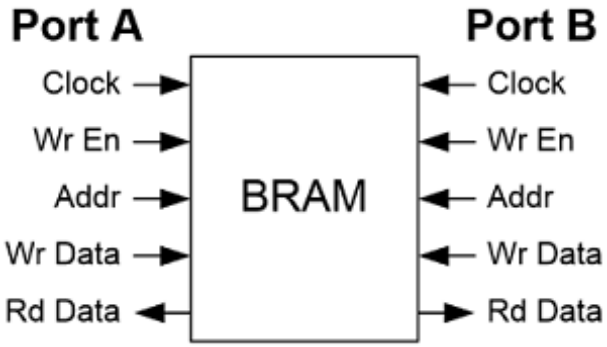
# 1 Block RAM

Ngoài nguồn lưu trữ dữ liệu là Distributed RAM với bản chất là một hình thức sử dụng khác của LUT thì trong Xilinx FPGA còn tích hợp các RAM (Block RAM) riêng biệt được cấu hình như một khối RAM hai cổng, số lượng bao gồm từ 4 đến 36 tùy theo từng IC cụ thể. Tất cả Block RAM hoạt động đồng bộ và có khả năng lưu trữ tập trung một khối lượng lớn thông tin.

BRAM trên thực tế đều là các khối RAM hai cổng nhưng các phần tử tương ứng được mô tả trong thư viện chuẩn của Xilinx và có thể được khởi tạo để hoạt động như RAM 2 cổng (Dual-port RAM) hoặc RAM 1 cổng (Single-port RAM).



