: L05 -- Nhóm 3 – HK 222

Giảng viên hướng dẫn: Đoàn Thiên Ân

Thành viên thực hiện:

Họ và Tên	MSSV
Lê Xuân Hải	2210885
Đặng Quốc An	2210002
Trần Trương Trung Hiếu	2211020
Dương Văn Nghĩa	2013864

Thành phố Hồ Chí Minh, tháng 3 năm 2023

Mục Lục

I. Introduction

Trong thế giới phức tạp và đầy thông tin của ngày nay, việc tổ chức và xử lý dữ liệu một cách hiệu quả là một thách thức quan trọng. Trong quá trình này, nguyên tắc FIFO nổi lên như một phương pháp đáng tin cậy và phổ biến để xử lý các yêu cầu và dữ liệu theo thứ tự chúng được thêm vào. Hôm nay, chúng ta sẽ khám phá sâu hơn về nguyên tắc FIFO, các ứng dụng của nó và tầm quan trọng của nó trong lĩnh vực công nghệ và quản lý dữ liệu.

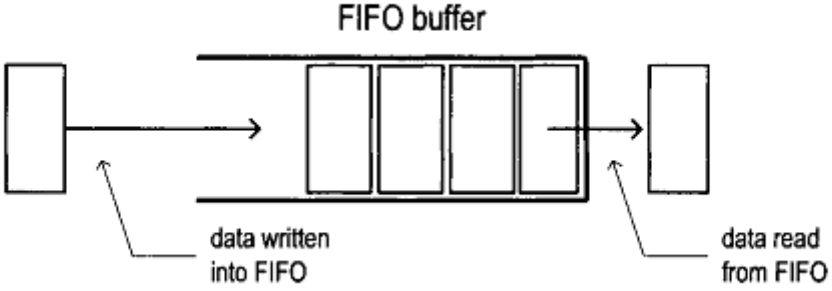
Nguyên tắc FIFO – "First in, First out" là một nguyên tắc cơ bản và quan trọng trong các hệ thống xếp hàng và quản lý dữ liệu. Ý tưởng cơ bản của FIFO là sự tuân thủ thứ tự đầu vào, đồng nghĩa với việc phần tử hoặc yêu cầu đến trước sẽ được xử lý hoặc truy cập trước.

Hãy tưởng tượng rằng chúng ta đang đứng trong một hàng chờ ở quầy thanh toán trong siêu thị. Những người đến trước sẽ được phục vụ trước. Đây chính là nguyên tắc FIFO hoạt động - đối tượng hay dữ liệu đến trước sẽ được xử lý hoặc truy cập trước.

