**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------

****

**BÁO CÁO ĐỒ ÁN 2**

**IMPLEMENTING AN INTERRUPT HANDLER FOR RISC-V ISA**

**GVHD: TRẦN HOÀNG LINH**

**SVTH: LÊ VIỆT CƯỜNG**

**NGUYỄN PHÚ CƯỜNG**

**MSSV: 2110878**

**2112715**

**TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2024**

***LỜI CẢM ƠN***

Trong quá trình thực hiện đề tài này, em đã nhận được sự giúp đỡ, góp ý và chỉ bảo tận tình từ các thầy cô, cũng như sự hỗ trợ từ bạn bè. Em xin gửi lời cảm ơn chân thành đến thầy TS. Trần Hoàng Linh, người đã đóng góp ý kiến và hỗ trợ em trong quá trình thực hiện đề tài.

Bên cạnh đó, em cũng xin gửi lời cảm ơn chân thành đến các thầy cô trường Đại học Bách khoa TP.HCM, những người đã truyền đạt cho em những kiến thức và kinh nghiệm quý báu để có thể hoàn thành đề tài này.

Trong quá trình thực hiện, em đã cố gắng hết mình để hoàn thành đề tài. Tuy nhiên, do kiến thức và kinh nghiệm còn hạn chế nên không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý từ quý thầy cô. Em xin chân thành cảm ơn!

*Tp. Hồ Chí Minh, ngày 27 tháng 12 năm 2024 .*

**Sinh viên**

**Lê Việt Cường**

**Nguyễn Phú Cường**

**TÓM TẮT ĐỒ ÁN**

**Đồ án trình bày về thiết kế một bộ xử lý ngắt (interrupt handler) trong kiến trúc RISC-V. Bộ xử lý ngắt này chịu trách nhiệm quản lý và xử lý các ngắt từ các thiết bị ngoại vi trong hệ thống MCU, bao gồm bộ đếm thời gian (timer) và các yêu cầu ngắt từ các thiết bị ngoại vi khác.**

Trong thiết kế này, CPU RISC-V 32I được triển khai với bộ xử lý ngắt cơ bản, giúp xử lý các ngắt mà không cần phải sử dụng các cấu trúc phức tạp như CLINT (Core Local Interruptor) và PLIC (Platform-Level Interrupt Controller). Bộ xử lý ngắt này trực tiếp quản lý các tín hiệu ngắt từ các thiết bị ngoại vi, bao gồm cả bộ đếm thời gian (timer) và các tín hiệu ngắt khác từ các thiết bị ngoại vi trong hệ thống.

Bộ đếm thời gian (timer) trong hệ thống được sử dụng để tạo ra các tín hiệu ngắt định kỳ. Khi bộ đếm thời gian đạt đến giá trị giới hạn, nó sẽ phát sinh một ngắt, giúp hệ thống thực hiện các tác vụ định kỳ hoặc quản lý thời gian. Bộ xử lý ngắt sẽ nhận và xử lý ngắt từ bộ đếm thời gian, từ đó cho phép hệ thống thực hiện các nhiệm vụ thời gian thực (real-time tasks) một cách hiệu quả.

Để tối ưu hóa hiệu suất, CPU RISC-V 32I được triển khai với 5 tầng pipeline, giúp tăng cường khả năng xử lý và giảm thiểu độ trễ khi xử lý các ngắt. Bộ dự đoán nhánh được tích hợp vào hệ thống giúp giảm thiểu độ trễ khi xảy ra các nhánh điều kiện, tăng số lệnh xử lý trên mỗi chu kỳ (IPC) và tối ưu hóa luồng xử lý lệnh.

Hệ thống MCU sử dụng giao thức APB thuộc họ AMBA cho các module ngoại vi, đặc biệt là UART, để đơn giản hóa giao tiếp và tối ưu hóa năng lượng. Ngoại vi UART hỗ trợ chuẩn APB, giúp giảm độ phức tạp trong việc giao tiếp và tăng cường sự tích hợp của hệ thống.

**MỤC LỤC**

[1. GIỚI THIỆU 6](#_Toc186150191)

[2. LÝ THUYẾT 8](#_Toc186150192)

[2.1. Tập lệnh RISC-V 9](#_Toc186150193)

[2.2. 2-bit prediction 12](#_Toc186150194)

[2.3. Interrupt Controller 13](#_Toc186150195)

[3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG 15](#_Toc186150196)

[3.1. Ý tưởng thiết kế CPU 15](#_Toc186150197)

[3.2. Thiết kế interrupt controller: 23](#_Toc186150198)

[3.3. Thiết kế Timer 8 bit 29](#_Toc186150199)

[3.4. THIẾT KẾ UART 36](#_Toc186150200)

[4. XÁC MINH 45](#_Toc186150201)

[4.1. CPU 45](#_Toc186150202)

[4.2. Interrupt controller 46](#_Toc186150203)

[4.3. TIMER 8 BIT 51](#_Toc186150204)

[4.4. UART 54](#_Toc186150205)

[TÀI LIỆU THAM KHẢO 58](#_Toc186150206)

**DANH SÁCH HÌNH MINH HỌA**

[Hình 2- 1 Định dạng của các lệnh cơ bản 7](#_Toc186150018)

[Hình 2- 2 Định dạng của R-type format 7](#_Toc186150019)

[Hình 2- 3 Định dạng của lệnh nhảy có điều kiện 8](#_Toc186150020)

[Hình 2- 4 Định dạng của lệnh JAL và JALR 8](#_Toc186150021)

[Hình 2- 5 Định dạng của lệnh LOAD và STORE 11](#_Toc186150022)

[Hình 2- 6 Sơ đồ khối chung của interrupt controller 12](#_Toc186150023)

[Hình 3- 1 Data hazard nếu không có Forwarding……………………………………………………………………………. 18](#_Toc186150126)

[Hình 3- 2 Data hazard sau khi giải quyết bằng Forwarding Unit 18](#_Toc186150127)

[Hình 3- 3 Sơ đồ khối của mạch 2-bit prediction với RISC-V 20](file:///C:\Users\Cuongle3012\Documents\Zalo%20Received%20Files\Interrupt%20Handler%20Report.docx#_Toc186150128)

[Hình 3- 4 Sơ đồ khối của APB\_RW 21](#_Toc186150129)

[Hình 3- 5 Thực thi cách ghi data của thanh ghi ISRA/ISRB 22](#_Toc186150130)

[Hình 3- 6 Sơ đồ khối của mạch phát hiện loại tín hiệu mà ngắt có thể bắt 23](#_Toc186150131)

[Hình 3- 7 Mạch phát hiện ngắt 24](#_Toc186150132)

[Hình 3- 8 Sơ đồ khối của mạch đã phát hiện ngắt và gửi tín hiệu yêu cầu ngắt 24](#_Toc186150133)

[Hình 3- 9 Mạch sắp xếp để chọn ra tín hiệu ngắt nào có độ ưu tiên cao nhất 25](#_Toc186150134)

[Hình 3- 10 Sơ đồ khối của IP TIMER 8 BIT 27](#_Toc186150135)

[Hình 3- 11Các thanh ghi sử dụng trong thiết kế 27](#_Toc186150136)

[Hình 3- 12 Cấu trúc của thanh ghi TCR 28](#_Toc186150137)

[Hình 3- 13 Cấu trúc của thanh ghi TSR 28](#_Toc186150138)

[Hình 3- 14 Mô tả hoạt động của Clock Selection 29](#_Toc186150139)

[Hình 3- 15 Mô tả hoạt động của bộ đếm Timer 30](#_Toc186150140)

[Hình 3- 16Cách thức hoạt động của APB controller 31](#_Toc186150141)

[Hình 3- 17 Data Framing Uart 33](#_Toc186150142)

[Hình 3- 18 Một số ứng dụng của UART 34](#_Toc186150143)

[Hình 3- 19 UART Top Module 35](#_Toc186150144)

[Hình 3- 20 APB UART block diagram 36](#_Toc186150145)

[Hình 3- 21 UART transmiter block diagram 37](#_Toc186150146)

[Hình 3- 22 UART transmiter FSMD 38](#_Toc186150147)

[Hình 3- 23 UART receiver block diagram 39](#_Toc186150148)

[Hình 3- 24 Vị trí lấy mẫu của 1 bit dữ liệu Uart 40](#_Toc186150149)

[Hình 3- 25 UART Receiver FSMD 40](#_Toc186150150)

[Hình 4- 1 Các testcase đã được kiểm chứng chính xác](file:///C:\Users\Cuongle3012\Documents\Zalo%20Received%20Files\Interrupt%20Handler%20Report.docx#_Toc186150151).........................................................................45

[Hình 4- 2 Sơ đồ của môi trường testbench interrupt controller 45](#_Toc186150152)

[Hình 4- 3 Hàm ghi data vào thanh ghi thông qua giao thức APB 46](#_Toc186150153)

[Hình 4- 4 Hàm đọc data vào thanh ghi thông qua giao thức APB 46](#_Toc186150154)

[Hình 4- 5 Kết quả sau khi test trường hợp ghi và đọc data cơ bản 47](#_Toc186150155)

[Hình 4- 6 Test cơ bản ghi data vào thanh ghi trạng thái 48](#_Toc186150156)

[Hình 4- 7 Kết quả sau khi đã test 5 testcase cơ bản. 48](#_Toc186150157)

[Hình 4- 8 Trường hợp nguồn ngắt là loại cạnh xuống và giả định cho phép toàn bộ ngắt đều được thông qua. 49](#_Toc186150158)

[Hình 4- 9 Kết quả sau khi test nguồn ngắt là loại cạnh xuống. 49](#_Toc186150159)

