|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  Ảnh có chứa văn bản, ký hiệu, thực phẩm  Mô tả được tạo tự động  **BÁO CÁO VLSI**  **Đề tài: Implementation of Universal Asynchronous Receiver Transmitter using System Verilog code**  **Nhóm sinh viên thực hiện:**   |  |  |  | | --- | --- | --- | | **Tên sinh viên** | **MSSV** | **Mã lớp** | | Phạm Ngọc Lâm | 20182628 | 129281 | | Phạm Minh Đức | 20172476 | 129281 |   **Giảng viên hướng dẫn:** TS. Phan Xuân Vũ  Hà Nội - 2022 |

MỤC LỤC

[MỤC LỤC 1](#_Toc92899598)

[DANH MỤC HÌNH ẢNH i](#_Toc92899599)

[DANH MỤC BẢNG iii](#_Toc92899600)

[LỜI NÓI ĐẦU iv](#_Toc92899601)

[CHƯƠNG 1. GIỚI THIỆU (INTRODUCTION) 1](#_Toc92899602)

[1.1 Giới thiệu chung 1](#_Toc92899603)

[1.2 Pipeline 1](#_Toc92899604)

[CHƯƠNG 2. ĐẶC TẢ THÔNG SỐ KĨ THUẬT (SPECIFICATION) 2](#_Toc92899605)

[2.1 RISC-V pipeline architecture 2](#_Toc92899606)

[2.1.1 Architecture 2](#_Toc92899607)

[2.1.2 Interface signals 2](#_Toc92899608)

[2.2 Module Intruction Fetch (instruction\_fetch) 2](#_Toc92899609)

[2.2.1 Interface signals 3](#_Toc92899610)

[2.2.2 Function description 3](#_Toc92899611)

[2.2.3 Instruction fetch architecture 4](#_Toc92899612)

[2.3 Module Instruction Decode (instruction\_decode) 5](#_Toc92899613)

[2.3.1 Interface signals 5](#_Toc92899614)

[2.3.2 Function description 6](#_Toc92899615)

[2.3.3 Instruction Decode architecture 7](#_Toc92899616)

[2.4 Module Excute (execute) 8](#_Toc92899617)

[2.4.1 Interface signals 9](#_Toc92899618)

[2.4.2 Function description 10](#_Toc92899619)

[2.4.3 Execute architecture 10](#_Toc92899620)

[2.5 Module Memory Access (memory\_access) 11](#_Toc92899621)

[2.5.1 Interface signals 11](#_Toc92899622)

[2.5.2 Function description 12](#_Toc92899623)

[2.5.3 Memory access architecture 12](#_Toc92899624)

[2.6 Module Register Write (register\_write) 13](#_Toc92899625)

[2.6.1 Interface signals 13](#_Toc92899626)

[2.6.2 Function description 13](#_Toc92899627)

[2.6.3 Register Write architecture 13](#_Toc92899628)

[2.7 Module Control (control) 14](#_Toc92899629)

[2.7.1 Interface signals 14](#_Toc92899630)

[2.7.2 Function description 14](#_Toc92899631)

[2.8 Module Hazard Detection Unit (hazard\_detection\_unit) 15](#_Toc92899632)

[2.8.1 Interface signals 15](#_Toc92899633)

[2.8.2 Function description 15](#_Toc92899634)

[2.9 Module Forwarding Unit (forwarding\_unit) 15](#_Toc92899635)

[2.9.1 Interface signals 16](#_Toc92899636)

[2.9.2 Function description 16](#_Toc92899637)

[2.10 Module Top (riscv\_pipeline\_top) 18](#_Toc92899638)

[2.10.1 Interface signals 18](#_Toc92899639)

[2.10.2 Function description 19](#_Toc92899640)

[CHƯƠNG 3. KIỂM THỬ (VERIFICATION) 20](#_Toc92899641)

[3.1 Kế hoạch kiểm thử 20](#_Toc92899642)

[3.1.1 Kịch bản – Test case 20](#_Toc92899643)

[3.1.2 Kích thích đầu vào – Stimulus 20](#_Toc92899644)

[3.2 Kết quả và đánh giá 21](#_Toc92899645)

[3.2.1 Module instruction\_fetch 21](#_Toc92899646)

[3.2.2 Module instruction\_decode 21](#_Toc92899647)

[3.2.3 Module execute 22](#_Toc92899648)

[3.2.4 Module memory\_access 22](#_Toc92899649)

[3.2.5 Module register\_write 22](#_Toc92899650)

[3.2.6 Module control 23](#_Toc92899651)

[3.2.7 Module forwarding\_unit 23](#_Toc92899652)

[3.2.8 Module hazard\_detection\_unit 23](#_Toc92899653)

[3.2.9 Register 24](#_Toc92899654)

[3.2.10 Data\_Memory 24](#_Toc92899655)

[CHƯƠNG 4. KẾT LUẬN 26](#_Toc92899656)

[TÀI LIỆU THAM KHẢO 27](#_Toc92899657)

DANH MỤC HÌNH ẢNH

[Hình 2.1 riscv\_pipeline architecture 2](#_Toc92899573)

[Hình 2.2 instruction\_fetch block diagram 2](#_Toc92899574)

[Hình 2.3 Instruction fetch architecture 4](#_Toc92899575)

[Hình 2.4 instruction\_decode block diagram 5](#_Toc92899576)

[Hình 2.5 Instruction Decode architecture 7](#_Toc92899577)

[Hình 2.6 execute block diagram 8](#_Toc92899578)

[Hình 2.7 Execute architecture 10](#_Toc92899579)

[Hình 2.8 memory\_access block diagram 11](#_Toc92899580)

[Hình 2.9 Memory access architecture 12](#_Toc92899581)

[Hình 2.10 register\_write block diagram 13](#_Toc92899582)

[Hình 2.11 Register write architecture 13](#_Toc92899583)

[Hình 2.12 control block diagram 14](#_Toc92899584)

[Hình 2.13 hazard\_detection\_unit block diagram 15](#_Toc92899585)

[Hình 2.14 forwarding\_unit block diagram 15](#_Toc92899586)

[Hình 2.15 riscv\_pipeline\_top block diagram 18](#_Toc92899587)

[Hình 3.1 Mô phỏng timing diagram khối instruction\_fetch 21](#_Toc92899588)

[Hình 3.2 Mô phỏng timing diagram khối instruction\_decode 21](#_Toc92899589)

[Hình 3.3 Mô phỏng timing diagram khối execute 22](#_Toc92899590)

[Hình 3.4 Mô phỏng timing diagram khối memory\_access 22](#_Toc92899591)

[Hình 3.5 Mô phỏng timing diagram khối register\_write 22](#_Toc92899592)

[Hình 3.6 Mô phỏng timing diagram khối control 23](#_Toc92899593)

[Hình 3.7 Mô phỏng timing diagram khối forwarding\_unit 23](#_Toc92899594)

[Hình 3.8 Mô phỏng timing diagram khối hazard\_detection\_unit 23](#_Toc92899595)

[Hình 3.9 Mô phỏng 32 Registers 24](#_Toc92899596)

[Hình 3.10 Mô phỏng Data\_Memory 24](#_Toc92899597)

DANH MỤC BẢNG

[Bảng 2.1 riscv\_pipeline port description 2](#_Toc92899563)

[Bảng 2.2 instruction\_decode port description 3](#_Toc92899564)

[Bảng 2.3 instruction\_decode port description 5](#_Toc92899565)

[Bảng 2.4 execute port description 9](#_Toc92899566)

[Bảng 2.5 memory\_access port description 11](#_Toc92899567)

[Bảng 2.6 register\_write port description 13](#_Toc92899568)

[Bảng 2.7 control port description 14](#_Toc92899569)

[Bảng 2.8 forwarding\_unit port description 16](#_Toc92899570)

[Bảng 2.9 Forwarding output ports 17](#_Toc92899571)

[Bảng 2.10 riscv\_pipeline\_top port description 18](#_Toc92899572)

LỜI NÓI ĐẦU

Mã nguồn mở đang dần trở thành một phần quan trọng của thế giới IT khi mà nó góp mặt trong khoảng 96% phần mềm thương mại. Tương tự với phần cứng, vi xử lý mã nguồn mở RISC-V đang dần được quan tâm và hứa hẹn sẽ mang tới thay đổi lớn về bối cảnh của ngành điện toán. Chương trình học môn Kiến trúc máy tính (ET4041), RISC-V Processor Design là một phần quan trọng giúp hiểu rõ về quá trình hoạt động khi thực hiện các lệnh. Trong báo cáo này, chúng em triển khai kiến trúc RISC-V áp dụng kĩ thuật pipeline nhằm tăng tốc độ xử lí, bên cạnh đó, thiết kế đã xữ lí được toàn bộ các hazard có thể xảy ra trong quá trình xử lí thực hiện lệnh của CPU bao gồm data hazard, mem hazard và control hazard. Thiết kế được tiến hành triển khai bằng ngôn ngữ mô tả phần cứng SystemVerilog và mô phỏng kiểm thử trên phần mềm ModelSim. Cho ra kết quả hoạt động đúng với yêu cầu vào ra. Chúng em sẽ đi trình bày cụ thể những gì chúng em đã làm được thông qua 4 chương sau:

**Chương 1: Giới thiệu**

**Chương 2: Đặc tả thông số kĩ thuật**

**Chương 3: Kiểm thử**

**Chương 4: Kết luận**

Nhóm chúng em xin chân thành cảm ơn PGS.TS Nguyễn Đức Minh đã tận tâm hướng dẫn chúng em trong quá trình thực hiện đồ án cũng như hoàn hiện báo cáo này !

# GIỚI THIỆU (INTRODUCTION)

*Chương này giới thiệu khái quát và các kiến thức cơ bản về UART.*

## Giới thiệu chung

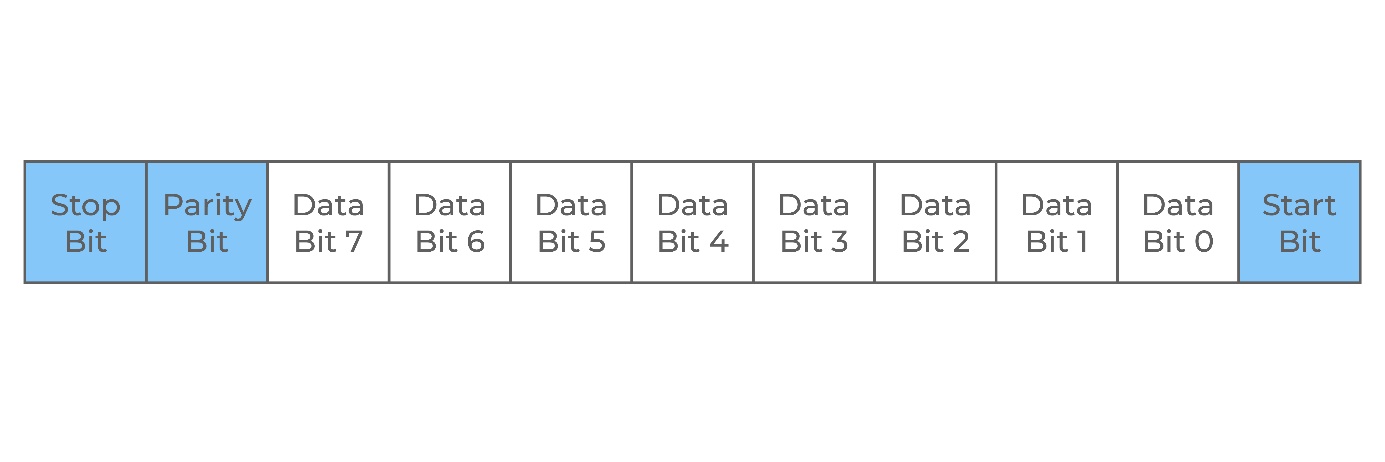
Universal asynchronous receiver – transmitter (UART) là một thiết bị cho truyền thông nối tiếp không đồng bộ, trong đó định dạng dữ liệu và tốc độ truyền tải được cấu hình. Nó gửi từng bit dữ liệu một, từ LSB đến MSB, được gói trong một khung truyền có các bit start và stop.

