JTE HCMUTE
ISSN: 1859-1272

TẠP CHÍ KHOA HỌC GIÁO DỤC KỸ THUẬT
Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh
Website: https://jte.hcmute.edu.vn/index.php/jte/index
Email: jte@hcmute.edu.vn

Implementation, Operation Analysis, and Performance Evaluation for a Wishbone-Based System-on-Chip

Pham Van Khoa[1*], Dang Phuoc Hai Trang[1], Ngo Dinh Thanh[2], Nguyen Quoc Khai[1], Nguyen Thanh Huy[1]
[1] HCMC University of Technology and Education (HCMUTE), Vietnam
[2] University of Science and Technology – University of Danang, Vietnam

[*] Corresponding author. *Email: khoapv@hcmute.edu.vn*

ARTICLE INFO

Received: 11/01/2022
Revised: 19/04/2022
Accepted: 10/01/2023
Published: 28/04/2023

KEYWORDS

System on chip;
Wishbone;
Arbiter;
Operation frequency;
Power consumption.

ABSTRACT

As integrated circuit technology advances, many processing cores can be built into a single silicon chip. This method optimizes design parameters like cost, size, and power consumption. Integrated circuit solutions support multiple connection types and shared interfaces for the cores to standardize data communication. Several popular buses, such as CoreConnect, AMBA, SiliconBackplane, and Wishbone, have been developed to accommodate many processor cores. The main challenge in designing the bus architecture is determining how to connect the processing cores in a simple, flexible, and scalable manner. Among the proposed methods, Wishbone architecture is a highly efficient solution for linking cores. This study implemented, analyzed, and evaluated the performance of a Wishbone-based System-on-Chip using simulation waveforms and FPGA hardware implementation to demonstrate the effectiveness of the Wishbone architecture in SoC design. According to simulation and realization results, Wishbone has a simple structure, requires fewer hardware resources, and is scalable for multi-core designs.

Xây Dựng, Phân Tích Hoạt Động và Đánh Giá Hiệu Năng Của Hệ Thống Trên Chip Sử Dụng Wishbone

Phạm Văn Khoa[1*], Đặng Phước Hải Trang[1], Ngô Đình Thanh[2], Nguyễn Quốc Khải[1], Nguyễn Thành Huy[1]
[1]Đại học Sư Phạm Kỹ Thuật Tp. Hồ Chí Minh, Việt Nam.
[2]Trường Đại học Bách Khoa - Đại học Đà Nẵng, Việt Nam.

[*]Tác giả liên hệ. *Email: khoapv@hcmute.edu.vn*

THÔNG TIN BÀI BÁO

Ngày nhận bài: 11/01/2022
Ngày hoàn thiện: 19/04/2022
Ngày chấp nhận đăng: 10/01/2023
Ngày đăng: 28/04/2023

TỪ KHÓA

Hệ thống trên chip;
Wishbone;
Bộ phân xử;
Tần số hoạt động;
Công suất tiêu thụ.

TÓM TẮT

Với sự phát triển của công nghệ vi mạch tích hợp, số lượng lớn các thành phần xử lý có thể được tích hợp trên một vi mạch đơn. Điều này mang lại ưu điểm như giảm giá thành, kích thước thiết kế và công suất tiêu thụ. Các giải pháp tích hợp hỗ trợ các kết nối đa điểm cho các lõi nhằm chuẩn hóa khả năng truyền dữ liệu. Để tạo kết nối trên các hệ thống đa lõi, một số thiết kế bus phổ biến như CoreConnect, AMBA, SiliconBackplane, và Wishbone đã được phát triển. Kiến trúc Wishbone là một phương pháp liên kết các lõi mang lại hiệu quả cao vì hỗ trợ nhiều dạng kết nối và các giao diện dùng chung cho các lõi làm chuẩn hóa và giúp giảm thiểu được vấn đề về khả năng tích hợp của hệ thống. Nhằm có thể kiểm chứng hiệu quả của kiến trúc Wishbone trong thiết kế SoC, nghiên cứu này đã thực thi, phân tích hoạt động và đánh giá hiệu năng một thiết kế SoC hoàn chỉnh ứng dụng kết nối Wishbone với phương pháp mô phỏng dạng sóng cũng như thực nghiệm trên phần cứng FPGA. Các kết quả mô phỏng và thực thi cho thấy rằng kiến trúc Wishbone có thiết kế đơn giản, yêu cầu một lượng tài nguyên phần cứng ít và phù hợp và có khả năng mở rộng liên kết dành cho các thiết kế đa lõi.

Doi: https://doi.org/10.54644/jte.76.2023.1122

JTE
HCMUTE
ISSN: 1859-1272

TẠP CHÍ KHOA HỌC GIÁO DỤC KỸ THUẬT
Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh
Website: https://jte.hcmute.edu.vn/index.php/jte/index
Email: jte@hcmute.edu.vn

1. Giới thiệu

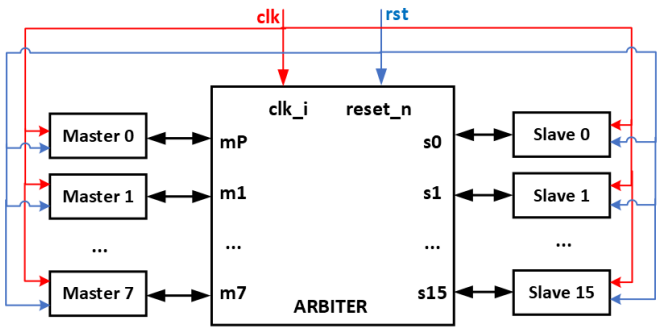
Với sự phát triển của công nghệ vi mạch tích hợp, số lượng lớn các thành phần xử lý có thể được tích hợp trên một vi mạch đơn [1]. Điều này mang lại ưu điểm như giảm giá thành, kích thước thiết kế và công suất tiêu thụ. Phương pháp tích hợp nhiều thành phần trên một chip đơn còn gọi là hệ thống trên chip (System-on-Chip SoC) sử dụng kiến trúc kết nối dạng bus để cung cấp giao diện chung giữa các lõi xử lý trong thiết kế. Hiện nay, một số kiến trúc tập hợp đường truyền tín hiệu (bus) phổ biến như CoreConnect, AMBA, SiliconBackplane và Wishbone đã được phát triển để gắn kết một lượng lớn lõi xử lý. Vấn đề chính khi thiết kế kiến trúc bus là làm thế nào để liên kết các lõi một cách đơn giản, linh hoạt và có khả năng mở rộng số lượng lõi kết nối. Thông thường thì các lõi trong thiết kế được phát triển riêng lẻ, và độc lập với nhau. Kiến trúc Wishbone được nhiều nhà nghiên cứu quan tâm bởi Wishbone hỗ trợ nhiều dạng kết nối và các giao diện dùng chung cho các lõi làm chuẩn hóa và giúp giảm thiểu được vấn đề về khả năng tích hợp của hệ thống. Điều này giúp việc kết nối các lõi trong các SoC trở nên dễ dàng hơn [2]-[3]. Như vậy, Wishbone cung cấp giao diện chung cho các lõi kết nối và giảm bớt các vấn đề tích hợp trên các thiết kế đa lõi [1]-[4].