II. Backgrounds and applications

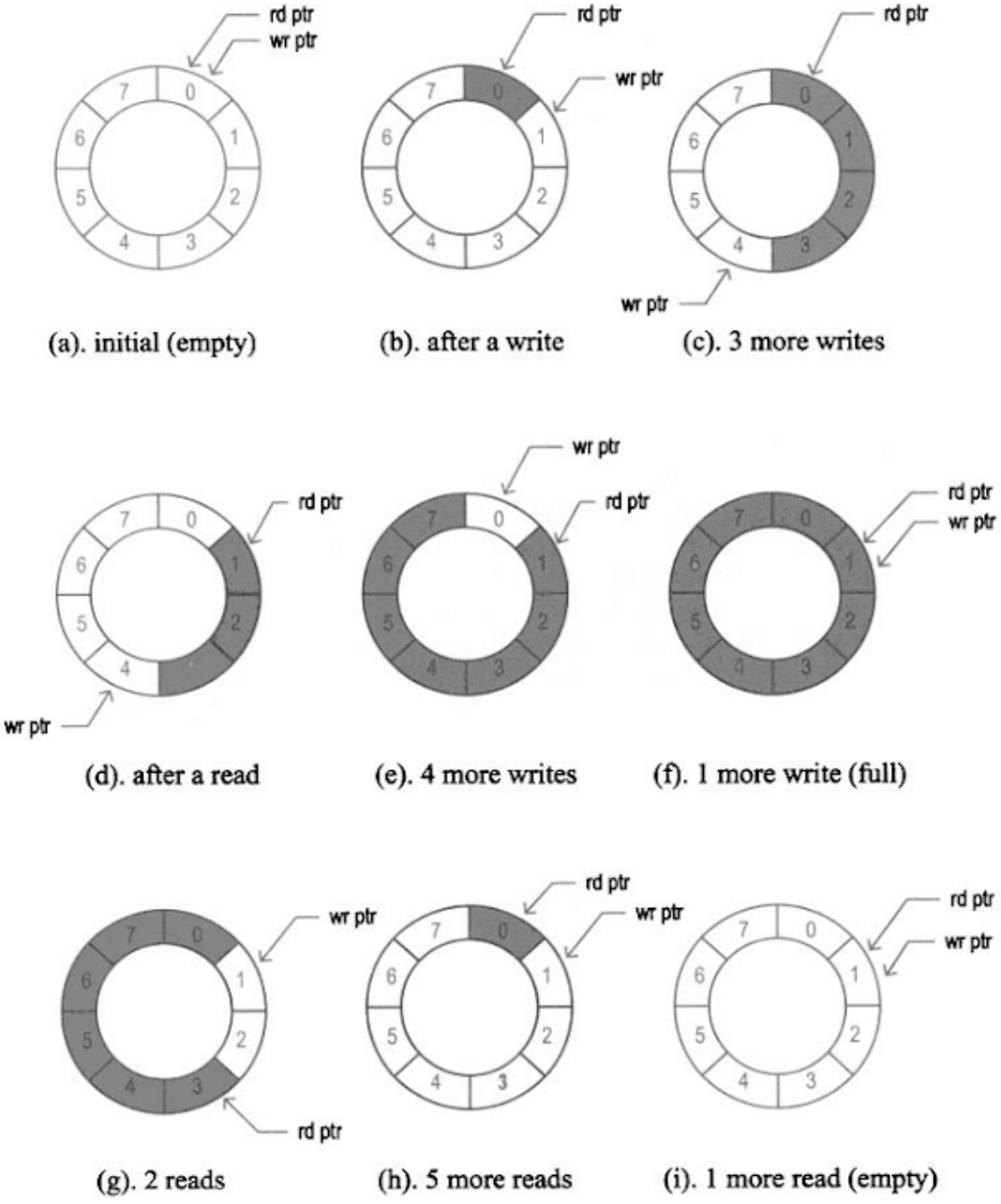
1. Giới thiệu

   FIFO (First in first out) như tên gọi đây là một nguyên tắc ám chỉ thứ tự đọc và ghi dữ liệu vào ra trong hàng đợi. Dữ liệu nào ghi trước thì được được đọc trước. Đối với FIFO không còn là khái niệm địa chỉ mà chỉ còn các cổng điều khiển đọc và ghi dữ liệu. Khi được phép ghi, dữ liệu bên ngoài sẽ ghi vào bộ nhớ đệm (buffer). Khi có tín hiệu đọc, dữ liệu được đọc từ bộ nhớ đệm (buffer) ra ngoài theo thứ tự đã ghi. Tùy theo những yêu cầu cụ thể của người dùng nó mà FIFO có thể được thiết kế bằng các cách khác nhau



2. Ý tưởng của FIFO

Để thiết kế được FIFO chúng ta hãy tưởng tượng các thành phần bộ nhớ trong vòng tròn với hai con trỏ write (ghi) và read (đọc). Đầu tiên ta có 2 con trỏ ghi và đọc ở đầu vòng tròn. Ta tăng lần lượt con trỏ đọc lên để ghi dữ liệu vào bộ nhớ đệm. Sau khi đã ghi được ô nhỏ trong vòng tròn ta bắt đầu tăng con trỏ đọc lên để đọc dữ liệu ra. Cứ như thế cho hết vòng tròn ta được dữ liệu ra y như giá trị dữ liệu ban đầu.



- Bộ nhớ đệm này cũng cần có 2 cờ là: empty và full.