Nó là một trong những thiết bị giao tiếp máy tính sớm nhất, được sử dụng để gắn máy viết chữ từ xa cho một bảng điều khiển. Nó cũng là một hệ thống phần cứng ban đầu cho Internet.

UART thường là một mạch tích hợp (IC) được sử dụng cho giao tiếp nối tiếp qua máy tính hoặc cổng nối tiếp thiết bị ngoại vi.

## Truyền thông UART

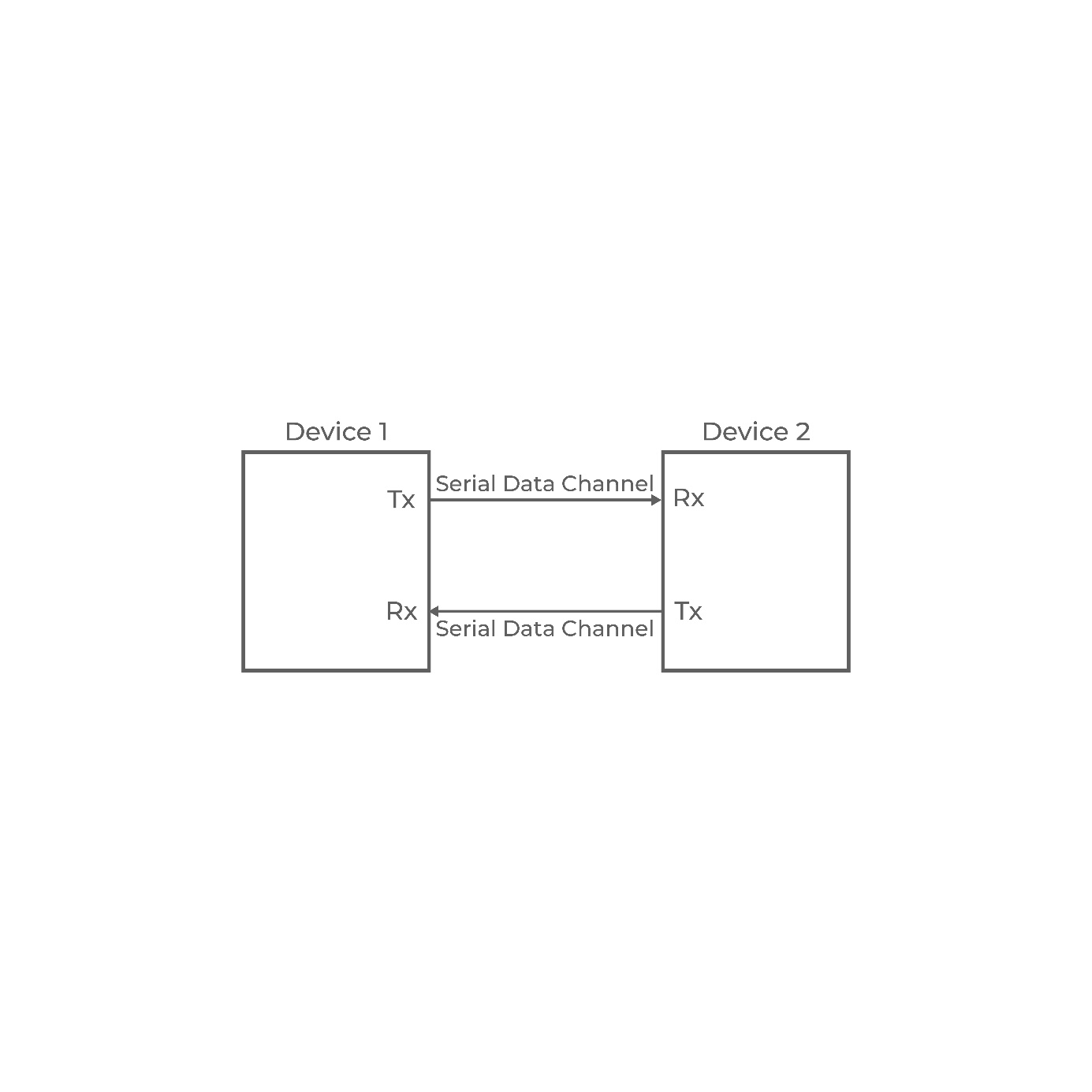
### Cấu trúc của Data framing



Hình .. Data Framing UART

* Start Bit: Bit đồng bộ hóa được đặt trước dữ liệu. Để bắt đầu truyền dữ liệu, UART kéo đường dữ liệu từ mức điện áp cao (Mức 1) xuống mức điện áp thấp (Mức 0). Chỉ có một Start Bit.
* Stop Bit: Bit dừng được đặt ở cuối của gói dữ liệu. Thường chỉ sử dụng 1 bit. Đề dừng truyền dữ liệu, UART giữ đường dữ liệu ở mức điện áp cao.
* Parity Bit: Bit chẵn lẻ cho phép người nhận đảm bảo dữ liệu được thu thập có đúng hay không. Bit này không được sử dụng rộng rãi nên không bắt buộc.

### UART Communication



Hình .. UART Communication

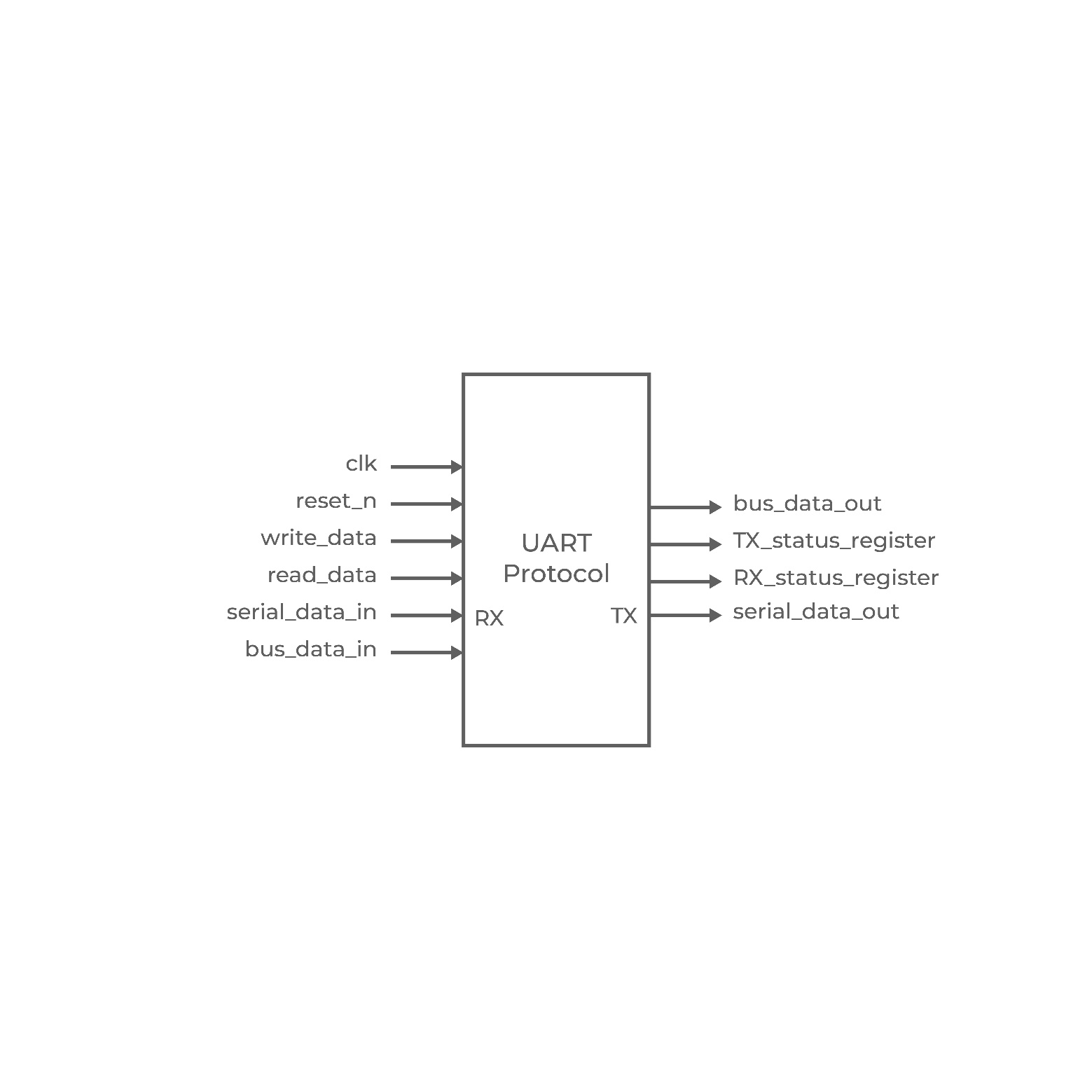
Trong giao tiếp dữ liệu nối tiếp, dữ liệu có thể được truyền qua một cáp hoặc một đường dây ở dạng bit-bit và nó chỉ cần hai cáp.

# ĐẶC TẢ THÔNG SỐ KĨ THUẬT (SPECIFICATION)

*Chương này mô tả thông số kĩ thuật và kiến trúc chi tiết của từng phần có trong kiến trúc UART được nhóm triển khai.*

## UART Architecture

### Block diagram



Hình . UART Block diagram

Hình 2.1 mô tả các Input và Output của module UART protocol.