Nghiên cứu này thực thi, phân tích hoạt động và đánh giá hiệu năng một SoC sử dụng kết nối Wishbone với các đặc điểm như kết nối kiểu chia sẻ đường truyền giữa chủ và tớ (Master và Slave), cơ chế phân xử độ ưu tiên theo chế độ ưu tiên (Priority) và xoay vòng (Round - Robin). Bên cạnh đó, nghiên cứu này cũng khảo sát hoạt động của các SoC sử dụng Wishbone ở nhiều cấu hình khác nhau nhằm đánh giá tần số hoạt động tối đa và công suất tiêu thụ trung bình trên mỗi trường hợp.

2. Thiết kế

2.1. Thiết kế tổng quát

Thông thường, Wishbone sử dụng mô hình kết nối kiểu giữa chủ và tớ (Master và Slave). Giữa Master và Slave sẽ giao tiếp với nhau thông qua một giao diện kết nối gọi là INTERCON. Wishbone thực hiện thay đổi dữ liệu tại cạnh lên của xung đồng hồ và tất cả tín hiệu của Wishbone đều tác động tích cực mức cao [5]-[7]. Kiến trúc Wishbone hỗ trợ bốn dạng kết nối gồm (1) Điểm đến điểm (Point to Point), (2) luồng dữ liệu (Data flow), (3) chia sẻ đường truyền (Shared bus) và (4) chuyển mạch tiếp điểm (Crossbar switch) [5]-[6]. Nghiên cứu này thực thi và phân tích mô hình SoC dựa trên kiến trúc Wishone ở dạng kết nối chia sẻ đường truyền (Shared bus).



Hình 1. *Hệ thống trên chip sử dụng kết nối Wishbone*

Sơ đồ tổng quát của hệ thống SoC sử dụng bus Wishbone kết nối 8 Master và 16 Slave được thể hiện bằng Hình 1. Trong đó, thiết kế SoC với 3 khối chính gồm (1) Khối phân xử (Arbiter - ARB) sẽ tương tác với tín hiệu của các Master và Slave. Khối ARB có nhiệm vụ truyền các đường dữ liệu và tín hiệu từ khối Master đến khối Slave và ngược lại. Đồng thời, khối ARB sử dụng thuật toán phân xử cho phép tại một thời điểm chỉ 1 Master được quyền truy cập bus và 1 Slave được kết nối đến Master đó. (2) Khối lõi xử lý chủ (Master) có chức năng yêu cầu một chu kỳ trao đổi dữ liệu trên bus. Khối này gửi các tín hiệu yêu cầu bắt đầu chu kỳ ghi/đọc, các tín hiệu địa chỉ và dữ liệu. Khối Master có thể là các bộ xử lý (Processor), bộ truy cập bộ nhớ trực tiếp (Direct Memory Access - DMA)...(3) Khối lõi tớ (Slave) có chức năng lưu trữ dữ liệu và thực thi yêu cầu từ Master truyền đến. Trong bus Wishbone, Slave nhận tín

hiệu yêu cầu bắt đầu chu kỳ trao đổi dữ liệu từ Master. Slave sẽ thực hiện và phản hồi đến Master thông tin trạng thái quá trình thực thi thành công, gửi lại hay bị lỗi. Khối Slave có thể là các bộ điều khiển ngoại vi (I/O module), bộ nhớ (Memory)...

2.2. Thuật toán phân xử quyền truy cập trong bus

Trong các thiết kế SoC, một hoặc nhiều Master có thể thực hiện các chu kỳ trao đổi dữ liệu với Slave thì phải được quyền thực thi và yêu cầu sử dụng đường truyền. Tại một thời điểm nếu có nhiều Master cùng yêu cầu quyền truy cập bus thì sẽ sinh ra xung đột và tranh chấp đường truyền, ảnh hưởng đến hoạt động chung của cả hệ thống. Vì vậy, bộ phân xử được thiết kế để có thể giải quyết vấn đề nêu trên và làm tăng hiệu năng xử lý của toàn hệ thống [8]-[9]. Kiến trúc bus được sử dụng trên các SoC thông thường sử dụng 02 loại phân xử gồm phân xử theo độ ưu tiên và phân xử theo vòng.

Ở phương pháp phân xử theo độ ưu tiên, bộ phân xử khi nhận yêu cầu từ lõi nguồn (Master) sẽ xem xét mức độ ưu tiên của tín hiệu nhận được cấu hình cho mỗi nguồn. Lõi nguồn có độ ưu tiên cao sẽ được cấp quyền truy cập bus trước. Lõi nguồn với độ ưu tiên cao kế tiếp sẽ đợi và được cấp phát quyền truy cập tiếp theo. Vòng lặp diễn ra cho đến khi Master cuối hoàn tất tác vụ và vòng lặp sẽ lại trở về với Master ban đầu [9]. Phương pháp này phù hợp với thiết kế trong đó phân quyền cao cho các lõi xử lý quan trọng và tần suất truy cập bus cao.

Ở phương pháp phân xử xoay vòng, mỗi lõi nguồn có mức ưu tiên truy cập bus là ngang nhau. Lõi nguồn ban đầu được cấp phát quyền truy cập, lõi kế tiếp sẽ phải đợi nguồn ban đầu hoàn thành xong quá trình truy cập và sau đó sẽ được cấp quyền truy cập sau đó. Sự tuần tự diễn ra cho các Master ở phía sau cho đến khi nguồn cuối cùng hoàn tất vòng lập sẽ lại trở về với nguồn ban đầu [8].

2.3. Thiết kế bus Wishbone

Hình 2a thể hiện giao tiếp giữa khối phân xử và các khối thành phần. Vai trò của khối phân xử rất quan trọng trong kết nối đa điểm của Wishbone khi chia sẻ đường truyền cho nhiều lõi xử lý khác nhau. Các tín hiệu và dữ liệu trên bus từ Master đến Slave và ngược lại đều phải thông qua khối phân xử. Khối phân xử sử dụng thuật toán xử lý để cấp quyền cho các lõi Master truy cập đường truyền.