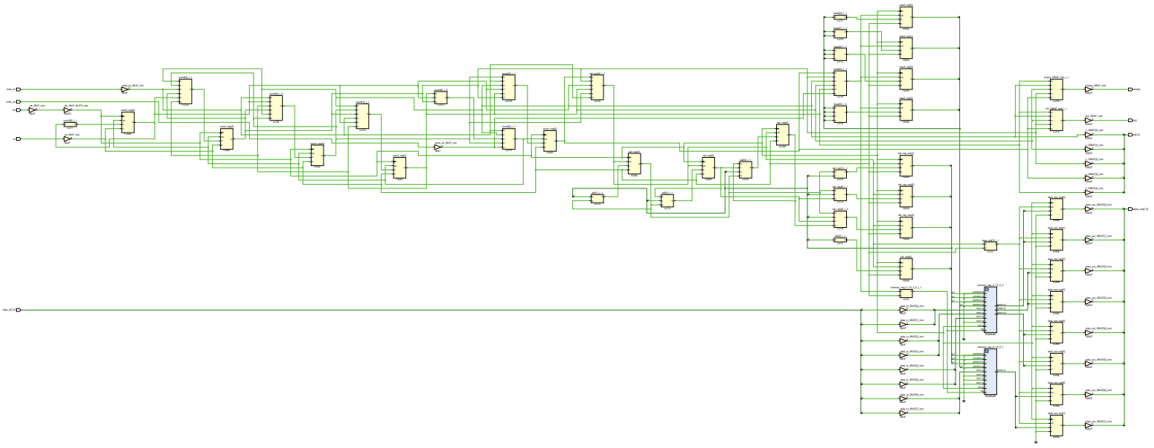
- Cờ full là trạng thái khi con trỏ ghi đã thực hiện ghi dữ liệu được một vòng tròn và gặp con trỏ đọc tại vòng tròn thứ 2. Nói cách khác, con trỏ đọc trùng với con trỏ ghi khi hay vòng quay con trỏ ghi lớn hơn con trỏ đọc 1 vòng. Dữ liệu chưa được đọc ra mà đã có tín hiệu ghi vào ô nhớ đó. Khi đó ta sẽ không được phép ghi dữ liệu vào nữa.
- Flags empty: là trạng thái con trỏ đọc trùng với con trỏ ghi khi cả 2 con trỏ cùng một vòng. Khi đó không có dữ liệu để đọc.

3. Ứng dụng của mạch FIFO

- Hệ thống xếp hàng: FIFO được sử dụng rộng rãi trong các hệ thống xếp hàng, như hàng chờ tại quầy thanh toán, hàng đợi gọi điện tử, hay hệ thống xếp hàng trong các công ty dịch vụ khách hàng. Nguyên tắc FIFO đảm bảo tính công bằng khi phục vụ khách hàng theo thứ tự đến trước.
- Quản lý dữ liệu: Trong quản lý dữ liệu, FIFO được sử dụng để theo dõi thứ tự các giao dịch hoặc sự kiện. Ví dụ, trong hệ thống ghi nhật ký (logging), các sự kiện được ghi lại theo thứ tự chúng xảy ra, và nguyên tắc FIFO đảm bảo rằng các sự kiện được truy xuất và xử lý theo thứ tự.
- Cấu trúc dữ liệu hàng đợi (queue): FIFO là nguyên tắc cơ bản trong cấu trúc dữ liệu hàng đợi. Hàng đợi được sử dụng để lưu trữ và xử lý dữ liệu theo thứ tự chúng được thêm vào. Ví dụ, trong các thuật toán tìm kiếm theo chiều rộng (BFS) trong đồ thị, hàng đợi FIFO được sử dụng để duyệt qua các đỉnh theo thứ tự theo cấp độ.
- Bộ đệm (buffer): FIFO cũng được sử dụng trong các bộ đệm (buffer) để quản lý dữ liệu trong các quá trình truyền thông, xử lý tín hiệu âm thanh hoặc video, hoặc trong việc tương tác với thiết bị ngoại vi. Dữ liệu mới nhất được ghi vào bộ đệm, trong khi dữ liệu cũ nhất được đọc hoặc xử lý đầu tiên.
- Quản lý tác vụ và tiến trình: Trong hệ điều hành, nguyên tắc FIFO được sử dụng để quản lý hàng đợi các tác vụ hoặc tiến trình. Các tác vụ được thực thi theo thứ tự chúng được thêm vào, đảm bảo tính công bằng và ưu tiên cho các tác vụ đã đến trước.
- Đối với ngành sản xuất: FIFO cũng được sử dụng trong các hoạt động sản xuất như quản lý hàng tồn kho và quản lý quá trình sản xuất. Nó đảm bảo rằng hàng hóa hoặc thành phẩm được xử lý theo thứ tự chúng được sản xuất hoặc nhập kho.