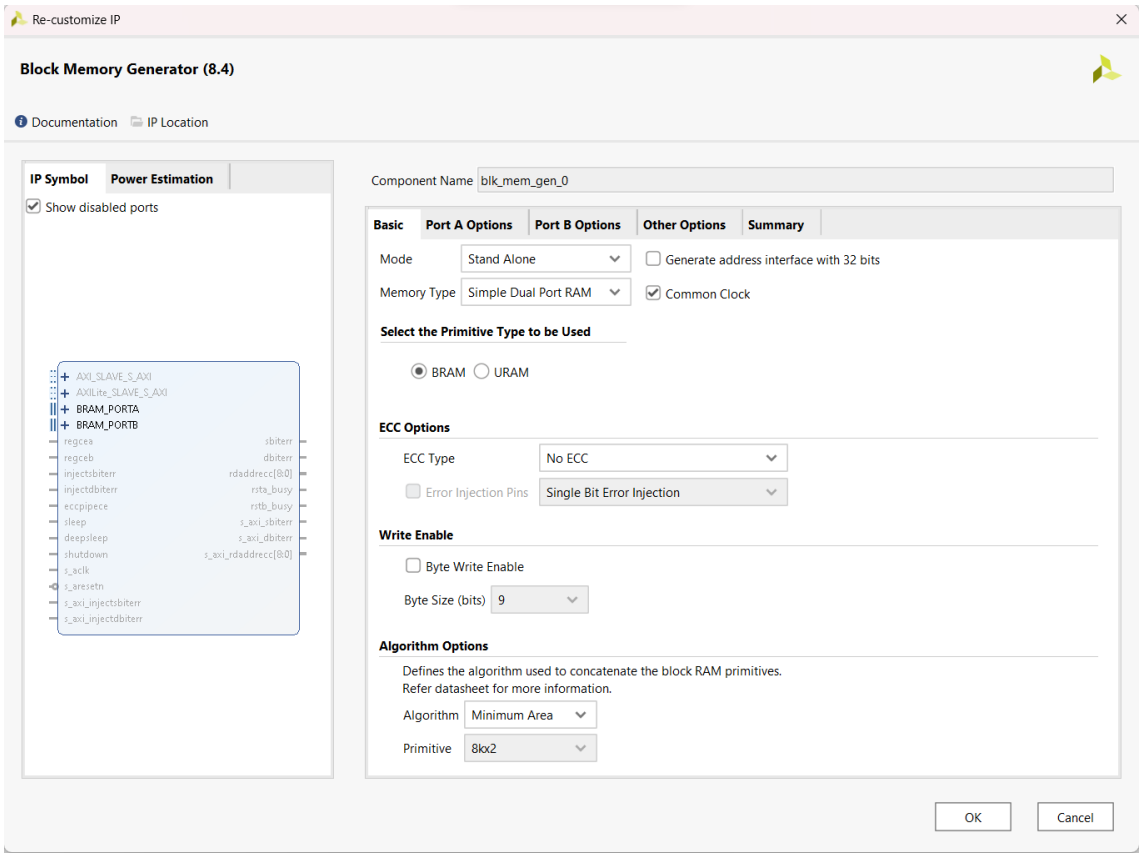

Hình 1: Giao diện BRAM hai cổng

Khối RAM có hai cổng A và B vào ra cho phép thực hiện các thao tác đọc ghi độc lập với nhau, mỗi một cổng có các tín hiệu xung nhịp đồng bộ, kênh dữ liệu và các tín hiệu điều khiển riêng. Có 4 đường dữ liệu cơ bản như sau:

1. Đọc ghi cổng A
2. Đọc ghi cổng B
3. Truyền dữ liệu từ A sang B
4. Truyền dữ liệu từ B sang A

## 1.1 Cấu hình IP

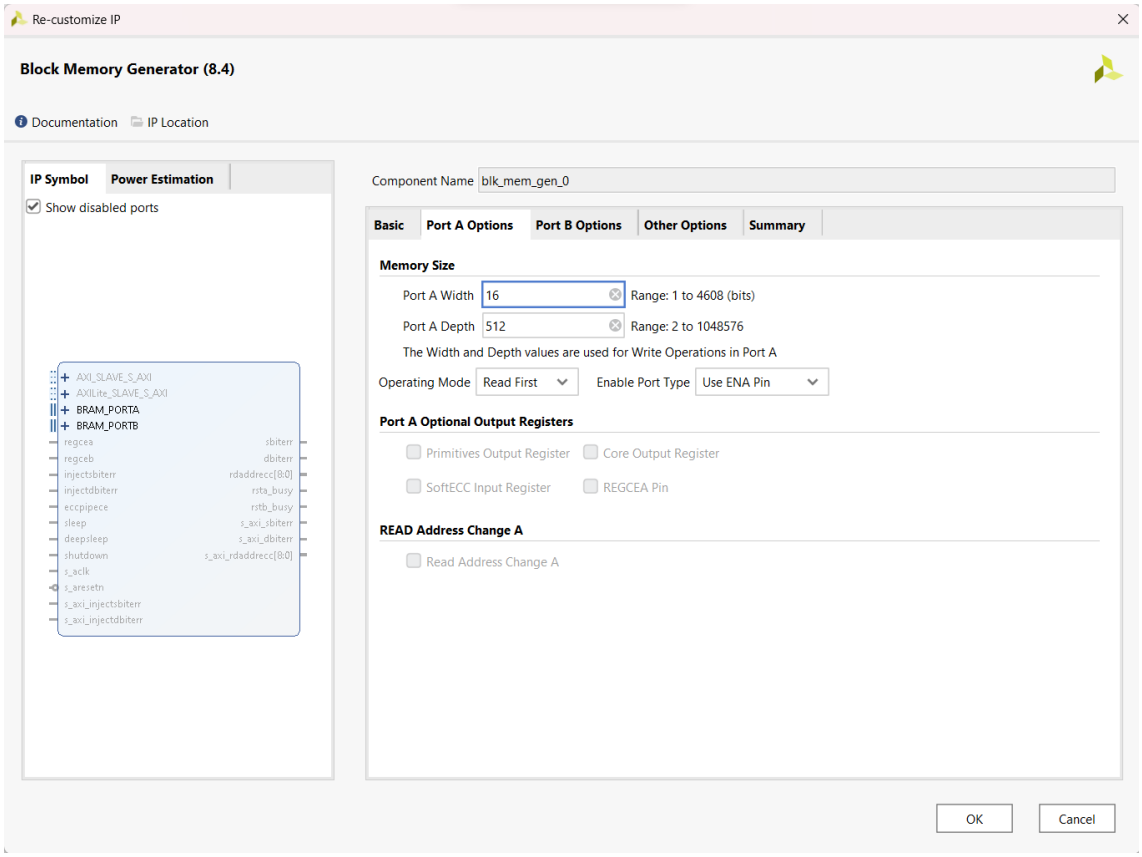
Trong Vivado, ta có thể tạo BRAM với IP Block Memory Generator.