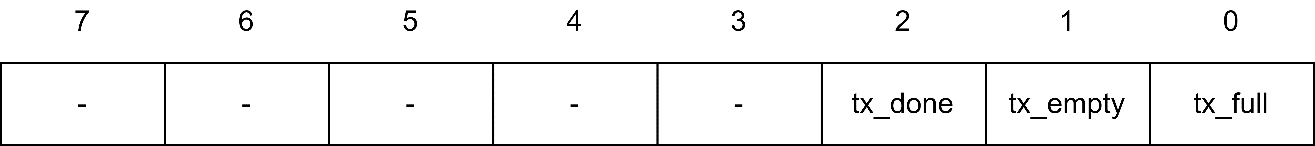
### Interface signals

Bảng . uart\_protocol port description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **write\_data** | 1 | Input | Tín hiệu ghi vào FIFO từ CPU |
| **read\_data** | 1 | Input | Tín hiệu lấy dữ liệu trong FIFO từ CPU |
| **serial\_data\_in** | 1 | Input | Dữ liệu vào nối tiếp |
| **bus\_data\_in** | 8 | Input | Bus dữ liệu phía RX |
| **bus\_data\_out** | 8 | Output | Bus dữ liệu phía TX |
| **TX\_status\_register** | 8 | Output | Thanh ghi trạng thái TX |
| **RX\_status\_register** | 8 | Output | Thanh ghi trạng thái RX |
| **serial\_data\_out** | 1 | Ouput | Dữ liệu ra nối tiếp |

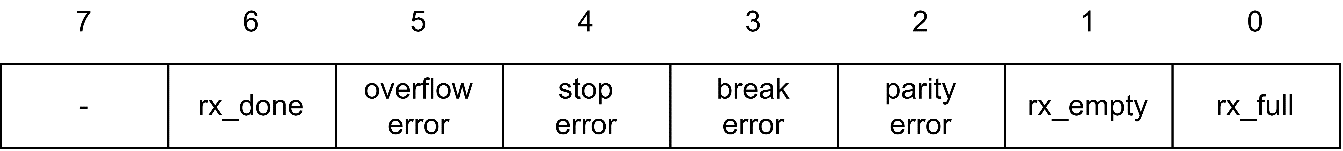
Thanh ghi trạng thái được biểu diễn như sau:

* TX\_status\_register:



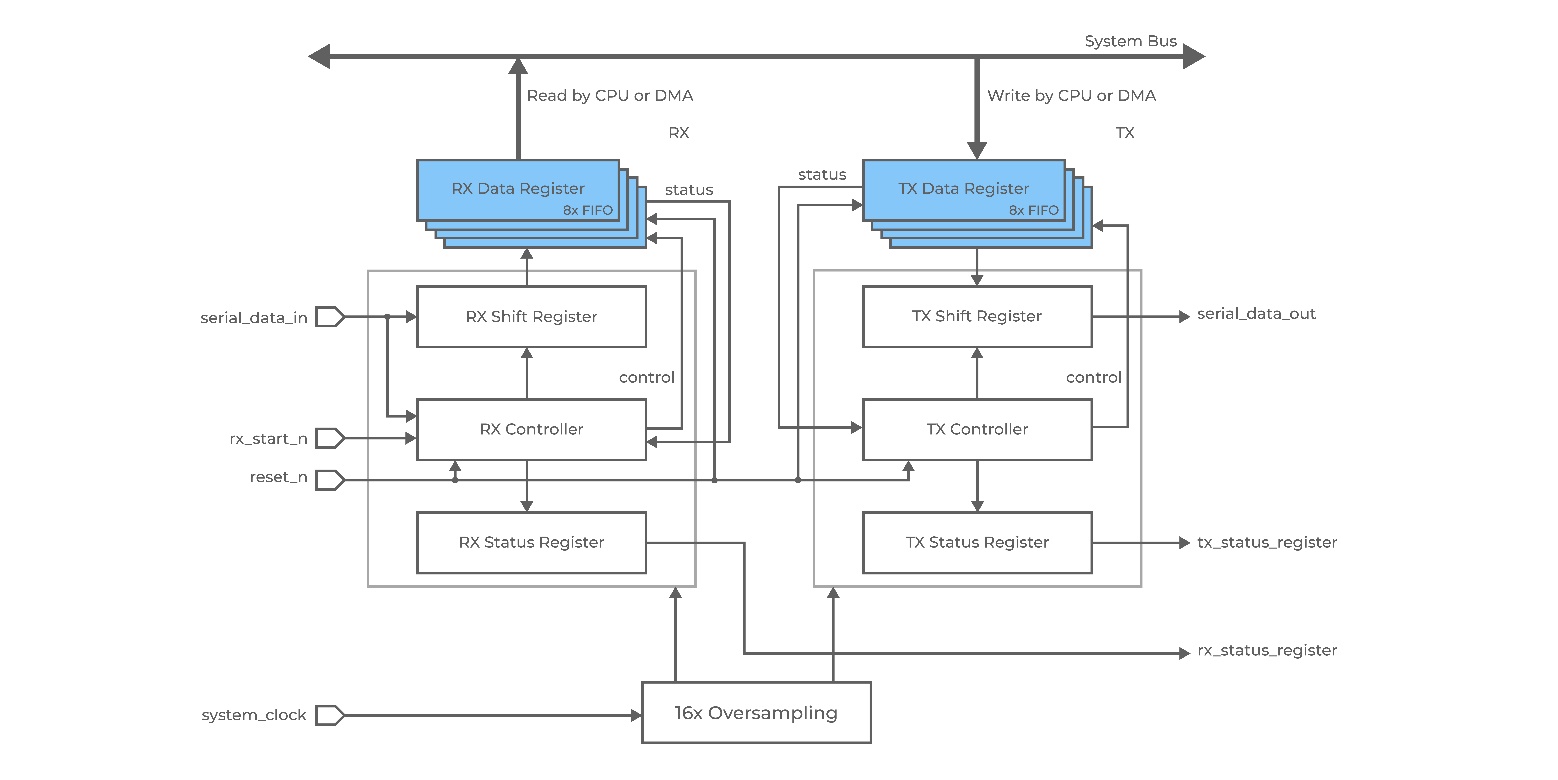
Hình .. TX\_status\_register

* RX\_status\_register

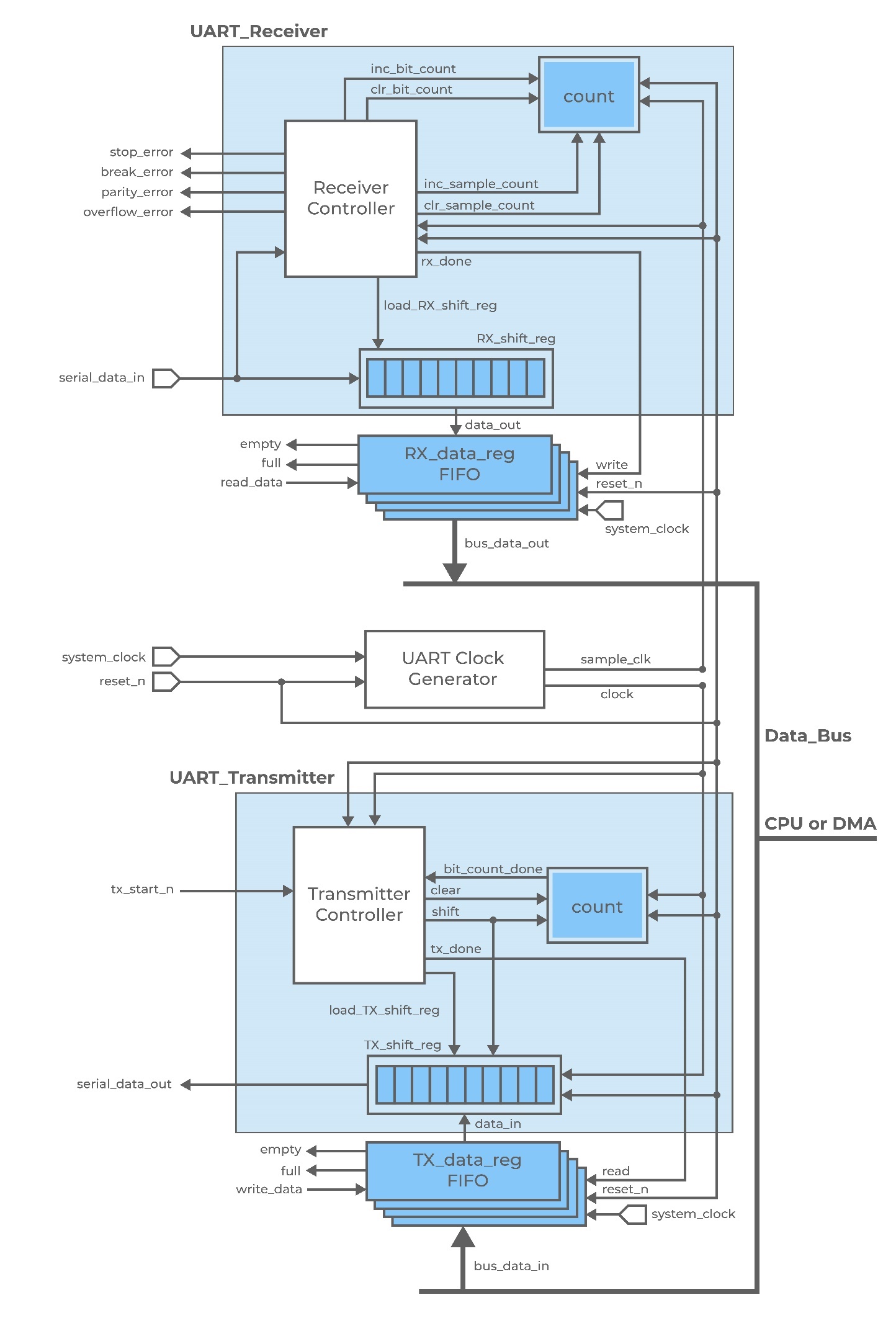


Hình .. RX\_status\_register

### Architecture

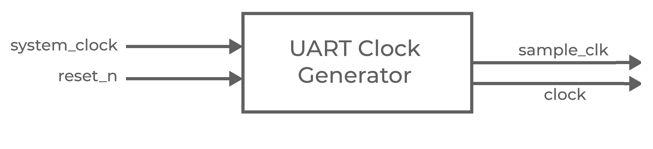


Hình .. Sơ đồ kiến trúc tổng quát



Hình .. Sơ đồ kiến trúc chi tiết

## Module Clock Generator (uart\_generator\_clock)



Hình .. uart\_generator\_clock block diagram

### Interface signals

Bảng .. uart\_generator\_clock port description

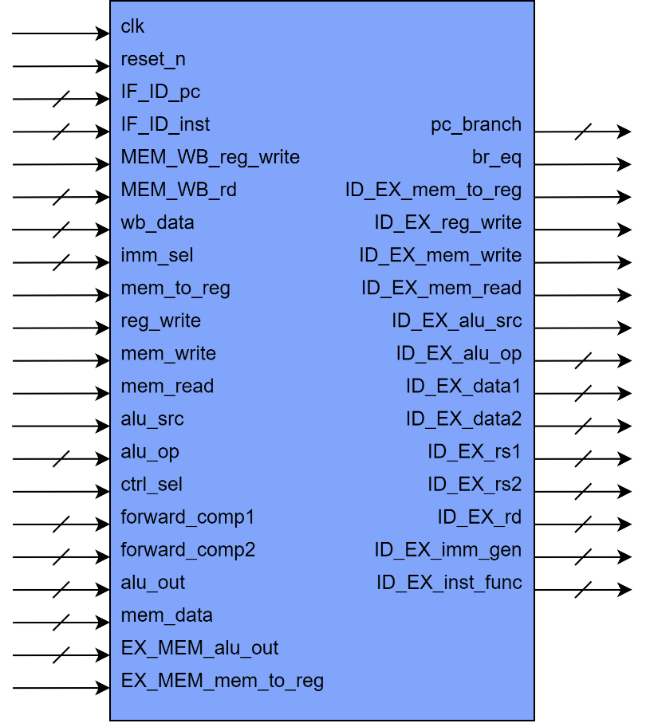
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **clock** | 1 | Output | Clock f = BAUD\_RATE cho Transmitter |
| **sample\_clk** | 1 | Output | Clock f = 16\*BAUD\_RATE cho Receiver |