III. Design

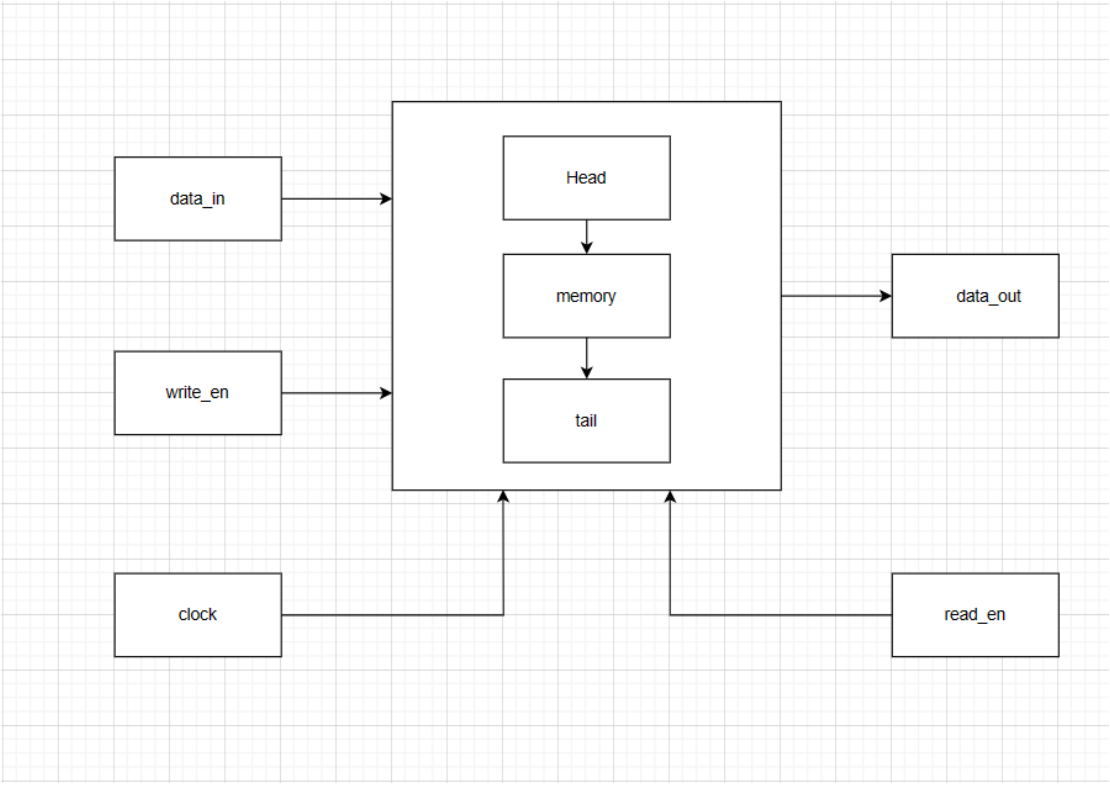
1. Schematic



(link ảnh: https://drive.google.com/file/d/13VRmsJkmjRBXUlWAsjc4nA4hNKlYsK4C/view?usp=sharing)

IV. Implementation

1. Block diagram FIFO



2. Module FIFO

```
`timescale 1ns / 1ps

    module fifo ( output [4:0] c,
      input clk,  // Clock input
      input rst,  // Reset input
      input read_en,  // Read enable input
      input write_en,  // Write enable input
      input [7:0] data_in,  // Data input
      output reg [7:0] data_out,  // Data output
      output empty,  // Empty flag output
      output full  // Full flag output
    );

      parameter depth = 16;  // Depth of FIFO
      reg [7:0] memory [0:depth-1];  // Memory array
      reg [4:0] head, tail;  // Head and tail pointers
       reg [4:0] count;  // Count of elements in FIFO

      always @(posedge clk) begin
        if (rst) begin
          head <= 0;
          tail <= 0;
          count <= 0;
        end else begin
          if (write_en && !full) begin
            memory[head] <= data_in;
            head <= (head == depth-1) ? 0 : head + 1;
            count <= count + 1;
          end
          if (read_en && !empty) begin
            data_out <= memory[tail];
```

```
        tail <= (tail == depth-1) ? 0 : tail + 1;
        count <= count - 1;
      end
    end

  end
    assign empty = (count == 0) ? 1 : 0;
   assign full = (count == depth ? 1 : 0);
  assign c = count;

endmodule
```