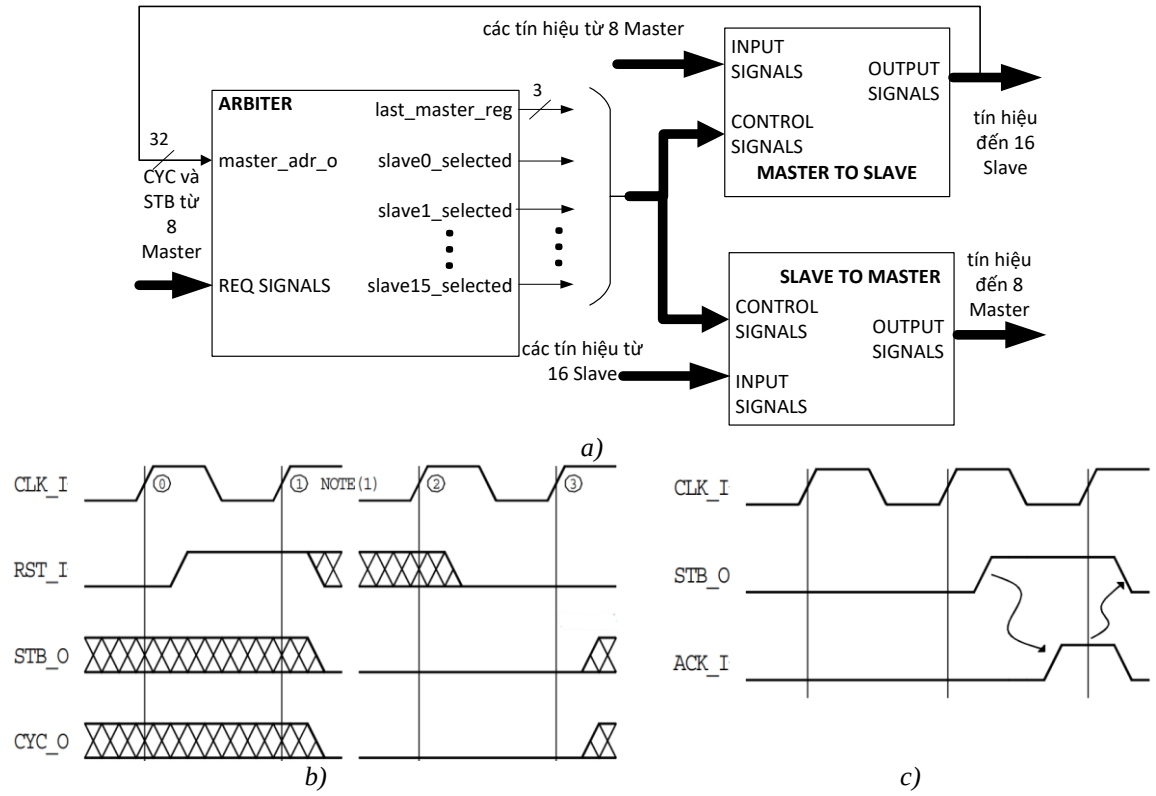
Khối **ARB** đảm nhận cấp quyền cho Master truy cập bus và lựa chọn **Slave** để giao tiếp tại một thời điểm. Ngoài các tín hiệu ngõ vào với chức năng thể hiện yêu cầu chu kỳ của **Master** được kết nối trực tiếp đến các ngõ ra của các **Master**, thì tín hiệu **master_adr_o** nhận từ khối **MASTER TO SLAVE** sẽ giúp lựa chọn **Slave** dựa trên địa chỉ bus. Các tín hiệu ngõ ra được kết nối trực tiếp đến các ngõ vào của 2 khối **MASTER TO SLAVE** và **SLAVE TO MASTER** giúp cho 2 khối trên có thể điều khiển và lựa chọn tín hiệu theo thuật toán phân quyền và khối ARB đã thực hiện. Trong các lõi xử lý thì **Master0** có độ ưu tiên cao nhất (**MasterP**), các lõi khác (**MasterR1** đến **MasterR7**) có mức độ ưu tiên như nhau.

Khối **MASTER TO SLAVE (M2S)** dựa trên các tín hiệu phân quyền nhận từ khối **ARB** để lựa chọn các tín hiệu và dữ liệu của **Master** được cấp quyền truy cập bus và giao tiếp với **Slave** đã lựa chọn. Ngoài các tín hiệu ngõ vào được kết nối trực tiếp đến các ngõ ra của **Master** và các tín hiệu ngõ ra được kết nối trực tiếp đến các ngõ vào của Slave thì khối **M2S** cũng nhận được các tín hiệu điều kiển được nhận từ khối **ARB**. Các tín hiệu được nhận từ khối **ARB** giúp cho khối **M2S** xác định được sẽ truyền các tín hiệu của **Master** được cấp quyền (**ADR, DAT_O, SEL, CYC, STB, WE**) đến ngõ vào của **Slave** được lựa chọn.

Khối **SLAVE TO MASTER (S2M)** có chức năng ngược lại với khối **M2S**. Ngoài các tín hiệu ngõ vào được kết nối trực tiếp đến các ngõ ra của các Slave và các tín hiệu ngõ ra được kết nối trực tiếp đến các ngõ vào của các **Master** thì khối **S2M** cũng nhận được các tín hiệu điều kiển được nhận từ khối **ARB**. Các tín hiệu này được nhận từ khối **ARB** giúp cho khối **S2M** xác định được sẽ truyền các tín hiệu của **Slave** được lựa chọn (**DAT_O, ACK, ERR, RTY**) đến ngõ vào **Master** được cấp quyền. Các tín hiệu cơ bản được sử trong Wishbone được thể hiện trong Bảng I.

Nghiên cứu này áp dụng thiết kế Wishbone tiêu chuẩn để xây dựng một mô hình SoC. Các giao thức chuẩn giao tiếp giữa **MASTER** và **SLAVE** được thể hiện trong Hình 2b và 2c. Trong đó, Hình 2b minh họa cho hoạt động reset trên thiết kế Wishbone. Các giao diện phần cứng sẽ được khởi tạo đến một trạng thái được định nghĩa trước. Điều này được thực hiện bởi tín hiệu **RST**. Độ rộng của xung **RST** có thể

thay đổi không cố định. Trên khối **MASTER**, tín hiệu ngõ ra **STB** và **CYC** có thể được xác nhận tại cạnh lên của xung **CLK** theo sau tín hiệu **RST**. Hình 2c, thể hiện giao thức bắt tay giữa **MASTER** và **SLAVE**. Tín hiệu ngõ ra **STB** trên **MASTER** sẽ xác nhận khi **MASTER** sẵn sàng truyền dữ liệu. Tín hiệu này được duy trì cho đến khi **SLAVE** xác nhận bằng một tín hiệu kết thúc **ACK** hoặc **ERR** tích cực mức **'1'** được lấy mẫu tại mỗi thời điểm cạnh lên của xung **CLK**. Nếu thỏa có tín hiệu **ACK** hoặc **ERR** tích cực mức **'1'** thì tín hiệu ngõ ra **STB** sẽ được chuyển thành mức **'0'**. Điều này giúp cho cả khối **MASTER** và **SLAVE** có thể kiểm soát được tốc độ tại mỗi thời điểm dữ liệu được truyền.



Hình 2. *a) Sơ đồ khối giao tiếp giữa khối phân xử và các khối thành phần b) Dạng sóng cho hoạt động reset [5] c) Dạng sóng cho giao tiếp bắt tay [5]*

Bảng 1. *Chức năng các tín hiệu cơ bản trong bus Wishbone*

Tên tín hiệu	Độ rộng	Mô tả chức năng
CYC	1	Tín hiệu yêu cầu chu kỳ bus
STB	1	Tín hiệu cho biết một chu kỳ đọc/ghi hợp lệ
WE	1	Tín hiệu quyết định chế độ đọc ghi
SEL	4	Tín hiệu chọn mảng dữ liệu ngõ ra
ADR	32	Đường địa chỉ truy xuất
DAT_I/O	32	Đường dữ liệu vào/ra
ACK	1	Tín hiệu phản hồi của Slave đến Master thông qua giao thức bắt tay
ERR	1	Tín hiệu thông báo chu kỳ lỗi
RTY	1	Tín hiệu yêu cầu thực hiện lại chu kỳ