**Có hai loại chính:** BRAM controller và Stand Alone. Ở đây ta chọn loại thứ 2 vì tính đơn giản cũng như dễ tiếp cận. Phần Controller có thêm các tính năng để có thể giao tiếp và điều khiển thông qua CPU bus. Điều này là hữu ích như không phải lúc nào cũng cần. Do đó, ta sẽ bắt đầu với mode Stand Alone để việc tìm hiểu được đơn giản hơn.

ECC option: Tùy chọn cho tính năng sửa lỗi, thường thấy trong các loại bộ nhớ thông dụng. Ở đây, chọn mặc định vì mục đích chỉ để simulation. Trong trường hợp cần ưu tiên khả năng tự khắc phục lỗi thì có thể áp dụng.

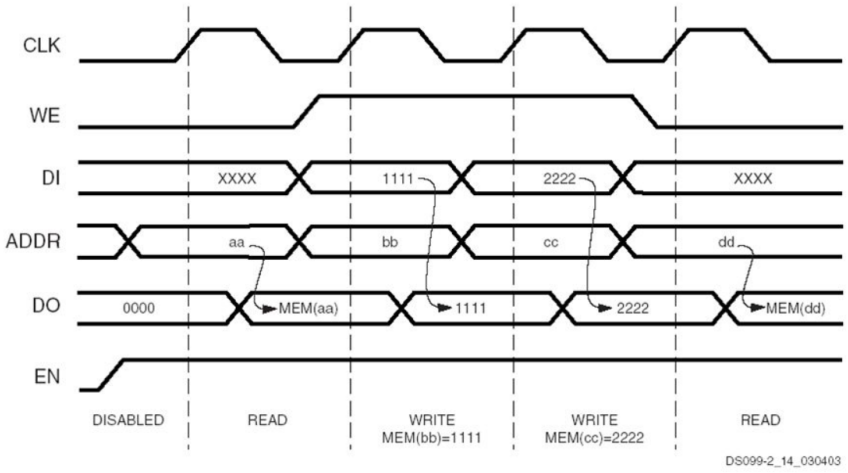
**Write Enable:** Đây là tùy chọn về chế độ ghi cho từng byte riêng lẻ.




Hình 2: Cấu hình Port A

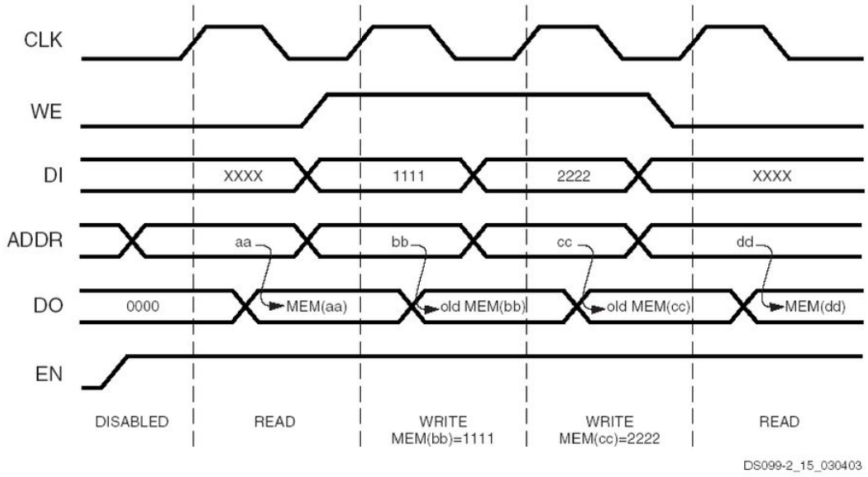
Phần cấu hình cho Port A. Ở phần **Operating Mode** thì có 3 chế độ:

+ **Write First:** ở chế độ này, khi tín hiệu write_enable được kích hoạt, RAM sẽ ghi dữ liệu (data_In) vào địa chỉ trong ô nhớ (addr), đồng thời sẽ truyền dữ liệu ra trong khoảng thời gian đó.