3. Module testbench_FIFO

```
module testbench;
    wire [4:0] c;
    // Define the clock and reset signals
    reg clk;
    reg rst;

    // Define the signals for interfacing with the FIFO
    reg wr_en;
    reg rd_en;
    reg [7:0] data_in;
    wire [7:0] data_out;
    wire empty;
    wire full;

    // Instantiate the FIFO module
    fifo UTT(.clk(clk), .rst(rst), .write_en(wr_en), .read_en(rd_en),
.data_in(data_in), .data_out(data_out), .empty(empty), .full(full), .c(c));

    initial begin
```

```
    $monitor("Time: %t, data_out = %d", $time, data_out);

    if(empty == 1 && rd_en == 1) begin
        $monitor("FIFO is empty !");
    end

    if(full == 1 && wr_en == 1) begin
        $monitor("FIFO is full !");
    end
end
reg[4:0] i;
initial begin
    // Initialize the clock signal
    clk = 0;
    rd_en = 0;
    // Reset the FIFO
    rst = 1;
    #10;
    rst = 0;

    // Write some data to the FIFO
    wr_en = 1;

    for( i=0 ; i < 15; i = i + 1) begin
            #10;
            data_in = $random();
    end

    #10;
    wr_en = 0;
```

```
        // Read the data back from the FIFO
        #10;
        rd_en = 1;
        #200;
        $finish;
    end

    always #5 clk = ~clk;

endmodule
```

4. Giải thích các module
   - Module `fifo` được định nghĩa để triển khai một FIFO (First-In, First-Out) với độ sâu 16, có khả năng nhận và truyền các tín hiệu điều khiển đọc/ghi, dữ liệu đầu vào và dữ liệu đầu ra.
   - Đầu vào của module `fifo` bao gồm:
     - `clk`: tín hiệu xung clock.
     - `rst`: tín hiệu reset.
     - `read_en`: tín hiệu cho phép đọc.
     - `write_en`: tín hiệu cho phép ghi.
     - `data_in`: dữ liệu đầu vào.
   - Đầu ra của module `fifo` bao gồm:
     - `data_out`: dữ liệu đầu ra.
     - `empty`: tín hiệu chỉ ra FIFO đang trống.
     - `full`: tín hiệu chỉ ra FIFO đã đầy.
     - `c`: số lượng phần tử hiện tại trong FIFO.
   - Mỗi lần nhận được xung clock, module `fifo` sẽ kiểm tra tín hiệu reset. Nếu reset được kích hoạt, con trỏ head (trỏ đến phần tử đầu tiên của FIFO) và con trỏ tail (trỏ đến phần tử cuối cùng của FIFO) sẽ được thiết lập về giá trị ban đầu và biến đếm count (số lượng phần tử trong FIFO) sẽ được đặt về 0.
   - Nếu không có reset được kích hoạt, module `fifo` sẽ kiểm tra tín hiệu ghi và tín hiệu đọc. Nếu tín hiệu ghi được kích hoạt và FIFO chưa đầy, dữ liệu đầu vào sẽ được ghi vào vị trí trỏ bởi con trỏ head, con trỏ head sẽ được cập nhật đến vị trí tiếp theo trong FIFO và biến đếm count sẽ tăng lên 1.
   - Nếu tín hiệu đọc được kích hoạt và FIFO không trống, dữ liệu tại vị trí trỏ bởi con trỏ tail sẽ được đọc, con trỏ tail sẽ được cập nhật đến vị trí tiếp theo trong FIFO và biến đếm count sẽ giảm đi 1.
   - Sau đó, module `fifo` sẽ cập nhật tín hiệu empty và full dựa trên giá trị của biến đếm count.
   - Module `testbench` được sử dụng để kiểm tra module `fifo`. Nó cung cấp các tín hiệu điều khiển đọc/ghi, dữ liệu đầu vào và nhận dữ liệu đầu ra từ module `fifo`. Các tín hiệu empty và full cũng được theo dõi để đảm bảo FIFO hoạt động đúng.