Module Clock Generator chia tần để giảm tần số 100MHz về Baud Rate chuẩn của UART

Bảng .. uart\_generator\_clock parameter

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| **SYS\_FREQ** | 100000000 | Tần số hoạt động của System Clock |
| **BAUD\_RATE** | 9600 | Tốc độ Baud trên kênh truyền |
| **CLOCK** | SYS\_FREQ/BAUD\_RATE | Độ chia clock |
| **SAMPLE** | 16 | Lấy mẫu x16 |
| **BAUD\_DVSR** | SYS\_FREQ/(SAMPLE\*BAUD\_RATE) | Độ chia tần số lấy mẫu |

## Module FIFO (uart\_fifo)



Hình . instruction\_decode block diagram

### Interface signals

Bảng . instruction\_decode port description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **IF\_ID\_pc** | 32 | Input | Giá trị PC lưu trong thanh ghi IF\_ID |
| **IF\_ID\_inst** | 32 | Input | Mã lệnh lưu trong thanh ghi IF\_ID |
| **MEM\_WB\_reg\_write** | 1 | Input | Tín hiệu cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_rd** | 5 | Input | Địa chỉ thanh ghi đích lưu trong thanh ghi MEM/WB |
| **wb\_data** | 32 | Input | Dữ liệu được write back |
| **imm\_sel** | 3 | Input | Chọn kiểu cho Immidiate Generate |
| **mem\_to\_reg** | 1 | Input | Tín hiệu cho phép write back |
| **reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi |
| **mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu |
| **mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu |
| **alu\_src** | 1 | Input | Chọn chế độ địa chỉ trực tiếp |
| **alu\_op** | 2 | Input | Opcode chọn phép toán |
| **ctrl\_sel** | 1 | Input | Chọn đầu ra control |
| **forward\_comp1** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs1 |
| **forward\_comp2** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs2 |
| **alu\_out** | 32 | Input | Kết quả alu được forward về |
| **mem\_data** | 32 | Input | Dữ liệu memory được forward về |
| **EX\_MEM\_alu\_out** | 32 | Input | Kết quả của alu trong thanh ghi EX/MEM được forward về |
| **EX\_MEM\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **pc\_branch** | 32 | Output | Giá trị pc cần nhảy đến |
| **br\_eq** | 1 | Output | Kết quả so sánh data1 và data2 |
| **ID\_EX\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi ID/EX |
| **ID\_EX\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_src** | 1 | Output | Chọn chế độ địa chỉ trực tiếp lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_op** | 2 | Output | Opcode chọn phép toán lưu trong thanh ghi ID/EX |
| **ID\_EX\_data1** | 32 | Output | Dữ liệu từ thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_data2** | 32 | Output | Dữ liệu từ thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs1** | 5 | Output | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Output | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Output | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **ID\_EX\_imm\_gen** | 32 | Output | Kết quả immidiate lưu trong thanh ghi ID/EX |
| **ID\_EX\_inst\_func** | 4 | Output | Function chọn phép toán lưu trong thanh ghi ID/EX |

### Function description

Khối Instruction Decode lấy đầu vào là instruction đã được nạp vào thanh ghi IF/ID và thực hiện:

* Đọc dữ liệu từ tệp thanh ghi dựa vào địa chỉ rs1, rs2, rd lấy từ instruction
* Gen ra 32bit immediate từ trường imm trong instruction
* Gửi trường opcode của instruction sang khối control để đưa ra các tín hiệu điều khiển đến các khối khác
* So sánh giá trị data của của 2 thanh ghi rs1, rs2 trong lệnh branch và gửi kết quả so sanh sang khối control. Nhận tín hiệu forward từ khối forwarding unit để quyết định xem sẽ so sánh giá trị thanh ghi hay giá trị được forward về trong trường hợp có hazard

### Instruction Decode architecture

Diagram, schematic

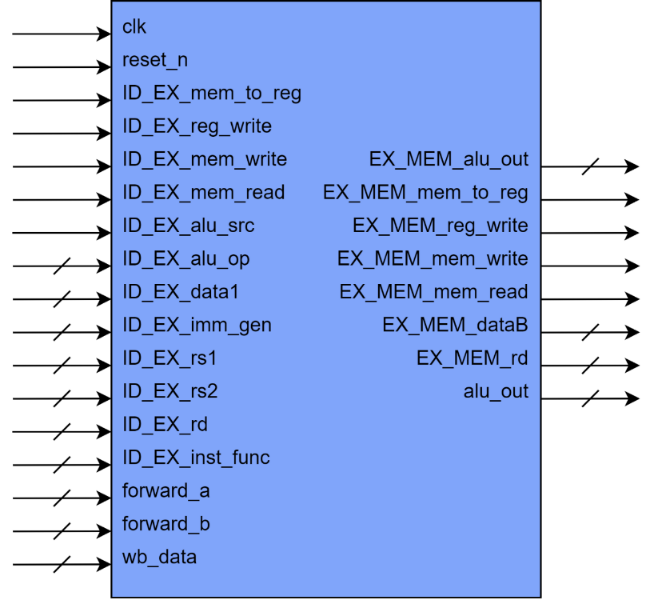
Description automatically generated

Hình . Instruction Decode architecture

Hình 2.5 mô tả kiến trúc của khối **Instruction decode** gồm:

* Tệp thanh ghi nhận các giá trị địa chỉ thanh ghi **rs1**, **rs2** từ **inst** để đưa ra giá trị **data1, data2** tương ứng cho 2 thanh ghi. Nhận địa chỉ **rd**, **wb\_data** từ khối **Register write** (thực hiện giai đoạn write back) và tín hiệu **reg\_write** từ **control** để ghi kết quả tính toán vào thanh ghi **rd**
* Bộ cộng nhằm tính toán địa chỉ **pc** cho lệnh brach
* Imm Gen nhằm signed extend cho giá trị **immediate**
* 2 bộ **mux4to1** nhận tín hiệu điều khiển từ khối **Forwarding unit** nhằm forward các kết quả tính toán của giai đoạn EX và MEM về bộ so sánh (branch compare) nhằm khắc phục data hazard khi thực hiện lệnh branch
* Branch comp là bộ so sánh so sánh giá trị của 2 thanh ghi **rs1**, **rs2** từ lệnh branch và đưa ra kết quả đến khối **control** để xác định xem chương trình sẽ nhảy hay sẽ thực hiện lệnh tiếp theo có địa chỉ **pc + 4**
* Thanh ghi **ID/EX** sẽ nhận kết quả từ khối **Instruction decode** đồng thời nhận các tín hiệu điều khiển được đưa ra từ khối **control** để gửi đến các khối **Execute, Memory access** và **Register write** để thực hiện tính toán cho các giai đoạn pipeline sau của lệnh

## Module Excute (execute)



Hình . execute block diagram

### Interface signals

Bảng . execute port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **ID\_EX\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi ID/EX |
| **ID\_EX\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_src** | 1 | Input | Chọn chế độ địa chỉ trực tiếp lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_op** | 2 | Input | Opcode chọn phép toán lưu trong thanh ghi ID/EX |
| **ID\_EX\_data1** | 32 | Input | Dữ liệu từ thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_data2** | 32 | Input | Dữ liệu từ thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_imm\_gen** | 32 | Input | Kết quả immidiate lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **ID\_EX\_inst\_func** | 4 | Input | Function chọn phép toán lưu trong thanh ghi ID/EX |
| **forward\_a** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs1 |
| **forward\_b** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs2 |
| **wb\_data** | 32 | Input | Dữ liệu được write back |
| **EX\_MEM\_alu\_out** | 32 | Output | Kết quả alu được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_dataB** | 32 | Output | Dữ liệu thanh ghi 2 được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_rd** | 5 | Output | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **alu\_out** | 32 | Output | Kết quả alu |

### Function description

Khối Execute sẽ nhận các tín hiệu điều khiển cho giai đoạn EX đã được lưu trong thanh ghi ID/EX và thực hiện tính toán dữ liệu hoặc địa chỉ tương ứng với yêu cầu của các instruction bằng khối ALU và ALU control.

### Execute architecture

Diagram

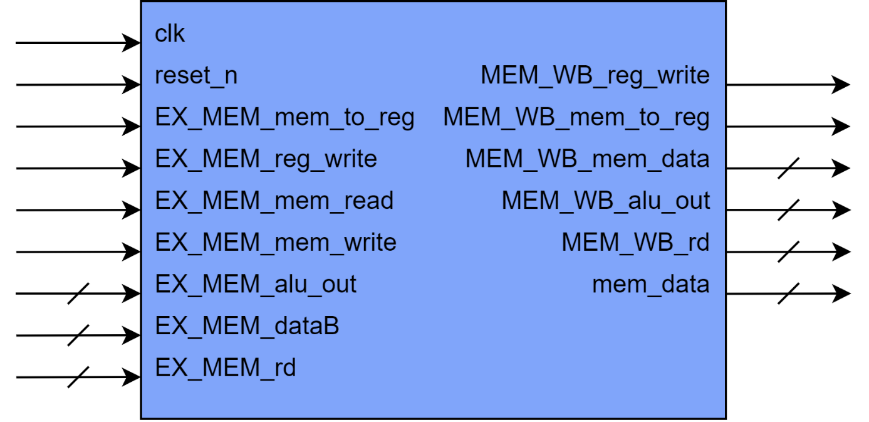
Description automatically generated

Hình . Execute architecture

Hình 2.7 mô tả kiến trúc của khối **Execute** gồm:

* 2 mộ **mux4to1** nhận tín hiệu từ khối **Forwarding unit** nhằm forward đầu vào cho khối **ALU** trong trường hợp hazard
* **Mux2to1** nhận tín hiệu **alu\_src** từ khối **control** để chọn giữa dữ liệu thanh ghi **rs2** hoặc giá trị đầu ra của khối **Imm Gen** đã được lưu trong thanh ghi **ID/EX**
* **ALU control** nhận tín hiệu **alu\_op** từ khối **control** và đưa ra tín hiệu **alu­\_ctrl** đến khối **ALU** để điều khiển hoạt động của **ALU** theo bảng sự thật Bảng 2.9
* **ALU** sẽ dựa vào tín hiệu **alu­\_ctrl** nhận được từ khối **ALU control** để thực hiện các phép toán tương ứng (add, sub, AND, OR, XOR,…)
* **ID/EX** là thanh ghi pipeline nhận các kết quả tính toán từ **ALU, dataB** và các giá trị của tín hiệu điều khiển để phục vụ cho các giai đoạn pipeline sau của lệnh. Bên cạnh đó còn nhận giá trị địa chỉ thanh ghi **rd** phục vụ cho việc nhận diện các trường hợp có hazard

## Module Memory Access (memory\_access)



Hình . memory\_access block diagram

### Interface signals

Bảng . memory\_access port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **EX\_MEM\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_alu\_out** | 32 | Input | Kết quả alu được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_dataB** | 32 | Input | Dữ liệu thanh ghi 2 được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **MEM\_WB\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi MEM/WB |
| **MEM\_WB\_mem\_data** | 32 | Output | Dữ liệu từ memory được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_alu\_out** | 32 | Output | Kết quả alu được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_rd** | 5 | Output | Địa chỉ thanh ghi đích được lưu trong thanh ghi MEM/WB |
| **mem\_data** | 32 | Output | Dữ liệu từ memory |

### Function description

Khối Memory access nhận các tín hiệu điều khiển đọc ghi cho giai đoạn MEM từ thanh ghi EX/MEM để đọc hoặc ghi dữ liệu từ Data memory.

### Memory access architecture

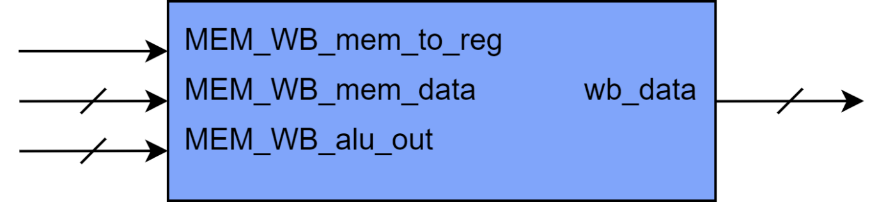
Diagram

Description automatically generated

Hình . Memory access architecture

Hình 2.9 mô tả kiến trúc khối **Memory access**, gồm một bộ nhớ **Data Memory** nhận tín hiệu điều khiển **mem\_read**, **mem\_write** điều khiển việc đọc ghi memory. **Data Memory** nhận địa chỉ đã được tính toán từ **ALU** trong giai đoạn pipeline trước đó được lưu vào thanh ghi **EX/MEM** và đưa ra data tại địa chỉ đó nếu có tín hiệu đọc, hoặc sẽ ghi data được chuyển từ **rs2** (**DataB**) sang thanh ghi **EX/MEM** khi có tín hiệu ghi. Thanh ghi **MEM/WB** lưu lại dữ liệu được đọc ra từ memory và kết quả tính toán từ **ALU** được lấy ra ở thanh ghi **EX/MEM** để làm dữ liệu thực hiện giai đoạn pipeline tiếp theo là **write back**.

## Module Register Write (register\_write)



Hình . register\_write block diagram

### Interface signals

Bảng . register\_write port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **MEM\_WB\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi MEM/WB |
| **MEM\_WB\_mem\_data** | 32 | Input | Dữ liệu từ memory được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_alu\_out** | 32 | Input | Kết quả alu được lưu trong thanh ghi MEM/WB |
| **wb\_data** | 32 | Output | Dữ liệu write back |

### Function description

Khối Register Write thực hiện việc quyết định xem ghi kết quả vào thanh ghi hay không dựa vào tín hiệu **reg\_write** nhận từ MEM/WB và quyết định xem ghi dữ liệu từ Data memory về thanh ghi hay giá trị được tính toán từ ALU chuyển sang.

### Register Write architecture

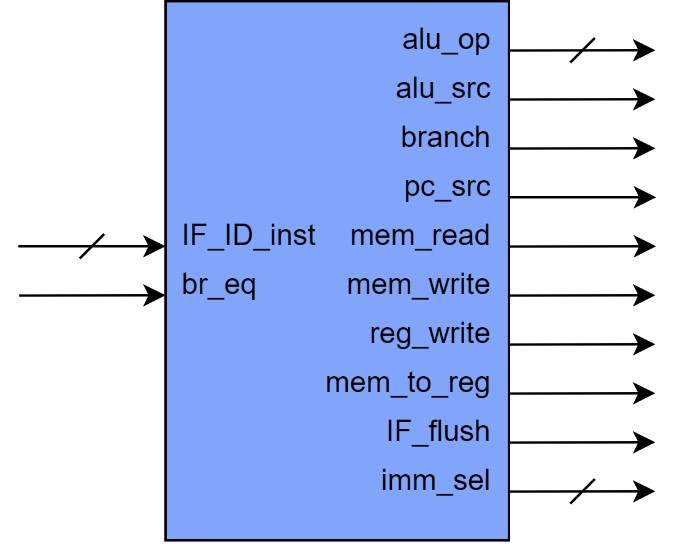
Diagram, schematic

Description automatically generated

Hình . Register write architecture

Hình 2.11 mô mô tả kiến trúc khối **Register Write** thực hiện giai đoạn pipeline write back. Bộ **mux2to1** nhận tín hiệu điều khiển **mem\_to\_reg** để thực hiện việc quyết định giữa đưa giá trị dữ liệu từ **Data memory** về thanh ghi hay kết quả tính toán của **ALU**.

## Module Control (control)



Hình . control block diagram

### Interface signals

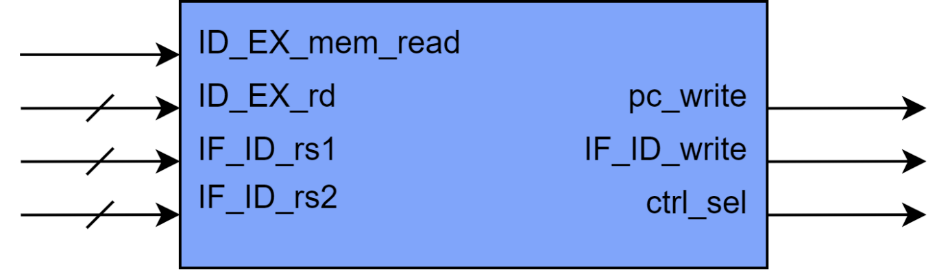
Bảng . control port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **IF\_ID\_inst** | 32 | Input | Mã lệnh được lưu trong thanh ghi IF/ID |
| **br\_eq** | 1 | Input | Kết quả so sánh data1 và data2 |
| **alu\_op** | 2 | Output | Opcode chọn phép toán |
| **alu\_src** | 1 | Output | Chọn chế độ địa chỉ trực tiếp |
| **branch** | 1 | Output | Cho phép nhảy đến địa chỉ offset |
| **pc\_src** | 1 | Output | Cho phép chọn PC+4 hoặc PC\_branch |
| **mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu |
| **mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu |
| **reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi |
| **mem\_to\_reg** | 1 | Output | Tín hiệu cho phép write back |
| **IF\_flush** | 1 | Output | Cho phép xóa thanh ghi IF\_ID |
| **imm\_sel** | 3 | Output | Chọn kiểu cho Immidiate Generate |

### Function description

Khối **control** nhận 32 bit instruction làm đầu vào và dựa vào phần opcode của instruction rồi đưa ra các tín hiệu điều khiển tương ứng theo bảng sự thật Bảng 2.9

## Module Hazard Detection Unit (hazard\_detection\_unit)



Hình . hazard\_detection\_unit block diagram

### Interface signals

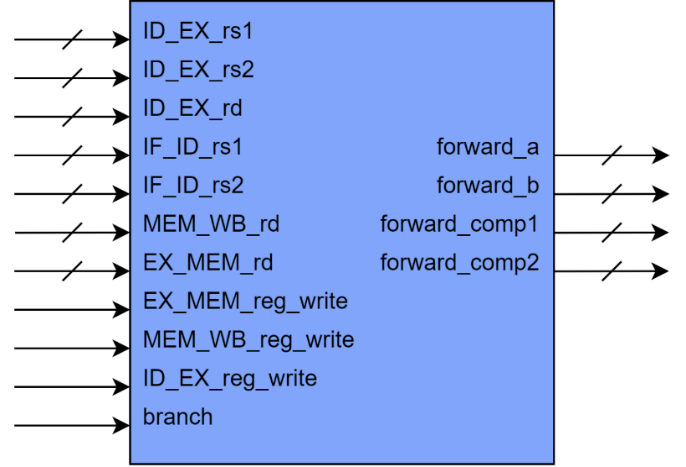
| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **ID\_EX\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **pc\_write** | 1 | Output | Cho phép thay đổi đầu ra PC |
| **IF\_ID\_write** | 1 | Output | Cho phép thanh ghi IF\_ID ghi dữ liệu mới |
| **ctrl\_sel** | 1 | Output | Chọn đầu ra control |

### Function description

Khối **Hazard detection unit** có nhiệm vụ xác định có hazard mà không thể sử dụng kĩ thuật forwarding để xử lí mà cần stall pipeline 1 chu kì clk. Khối **Hazard detection unit** xác định trường hợp cần stall như sau:

|  |
| --- |
| if (ID\_EX\_mem\_read) and (ID\_EX\_rd == IF\_ID\_rs1 or ID\_EX\_rd == IF\_ID\_rs2) then stall the pipline. |

## Module Forwarding Unit (forwarding\_unit)



Hình . forwarding\_unit block diagram

### Interface signals

Bảng . forwarding\_unit port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **ID\_EX\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi IF/ID |
| **IF\_ID\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi IF/ID |
| **MEM\_WB\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi MEM/WB |
| **EX\_MEM\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **MEM\_WB\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **ID\_EX\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **branch** | 1 | Input | Cho phép nhảy đến địa chỉ offset |
| **forward\_a** | 2 | Output | Forwarding cho ALU khi có hazard tại thanh ghi rs1 EX |
| **forward\_b** | 2 | Output | Forwarding cho ALU khi có hazard tại thanh ghi rs2 EX |
| **forward\_comp1** | 2 | Output | Forwarding cho bộ branch compare khi có hazard tại thanh ghi rs1 ID |
| **forward\_comp2** | 2 | Output | Forwarding cho bộ branch compare khi có hazard tại thanh ghi rs2 ID |

### Function description

Khối **Forwarding unit** sẽ nhận các giá trị địa chỉ **rs1**, **rs2**, **rd** và các tín hiệu điều khiển từ các thanh ghi pipeline nhằm xác định các trường hợp các lệnh liên tiếp nhau có xảy ra hazard (data hazard, mem hazard, control hazard) để đưa ra các tín hiệu điều khiển cho các bộ **mux** nhằm forward các dữ liệu bị hazard tương ứng để có được kết quả tính toán chính xác nhất mà không cần phải stall pipeline quá nhiều chu kì clk.