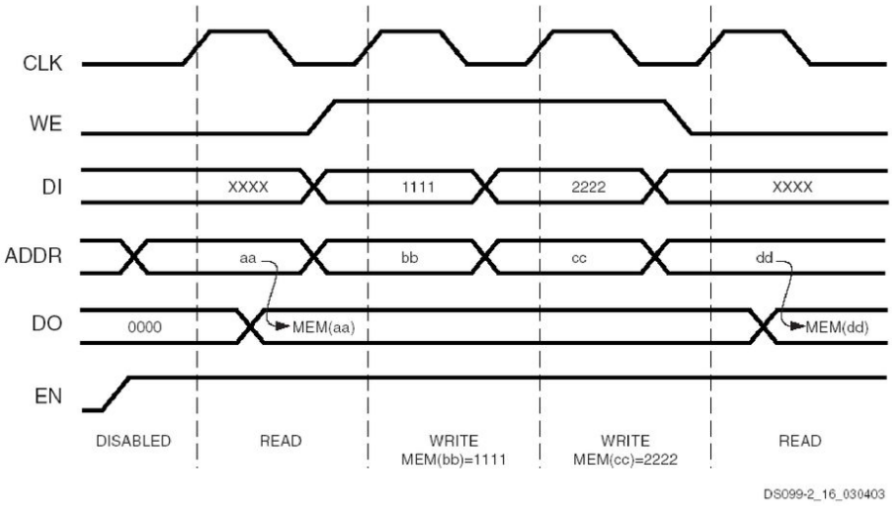

Hình 3: Chế độ Write First

+ **Read First:** ở chế độ này, giá trị được ghi lại (data_In) tại địa chỉ mong muốn (addr) sẽ được lưu tại địa chỉ đó sau khi giá trị được lưu trước đó tại địa chỉ đó được xuất ra (data_Out).