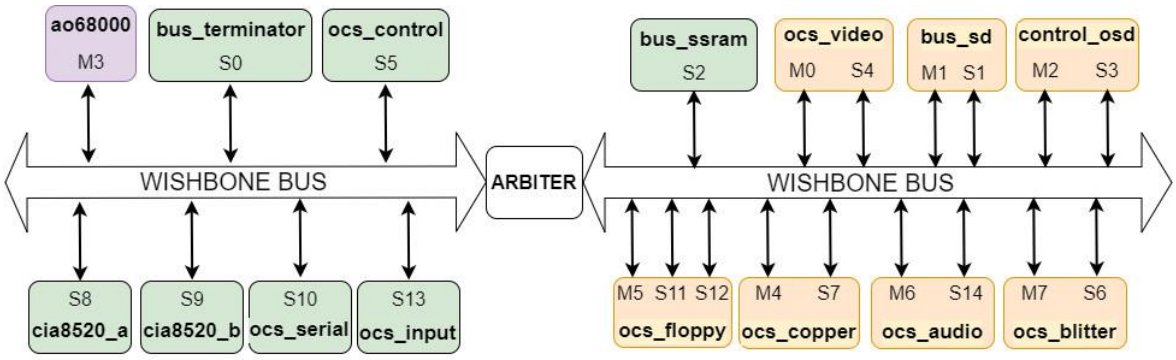
2.4. Xây dựng mô hình SoC ứng dụng Wishbone

Nhằm đánh giá hiệu năng của Wishbone trong một hệ thống SoC, nghiên cứu này đã xây dựng một mô hình SoC đa lõi hoàn chỉnh ứng dụng kiến trúc Wishbone. Chức năng chính của SoC là một hệ thống trò chơi điện tử 2D với khả năng đọc và xử lý các tập tin hình ảnh định dạng ADF (Amiga Disk File) được lưu trữ từ thẻ nhớ SD, được điều khiển bởi các bảng điều khiển từ người dùng và hiển thị lên màn

JTE
HCMUTE
ISSN: 1859-1272

TẠP CHÍ KHOA HỌC GIÁO DỤC KỸ THUẬT
Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh
Website: https://jte.hcmute.edu.vn/index.php/jte/index
Email: jte@hcmute.edu.vn

hình thông qua chuẩn giao tiếp VGA (Video Graphics Array). Như thể hiện trong Hình 3, mô hình SoC trong nghiên cứu này có 8 Master và 15 Slave được kết nối thông qua bus Wishbone kiểu kết nối chia sẻ đường truyền và có bộ phân xử áp dụng cả hai thuật toán phân xử Priority và Round - robin.



Hình 3. *Sơ đồ khối thiết kế SoC ứng dụng Wishbone*

Mô hình SoC được xây dựng như Hình 3 bao gồm 1 bộ phân xử **ARBITER** và 15 lõi. Trong đó, 1 lõi có vai trò như Master được thể hiện màu tím, 8 lõi có vai trò như Slave được thể hiện màu xanh và 7 lõi có vai trò như Master và Slave được thể hiện màu vàng. Chức năng của từng lõi được thể hiện bằng Bảng II dưới đây.

Bảng 2. *Chức năng các lõi trong mô hình SoC*

STT	Tên lõi IP	Loại (M Master S Slave)	Chức năng
1	**ao68000**	M3	Bộ vi xử lí cho cả hệ thống
2	**bus_terminator**	S0	Kết thúc cho các chu kỳ bus Wishbone không được xử lý
3	**bus_ssram**	S2	Bộ nhớ lưu trữ SRAM
4	**ocs_control**	S5	Quản lý và điều khiển hệ thống giao tiếp, trao đổi với giao diện Wishbone Slave.
5	**cia8520**	S8 & 9	Triển khai bộ điều hợp (Adapter) Commodore 8520
6	**ocs_serial**	S10	giao tiếp các port bên ngoài với giao diện Wishbone Slave
7	**ocs_input**	S13	Nhận đầu vào từ người dùng như bàn phím, joystick
8	**ocs_floppy**	M5, S11 & 12	Thực hiện đọc ghi các tập tin ADF
9	**ocs_video**	M0, S4	Triển khai phần Video của hệ thống
10	**bus_sd**	M1, S1	Triển khai thực hiện đọc ghi dữ liệu từ thẻ nhớ
11	**control_osd**	M2, S3	Quản lý phần hiển thị tổng thể trên màn hình.
12	**ocs_copper**	M4, S7	Triển khai mạch copper cho hệ thống (đặt lại giá trị các thanh ghi cho đầu mỗi khung hình video trước khi truyền đến khối ocs_video)
13	**ocs_audio**	M6, S14	Triển khai phần âm thanh của hệ thống
14	**ocs_blitter**	M7, S6	Triển khai mạch blitter cho hệ thống (sao chép một lượng lớn dữ liệu từ vùng nhớ này sang vùng nhớ khác một cách tương đối nhanh chóng, song song với CPU)

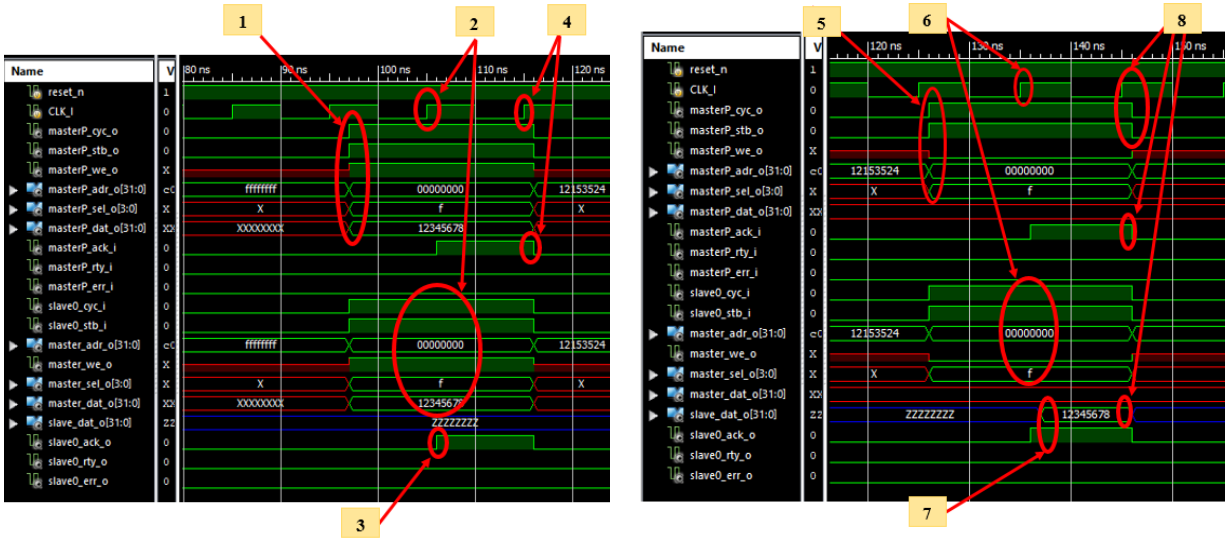
3. Phân tích hoạt động

Trước khi đi vào đánh giá các tham số thiết kế như tần số hoạt động, công suất tiêu thụ... thì thiết kế cần được xác thực khả năng hoạt động ổn định ở các chế độ truyền dẫn dữ liệu cơ bản như ghi và đọc trên dữ liệu đơn cũng như chuỗi dữ liệu. Bộ công cụ Xilinx ISE [10], [11] được sử dụng trong nghiên cứu này để xác thực hoạt động, đánh giá tần số, khảo sát tài nguyên phần cứng cũng như công suất tiêu thụ trung bình của hệ thống. Việc xác thực hoạt động của hệ thống được thực hiện thông qua các trường

JTE
HCMUTE
ISSN: 1859-1272

TẠP CHÍ KHOA HỌC GIÁO DỤC KỸ THUẬT
Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh
Website: https://jte.hcmute.edu.vn/index.php/jte/index
Email: jte@hcmute.edu.vn

hợp mô phỏng khi đưa các kích thích ngõ vào và giám sát trạng thái của tín hiệu ngõ ra. Công cụ Xilinx ISE sẽ mô phỏng hoạt động của thiết kế dựa trên các điều kiện ngõ vào và hiển thị kết quả dạng sóng ngõ ra một cách trực quan.