Khối **Forwarding unit** sẽ xác định các trường hợp cần forward như sau:

|  |
| --- |
| if (MEM\_WB\_reg\_write  and (MEM\_WB\_reg\_write ≠ 0)  and not(EX\_MEM\_reg\_write and (EX\_MEM\_rd ≠ 0)  and (EX\_MEM\_rd = ID\_EX\_rs1))  and (MEM\_WB\_reg\_write = ID\_EX\_rs1)) forward\_a = 01  if (MEM\_WB\_reg\_write  and (MEM\_WB\_reg\_write ≠ 0)  and not(EX\_MEM\_reg\_write and (EX\_MEM\_rd ≠ 0)  and (EX\_MEM\_rd = ID\_EX\_rs2))  and (MEM\_WB\_reg\_write = ID\_EX\_rs2)) forward\_b = 01  if (branch && (ID\_EX\_rd != 0) && ID\_EX\_reg\_write && (IF\_ID\_rs1 == ID\_EX\_rd))  forward\_comp1 = 01;  else if (branch && (EX\_MEM\_rd != 0) && ~(ID\_EX\_reg\_write && (IF\_ID\_rs1 == ID\_EX\_rd)) && EX\_MEM\_reg\_write && (EX\_MEM\_rd == IF\_ID\_rs1))  forward\_comp1 = 10;  else forward\_comp1 = 00;  if (branch && (ID\_EX\_rd != 0) && ID\_EX\_reg\_write && (IF\_ID\_rs2 == ID\_EX\_rd))  forward\_comp2 = 01;  else if (branch && (EX\_MEM\_rd != 0) && ~(ID\_EX\_reg\_write && (IF\_ID\_rs2 == ID\_EX\_rd)) && EX\_MEM\_reg\_write && (EX\_MEM\_rd == IF\_ID\_rs2))  forward\_comp2 = 10;  else forward\_comp2 = 00; |

Bảng . Forwarding output ports

|  |  |  |
| --- | --- | --- |
| **Mux control** | **Source** | **Explaination** |
| **forward\_a = 00** | **ID/EX** | Đường input thứ nhất của **ALU** đến từ tệp thanh ghi. |
| **forward\_a = 10** | **EX/MEM** | Đường input thứ nhất của **ALU** được foward từ kết quả tính toán của **ALU** cho lệnh ngay trước đó. |
| **forward\_a = 01** | **MEM/WB** | Đường input thứ nhất của **ALU** được forward từ kết quả đọc data từ **memory** hoặc kết quả tính toán của **ALU** cho lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_b = 00** | **ID/EX** | Đường input thứ hai của **ALU** đến từ tệp thanh ghi. |
| **forward\_b = 10** | **EX/MEM** | Đường input thứ hai của **ALU** được foward từ kết quả tính toán của **ALU** cho lệnh ngay trước đó. |
| **forward\_b = 01** | **MEM/WB** | Đường input thứ hai của **ALU** được forward từ kết quả đọc data từ memory hoặc kết quả tính toán của **ALU** cho lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp1 = 00** | **Register** | Đường input thứ nhất của **branch compare** đến từ tệp thanh ghi. |
| **forward\_comp1 = 10** | **mem** or **EX/MEM** | Đường input thứ nhất của **branch compare** đến từ kết quả đọc data từ **memory** hoặc kết quả **ALU** của lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp1 = 01** | **ALU** | Đường input thứ nhất của **branch compare** đến từ kết quả tính toán **ALU** của lệnh trước đó. |
| **forward\_comp2 = 00** | **Register** | Đường input thứ hai của **branch compare** đến từ tệp thanh ghi. |
| **forward\_comp2 = 10** | **mem** or **EX/MEM** | Đường input thứ hai của **branch compare** đến từ kết quả đọc data từ **memory** hoặc kết quả **ALU** của lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp2 = 01** | **ALU** | Đường input thứ hai của **branch compare** đến từ kết quả tính toán **ALU** của lệnh trước đó. |

## Module Top (riscv\_pipeline\_top)

Diagram

Description automatically generated

Hình . riscv\_pipeline\_top block diagram

Hình 2.15 mô tả toàn bộ các module con có trong khối top và các interface tương ứng được kết nối với nhau.

### Interface signals

Bảng . riscv\_pipeline\_top port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 5 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 5 | Input | Tín hiệu reset không đồng bộ, tích cực mức thấp |

### Function description

Khối top là khối kết nối tất cả các module con của CPU, nhận xung **clk** và tín hiệu **reset** để hoạt động, chương trình cần CPU thực hiện sẽ được nạp sẵn vào bộ nhớ **Instruction memory**.

# KIỂM THỬ (VERIFICATION)

*Chương này trình bày các kết quả mô phỏng cho từng khối và toàn bộ thiết kế được triển khai bằng ngôn ngữ SystemVerilog trên phần mềm ModelSim.*

## Kế hoạch kiểm thử

### Kịch bản – Test case

* Kiểm tra quá trình reset, đảm bảo các đầu ra được thiết lập về đúng giá trị
* Kiểm tra trường hợp reset bất thường khi mạch đang hoạt động
* Kiểm tra trường hợp tín hiệu reset được đặt tích cực trong nhiều chu kỳ
* Kiểm tra các trường hợp instruction không có hazard
* Kiểm tra các trường hợp instruction có data hazard
* Kiểm tra các trường hợp instruction có mem hazard
* Kiểm tra các trường hợp instruction có control hazard

### Kích thích đầu vào – Stimulus

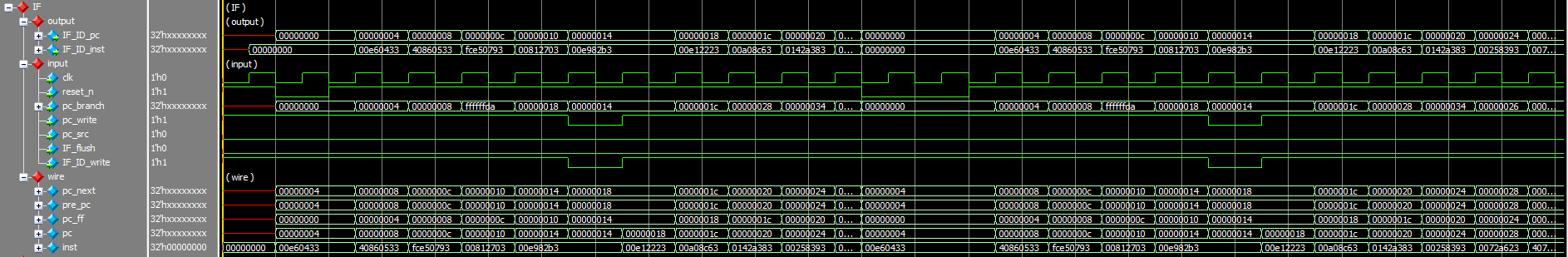
* Tạo sẵn các mã lệnh trong instruction memory như bảng

|  |  |  |  |
| --- | --- | --- | --- |
| **Localparam** | **Address** | **Assembly code** | **Instruction** |
| NONE | 32'h0000 | Nothing | Nothing |
| INST1 | 32'h0004 | add x8, x12, x14 | 0000000\_01110\_01100\_000\_01000\_0110011 |
| INST2 | 32'h0008 | sub x10, x12, x8 | 0100000\_01000\_01100\_000\_01010\_0110011 |
| INST3 | 32'h000C | addi x15, x10, -50 | 111111001110\_01010\_000\_01111\_0010011 |
| INST4 | 32'h0010 | lw x14, 8(x2) | 000000001000\_00010\_010\_01110\_0000011 |
| INST5 | 32'h0014 | add x5, x19, x14 | 0000000\_01110\_10011\_000\_00101\_0110011 |
| INST6 | 32'h0018 | sw x14, 4(x2) | 0000000\_01110\_00010\_010\_00100\_0100011 |
| INST7 | 32'h001C | beq x1, x10, offset(12) | 0\_000000\_01010\_00001\_000\_1100\_0\_1100011 |
| INST8 | 32'h0020 | lw x7, 20(x5) | 000000010100\_00101\_010\_00111\_0000011 |
| INST9 | 32'h0024 | addi x7, x11, 2 | 000000000010\_01011\_000\_00111\_0010011 |
| INST10 | 32'h0028 | sw x7, 12(x5) | 000000000111\_00101\_010\_01100\_0100011 |
| INST11 | 32'h002C | sub x2, x11, x7 | 0100000\_00111\_01011\_000\_00010\_0110011 |
| INST12 | 32'h0030 | and x14, x5, x3 | 0000000\_00011\_00101\_111\_01110\_0110011 |
| INST13 | 32'h0034 | sw x14, 16(x5) | 000000001110\_00101\_010\_10000\_0100011 |
| INST14 | 32'h0038 | beq x1, x14, offset(12) | 0\_000000\_01110\_00001\_000\_1100\_0\_1100011 |
| INST15 | 32'h003C | add x8, x12, x14 | 0000000\_01110\_01100\_000\_01000\_0110011 |
| INST16 | 32'h0040 | sub x10, x12, x8 | 0100000\_01000\_01100\_000\_01010\_0110011 |
| INST17 | 32'h0044 | addi x15, x10, -50 | 111111001110\_01010\_000\_01111\_0010011 |
| INST18 | 32'h0048 | lw x14, 8(x2) | 000000001000\_00010\_010\_01110\_0000011 |
| INST19 | 32'h004C | add x5, x19, x14 | 0000000\_01110\_10011\_000\_00101\_0110011 |
| INST20 | 32'h0050 | lw x14, 10(x2) | 000000001010\_00010\_010\_01110\_0000011 |
| INST21 | 32'h0054 | beq x1, x14 , offset(12) | 0\_000000\_01110\_00001\_000\_1100\_0\_1100011 |
| INST22 | 32'h0058 | add x15, x12, x14 | 0000000\_01110\_01100\_000\_01111\_0110011 |

Các lệnh trên được tạo ra để test các kịch bản có thể xảy ra khi thực hiện chương trình bằng kiến trúc vi xử lí RISCV sử dụng kĩ thuật pipeline. Kịch bản test trên đã bao gồm data hazard, mem hazard và cuối cùng là control hazard. Các thanh ghi được highlight thể hiện có hazard trong chuỗi các instruction.