[Hình 4- 10 Testbench đếm lên với hệ số clock là 2 50](#_Toc186150160)

[Hình 4- 11Kết quả với trường hợp đếm lên với hệ số clock là 2 51](#_Toc186150161)

[Hình 4- 12 Testbench đếm xuống với hệ số clock là 2 51](#_Toc186150162)

[Hình 4- 13 Kết quả với trường hợp đếm xuống với hệ số clock là 2 52](#_Toc186150163)

[Hình 4- 14 Đánh giá tài nguyên Timer với Quartus 53](#_Toc186150164)

[Hình 4- 15 Waveform lý thuyết 53](#_Toc186150165)

[Hình 4- 16 Phân tích thời gian của 1 bit UART ở khối Transmitter 54](#_Toc186150166)

[Hình 4- 17 Phân tích output ở khối Transmitter 54](#_Toc186150167)

[Hình 4- 18 Phân tích 1 bit ở khối Receiver 54](#_Toc186150168)

[Hình 4- 19 Phân tích output ở khối Receiver 55](#_Toc186150169)

[Hình 4- 20 Đánh giá tài nguyên bộ Uart Transmitter 55](#_Toc186150170)

[Hình 4- 21 Đánh giá tài nguyên bộ Uart Receiver 56](#_Toc186150171)

**DANH SÁCH BẢNG MINH HỌA**

[Bảng 1 Các tín hiệu của interrupt controller 15](#_Toc186150172)

[Bảng 2 Các tín hiệu của khối IF\_ID 16](#_Toc186150173)

[Bảng 3 Các tín hiệu của khối ID\_EX 18](#_Toc186150174)

[Bảng 4 Các tín hiệu của khối EX\_MEM 18](#_Toc186150175)

[Bảng 5 Các tín hiệu của khối MEM\_WB 19](#_Toc186150176)

[Bảng 6 Các tín hiệu của khối hazard\_unit 19](#_Toc186150177)

[Bảng 7 Các tín hiệu của khối forwarding\_unit 20](#_Toc186150178)

[Bảng 8 Các tín hiệu của khối branch\_prediction 21](#_Toc186150179)

[Bảng 9 Các tín hiệu đầu vào và ra của APB\_RW 25](#_Toc186150180)

[Bảng 10 Các tín hiệu đầu vào và ra của edge\_detection 26](#_Toc186150181)

[Bảng 11 Bảng giá trị của mạch phát hiện yêu cầu ngắt 28](#_Toc186150182)

[Bảng 12 Các tín hiệu của mạch Prio\_deter 29](#_Toc186150183)

[Bảng 13 Signal description APB UART 40](#_Toc186150184)

[Bảng 14 UART Transmiter signal description 41](#_Toc186150185)

[Bảng 15 UART Receiver signal description 42](#_Toc186150186)

[Bảng 16 Signal description of baud clock generator module 44](#_Toc186150187)

[Bảng 17 Control Register 44](#_Toc186150188)

[Bảng 18 Các tín hiệu Enable 45](#_Toc186150189)

[Bảng 19 Các tín hiệu ngắt có trong thiết kế 45](#_Toc186150190)

# 1. GIỚI THIỆU

**1.1 Tổng quan**

Kiến trúc tập lệnh đơn giản RISC (Reduced Instruction Set Computing) ra đời vào những năm 1980 nhằm đáp ứng nhu cầu thiết kế bộ xử lý với hiệu suất cao hơn bằng cách giảm độ phức tạp của tập lệnh. Thay vì tích hợp nhiều tác vụ phức tạp vào một câu lệnh như trong CISC (Complex Instruction Set Computing), RISC tối giản tập lệnh, giúp tăng tốc độ xử lý và giảm tải cho bộ vi xử lý. Cách tiếp cận của RISC giúp đơn giản hóa quá trình thực thi và tăng khả năng thực hiện nhiều lệnh song song, từ đó cải thiện hiệu suất tổng thể của hệ thống.

Các CPU sử dụng kiến trúc RISC, như ARM, đã đạt được thành công đáng kể nhờ vào thiết kế tập lệnh hiệu quả và tối ưu hóa cho nhiều ứng dụng, từ thiết bị di động đến hệ thống nhúng. ARM nổi tiếng vì tập lệnh dễ mở rộng và khả năng tiết kiệm năng lượng, phù hợp với các thiết bị yêu cầu độ linh hoạt cao nhưng tiêu thụ ít năng lượng, đặc biệt là trong các thiết bị di động. Tuy nhiên, vì là một thiết kế độc quyền, ARM yêu cầu phí bản quyền khi các công ty muốn tùy chỉnh hoặc mở rộng kiến trúc này cho các mục đích riêng.

Để khắc phục các hạn chế của thiết kế độc quyền, năm 2010, một nhóm các nhà nghiên cứu từ Đại học California, Berkeley đã giới thiệu kiến trúc RISC-V. Khác với ARM, RISC-V là một tập lệnh mã nguồn mở, cho phép bất kỳ cá nhân hoặc tổ chức nào cũng có thể truy cập, nghiên cứu, và triển khai mà không yêu cầu chi phí bản quyền. Kiến trúc RISC-V không bị giới hạn trong một ứng dụng cụ thể nào, mà có thể tùy chỉnh và mở rộng để đáp ứng các yêu cầu đa dạng trong ngành công nghiệp bán dẫn. Điều này mang lại sự linh hoạt cho các công ty phát triển chip, đặc biệt là trong bối cảnh xu hướng sản xuất các vi mạch tùy biến ngày càng phổ biến.

RISC-V được thiết kế với cấu trúc tập lệnh đơn giản, dễ học và dễ triển khai. Khả năng mở rộng của RISC-V cho phép các nhà phát triển thêm các tính năng hoặc các lệnh mới mà không làm phức tạp kiến trúc cơ bản. Các tập lệnh của RISC-V bao gồm các bộ lệnh cơ bản (base instruction set) như RISC-V 32I, cùng với các phần mở rộng tùy chọn cho các tính năng cụ thể như xử lý dấu phẩy động (floating-point) hoặc nhân và chia (multiply/divide). Nhờ tính linh hoạt này, RISC-V phù hợp cho cả hệ thống nhỏ (như các thiết bị IoT) và hệ thống lớn hơn đòi hỏi hiệu suất cao.

Mặc dù có nhiều tiềm năng, RISC-V vẫn đang trong quá trình phát triển hệ sinh thái phần mềm để hỗ trợ đầy đủ cho các ứng dụng phức tạp. So với ARM, hệ sinh thái phần mềm của RISC-V hiện chưa phong phú, nhưng đang phát triển mạnh mẽ nhờ cộng đồng mã nguồn mở. Ngày nay, RISC-V không chỉ được sử dụng trong học thuật mà còn dần được ứng dụng trong các sản phẩm thương mại, nhờ khả năng tùy chỉnh cao và không có chi phí bản quyền. Việc sử dụng RISC-V ngày càng được mở rộng, đặc biệt khi các công ty ngày càng quan tâm đến các giải pháp thiết kế vi mạch không phụ thuộc vào các nhà cung cấp độc quyền.

**1.2 Nhiệm vụ đề tài**

Nội dung 1: RISCV-CPU

* Nghiên cứu tập lệnh, kiến trúc RISC-V 32I
* Xây dựng CPU RISC-V với kiến trúc 5 tầng pipeline, tích hợp bộ Branch Prediction.

Nội dung 2: APB Bus

* Nghiên cứu lý thuyết về APB Bus.
* Xây dựng hệ thống Bus có thể giúp CPU giao tiếp với ngoại vi.

Nội dung 3: Interrupt Controller

* Tìm hiểu và phân tích các khái niệm, nguyên lý hoạt động và các thành phần chính của hệ thống điều khiển ngắt, bao gồm các loại ngắt và quy trình xử lý ngắt.
* Thiết kế và triển khai một mô hình ngắt có khả năng nhận và xử lý các tín hiệu ngắt từ các thiết bị ngoại vi, đảm bảo tính chính xác và hiệu quả

Nội dung 4: IP Timer và Uart

* Tìm hiểu về các đặc điểm, nguyên lý của IP Timer và Uart
* Xây dựng IP Timer và Uart tương thích với yêu cầu cầu MCU

Nội dung 5: Thực hiện trên FPGA

**1.3 Phân chia công việc trong nhóm**

|  |  |
| --- | --- |
| **Nội dung** | **Người thực hiện** |
| Thiết kế CPU RISCV | Việt Cường, Phú Cường |
| Interrupt Controller | Phú Cường |
| APB Bus System | Việt Cường |
| IP Timer | Việt Cường, Phú Cường |
| IP Uart | Việt Cường |
| Thực hiện trên FPGA | Việt Cường, Phú Cường |

# 

# 2. LÝ THUYẾT

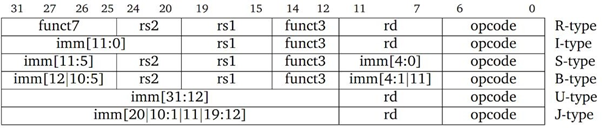
## 2.1. Tập lệnh RISC-V

Kiến trúc tập lệnh RISC-V (ISA) là một kiến trúc máy tính mã nguồn mở được thiết kế đơn giản, linh hoạt và dễ mở rộng. Đây là một ISA kiểu load-store, nghĩa là hầu hết các lệnh hoạt động trên các thanh ghi.

Trong số đó, tập lệnh RV32I (Tập lệnh cơ sở số nguyên 32-bit của RISC-V) là tập lệnh cơ bản nhất trong kiến trúc RISC-V, với "32" biểu thị kích thước mỗi từ (32 bit) và "I" đại diện cho "Integer" (số nguyên). Điều này có nghĩa đây là tập lệnh 32-bit được thiết kế cho các thao tác số nguyên. Mỗi lệnh bao gồm một opcode, các địa chỉ thanh ghi nguồn, và địa chỉ thanh ghi đích, cho phép thực hiện các thao tác số học, logic, điều khiển luồng, và truy cập bộ nhớ.