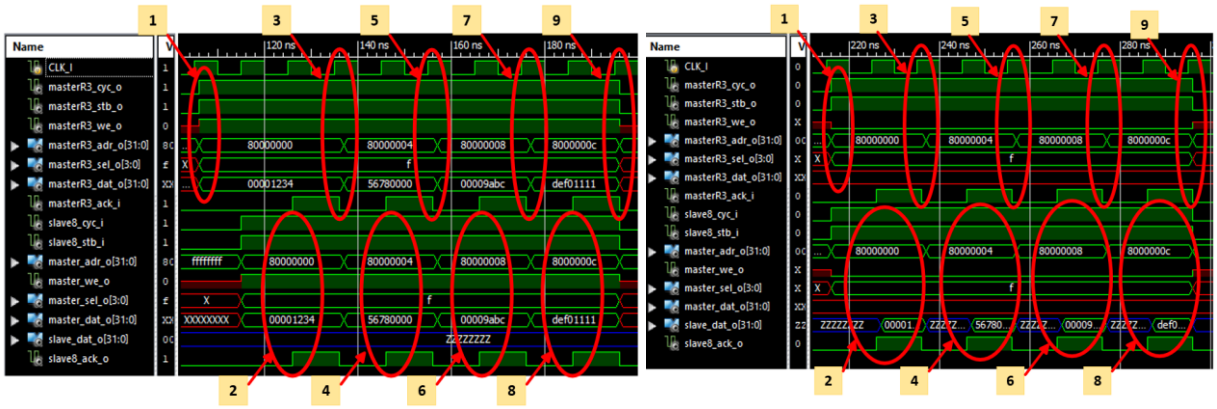


(a) *(b)*

Hình 4. *Dạng sóng cho a) chu kỳ ghi đơn b) chu kỳ đọc đơn*

3.1. Hoạt động ghi/đọc đơn

Một chu kỳ ghi/đọc đơn trên hệ thống sử dụng Wishbone được thể hiện thông qua Hình 4a và 4b tương ứng. Chu kỳ ghi diễn ra giữa lõi chủ **MasterP** và lõi tớ **Slave0** được yêu cầu ở thời điểm (1) khi **MasterP** đưa 2 tín hiệu **masterP_cyc_o** và **masterP_stb_o** tích cực ở mức **'1'**, đồng thời tín hiệu cho phép ghi **masterP_we_o** cũng tích cực mức **'1'**, dữ liệu được đưa ra ở đường tín hiệu **masterP_dat_o** với tín hiệu lựa chọn ngõ ra trên **masterP_sel_o** xác định các byte có giá trị **0x12345678** trên đường dữ liệu **masterP_dat_o** và địa chỉ ô nhớ thứ **0** trên đường tín hiệu **masterP_adr_o**. Tại thời điểm (2), khi có xung clock kích cạnh lên, **Slave0** xác nhận yêu cầu ghi của **MasterP**. Tại thời điểm (3), khi **Slave0** thực hiện xong yêu cầu, **Slave0** sẽ phản hồi lại đã thực hiện thành công bởi tín hiệu **slave0_ack_o** mức **'1'**. Tại thời điểm (4), khi có xung clock kích cạnh lên tiếp theo, **MasterP** sẽ nhận tín hiệu **ack** từ **Slave0** và kết thúc chu kỳ bằng cách cho 2 tín hiệu **masterP_cyc_o** và **masterP_stb_o** xuống mức **'0'**.



(a) *(b)*

Hình 5. *Dạng sóng cho a) chu kỳ ghi chuỗi b) chu kỳ đọc chuỗi*

Chu kỳ đọc đơn được thể hiện trong Hình 4b. Chu kỳ này được yêu cầu tại thời điểm (5), khi **MasterP** đưa 2 tín hiệu **masterP_cyc_o** và **masterP_stb_o** tích cực mức **'1'**, đồng thời thời tín hiệu cho phép ghi **masterP_we_o** ở mức **'0'**, địa chỉ đọc được thể hiện trên đường tín hiệu **masterP_adr_o** là ô nhớ thứ

0. Tại thời điểm (6), khi có xung nhịp kích cạnh lên, **Slave0** sẽ xác nhận yêu cầu đọc của **MasterP**. Tại thời điểm (7), khi **Slave0** thực hiện xong yêu cầu, **Slave0** sẽ phản hồi lại trên đường tín hiệu **slave0_ack_o** và dữ liệu **MasterP** muốn đọc trên đường dữ liệu **slave0_dat_o** và qua khối **ARBITER** đến đường dữ liệu **slave_dat_o**. Tại thời điểm (8), khi có xung clock kích cạnh lên tiếp theo, **MasterP** sẽ xác nhận tín hiệu **ack** phản hồi từ **Slave0** đồng thời đọc dữ liệu đã nhận được từ **Slave0** và kết thúc chu kỳ bằng cách cho hai tín hiệu **masterP_cyc_o** và **masterP_stb_o** xuống mức **'0'**. Hình 4a và 4b minh họa cho trường hợp ghi vào **Slave0** tại ô nhớ thứ 0 với dữ liệu 32 bit là **0x12345678** và sau đó đọc từ ô nhớ thứ 0 của **Slave0** thu được kết quả là **0x12345678**. Điều này cho thấy mô hình SoC được thiết kế đã hoạt động đúng chức năng như mô tả.

3.2. Hoạt động ghi/đọc chuỗi

Hoạt động ghi/đọc một chuỗi dữ liệu được thể hiện chi tiết bằng dạng sóng ở Hình 5a và 5b. Tại thời điểm (1), chu kỳ ghi chuỗi giữa lõi chủ **MasterR3** và lõi tớ **Slave8** được yêu cầu khi **MasterR3** đưa 2 tín hiệu **masterR3_cyc_o** và **masterR3_stb_o** ở mức **'1'**, đồng thời tín hiệu cho phép ghi **masterR3_we_o** tích cực mức **'1'** cho đến khi chuỗi dữ liệu được ghi xong, dữ liệu được xuất ra ở đường tín hiệu **masterR3_dat_o** với tín hiệu lựa chọn mảng ngõ ra **masterR3_sel_o** xác định các byte có giá trị trên đường dữ liệu **masterR3_dat_o** và địa chỉ trên đường tín hiệu **masterR3_adr_o**. Tại các thời điểm (3) (5) (7), khi có xung clock kích cạnh lên, **Slave8** sẽ xác nhận yêu cầu ghi của **MasterR3**. Khi **Slave8** thực hiện xong yêu cầu, **Slave8** sẽ phản hồi lại trên đường tín hiệu **slave8_ack_o** bằng mức **'1'** báo đã thực thiện thành công 1 chu kỳ ghi đơn ở các thời điểm (2) (4) (6) (8). Đồng thời, ở các thời điểm (3) (5) (7), ở cạnh lên xung clock tiếp theo, **MasterR3** sẽ xác nhận tín hiệu **ack** phản hồi từ **Slave8** và yêu cầu chu kỳ ghi tiếp theo. Công việc thực hiện ghi chuỗi sẽ được lặp lại cho đến khi chuỗi dữ liệu được ghi xong, chu kỳ ghi chuỗi sẽ được kết thúc bằng cách cho 2 tín hiệu **masterR3_cyc_o** và **masterR3_stb_o** xuống mức **'0'** tại thời điểm (9). Ở hoạt động ghi chuỗi, lõi **MasterR3** thực hiện 4 hoạt động ghi đơn đến lõi **Slave8**. Trong đó, mỗi hoạt động ghi đơn sẽ truyền tải 4 Byte dữ liệu trên đường tín hiệu **masterR3_dat_o** và hoạt động này diễn ra trong 2 chu kỳ xung nhịp. Đồng thời, khối **ARBITER** cần 1 chu kỳ để phân xử cho **MasterR3** có quyền truy cập đường truyền. Tóm lại, hoạt động ghi chuỗi với 4 chu kỳ ghi đơn được thực hiện trong 9 chu kỳ xung nhịp với tổng số dữ liệu ghi được là 16 Byte.