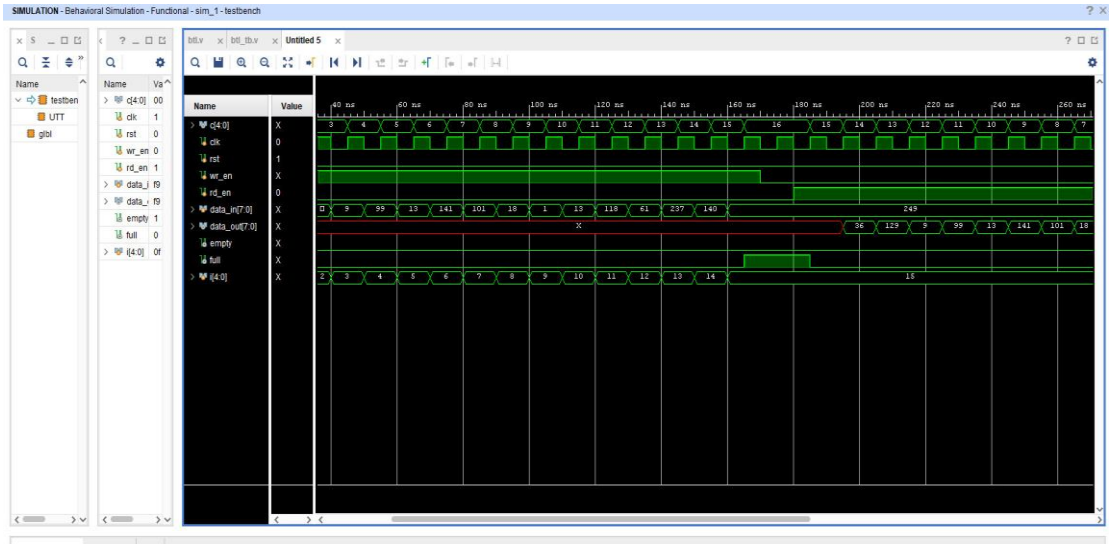
VI. Result

1. Kết quả thử nghiệm

```
`timescale 1ns / 1ps

module fifo ( output [4:0] c,
  input clk,  // Clock input
  input rst,  // Reset input
  input read_en,  // Read enable input
  input write_en,  // Write enable input
  input [7:0] data_in,  // Data input
  output reg [7:0] data_out,  // Data output
  output empty,  // Empty flag output
  output full  // Full flag output
);

  parameter depth = 16;  // Depth of FIFO
  reg [7:0] memory [0:depth-1];  // Memory array
  reg [4:0] head, tail;  // Head and tail pointers
   reg [4:0] count;  // Count of elements in FIFO

  always @(posedge clk) begin
    if (rst) begin
      head <= 0;
      tail <= 0;
      count <= 0;
    end else begin
      if (write_en && !full) begin
        memory[head] <= data_in;
        head <= (head == depth-1) ? 0 : head + 1;
        count <= count + 1;
```

```
Time:                    0, data_out =   x
Time:               195000, data_out =  36
Time:               205000, data_out = 129
Time:               215000, data_out =   9
Time:               225000, data_out =  99
Time:               235000, data_out =  13
Time:               245000, data_out = 141
Time:               255000, data_out = 101
Time:               265000, data_out =  18
Time:               275000, data_out =   1
Time:               285000, data_out =  13
Time:               295000, data_out = 118
Time:               305000, data_out =  61
Time:               315000, data_out = 237
Time:               325000, data_out = 140
Time:               335000, data_out = 249
main.v:96: $finish called at 380000 (1ps)
```

2. Waveform



3. Giải thích:

- Thiết lập các giá trị đầu vào **clock** =0, **rd_en**=0, **reset**=1 -> khởi tạo các giá trị head, tail, count.
- Delay 10ns sau đó giá trị **reset**=0, **wr_en**=1 -> cho phép thực hiện quá trình ghi dữ liệu đầu vào ngẫu nhiên từ vòng lập, cứ mỗi vòng lập delay thời gian ghi giá trị vào 10ns (tổng thời gian cho vòng lập là 160ns).
- Sau khi vòng lập kết thúc tương ứng quá trình ghi giá trị đầu vào cũng dừng sau 10ns kế tiếp. Tiếp tục delay 10ns và khởi tạo giá trị **rd_en** = 1 trong 200ns tiếp theo, bắt đầu quá trình đọc các giá trị đầu vào **data_in** và xuất vào **data_out** tại các cạnh lên của xung clock. Sau mỗi 5ns clock sẽ đảo giá

trị, mỗi cạnh lên cách nhau 10ns <=> các data_out delay 10ns cho đến hết thời gian khởi tạo và chương trình kết thúc.