RV32I bao gồm các loại lệnh chính sau:

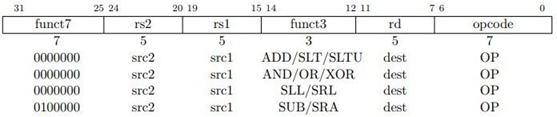
* **Lệnh số học và logic:** Các lệnh như ADD, SUB, AND, OR, và XOR thực hiện các phép toán trên số nguyên và bit.
* **Lệnh điều khiển luồng:** Các lệnh như BEQ, BNE, JAL, và JR điều khiển luồng thực thi của chương trình.
* **Lệnh truy cập bộ nhớ:** Các lệnh như LW và SW thực hiện việc đọc và ghi dữ liệu từ và vào bộ nhớ.



Hình 2- 1 Định dạng của các lệnh cơ bản

**2.1.1. Nhóm lệnh tính toán**

Hầu hết các lệnh tính toán trong RISC-V lưu giá trị vào các thanh ghi, với hai dạng chính: **Register-Immediate** (sử dụng định dạng I-type) và **Register-Register** (sử dụng định dạng R-type). Cả hai dạng lệnh này đều sử dụng thanh ghi đích là rd.



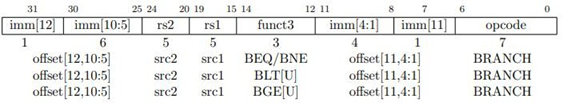
Hình 2- 2 Định dạng của R-type format

* Với lệnh **I-type**, giá trị trong thanh ghi rs1 sẽ được tính toán với một giá trị tức thời (immediate), và kết quả được lưu vào rd. Ví dụ, lệnh **ADDI** cộng giá trị tức thời 12-bit (được mở rộng dấu) với giá trị trong rs1. Kết quả phép cộng (bỏ qua tràn số học) được lưu vào thanh ghi rd, chỉ lấy 32 bit thấp nhất.
* Với lệnh **R-type**, phép tính được thực hiện giữa hai thanh ghi rs1 và rs2, sau đó lưu kết quả vào rd.

Một lệnh đặc biệt trong nhóm này là **NOP** (No Operation), không thay đổi giá trị nào trong các thanh ghi ngoại trừ tăng giá trị của bộ đếm chương trình (PC).

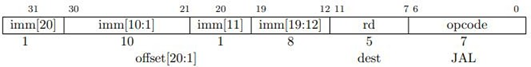
**2.1.2. Nhóm lệnh nhảy**

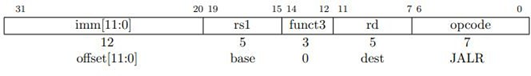
RV32I hỗ trợ hai loại lệnh nhảy: **có điều kiện** và **không điều kiện**.



Hình 2- 3 Định dạng của lệnh nhảy có điều kiện

* Lệnh nhảy **có điều kiện** chuyển giá trị PC tới một nhãn nếu thỏa mãn điều kiện của lệnh. Các lệnh này sử dụng định dạng **B-type**, với giá trị tức thời 12-bit có dấu, cộng với giá trị PC để xác định địa chỉ đích. Phạm vi nhảy của lệnh này là ±4 KiB.





Hình 2- 4 Định dạng của lệnh JAL và JALR

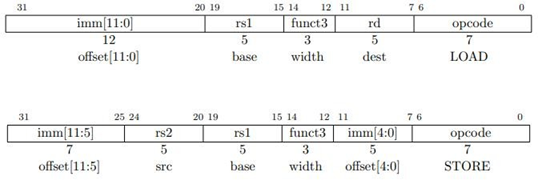
* Lệnh nhảy **không điều kiện** luôn chuyển PC tới nhãn đích. Lệnh **JAL** sử dụng định dạng **J-type**, với giá trị offset 20-bit có dấu được mở rộng và cộng với PC. Phạm vi nhảy là ±1 MiB, và địa chỉ lệnh tiếp theo (PC+4) được lưu vào thanh ghi rd. Lệnh **JALR** sử dụng định dạng **I-type**, với giá trị offset 12-bit có dấu cộng với giá trị của rs1. Kết quả được làm tròn LSB về 0 và PC+4 cũng được lưu vào rd.

**2.1.3. Nhóm lệnh load và store**

Kiến trúc RV32I là một kiến trúc **load-store**, trong đó chỉ các lệnh **load** và **store** có quyền truy cập vào bộ nhớ, còn các lệnh tính toán khác chỉ thao tác trên thanh ghi CPU.

Các lệnh **Load** và **Store** thực hiện việc trao đổi dữ liệu giữa thanh ghi và bộ nhớ.

* **Lệnh Load** được mã hóa theo định dạng **I-type format**.
  + **LW** tải một giá trị 32-bit từ bộ nhớ vào thanh ghi rd.
  + **LH** tải một giá trị 16-bit từ bộ nhớ, mở rộng dấu lên 32-bit, rồi lưu vào rd.
  + **LHU** tải một giá trị 16-bit từ bộ nhớ, thêm các bit 0 ở đầu để tạo thành 32-bit, sau đó lưu vào rd.
  + **LB** và **LBU** thực hiện tương tự đối với các giá trị 8-bit, với **LB** mở rộng dấu và **LBU** thêm bit 0.
* **Lệnh Store** được mã hóa theo định dạng **S-type format**.
  + **SW**, **SH**, và **SB** lưu giá trị 32-bit, 16-bit, và 8-bit (lấy từ các bit thấp của thanh ghi rs2) vào bộ nhớ.



Hình 2- 5 Định dạng của lệnh LOAD và STORE

## 2.2 2-bit prediction

Thuật toán dự đoán hai bit là một phương pháp được sử dụng trong các bộ xử lý pipeline để cải thiện độ chính xác của dự đoán nhánh.

Trong một bộ xử lý pipeline, khi gặp lệnh nhánh, việc dự đoán nhánh sẽ được thực hiện (branch được thực thi hay không) là rất quan trọng để duy trì liên tục quá trình thực thi lệnh. Nếu dự đoán nhánh đúng, bộ xử lý có thể tiếp tục thực thi lệnh tiếp theo mà không cần chờ kết quả của lệnh nhánh. Ngược lại, nếu dự đoán sai, bộ xử lý phải thực hiện các thao tác để sửa sai và chờ kết quả thực sự của lệnh nhánh.

Thuật toán dự đoán hai bit sử dụng bộ đếm hai bit để theo dõi lịch sử của nhánh. Mỗi nhánh sẽ được liên kết với một bộ đếm hai bit, trong đó mỗi trạng thái đại diện cho một mức dự đoán, bao gồm:

1. **STRONG\_TAKEN (11):** Nhánh được dự đoán sẽ được thực thi ở mức mạnh (nếu lần dự đoán tiếp theo sai, BTB vẫn giữ trạng thái 0 và cập nhật sang trạng thái khác).
2. **WEAK\_TAKEN (10):** Nhánh được dự đoán sẽ được thực thi ở mức yếu (nếu dự đoán sai lần tiếp theo, BTB được cập nhật thành trạng thái 0 - không thực thi ở mức yếu; nếu dự đoán đúng, BTB tăng lên mức mạnh).
3. **STRONG\_NOT\_TAKEN (00):** Nhánh được dự đoán sẽ không thực thi ở mức mạnh (nếu dự đoán sai lần tiếp theo, BTB vẫn giữ trạng thái 0 và cập nhật trạng thái khác).
4. **WEAK\_NOT\_TAKEN (01):** Nhánh được dự đoán sẽ không thực thi ở mức yếu (nếu dự đoán sai lần tiếp theo, BTB được cập nhật thành trạng thái 1 - thực thi ở mức yếu; nếu dự đoán đúng, BTB tăng lên mức mạnh không thực thi).

Ban đầu, tất cả các bộ đếm được khởi tạo ở trạng thái **Strongly Not Taken (00)**. Khi xuất hiện lệnh nhánh, bộ đếm hai bit tương ứng với lệnh nhánh đó sẽ được kiểm tra.

* Nếu trạng thái dự đoán là **10** hoặc **11**, bộ xử lý dự đoán nhánh sẽ được thực thi và tiếp tục thực thi lệnh tiếp theo.
* Nếu trạng thái dự đoán là **00** hoặc **01**, bộ xử lý dự đoán nhánh sẽ không được thực thi và chờ kết quả của lệnh nhánh trước khi tiếp tục.

Khi lệnh nhánh hoàn tất, kết quả thực tế của nó sẽ được dùng để cập nhật bộ đếm hai bit.

* Nếu nhánh được thực thi, bộ đếm sẽ tăng lên trạng thái dự đoán cao hơn (**11** hoặc **10**).
* Nếu nhánh không được thực thi, bộ đếm sẽ giảm xuống trạng thái dự đoán thấp hơn (**01** hoặc **00**).

Cơ chế này giúp cải thiện độ chính xác của dự đoán nhánh trong tương lai dựa trên lịch sử thực thi trước đó.

Thuật toán dự đoán hai bit cung cấp sự cân bằng giữa độ chính xác và hiệu suất. Nhờ sử dụng bộ đếm hai bit, thuật toán này có khả năng thích ứng với các kiểu dự đoán nhánh khác nhau và giảm tần suất dự đoán sai. Tuy nhiên, nó cũng gặp một số vấn đề, chẳng hạn như khi gặp nhánh lần đầu hoặc khi có sự chuyển đổi nhanh giữa các trạng thái dự đoán.