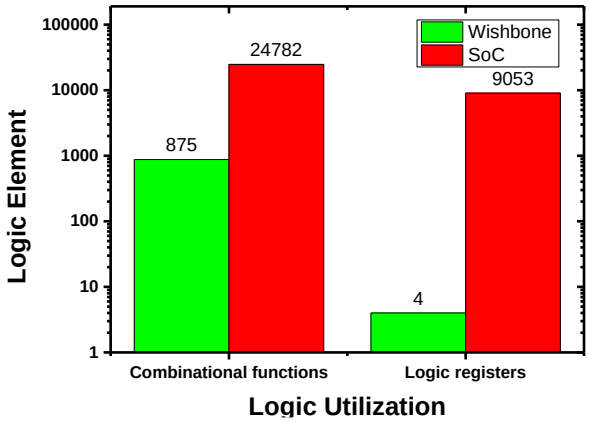
Dạng sóng mô phỏng chu kỳ đọc chuỗi giữa lõi chủ **MasterR3** và lõi tớ **Slave8** được thể hiện chi tiết trong Hình 5b. Tại thời điểm (1), chu kỳ đọc được yêu cầu khi **MasterR3** đưa 2 tín hiệu **masterR3_cyc_o** và **masterR3_stb_o** tích cực mức **'1'**, đồng thời thời tín hiệu cho phép ghi **masterR3_we_o** ở cực mức **'0'**, và đọc ở địa chỉ trên đường tín hiệu **masterR3_adr_o**. Tại các thời điểm (3) (5) (7), khi có xung nhịp kích cạnh lên, **Slave8** sẽ xác nhận yêu cầu chu kỳ đọc của **MasterR3**. Khi **Slave8** thực hiện xong yêu cầu, **Slave8** sẽ phản hồi lại trên đường tín hiệu **slave8_ack_o** và dữ liệu **MasterR3** muốn đọc trên đường dữ liệu **slave8_dat_o** và qua khối **ARBITER** đến đường dữ liệu **slave8_dat_o** ở các thời điểm (2) (4) (6) (8). Đồng thời, ở các thời điểm (3) (5) (7), ở cạnh lên xung clock tiếp theo, **MasterR3** sẽ xác nhận tín hiệu **ack** phản hồi từ **Slave8** đồng thời đọc dữ liệu đã nhận được từ **Slave8** và yêu cầu chu kỳ đọc tiếp theo. Công việc thực hiên đọc chuỗi sẽ được lặp lại cho đến khi chuỗi dữ liệu được đọc hoàn chỉnh, chu kỳ đọc chuỗi sẽ được kết thúc bằng cách cho 2 tín hiệu **masterR3_cyc_o** và **masterR3_stb_o** xuống mức **'0'** ở thời điểm (9). Hình 5a minh họa cho trường hợp ghi vào **Slave8** từ ô nhớ thứ 0 đến 3 với dữ liệu lần lượt là **0x00001234; 0x56780000; 0x00009abc; 0xdef01111** và sau đó đọc từ ô nhớ thứ 0 đến 3 của **Slave8** thu được kết quả là **0x00001234; 0x56780000; 0x00009abc; 0xdef01111** ở Hình 5b. Điều này cho thấy mô hình SoC được thiết kế đã hoạt động đúng chức năng như mô tả.

4. Đánh giá thiết kế

4.1. Tài nguyên phần cứng

Thiết kế trong nghiên cứu này được thực thi trên nền tảng phần cứng mảng cổng lập trình được dạng trường (Field programmable Gate Array - FPGA) trên bo mạch Spartan-3E của hãng Xilinx. Công cụ tổng hợp thiết kế ISE [10] được sử dụng để thống kê chi tiết số lượng tài nguyên cần thiết đối với kiến trúc Wishbone cũng như của thiết kế mô hình SoC hoàn chỉnh. Từ kết quả tổng hợp thiết kế có thể thấy

rằng tài nguyên phần cứng được yêu cầu đối với mỗi khối trong thiết kế là khác nhau. Chi tiết về tài nguyên phần cứng trên chip FPGA khi thực thi cấu hình hệ thống trên chip với 8 Master và 16 Slave (SoC) và kiến trúc Wishbone được thể hiện bằng Hình 6.

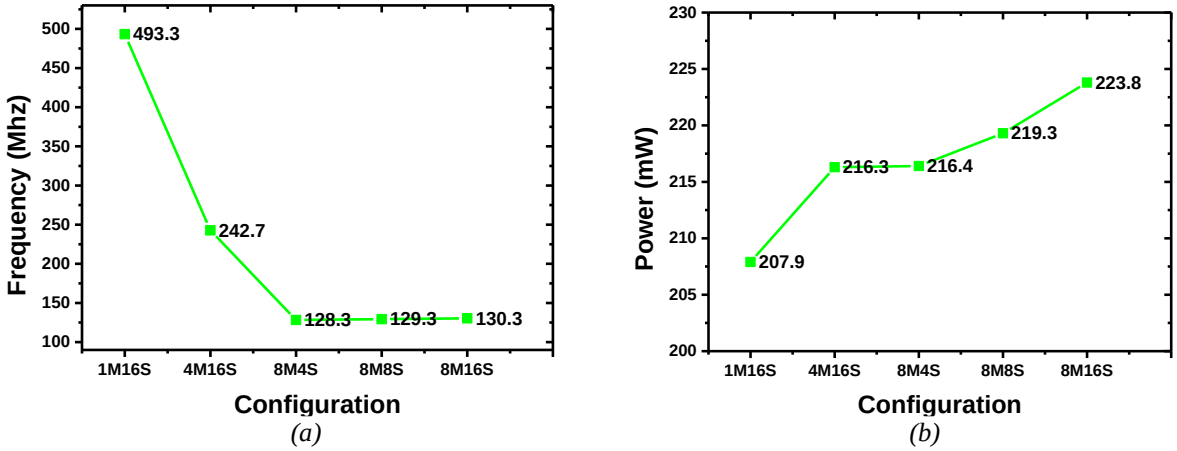


Hình 6. *Tài nguyên sử dụng đối với kiến trúc Wishbone và SoC*

Từ biểu đồ so sánh có thể thấy tài nguyên cần thiết để thực thi nền tảng kết nối Wishbone thấp hơn rất nhiều so với tài nguyên để thực thi mô hình SoC hoàn chỉnh sử dụng Wishbone. Tài nguyên phần cứng cần thiết để thực thi Wishbone chủ yếu là các thành phần cổng logic, số lượng thành phần thanh ghi được sử dụng rất thấp so với cổng logic. Hơn nữa, tỷ số phần trăm giữa tổng số lượng tài nguyên kiến trúc Wishbone và tổng số lượng tài nguyên SoC dựa trên Wishbone gần bằng 2.6%. Đồng thời, tổng số tài nguyên sử dụng cho kiến trúc bus Wishbone là khoảng 900 thành phần. Điều này cho thấy bus Wishbone rất phù hợp cho các hệ thống bị hạn chế về tài nguyên phần cứng.