## Kết quả và đánh giá

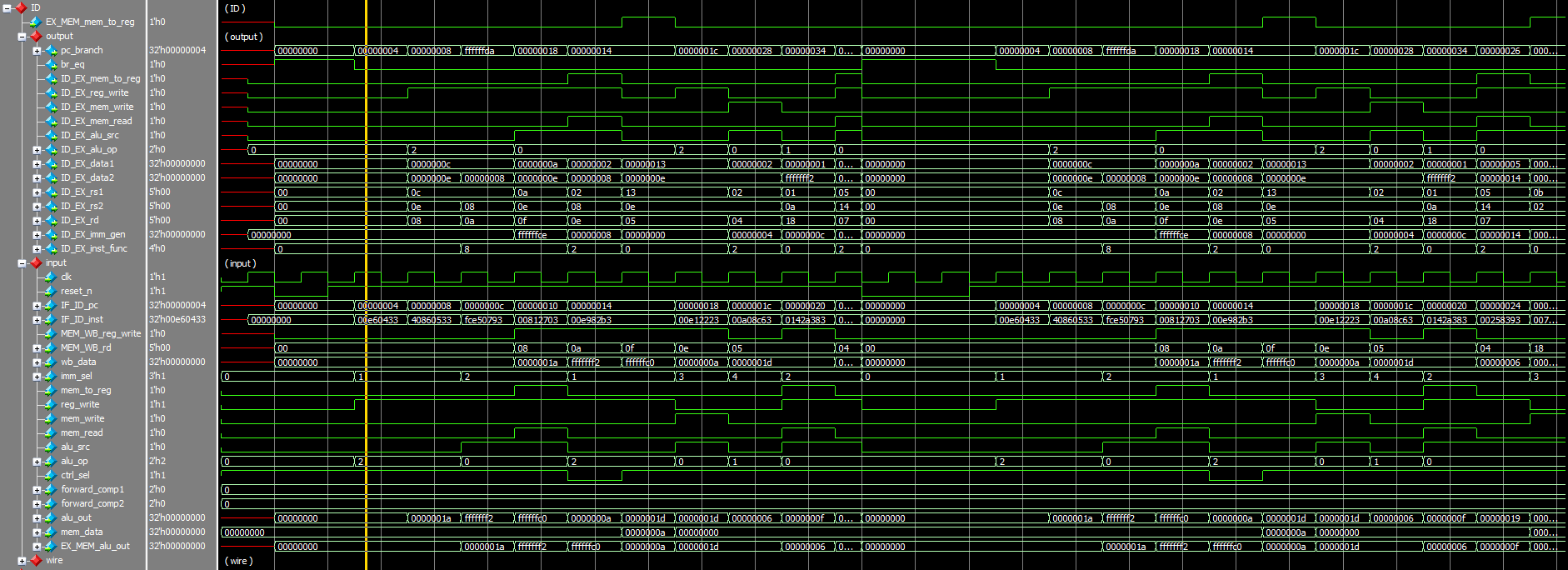
### Module instruction\_fetch



Hình . Mô phỏng timing diagram khối instruction\_fetch

Hình 3.1 cho thấy kết quả đầu ra hoạt động đúng logic khi thực hiện các lệnh trong kịch bản được nạp sẵn vào **Instruction Memory**.

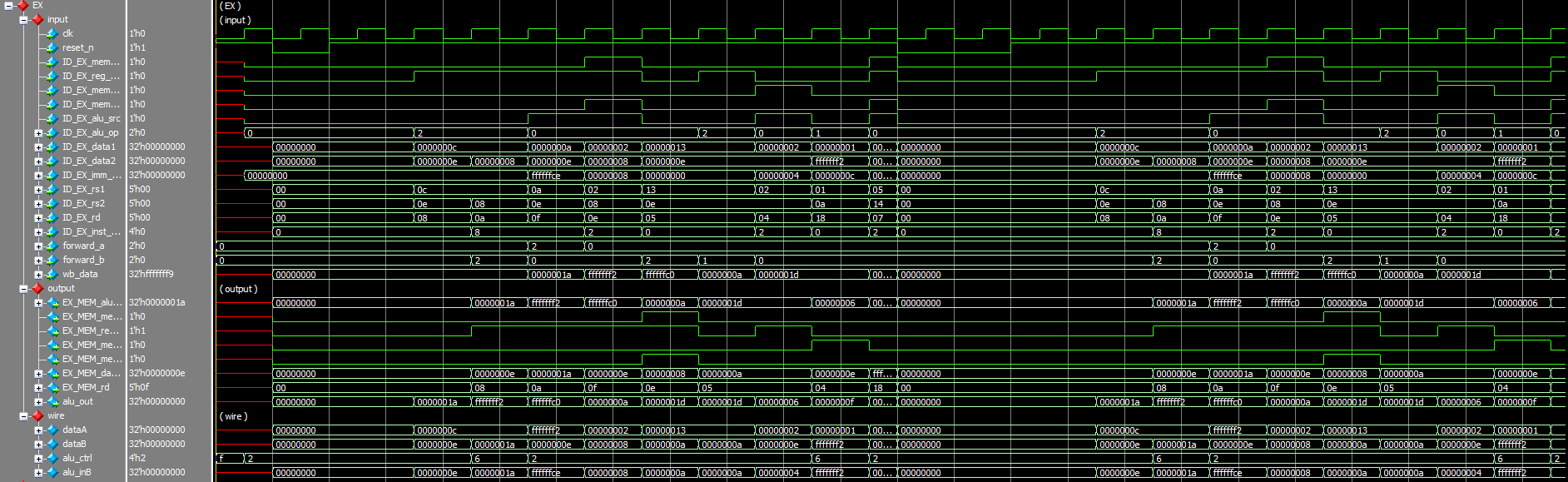
### Module instruction\_decode



Hình . Mô phỏng timing diagram khối instruction\_decode

Hình 3.2 cho thấy kết quả đầu ra của khối **Instruction Decode** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh trong kịch bản được nạp sẵn vào **Instruction Memory**.

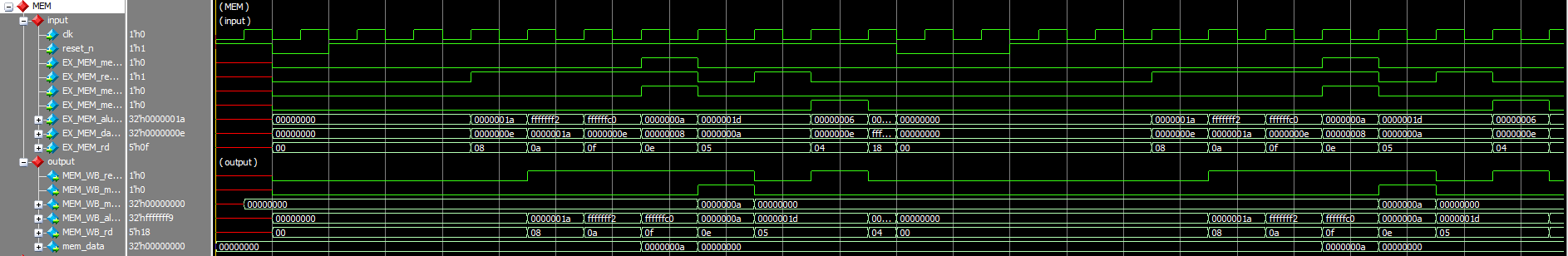
### Module execute



Hình . Mô phỏng timing diagram khối execute

Hình 3.3 cho thấy kết quả đầu ra của khối **Execute** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh trong kịch bản được nạp sẵn.

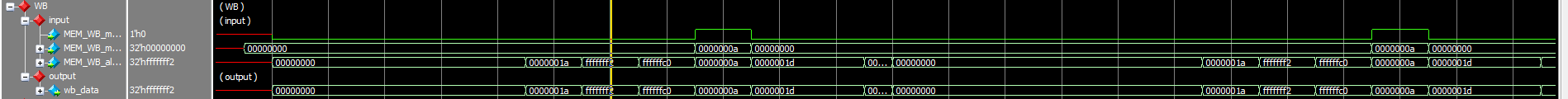
### Module memory\_access



Hình . Mô phỏng timing diagram khối memory\_access

Hình 3.4 cho thấy kết quả đầu ra của khối **Memory Access** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh trong kịch bản được nạp sẵn.

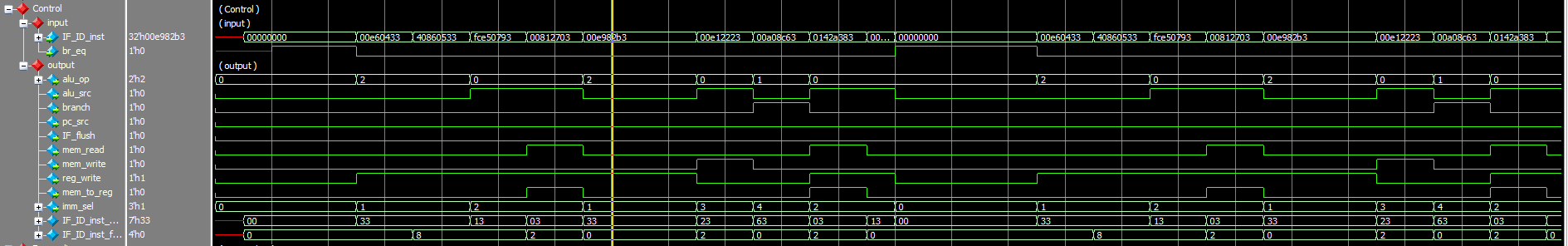
### Module register\_write



Hình . Mô phỏng timing diagram khối register\_write

Hình 3.5 cho thấy kết quả đầu ra của khối **Register Write** hoạt động đúng với logic của thiết kế.

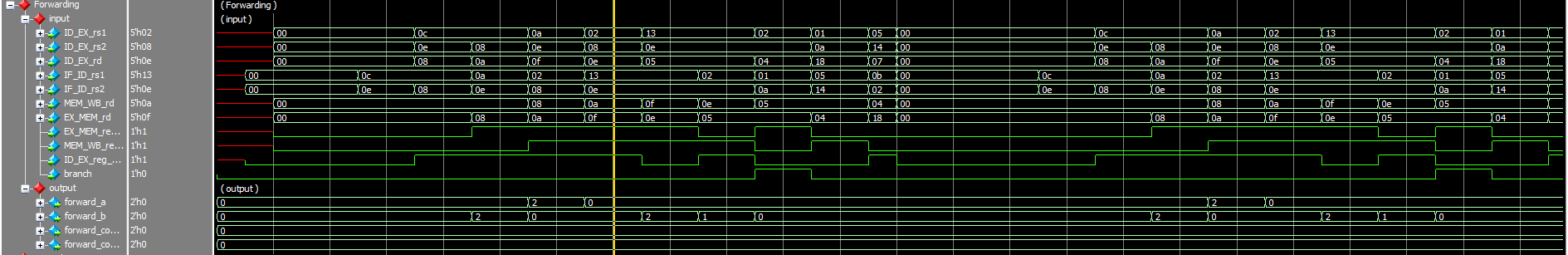
### Module control



Hình . Mô phỏng timing diagram khối control

Hình 3.6 cho thấy các tín hiệu điều khiển **control** gửi đến **datapath** đúng với logic thiết kế khi thực hiện các lệnh được nạp sẵn trong **Instruction Memory**.