Tóm lại, thuật toán dự đoán hai bit là một phương pháp dự đoán nhánh được sử dụng trong các bộ xử lý pipeline. Bằng cách sử dụng bộ đếm hai bit để theo dõi lịch sử dự đoán, thuật toán này cân bằng giữa độ chính xác và hiệu suất, đồng thời tối ưu hóa quá trình xử lý lệnh trong bộ xử lý nhiều giai đoạn.

## 2.3. Interrupt Controller

Ngắt được điều khiển bởi bộ điều khiển ngắt. Bộ điều khiển ngắt có hai chế độ kiểm soát ngắt và có thể gán các ngắt, ngoại trừ NMI (Ngắt Không Mặt Nạ), vào tám mức ưu tiên/mặt nạ để cho phép kiểm soát ngắt đa kênh. Nguồn kích hoạt xử lý ngoại lệ ngắt và địa chỉ vector khác nhau tùy thuộc vào sản phẩm. Để biết chi tiết, tham khảo mục 5, Bộ điều khiển ngắt.

Quá trình xử lý ngoại lệ ngắt như sau:

1. Các giá trị trong bộ đếm chương trình (PC), thanh ghi mã điều kiện (CCR), và thanh ghi mở rộng (EXR) được lưu vào ngăn xếp.
2. Bit mặt nạ ngắt được cập nhật và bit T được xóa về 0.
3. Một địa chỉ vector tương ứng với nguồn ngắt được tạo ra, địa chỉ bắt đầu được nạp từ bảng vector vào PC, và chương trình bắt đầu thực thi từ địa chỉ đó.

**2.3.1. Các khái niệm đặc biệt:**

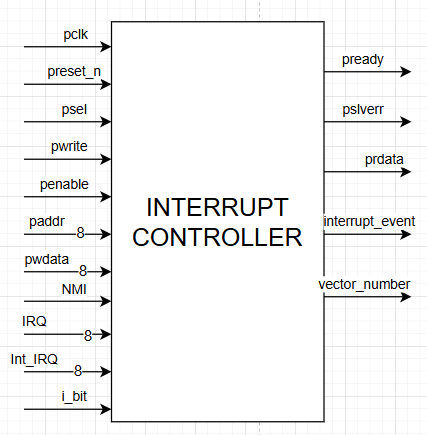
NMI(nonmaskable external interrupt): có tín hiệu ngắt xảy ra, thoát khỏi chương trình chính và bắt buộc nhảy qua chương trình con đó xong thì mới về lại chương trình chính. 🡪 Tín hiệu interrupt có độ ưu tiên cao nhất.

Maskable interrupt(internal and external): có tín hiệu ngắt xảy ra, nhưng chương trình chính vẫn hoạt động bình thường.

Exception handler: Quy định các vùng địa chỉ cho chương trình ngắt ứng với các điều kiện ngắt xảy ra.

Mỗi interrupt source(IRQ) sẽ có một vùng địa chỉ khác nhau. Sau khi xử lý xong chương trình đó(interrupt status là trạng thái không còn ngắt nữa) interrupt controller sẽ trả lại địa chỉ PC+4.

VD: có tín hiệu ngắt của vùng 0x06 thì chương trình sẽ tạm dừng chương trình chính, nó sẽ nhảy vào chương trình con đó và thực thi chuỗi lệnh. Sau khi cờ ngắt được xóa đi(tương đương với việc cờ ngắt đã được mask lại), chương trình ngắt đã được thực thi xong, và con trỏ sẽ quay lại chương trình chính.



Hình 2- 6 Sơ đồ khối chung của interrupt controller

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| Pclk | 1 | Input | Clock. Cạnh lên của pclk biển hiện tất cả transfer trên APB |
| Preset\_n | 1 | Input | Reset. Tích cực thấp. Tín hiệu này nối trực tiếp với system bus reset signal |
| Psel | 1 | Input | Tạo ra tín hiệu tới bus slave peripheral. |
| Penable | 1 | Input | Được bật lên tùy theo Wait\_cycle của của APB(thường là cycle thứ 2 hoặc sau đó) |
| Paddr | 8 | Input | Địa chỉ APB bus. Có thể lên tới 32 bit và được lái bởi peripheral bus bridge unit |
| Pwdata | 8 | Input | Lái bởi peripheral bus bridge unit trong chu kỳ ghi khi pwrite = 1 |
| pwrite | 1 | Input | Cho phép ghi APB khi tích cực cao và đọc khi tích cực thấp |
| NMI | 1 | Input | Ngắt ngoại không thể bị mask |
| IRQ | 8 | Input | Ngắt ngoại có thể bị mask |
| Int\_IRQ | 8 | Input | Ngắt nội |
| i\_bit | 1 | Input | Bit được gửi từ CPU cho biết interrupt source đó có được xử lý hay không? |
| vector\_number | 5 | Output | Địa chỉ của chương trình ngắt |
| Intr\_ev | 1 | Output | Tín hiệu ngắt cuối cùng gửi tới CPU |
| Prdata | 8 | Output | Slave được chọn lái bus này trong chu kỳ đọc khi pwrite =0 |
| Pready | 1 | Output | Slave sử dụng tín hiệu này để kéo dài APB transfer |
| Pslverr | 1 | Output | Báo lỗi về sự truy cập không thành công(không có địa chỉ thanh ghi) |

Bảng 1 Các tín hiệu của interrupt controller

# 3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

## 3.1. Ý tưởng thiết kế CPU

1. **Các tầng của pipeline:**

Nhóm sẽ thiết kế một bộ xử lý theo đường ống năm giai đoạn bao gồm:

1. IF (Instruction Fetch): Lấy lệnh từ bộ nhớ.

2. ID (Instruction Decode): Đọc các thanh ghi và giải mã lệnh.

3. EX (Execute): Thực hiện các phép toán hoặc tính toán địa chỉ.

4. MEM (Memory Access): Truy cập các toán hạng trong bộ nhớ dữ liệu.

5. WB (Write-back): Ghi kết quả trở lại các thanh ghi.

Đây là một thiết kế theo đường ống cơ bản, dễ triển khai và được phân vùng dựa trên bộ xử lý chu kỳ đơn được thiết kế trước đó. Có nhiều hazard khác nhau mà nhóm sẽ xử lý, chẳng hạn như Data Hazard, Load\_Store Hazard, Control Hazard.

* Khối IF\_ID:

| **Signal** | **Type** | **Size** | **Description** |
| --- | --- | --- | --- |
| i\_clk | Input | 1 | Clock signal to trigger the pipeline register updates. |
| i\_rst\_n | Input | 1 | Active-low reset signal to initialize the pipeline registers. |
| enable | Input | 1 | Enable signal to allow updating of the pipeline registers. |
| pc\_IF | Input | 32 | Program counter value from the IF stage. |
| instr\_IF | Input | 32 | Instruction fetched from the IF stage. |
| pc\_ID | Output | 32 | Program counter value passed to the ID stage. |
| instr\_ID | Output | 32 | Instruction passed to the ID stage. |
| i\_pc\_sel\_BTB | Input | 1 | Predicted whether the next pc is jumped or not from the IF stage |
| o\_pc\_sel\_BTB | Output | 1 | Predicted whether the next pc is jumped or not to the ID stage |
| i\_predicted\_pc | Input | 32 | Predicted PC of BTB from the IF stage |
| o\_predicted\_pc | Output | 32 | Predicted PC of BTB to the ID stage |

Bảng 2 Các tín hiệu của khối IF\_ID

**Khối ID\_EX:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Size** | **Description** |
| i\_clk | Input | 1 | Clock signal to trigger the pipeline register updates. |
| i\_rst\_n | Input | 1 | Active-low reset signal to initialize the pipeline registers. |
| enable | Input | 1 | Enable signal to allow updating of the pipeline registers. |
| i\_insn\_vld | Input | 1 | Instruction valid signal to indicate if instruction is valid. |
| i\_pc | Input | 32 | Program counter value from the ID stage. |
| i\_instr | Input | 32 | Instruction passed to the ID stage. |
| i\_rd\_wren | Input | 1 | Register write enable signal for the ID stage. |
| i\_ld\_en | Input | 3 | Load enable signal for the ID stage. |
| i\_opa\_sel | Input | 1 | Operand A selection signal for the ALU. |
| i\_opb\_sel | Input | 1 | Operand B selection signal for the ALU. |
| i\_lsu\_wren | Input | 1 | Load/store unit write enable signal. |
| i\_alu\_op | Input | 4 | ALU operation selection signal. |
| i\_br\_un | Input | 1 | Branch uncondtional signal for branch prediction. |
| i\_wb\_sel | Input | 2 | Write-back selection signal for the write-back stage. |
| i\_rs1\_data | Input | 32 | Data from register Rs1. |
| i\_rs2\_data | Input | 32 | Data from register Rs2. |
| o\_insn\_vld | Output | 1 | Output signal for instruction validity. |
| o\_pc | Output | 32 | Program counter output for the EX stage. |
| o\_instr | Output | 32 | Instruction passed to the EX stage. |
| o\_rd\_wren | Output | 1 | Register write enable signal for the EX stage. |
| o\_ld\_en | Output | 3 | Load enable signal for the EX stage. |
| o\_opa\_sel | Output | 1 | Operand A selection signal for the ALU in the EX stage. |
| o\_opb\_sel | Output | 1 | Operand B selection signal for the ALU in the EX stage. |
| o\_lsu\_wren | Output | 1 | Load/store unit write enable signal in the EX stage. |
| o\_alu\_op | Output | 4 | ALU operation selection signal in the EX stage. |
| o\_br\_un | Output | 1 | Branch uncondtional signal for branch prediction in the EX stage. |
| o\_wb\_sel | Output | 2 | Write-back selection signal in the EX stage. |
| i\_pc\_sel\_BTB | Input | 1 | Predicted whether the next pc is jumped or not from the ID stage |
| o\_pc\_sel\_BTB | Output | 1 | Predicted whether the next pc is jumped or not to the EX stage |
| i\_predicted\_pc | Input | 32 | Predicted PC of BTB from the ID stage |
| o\_predicted\_pc | Output | 32 | Predicted PC of BTB to the EX stage |