4.2. Tần số hoạt động

Về yếu tố phần cứng, tần số hoạt động tối đa được quyết định bởi nền tảng phần cứng của chip FPGA. Một khi lựa chọn được nền tảng phần cứng để thực thi thiết kế thì yếu tố chính ảnh hưởng đến tần số hoạt động là kiến trúc của thiết kế được xây dựng từ các mã mô tả thiết kế phần cứng. Tần số hoạt động trong một thiết kế đồng bộ được giới hạn bởi thời gian thiết lập (setup time), thời gian ổn định (hold time) trên mỗi flip-flop cùng độ trễ lan truyền (propagation delay). Ngoài ra, một số yếu tố khác cũng ảnh hưởng đến tần số khi hệ thống hoạt động là sự biến thiên về điện áp cung cấp và nhiệt độ hoạt động. Hình 7a minh họa các tần số hoạt động khác nhau của thiết kế khi khảo sát trên một số cấu hình SoC khác nhau như 1 Master 16 Slave (1M16S), 4 Master 16 Slave (4M16S), 8 Master 16 Slave (8M16S), 8 Master 8 Slave (8M8S) và 8 Master 4 Slave (8M4S).



Hình 7. *a) Tần số hoạt động tối đa trên các cấu hình SoC b) Công suất tiêu thụ trên các cấu hình SoC*

Hình 7a thể hiện lên sự chênh lệch tần số hoạt động tối đa trong 5 trường hợp. Có thể thấy, đối với 3 trường hợp có số lượng Slave được cố định là 16, số lượng Master được thay đổi từ 1, 4 và 8 thì tần số hoạt động của bus bị giảm gần 4 lần từ 493 Mhz xuống 128 Mhz. Điều này vì số lượng Master càng nhiều thì xác suất tồn tại các yêu cầu truy cập đường truyền từ các lõi Master sẽ tăng lên, từ đó bộ phân xử ARBITER cần nhiều thời gian để quyết định quyền truy cập bus sẽ dành cho Master nào. Đây là nguyên nhân làm cho hệ thống có độ trễ lớn dẫn đến tần số hoạt động của hệ thống bị giảm đáng kể. Tuy nhiên trong trường hợp số lượng Master được giữ nguyên là 8, và số lượng Slave được thay đổi từ 4, 8 và 16 thì tần số hoạt động của bus duy trì trung bình ở 129 Mhz. Kết quả này đúng với tính chất của việc giao tiếp giữa Master và Slave, vì Master là nguồn đưa ra các tín hiệu yêu cầu truy cập bus và nguồn giao tiếp với Slave, thời gian xử lý trên bộ phân xử ARBITER không bị ảnh hưởng nhiều với số lượng của các Slave. Khi so sánh với các nghiên cứu trước đó, mô hình SoC trong nghiên cứu này có tần số hoạt động tối đa cao hơn xấp xỉ 2.1 lần so với mô hình SoC sử dụng cấu hình Wishbone với 4 lõi chủ và 4 lõi tớ được kết nối dạng Shared-bus có tần số hoạt động là 118,3 MHz [7] và cao hơn xấp xỉ 2.6 lần so với mô hình Wishbone Conmax được kết nối dạng Crossbar có tần số hoạt động là 94,5 MHz [12]. Như vậy có thể thấy tần số hoạt động tối đa của hệ thống được quyết định chính bằng số lượng Master được thiết kế trong hệ thống.

4.3. Công suất tiêu thụ

Công suất tiêu thụ trung bình là một trong những tiêu chí quan trọng đánh giá hiệu năng của một thiết kế số. Phần đánh giá công suất được thực hiện bởi công cụ tổng hợp thiết kế ISE [10]. Để đánh giá công suất tiêu thụ một cách tổng quát trong nhiều trường hợp khác nhau, nghiên cứu đã thực hiện tổng hợp công suất tiêu thụ của thiết kế SoC với 5 cấu hình phần cứng gồm 1 Master 16 Slave (1M16S), 4 Master 16 Slave (4M16S), 8 Master 16 Slave (8M16S), 8 Master 8 Slave (8M8S) và 8 Master 4 Slave (8M4S). Các cấu hình hệ thống đều được khảo sát ở tần số hoạt động 50 MHz và điện áp cung cấp là 3.3V.

Kết quả tổng hợp công suất tiêu thụ tại tần số hoạt động f = 50 Mhz được thể hiện trong Hình 7b. Từ kết quả có được, có thể thấy công suất tiêu thụ của cấu hình 1 Master và 16 Slave là thấp nhất, số lượng Master càng tăng thì công suất tiêu thụ cũng tăng theo. Điều này là do càng nhiều Master sẽ phát sinh 2 vấn đề (1) Master là thành phần xử lý chính và tạo ra các yêu cầu về truy cập bus (2) bộ phân xử ARBITER sẽ cần nhiều tài nguyên hơn cho việc phân quyền các Master. Bên cạnh đó, nghiên cứu này cũng khảo sát sự ảnh hưởng của tần số hoạt động lên công suất tiêu thụ của thiết kế. Bằng cách tăng tần số hoạt động lên từ 50 Mhz đến 150 Mhz, Bảng III thể hiện công suất tiêu thụ trung bình của thiết kế SoC trong nhiều trường hợp hoạt động khác nhau.

Bảng 3. *Công suất tiêu thụ với các tần số hoạt động khác nhau.*

Cấu hình	50 MHz	100 MHz	150 MHz
1 Master 16 Slave	207.9 mW	210.4 mW	213 mW
4 Master 16 Slave	216.3 mW	223.4 mW	233.7 mW
8 Master 16 Slave	223.78 mW	238.53 mW	**X**

Công suất tiêu thụ trung bình của 3 trường hợp thay đổi số lượng Master tăng lên khi tần số hoạt động tăng, có thể thấy rằng tần số ảnh hưởng lớn đến công suất tiêu thụ của thiết kế SoC sử dụng kiến trúc bus Wishbone. Ở trường hợp 8 Master 16 Slave, khi tần số cung cấp là 150 MHz, công suất không thể đo đạt (đánh dấu X) vì giới hạn tần số của cấu hình này là 130.3 MHz như thể hiện trong Hình 7a. Bên cạnh đó, trong mô hình Wishbone Conmax [12] có công suất tiêu thụ là 357mW lớn hơn xấp xỉ 1.6 lần so với mô hình Wishbone được trình bày trong nghiên cứu này. Bởi vì, SoC kết nối bằng Wishbone đang được trình bày sử dụng kiểu kết nối Shared-bus, chỉ có một bộ **ARBITER**, tại một thời điểm chỉ có 1 lõi chủ và 1 lõi tớ được kết nối và trao đổi dữ liệu, trong khi đó Wishbone Conmax sử dụng kiểu kết nối Crossbar, có 16 bộ **ARBITER**, tại một thời điểm có thể có tối đa 8 lõi chủ và 8 lõi tớ thực hiện trao đổi dữ liệu, dẫn đến tần suất hoạt động cao hơn, điện áp cung cấp nhiều hơn và tiêu tốn năng lượng cao hơn. Dựa trên kết quả mô phỏng về tần số hoạt động và công suất tiêu thụ được thể hiện trong Hình 7, khi thực thi mô hình SoC ứng dụng Wishbone với 4 Master thì tần số hoạt động cao nhất xấp xỉ 250 Mhz trong khi công suất tiêu thụ gần tương đương với các cấu hình Master và Slave khác khi xem xét tại tần số hoạt động 50 Mhz.