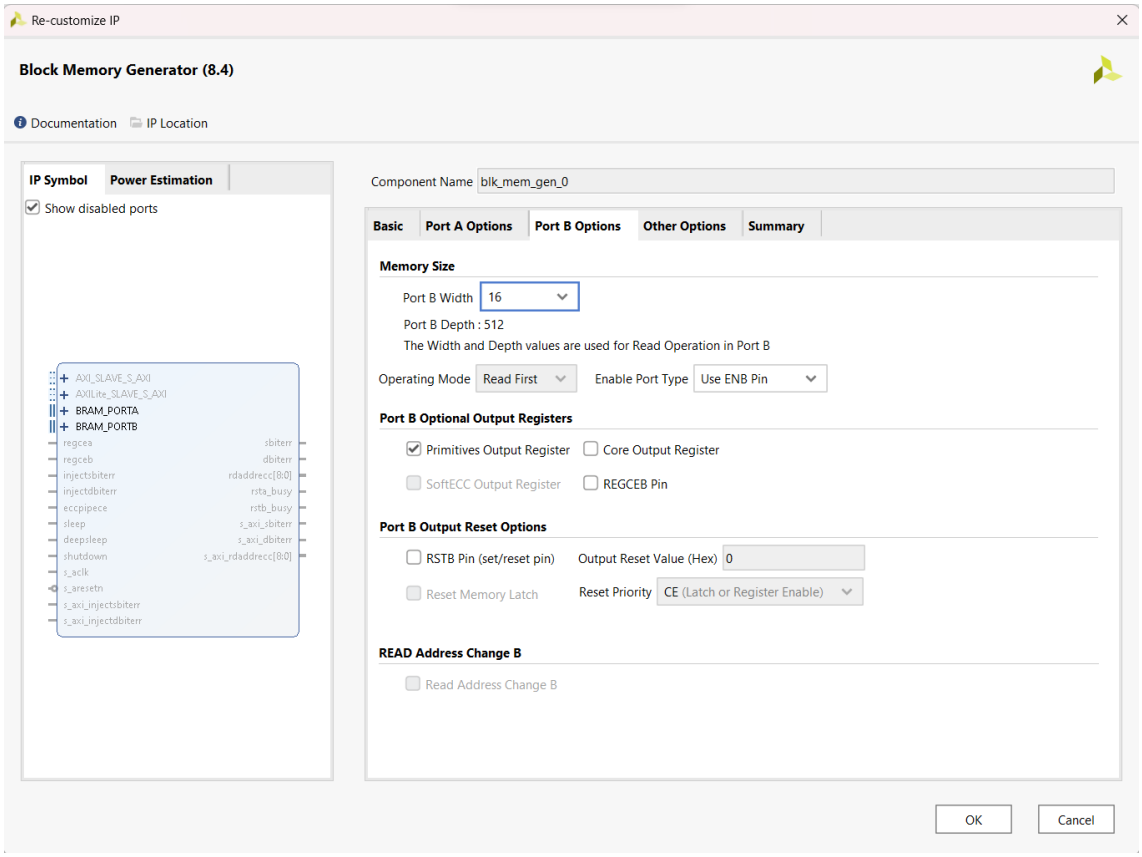

Hình 4: Chế độ Read First

+ **No change:** ở chế độ này, giá trị đầu ra sẽ được giữ nguyên trong suốt quá trình ghi dữ liệu, tức là khi chân write_enable được kích hoạt, khi chân write_enable dừng kích hoạt thì RAM mới bắt đầu đọc dữ liệu theo địa chỉ ở đầu vào.




Hình 5: Chế độ No Change

Vấn đề thường gặp khi sử dụng chế độ Dual-port đó là sẽ xảy ra trường hợp Read-write collision, là khi Port A ghi dữ liệu va Port B đọc dữ liệu tại cùng địa chỉ trong cùng khoảng thời gian. Vậy thì giá trị đầu ra sẽ là giá trị nào? Để giải quyết vấn đề đó, ta sẽ sử dụng chế độ **Read First** cho write port.

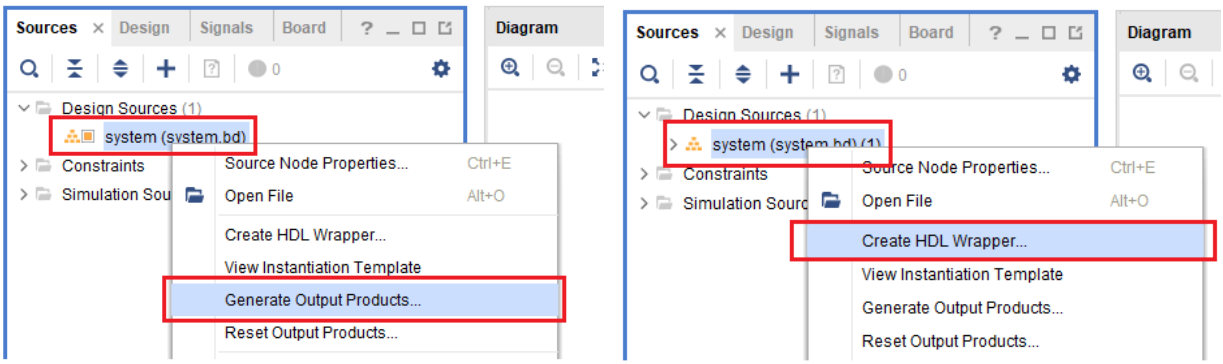



Hình 6: Cấu hình Port B

Tương tự như Port A, chúng ta có thể điệu chỉnh Width của Port B, ở lúc này thì nếu Width của Port B khác Port A, thì phần mềm sẽ tự điều chỉnh Depth của Port B cho phù hợp theo nguyên lý: **Width_A * Depth_A = Width_B * Depth_B**

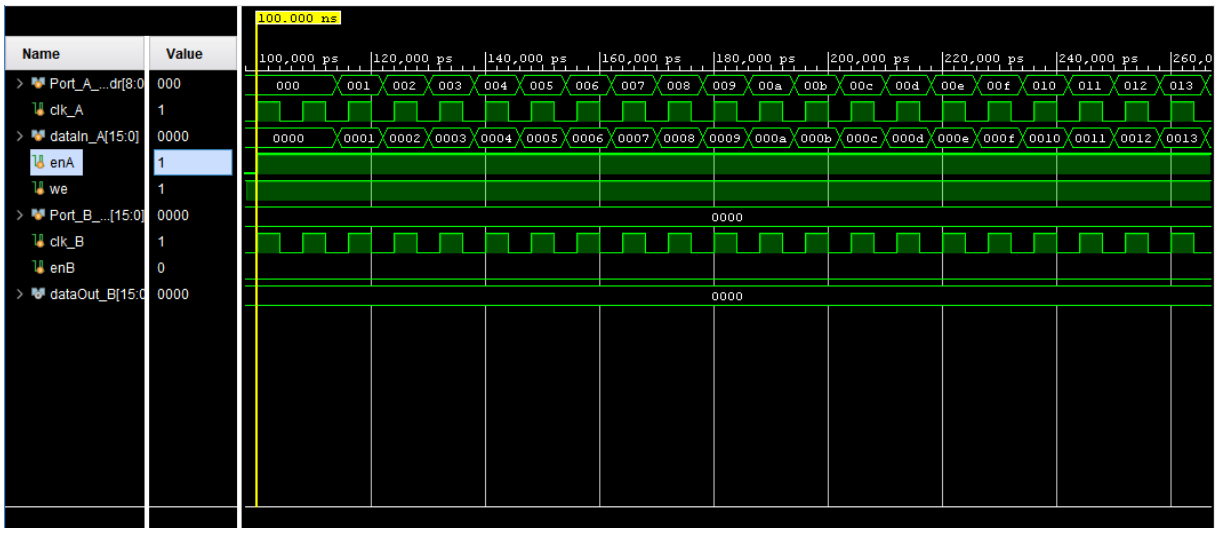
Tiếp đến là phần **Optional Register**, có thể chọn chế độ **Primitives Output Register** và **Core Output register**, được gọi là thanh ghi đệm, để cái thiện hiệu suất của RAM tốt hơn. Nó sẽ hạn chế ảnh hưởng delay của RAM và sẽ dùng rất tốt khi RAM hoạt động ở tần số cao.