Bảng 3 Các tín hiệu của khối ID\_EX

**Khối EX\_MEM:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Size** | **Description** |
| i\_clk | Input | 1 | Clock signal for synchronization |
| i\_rst\_n | Input | 1 | Active low reset signal |
| enable | Input | 1 | Enable signal for operation |
| i\_insn\_vld | Input | 1 | Instruction valid flag |
| i\_pc | Input | 32 | Program counter value at the EX stage |
| i\_rs2\_data | Input | 32 | Data from rs2 operand at EX stage |
| i\_instr | Input | 32 | Instruction at EX stage |
| i\_ld\_en | Input | 3 | Load enable signal |
| i\_lsu\_wren | Input | 1 | LSU write enable signal |
| i\_rd\_wren | Input | 1 | Register write enable signal |
| i\_wb\_sel | Input | 2 | Write-back selection (which data to write back) |
| i\_alu\_data | Input | 32 | ALU result at EX stage |
| o\_insn\_vld | Output | 1 | Instruction valid flag |
| o\_pc | Output | 32 | Program counter value at MEM stage |
| o\_rs2\_data | Output | 32 | Data from rs2 operand at MEM stage |
| o\_instr | Output | 32 | Instruction at MEM stage |
| o\_ld\_en | Output | 3 | Load enable signal |
| o\_rd\_wren | Output | 1 | Register write enable signal |
| o\_lsu\_wren | Output | 1 | LSU write enable signal |
| o\_wb\_sel | Output | 2 | Write-back selection (which data to write back) |

Bảng 4 Các tín hiệu của khối EX\_MEM

**Khối MEM\_WB:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Size** | **Description** |
| i\_clk | Input | 1 | Clock signal for synchronization |
| i\_rst\_n | Input | 1 | Active low reset signal |
| enable | Input | 1 | Enable signal for operation |
| i\_pc | Input | 32 | Program counter value at the MEM stage |
| i\_instr | Input | 32 | Instruction at MEM stage |
| i\_insn\_vld | Input | 1 | Instruction valid flag |
| i\_rd\_wren | Input | 1 | Register write enable signal |
| i\_ld\_data | Input | 32 | Load data at MEM stage |
| i\_wb\_sel | Input | 2 | Write-back selection (which data to write back) |
| i\_alu\_data | Input | 32 | ALU result at MEM stage |
| o\_pc | Output | 32 | Program counter value at WB stage |
| o\_instr | Output | 32 | Instruction at WB stage |
| o\_insn\_vld | Output | 1 | Instruction valid flag |
| o\_rd\_wren | Output | 1 | Register write enable signal |
| o\_ld\_data | Output | 32 | Load data at WB stage |
| o\_wb\_sel | Output | 2 | Write-back selection (which data to write back) |

Bảng 5 Các tín hiệu của khối MEM\_WB

Tuy nhiên, việc chỉ forward giá trị qua các tầng thanh ghi thôi là chưa đủ, vì vẫn còn các hazard phải giải quyết. Đó là lý do ta phải thiết kế khối hazard\_unit để phát hiện các hazard ấy.

**Khối Hazard\_unit:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Size** | **Description** |
| i\_pc\_sel | Input | 1 | Indicates branch condition: 1 for ALU data, 0 for PC+4. |
| ex\_rd\_wren | Input | 1 | Write enable signal for the register file in the EX stage. |
| mem\_rd\_wren | Input | 1 | Write enable signal for the memory (LSU) in the MEM stage. |
| wb\_rd\_wren | Input | 1 | Write enable signal for the register file in the WB stage. |
| ex\_rd\_addr | Input | 5 | Destination register address in the EX stage. |
| mem\_rd\_addr | Input | 5 | Destination register address in the MEM stage. |
| wb\_rd\_addr | Input | 5 | Destination register address in the WB stage. |
| id\_rs1\_addr | Input | 5 | Source register 1 address in the ID stage. |
| id\_rs2\_addr | Input | 5 | Source register 2 address in the ID stage. |
| id\_opcode | Input | 7 | Opcode of the instruction in the ID stage. |
| ex\_opcode | Input | 7 | Opcode of the instruction in the EX stage. |
| mem\_opcode | Input | 7 | Opcode of the instruction in the MEM stage. |
| stall\_ID | Output | 1 | Signal to stall the IF/ID pipeline stage. |
| stall\_EX | Output | 1 | Signal to stall the ID/EX pipeline stage. |
| stall\_MEM | Output | 1 | Signal to stall the EX/MEM pipeline stage. |
| stall\_WB | Output | 1 | Signal to stall the MEM/WB pipeline stage. |
| pc\_enable | Output | 1 | Enables the PC update: 0 for stall, 1 for normal operation. |
| flush\_ID | Output | 1 | Flushes the IF/ID pipeline stage. |
| flush\_EX | Output | 1 | Flushes the ID/EX pipeline stage. |
| flush\_MEM | Output | 1 | Flushes the EX/MEM pipeline stage. |
| flush\_WB | Output | 1 | Flushes the MEM/WB pipeline stage. |

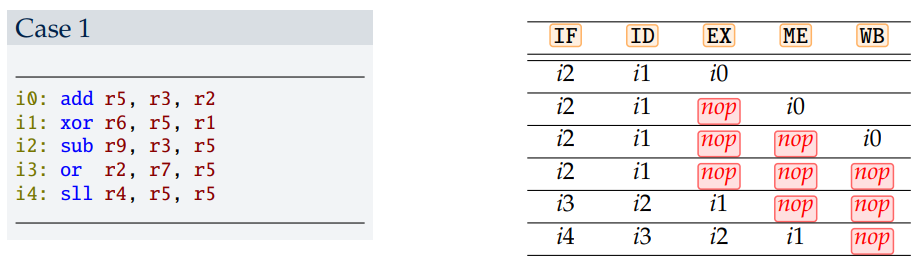
Bảng 6 Các tín hiệu của khối hazard\_unit

Forwarding unit đóng vai trò quan trọng trong việc giải quyết các Data hazard trong pipeline. Sau đây là cách thức hoạt động và cách quản lý các hazard khác nhau:

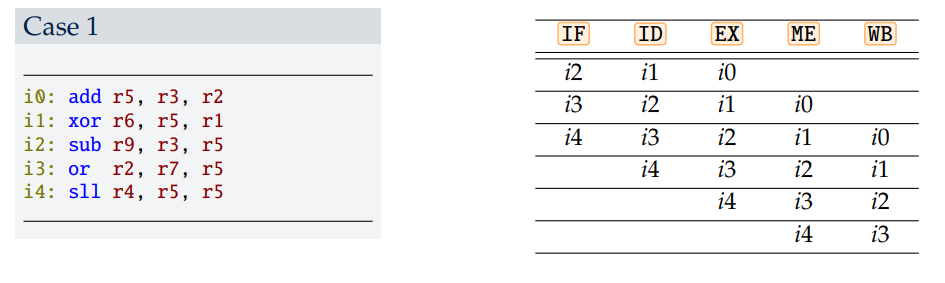
**Khối Forwarding\_unit:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Size** | **Type** | **Description** |
| instr\_MEM | 32 | Input | Instruction in the MEM stage, containing destination register information. |
| instr\_WB | 32 | Input | Instruction in the WB stage, containing destination register information. |
| instr\_EX | 32 | Input | Instruction in the EX stage, containing source register information currently being processed. |
| rd\_wren\_MEM | 1 | Input | Write-enable signal for the destination register in the MEM stage. |
| rd\_wren\_WB | 1 | Input | Write-enable signal for the destination register in the WB stage. |
| forward\_ASel | 2 | Output | Selection signal for forwarding to the source operand A in the EX stage. |
| forward\_BSel | 2 | Output | Selection signal for forwarding to the source operand B in the EX stage. |

Bảng 7 Các tín hiệu của khối forwarding\_unit



Hình 3- 1 Data hazard nếu không có Forwarding Unit



Hình 3- 2 Data hazard sau khi giải quyết bằng Forwarding Unit

Hazard khi tải/lưu trữ dữ liệu: hazard trong quá trình truy cập dữ liệu (ví dụ: tải dữ liệu vào bộ nhớ) được xử lý bằng cách forwarding. Tuy nhiên, hazard sử dụng sau khi tải xảy ra khi lệnh tải ghi dữ liệu vào thanh ghi và lệnh tiếp theo (sử dụng thanh ghi đó) cần dữ liệu đó. Không thể giải quyết vấn đề này bằng cách chuyển tiếp vì dữ liệu từ bộ nhớ vẫn chưa khả dụng tại thời điểm lệnh tiếp theo thực thi.

Để xử lý vấn đề này, pipeline sẽ dừng trong một chu kỳ để đảm bảo dữ liệu chính xác khả dụng. Điều này được quản lý bởi một đơn vị stall\_check, đơn vị này phát hiện ra rủi ro này và đưa ra lệnh NOP (không hoạt động) cho chu kỳ đó.

Với Control Hazard, để thiết kế để tối ưu nhất, nhóm quyết định sử dụng mode 2-bit Predictor đã được học ở bộ môn cấu trúc máy tính.