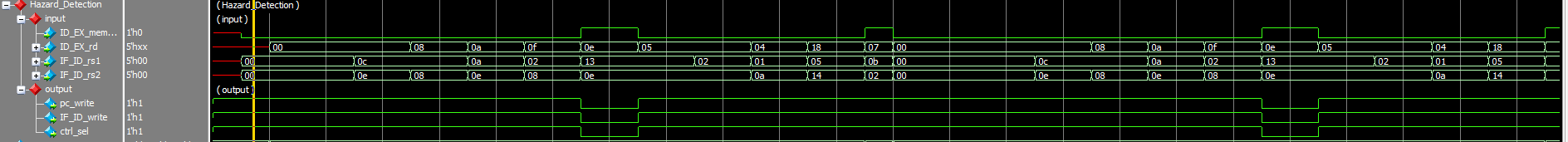
### Module forwarding\_unit



Hình . Mô phỏng timing diagram khối forwarding\_unit

Hình 3.7 mô tả tín hiệu đầu ra của module khối **forwarding unit** theo các kích thích đầu vào tương ứng, wave form cho thấy module hoạt động đúng logic đã định nghĩa.

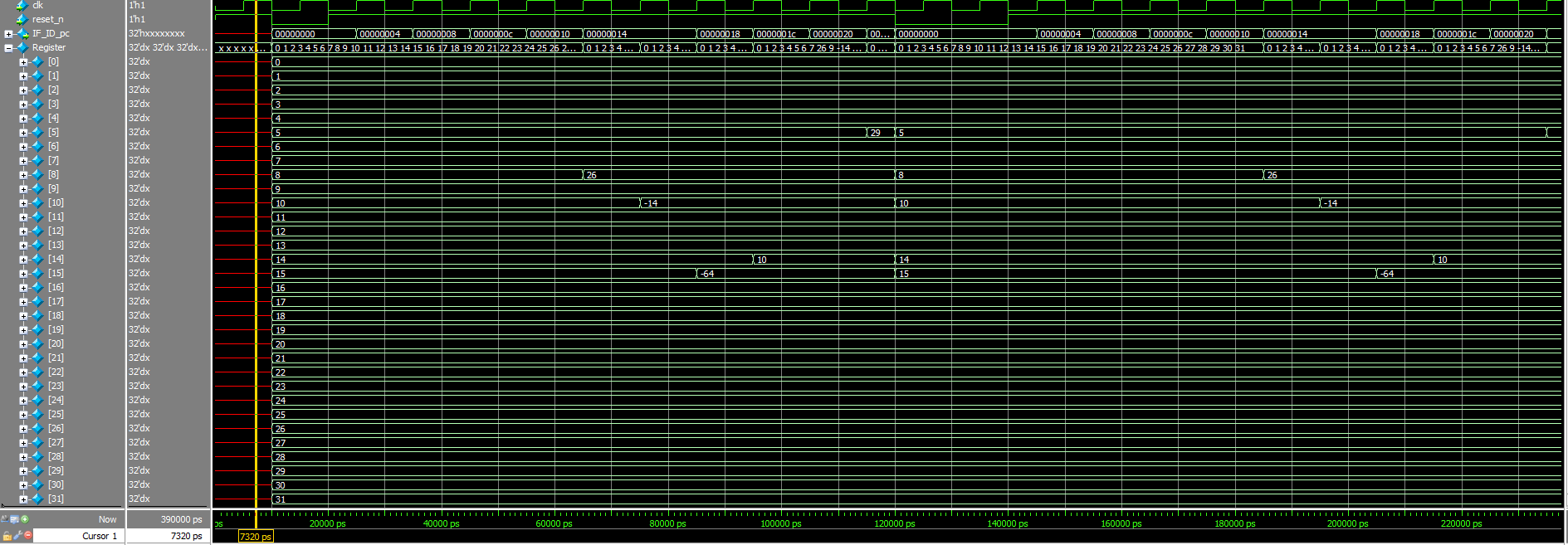
### Module hazard\_detection\_unit



Hình . Mô phỏng timing diagram khối hazard\_detection\_unit

Hình 3.8 mô tả tín hiệu đầu ra của module khối **hazard detection unit** theo các kích thích đầu vào tương ứng, wave form cho thấy module hoạt động đúng logic đã định nghĩa.

### Register

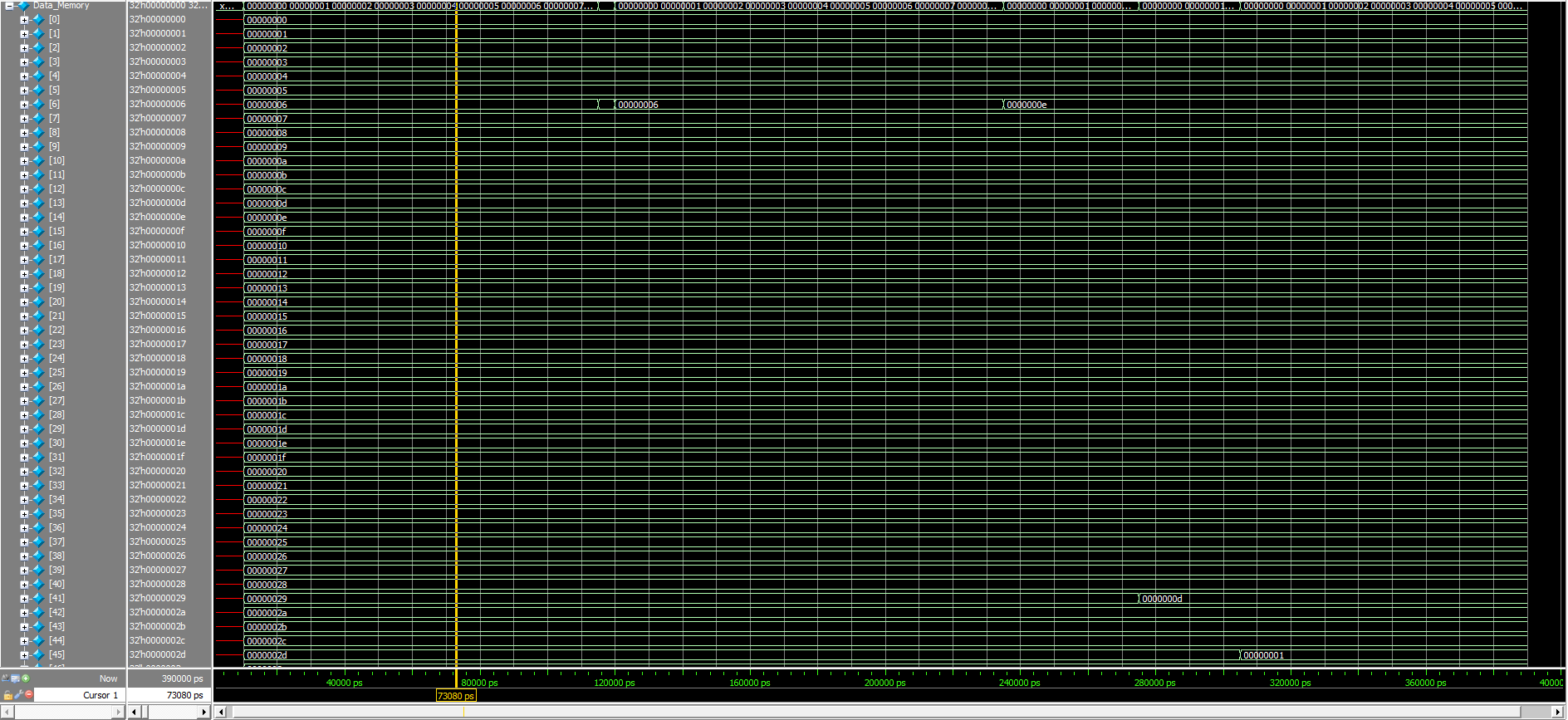


Hình . Mô phỏng 32 Registers

Hình 3.9 thể hiện giá trị của tệp thanh ghi đã được thay đổi sau khi thực hiện chương trình. Wave tín hiệu cho thấy kể từ khi lệnh được nạp vào CPU (tương ứng với giá trị **pc** thay đổi) thì sau 5 chu kì tiếp theo giá trị thanh ghi được thay đổi đúng như kết quả tính toán từ trước cho các lệnh nạp sẵn theo kịch bản cho trước.

Khi có tín hiệu **reset** bất thường, CPU được reset về trạng thái ban đầu, giá trị thanh ghi được khởi tạo tương ứng với địa chỉ và **pc** được reset về 0, chương trình trong **Instruction memory** được chạy lại từ đầu và vẫn đũng so với kết quả đã tính toán trước theo kịch bản lệnh được nạp sẵn.

### Data\_Memory



Hình . Mô phỏng Data\_Memory

Hình 3.10 mô tả sự thay đổi của dữ liệu bên trong **Data Memory**, dữ liệu thay đổi đúng như kịch bản tính toán của các câu lệnh được nạp sẵn trong **Instruction Memory**.

# KẾT LUẬN

Báo cáo này đã trình bày và triển khai kiến trúc của một RISCV32I processor áp dụng kĩ thuật pipeline nhằm tăng tốc độ xử lí, bên cạnh đó, thiết kế đã xữ lí được toàn bộ các hazard có thể xảy ra trong quá trình xử lí thực hiện lệnh của CPU bao gồm data hazard, mem hazard và control hazard. Thiết kế được tiến hành triển khai bằng ngôn ngữ mô tả phần cứng SystemVerilog và mô phỏng kiểm thử trên phần mềm ModelSim. Cho ra kết quả hoạt động đúng với yêu cầu vào ra. Kiến trúc có thể thực hiện 4 kiểu lệnh assembly trong tập lệnh của RISCV gồm: R-type, I-type, S-type, B-type, tuy nhiên chưa thể thực hiện được tất cả các lệnh có trong tập lệnh của RISCV (xấp xỉ 40 lệnh). Trong tương lai nhóm sẽ tiến hành hoàn thiện và triển khai kiến trúc của một RISCV32I processor hoàn chỉnh với đầy đủ các chức năng, thực hiện được đầy đủ các lệnh trong kiến trúc tập lệnh của RISCV một cách tối ưu nhất.

TÀI LIỆU THAM KHẢO

1. <https://en.wikipedia.org/wiki/RISC-V> Ngày truy cập cuối cùng: 16/5/2021
2. Slide: RISC – V 2021. PGS TS. Nguyễn Đức Minh
3. Computer Organization and Design Risc – V The Hardware Software Interface by David A. Patterson and John L. Hennessy
4. Link project: https://github.com/chien172431/riscv32.git