Sau khi cấu hình xong, ta sẽ generate output products và wrap nó

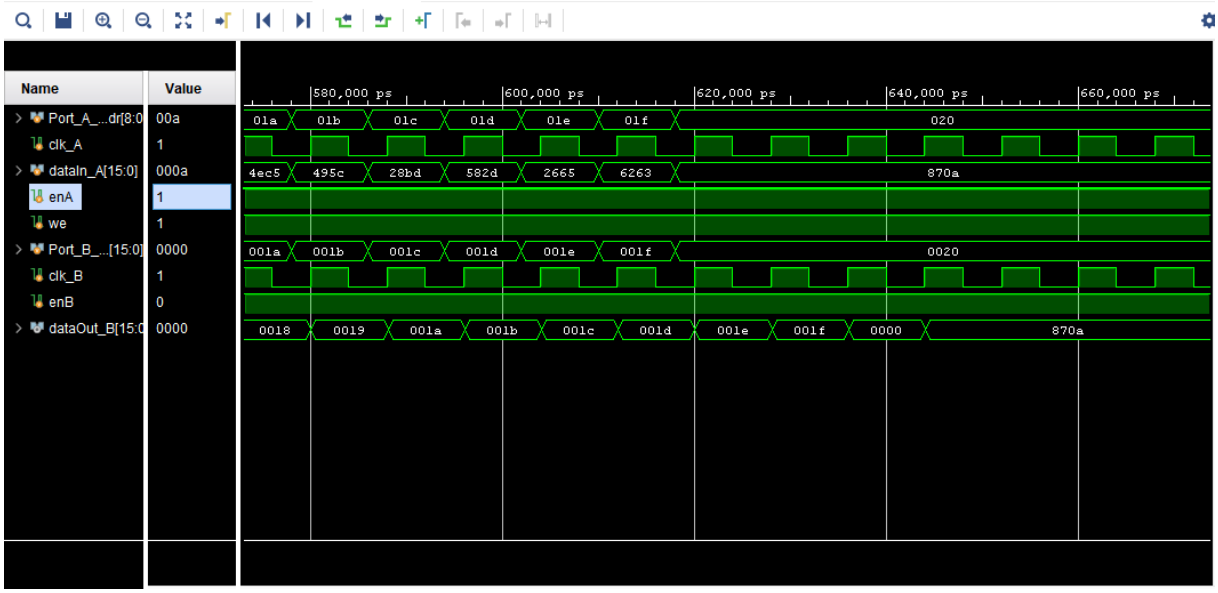


## 1.2 Test chức năng logic

Link testbench
Sau khi Run simulation, chúng ta có dạng sóng như sau:




Hình 8: Write data




Hình 9: Read data

Có thể thấy khi sử dụng 2 thanh ghi đệm **Primitives output register** và **Core output register**, dữ liệu đọc ở **data_Out** sẽ bị delay 2 chu kỳ

Khi sử dụng chế độ **Read First** sẽ không bị collision Read-Write, giá trị được đọc ra là giá trị được lưu từ trước trong bộ nhớ, sau đó giá trị mới sẽ được đưa vào sau đó.