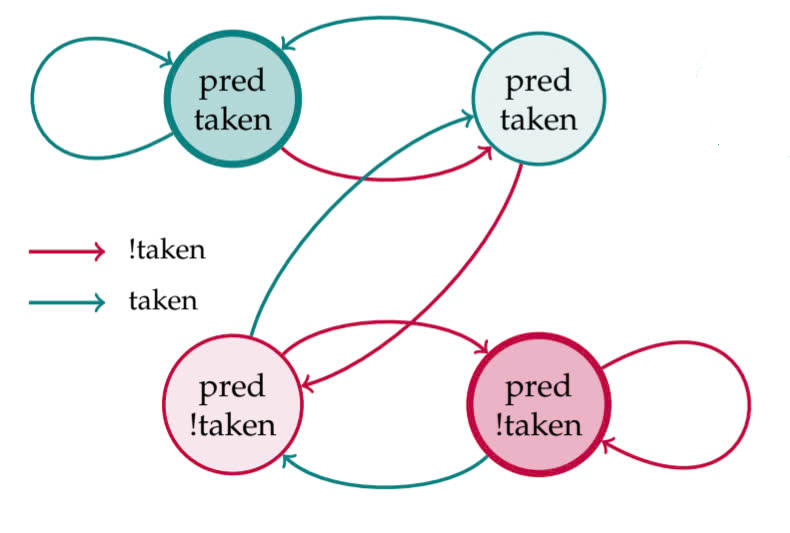
**Khối branch prediction:**

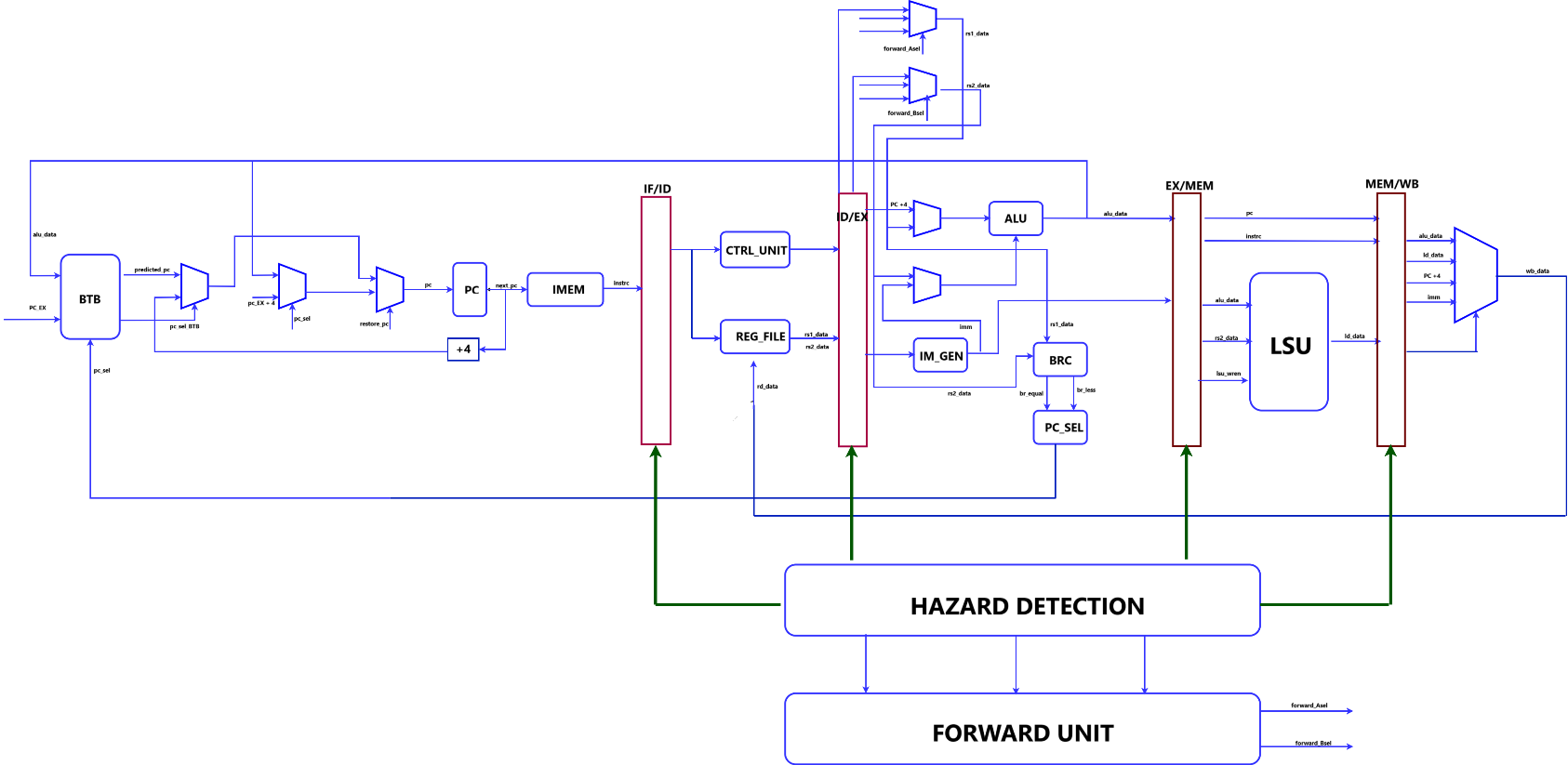
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Type** | **Size** | **Description** |
| i\_clk | Input | 1 | Clock signal to synchronize the operation of the branch predictor. |
| i\_rst\_n | Input | 1 | Active-low reset signal to initialize the predictor. |
| i\_alu\_data | Input | 32 | Data from the ALU used for branch calculations. |
| instr\_IF | Input | 32 | Instruction fetched in the instruction fetch (IF) stage. |
| instr\_EX | Input | 32 | Instruction in the execution (EX) stage. |
| pc\_IF | Input | 32 | Program counter (PC) value from the instruction fetch (IF) stage. |
| pc\_EX | Input | 32 | Program counter (PC) value from the execution (EX) stage. |
| i\_taken | Input | 1 | Signal indicating whether the branch was taken or not. |
| o\_pc | Output | 32 | Predicted program counter (PC) value based on the branch prediction algorithm. |
| o\_pc\_sel\_BTB | Output | 1 | Signal indicating whether the PC should be selected from the Branch Target Buffer (BTB). |
| o\_mispred | Output | 1 | Signal indicating a misprediction occurred. |

Bảng 8 Các tín hiệu của khối branch\_prediction

Thuật toán 2-bit prediction là một phương pháp được sử dụng trong bộ xử lý theo pipeline để cải thiện độ chính xác của dự đoán nhánh.

Trong bộ xử lý theo pipeline, khi gặp lệnh rẽ nhánh, việc dự đoán nhánh đó có được thực hiện hay không là rất quan trọng để duy trì tính liên tục của lệnh thực thi. Nếu dự đoán nhánh đúng, bộ xử lý có thể tiếp tục thực thi lệnh tiếp theo mà không cần chờ kết quả của lệnh rẽ nhánh. Mặt khác, nếu dự đoán không đúng, bộ xử lý phải thực hiện các thao tác để sửa dự đoán và chờ kết quả thực tế của lệnh rẽ nhánh được xác định.





Hình 3- 3 Sơ đồ khối của mạch 2-bit prediction với RISC-V

**b. IPC:**

- Giá trị IPC (Hướng dẫn trên mỗi chu kỳ) cho biết hiệu suất của bộ xử lý. Nếu bộ xử lý pipelined có quá nhiều chu kỳ "NOP" (Không hoạt động), IPC sẽ giảm theo công thức sau:

Với chương trình chuyển từ số HEX sang DEC và hiển thị lên LED 7 đoạn, ta có IPC sau:

|  |  |  |
| --- | --- | --- |
| Logic Utilization/Total logic elements | 8809 | 27% |
| Fmax | 43,71 MHz |  |

|  |  |
| --- | --- |
|  | 2-Bit Prediction |
| **Control Transfer Instruction** | 69 |
| **Mispredict** | 20 |
| **Accuracy** | 49 |

## 3.2. Thiết kế interrupt controller:

Interrupt controller bao gồm:

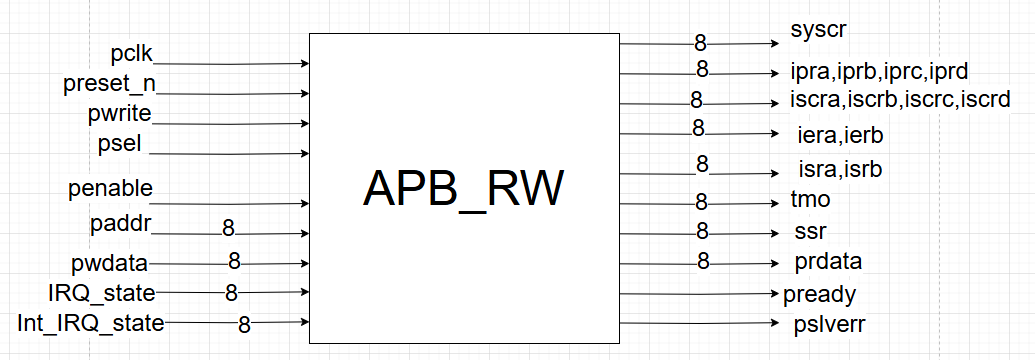
* Có giao thức APB bus interface của ARM bus
* Có thể được set mode cho SYSCR(NMI\_eg)
* Độ ưu tiên được set bởi IPRA,IPRB,IPRC,IPRD.