VI. Conclusion

Ưu điểm:

- Dễ hiểu và triển khai: Nguyên tắc FIFO khá đơn giản và dễ hiểu. Nó không đòi hỏi các quy tắc phức tạp hay thuật toán phức hợp để triển khai. Do đó, việc triển khai và sử dụng FIFO trong các hệ thống và ứng dụng thường rất dễ dàng.
- Tương thích với nhiều ứng dụng: FIFO có thể được áp dụng trong nhiều lĩnh vực và ứng dụng khác nhau. Từ hệ thống xếp hàng và quản lý dữ liệu cho đến xử lý tín hiệu và quá trình sản xuất, FIFO có thể được sử dụng để xử lý và tổ chức dữ liệu theo thứ tự chúng được nhận.
- Đơn giản hóa quản lý: Sử dụng nguyên tắc FIFO giúp đơn giản hóa quá trình quản lý dữ liệu và tài nguyên. Với sự tuân thủ nguyên tắc "Đầu vào đầu ra", việc theo dõi và kiểm soát các yêu cầu, giao dịch hoặc tác vụ trở nên dễ dàng và hợp lý hơn.

Nhược điểm:

- Không linh hoạt trong việc ưu tiên: FIFO không xem xét các yếu tố khác ngoài thứ tự đến trước. Điều này có thể dẫn đến việc xử lý các yêu cầu quan trọng hơn hoặc tài nguyên quan trọng hơn mà đến sau, trong khi các yêu cầu ít quan trọng hơn được xử lý trước. Trong những trường hợp cần đánh giá các yếu tố khác như mức độ ưu tiên hay độ quan trọng, FIFO có thể không đáp ứng được nhu cầu này.
- Trong một số trường hợp, FIFO có thể dẫn đến lãng phí tài nguyên. Ví dụ, trong hệ thống xếp hàng, các yêu cầu hoặc tác vụ quan trọng có thể bị chờ đợi lâu hơn nếu hàng đợi đã đầy và cần phải chờ đến khi các yêu cầu trước được xử lý. Điều này có thể gây gián đoạn và ảnh hưởng đến hiệu suất và thời gian đáp ứng của hệ thống.

Tổng kết lại, FIFO là một nguyên tắc quan trọng và hữu ích trong việc quản lý dữ liệu và xử lý yêu cầu theo thứ tự chúng được nhận. Điều quan trọng là hiểu rõ ưu điểm và

nhược điểm của FIFO để áp dụng nó một cách hợp lý trong các hệ thống và ứng dụng. Bằng cách tận dụng ưu điểm và vượt qua nhược điểm, chúng ta có thể tạo ra các quy trình và hệ thống hiệu quả và công bằng hơn trong việc xử lý dữ liệu và yêu cầu.

Qua bài tập lớn này, nhóm chúng em đã hiểu hơn về nguyên tắc FIFO – First in, First out, cũng như các ứng dụng của nó trong các lĩnh vực cảu cuộc sống, các ưu nhước điểm và từ đó biết cách áp dụng một các hợp lý. Trong tương lai, nhóm em sẽ tiếp tục tìm hiểu sâu hơn về nguyên lý FIFO để có thể áp dụng vào các lĩnh vực và môn học khác (cụ thể là môn Cấu Trúc Dữ Liệu – DSA có Cấu trúc dữ liệu hàng đợi – queue).

Chúng em xin gửi lời tri ân sâu sắc đến thầy Đoàn Thiên Ân - giảng viên môn học “Thiết Kế Luận Lý - TN”, người đã tận tình giảng dạy và đồng hành với chúng em trong suốt học kì vừa qua. Bên cạnh đó, qua báo cáo lần này, chúng em học hỏi được nhiều kinh nghiệm, kỹ năng và cách thức để cùng nhau làm việc nhóm một cách tối ưu và hiệu quả nhất. Do lượng kiến thức của chúng em còn nhiều hạn chế, chưa thực sự hiểu sâu về môn học nên dù có cố gắng hoàn thiện đề tài qua tham khảo tài liệu, trao đổi và tiếp thu ý kiến đóng góp nhưng cũng không thể tránh khỏi thiếu sót. Chúng em rất mong nhận được sự đóng góp từ thầy để từ đó có thể rút kinh nghiệm, hoàn thiện hơn trong những lần làm đề tài sau này.

Chúng em xin chân thành cảm ơn!