5. Kết luận

Với sự phát triển của công nghệ vi mạch tích hợp, số lượng lớn các thành phần xử lý có thể được tích hợp trên một vi mạch đơn. Điều này mang lại ưu điểm như giảm giá thành, kích thước thiết kế và công suất tiêu thụ. Kiến trúc Wishbone là một phương pháp liên kết các lõi mang lại hiệu quả cao vì hỗ trợ nhiều dạng kết nối và các giao diện dùng chung cho các lõi làm chuẩn hóa và giúp giảm thiểu được vấn đề về khả năng tích hợp của hệ thống. Nhằm có thể kiểm chứng hiệu quả của kiến trúc Wishbone trong thiết kế SoC, nghiên cứu này đã thực thi, phân tích hoạt động và đánh giá hiệu năng một thiết kế SoC hoàn chỉnh ứng dụng kết nối Wishbone với phương pháp mô phỏng dạng sóng cũng như thực nghiệm trên phần cứng FPGA. Các kết quả mô phỏng và thực thi cho thấy rằng kiến trúc Wishbone có thiết kế đơn giản, yêu cầu một lượng tài nguyên phần cứng ít và phù hợp và có khả năng mở rộng liên kết dành cho các thiết kế đa lõi.

Lời cám ơn

Nghiên cứu này thuộc đề tài năm 2023 được hỗ trợ kinh phí bởi Trường Đại học Sư phạm Kỹ thuật Tp. Hồ Chí Minh.

TÀI LIỆU THAM KHẢO

[1] A. Bharti, "Design, verification and comparison of Wishbone bus for SoC integration," *Lakshmi Narain College of Technology & Science*, Bhopal, 2012.
[2] M. Jovanovic and M. Stojcev, "A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone," *Communication and Energy Systems and Technologies*, Sofia, 2006.
[3] A. Bharti, A. Johari and S. Changlani, "Design of Wishbone Point to Point Architecture and Comparison with Shared Bus," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 4, no. 12, 2015.
[4] F. Abid and N. Izeboudjen, "Low power methodology for wishbone compatible IP cores based SoC design," *Seminar on Detection Systems Architectures and Technologies (DAT)*, Algeria, 2017.
[5] OpenCores, Wishbone SoC Architecture Specification, Revision B.3, 2002.
[6] OpenCores, Wishbone SoC Architecture Specification, Revision B.4, 2010.
[7] A. K. Swain and K. Mahapatra, "Design and verification of WISHBONE bus interface for System-on-Chip integration," *Annual IEEE India Conference (INDICON)*, India, 2010.
[8] E. S. Shin, V. J. Mooney and G. F. Riley, "Round-robin Arbiter Design and Generation," *15th International Symposium on System Synthesis, Japan*, pp. 243-248, doi: 10.1145/581199.581253, 2002.
[9] M. Weber, "Arbiters: Design Ideas and Coding Styles," *Synopsys User Group Conference*, Boston, 2001.
[10] Xilinx, ISE In-Depth Tutorial, 2011.
[11] Xilinx, Xilinx Power Tools Tutorial, 2013.
[12] C. Dongye, "Design of the On-chip Bus Based on Wishbone", *Electronics*, Communications and Control (ICECC), 2011.

TS. Phạm Văn Khoa
- Tốt nghiệp Đại học chuyên ngành Kỹ thuật máy tính và Thạc sĩ Kỹ thuật điện tử năm 2010 và 2014 tại đại học Sư Phạm Kỹ Thuật Tp.HCM, tiến sĩ Kỹ thuật điện tử năm 2019 tại Đại học Kookmin, Seoul, Hàn Quốc.
- Từ năm 2011 đến nay, giảng viên chính giảng dạy tại Bộ môn Kỹ thuật máy tính-viễn thông, Khoa Điện-Điện Tử. Từ năm 2022, trưởng ngành Máy Tính-Viễn Thông, Khoa Đào Tạo Quốc Tế, Trường Đại học Sư phạm kỹ thuật, tp Hồ chí minh.
- Lĩnh vực quan tâm: Thiết kế bộ nhớ; xử lý trên bộ nhớ (processing in memory); Neuromorphic; ứng dụng trí tuệ nhân tạo
https://orcid.org/0000-0002-6129-5856
https://scholar.google.co.kr/citations?user=t_abZ6kAAAAJ&hl=en
- Điện thoại: 0918-004-457

ThS. Đặng Phước Hải Trang
- Tốt nghiệp Đại học chuyên ngành Kỹ thuật Điện – Điện tử năm 2007 và cao học Kỹ thuật Điện tử năm 2011 tại trường Đại học Sư phạm Kỹ thuật Tp. Hồ Chí Minh.
- Từ năm 2007 đến nay là giảng viên tại Bộ môn Kỹ thuật Máy tính – Viễn thông, khoa Điện – Điện tử, trường Đại học Sư phạm Kỹ thuật Tp. Hồ Chí Minh.
- Lĩnh vực quan tâm: Xử lý tín hiệu, Internet of things, Thiết kế Vi mạch
- Email: trangdph@hcmute.edu.vn
- Điện thoại: 0909-913-376

TS. Ngô Đình Thanh
- Tốt nghiệp Tiến sĩ năm: 2015 chuyên ngành Hệ thống nhúng tại Đại học South Brittany
- Từ 2004 đến nay là giảng viên tại bộ môn Tự động hóa - Khoa Điện - Trường Đại học Bách Khoa - Đại học Đà Nẵng.
- Lĩnh vực quan tâm: Tự động hóa, Học sâu
- Email: tndthanh@dut.udn.vn
- Điện thoại: 0914-486-786

KS. Nguyễn Quốc Khải
- Tốt nghiệp chương trình ngành Công nghệ Kỹ thuật Điện tử Truyền thông, chuyên ngành Viễn thông - Vi mạch tại trường ĐH Sư Phạm Kỹ Thuật TP. HCM năm 2022.
- Kỹ sư Physical Implement, Công ty TNHH ADT & SNST Vietnam
- Lĩnh vực quan tâm: Thiết kế vi mạch số
- Email: nguyenkhai3339@gmail.com
- Điện thoại: 0373334971

KS. Nguyễn Thành Huy
- Tốt nghiệp chương trình ngành Công nghệ Kỹ thuật Điện tử Truyền thông, chuyên ngành Viễn thông - Vi mạch tại trường ĐH Sư Phạm Kỹ Thuật TP. HCM năm 2022.
- Kỹ sư Memory Layout Design, Công ty Faraday Technology Viet Nam
- Lĩnh vực quan tâm: Thiết kế vi mạch số
- Email: nguyenhuy29291@gmail.com
- Điện thoại: 0389445838