Bất cứ khi nào, quá trình interrupt khác(ngoại trừ NMI) đang xảy ra, cứ có tín hiệu NMI kích lên thì exception handler của interrupt sẽ phải dừng, và thực hiện exception handler của NMI.

* pslverr của giao thức APB(những địa chỉ khi qua bộ decoder của address mà không tồn tại).
* Time-out logic khi mà hệ thống đang bận. Khi đang có interrupt request đang được thực thi trong interrupt handler mà chương trình đó quá lâu thì time out sẽ đếm xuống.

Các khối của Interrupt controller:

1. **Khối ghi đọc theo giao thức APB(apb\_rw):**



Hình 3- 4 Sơ đồ khối của APB\_RW

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| Pclk | 1 | Input | Clock. Cạnh lên của pclk biển hiện tất cả transfer trên APB |
| Preset\_n | 1 | Input | Reset. Tích cực thấp. Tín hiệu này nối trực tiếp với system bus reset signal |
| Pwrite | 1 | Input | Cho phép ghi APB khi tích cực cao và đọc khi tích cực thấp |
| Psel | 1 | Input | Tạo ra tín hiệu tới bus slave peripheral. |
| Penable | 1 | Input | Được bật lên tùy theo Wait\_cycle của của APB(thường là cycle thứ 2 hoặc sau đó) |
| Paddr | 8 | Input | Địa chỉ APB bus. Có thể lên tới 32 bit và được lái bởi peripheral bus bridge unit |
| Pwdata | 8 | Input | Lái bởi peripheral bus bridge unit trong chu kỳ ghi khi pwrite = 1 |
| Prdata | 8 | Output | Slave được chọn lái bus này trong chu kỳ đọc khi pwrite =0 |
| Pready | 1 | Output | Slave sử dụng tín hiệu này để kéo dài APB transfer |
| Pslverr | 1 | Output | Báo lỗi về sự truy cập không thành công(không có địa chỉ thanh ghi) |
| Syscr | 8 | Output | Xác định loại ngắt NMI nào được bắt và báo tín hiệu timeout và slverr |
| Ipra, iprb, iprc, iprd | 8 | Output | Set mức độ ưu tiên của tín hiệu ngắt(ngoại trừ NMI) |
| Iscra, iscrb | 8 | Output | Bắt loại ngắt  IRQ[1:0]/Int\_IRQ[1:0]:  00: Yêu cầu ngắt được tạo ra nếu ngõ vào mức thấp  01: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh xuống  10: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh lên  11: Yêu cầu ngắt được tạo ra ở cả cạnh lên và xuống |
| Iera, ierb | 8 | Output | Điều khiển sự cho phép của ngắt nội, ngắt ngoại |
| Isra, isrb | 8 | Output | Chỉ rõ trạng thái của các tín hiệu ngắt(ngắt nội và ngoại) |
| Tmo | 8 | Output | Thanh ghi đếm xuống khi tín hiệu ngắt xảy ra |
| Ssr | 8 | Output |  |
| IRQ\_state | 8 | Output | Trạng thái ngắt ngoại |
| Int\_IRQ\_state | 8 | Input | Trạng thái ngắt nội |

Bảng 9 Các tín hiệu đầu vào và ra của APB\_RW

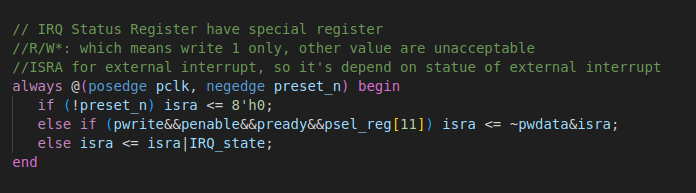
**Chú ý:**

Thanh ghi ISRA, ISRB:

R/W\*: Chỉ có thể ghi 0 vào được thôi, nếu ghi 1 vào sẽ xóa cờ.

* Điều kiện bật lên(Setting condition)

+ Khi có interrupt source được chọn bởi các thanh ghi ISCRA, ISCRB, ISCRC, ISCRD xảy ra thì các trạng thái này của thanh ghi sẽ được tự động kích hoạt lên 1.



Hình 3- 5 Thực thi cách ghi data của thanh ghi ISRA/ISRB

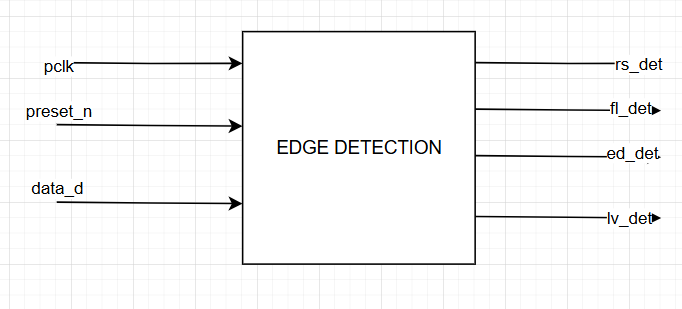
**Điều kiện xóa cờ(clearing conditions)**

Các bit của thanh ghi được xóa bởi việc ta ghi bit 1 vào thanh ghi ấy.

Có ba trường hợp dẫn đến trường hợp chương trình bước vào chương trình ngắt:

* Nếu nguồn ngắt là mức thấp(low level) và tín hiệu IRQ được phát hiện là mức thấp
* Nếu nguồn ngắt là cạnh lên(rising edge) và tín hiệu IRQ được phát hiện là cạnh lên
* Nếu nguồn ngắt là cạnh xuống(falling edge) và tín hiệu IRQ được phát hiện là cạnh xuống
* Nếu nguồn ngắt là cạnh(both edge) và tín hiệu IRQ được phát hiện là cạnh(lên hay xuống đều được).

1. **Khối phát hiện cạnh: (edge\_detection)**



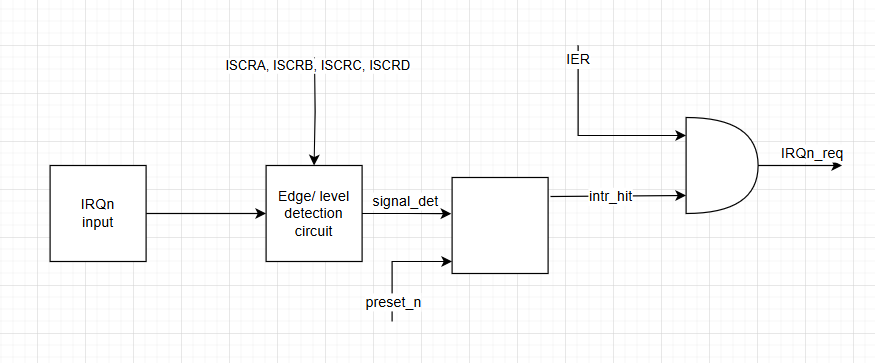
Hình 3- 6 Sơ đồ khối của mạch phát hiện loại tín hiệu mà ngắt có thể bắt

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| Pclk | 1 | Input | Clock. Cạnh lên của pclk biển hiện tất cả transfer trên APB |
| Preset\_n | 1 | Input | Reset. Tích cực thấp. Tín hiệu này nối trực tiếp với system bus reset signal |
| Data\_d | 1 | Input | Ngõ vào |
| Rs\_det | 1 | Output | Tín hiệu phát hiện có cạnh lên |
| Fl\_det | 1 | Output | Tín hiệu phát hiện có cạnh xuống |
| Ed\_det | 1 | Output | Tín hiệu phát hiện cạnh lên hoặc xuống |
| Lv\_det | 1 | Output | Tín hiệu phát hiện mức thấp tín hiệu |

Bảng 10 Các tín hiệu đầu vào và ra của edge\_detection

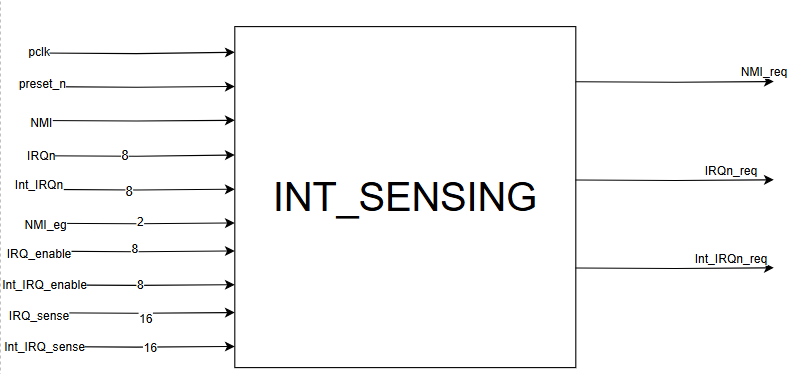
1. **Khối yêu cầu các tín hiệu ngắt: (int\_sensing)**

Khối tín hiệu ngắt theo mô hình chung:



Hình 3- 7 Mạch phát hiện ngắt

Từ quy tắc mô hình chung đó, nhóm quyết định xây dựng mạch phát hiện interrupt source theo sơ đồ khối như hình:



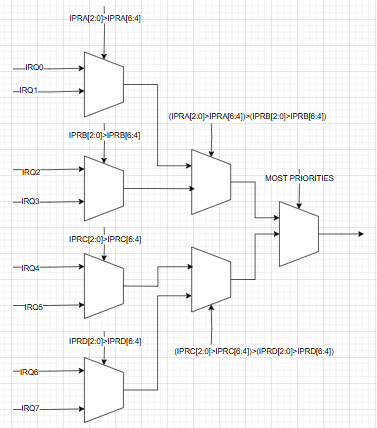
Hình 3- 8 Sơ đồ khối của mạch đã phát hiện ngắt và gửi tín hiệu yêu cầu ngắt

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| Pclk | 1 | Input | Clock. Cạnh lên của pclk biển hiện tất cả transfer trên APB |
| Preset\_n | 1 | Input | Reset. Tích cực thấp. Tín hiệu này nối trực tiếp với system bus reset signal |
| NMI | 1 | Input | Ngắt ngoại không thể bị mask |
| IRQn | 8 | Input | Ngắt ngoại có thể bị mask |
| Int\_IRQn | 8 | Input | Ngắt nội |
| NMI\_eg | 2 | Input | Chọn loại ngắt cho NMI |
| IRQ\_enable | 8 | Input | Cho phép ngắt ngoại(có 8 ngắt ngoại) |
| Int\_IRQ\_enable | 8 | Input | Cho phép ngắt nội(có 8 ngắt ngoại) |
| IRQ\_sense | 16 | Input | Bắt loại ngắt nội  00: Yêu cầu ngắt được tạo ra nếu ngõ vào mức thấp  01: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh xuống  10: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh lên  11: Yêu cầu ngắt được tạo ra ở cả cạnh lên và xuống |
| Int\_IRQ\_sense | 16 | Input | Bắt loại ngắt ngoại  00: Yêu cầu ngắt được tạo ra nếu ngõ vào mức thấp  01: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh xuống  10: Yêu cầu ngắt được tạo ra nếu ngõ vào là cạnh lên  11: Yêu cầu ngắt được tạo ra ở cả cạnh lên và xuống |
| NMI\_req | 1 | Output | Yêu cầu ngắt của NMI |
| IRQn\_req | 1 | Output | Yêu cầu ngắt của ngắt ngoại |
| Int\_IRQn\_req | 1 | Output | Yêu cầu ngắt của ngắt nội |

Bảng 11 Bảng giá trị của mạch phát hiện yêu cầu ngắt

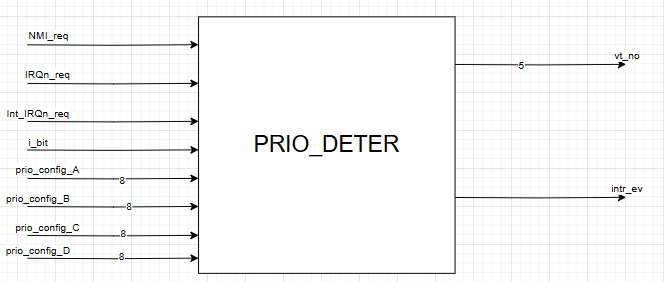
1. **Khối xác định mức độ ưu tiên của ngắt: (prio\_deter)**

Theo lý thuyết, bộ ngắt sẽ chọn giá trị nào có độ ưu tiên cao nhất. Như vậy, nếu có nhiều hơn một tín hiệu ngắt cùng xảy ra, ta sẽ phải tìm cách cho ra interrupt source nào có độ ưu tiên cao nhất, còn các interrupt source còn lại sẽ đợi đến khi interrupt source kia được xử lý xong thì sẽ xử lý. Từ đó, nhóm xây dựng mạch sắp xếp như sau:



Hình 3- 9 Mạch sắp xếp để chọn ra tín hiệu ngắt nào có độ ưu tiên cao nhất

Cấu trúc mạch trên sẽ vừa chọn ra nguồn interrupt nào sẽ được xử lý và cả vector địa chỉ của interrupt đó. Từ đó, nhóm sẽ có mạch xác định độ ưu tiên các interrupt source như sau:



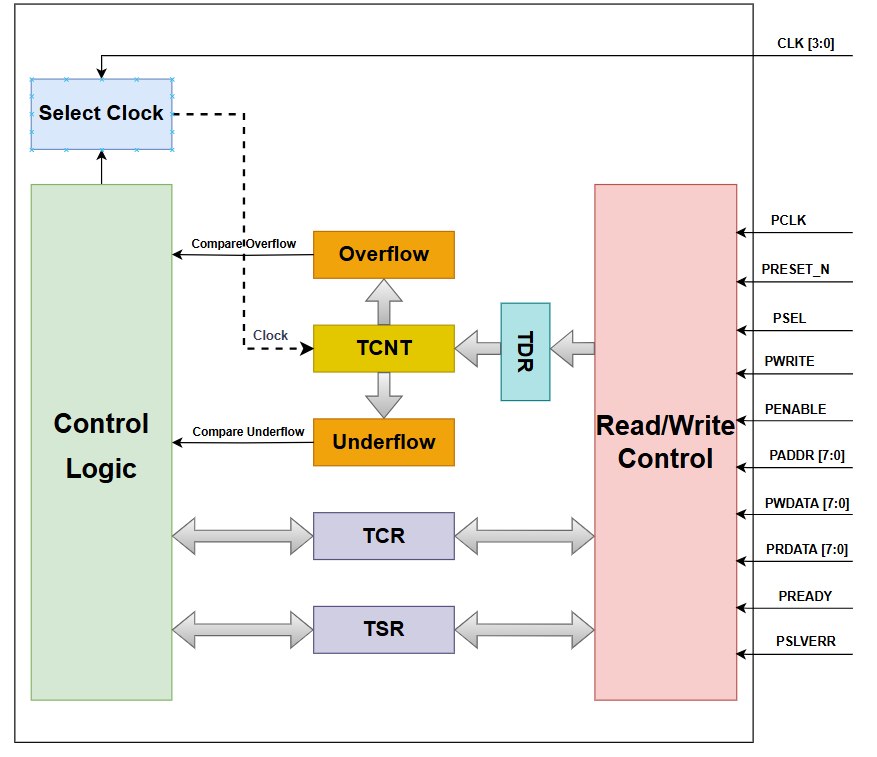
**Hình 3.11:** Mạch tìm ra ngắt có độ ưu tiên cao nhất đi kèm với đó là địa chỉ của nguồn ngắt

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| NMI\_req | 1 | Input | Yêu cầu ngắt NMI |
| IRQn\_req | 1 | Input | Yêu cầu ngắt ngoại |
| Int\_IRQn\_req | 1 | Input | Yêu cầu ngắt nội |
| I\_bit | 1 | Input | Bit được gửi từ CPU cho biết interrupt source đó có được xử lý hay không? |
| Prio\_config\_A | 8 | Input | Thanh ghi chứa mức độ ưu tiên(là thanh ghi IPRA) |
| Prio\_config\_B | 8 | Input | Thanh ghi chứa mức độ ưu tiên(là thanh ghi IPRB) |
| Prio\_config\_C | 8 | Input | Thanh ghi chứa mức độ ưu tiên(là thanh ghi IPRC) |
| Prio\_config\_D | 8 | Input | Thanh ghi chứa mức độ ưu tiên(là thanh ghi IPRD) |
| Vt\_no | 5 | Output | Địa chỉ của chương trình ngắt |
| Intr\_ev | 1 | Output | Tín hiệu ngắt cuối cùng gửi tới CPU |

Bảng 12 Các tín hiệu của mạch Prio\_deter

## 3.3 Thiết kế Timer 8 bit

**3.3.1 Sơ đồ khối**

****

Hình 3- 10 Sơ đồ khối của IP TIMER 8 BIT

**3.3.2 Các thanh ghi được sử dụng**

TDR (8-bit)

Timer Data Register

TCNT (8-bit)

Timer Counter

TCR (8-bit)

Timer Control Register

TSR (8-bit)

Timer Status Register

\

Hình 3- 11Các thanh ghi sử dụng trong thiết kế

**TCNT (Timer Counter Register):**

* Lưu giữ giá trị đếm hiện tại của bộ đếm thời gian.
* Bộ đếm thời gian tăng hoặc giảm giá trị của thanh ghi này dựa trên các thiết lập trong TCR.
* Có thể được tải giá trị từ TDR dựa trên các tín hiệu điều khiển.

**TDR (Timer Data Register):**

* Lưu trữ giá trị sẽ được nạp vào TCNT khi tín hiệu tải trong TCR được kích hoạt.
* Được sử dụng để đặt trước giá trị khởi đầu cho bộ đếm.

**TCR (Timer Control Register)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit position** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit function** | **Load** | **x** | **Up/Dw** | **enable** | **x** | **x** | **clks** | **clks** |

Hình 3- 12 Cấu trúc của thanh ghi TCR

* **Bit 7 (load):** Bit này điều khiển việc nạp giá trị từ **TDR (Timer Data Register)** vào **TCNT**. Khi được đặt là 1, bộ đếm thời gian sẽ nạp giá trị từ TDR.
* **Bit 6 (Unused/X):** Bit này không được sử dụng (được ký hiệu là X trong sơ đồ).
* **Bit 5 (Up/Down):** Bit này điều khiển việc đếm tăng hoặc đếm giảm của bộ đếm thời gian. Nếu được đặt là 0, bộ đếm sẽ đếm tăng. Nếu đặt là 1, bộ đếm sẽ đếm giảm.
* **Bit 4 (enable):** Bit này điều khiển việc kích hoạt bộ đếm thời gian. Khi được đặt là 1, bộ đếm sẽ hoạt động. Khi đặt là 0, bộ đếm sẽ ngừng hoạt động.
* **Bits 3 và 2 (Unused/X):** Các bit này không được sử dụng.
* **Bits 1-0 (clks):** Hai bit này được sử dụng để chọn xung nhịp. Giá trị của **clks** xác định hệ số chia xung nhịp được sử dụng cho bộ đếm thời gian.

Điều khiển hoạt động của bộ đếm thời gian, bao gồm bật/tắt bộ đếm, chọn nguồn xung nhịp và thiết lập chế độ (đếm tăng hoặc đếm giảm). Các bit trong thanh ghi này quyết định cách bộ đếm thời gian hoạt động.

**TSR (Timer Status Register)**

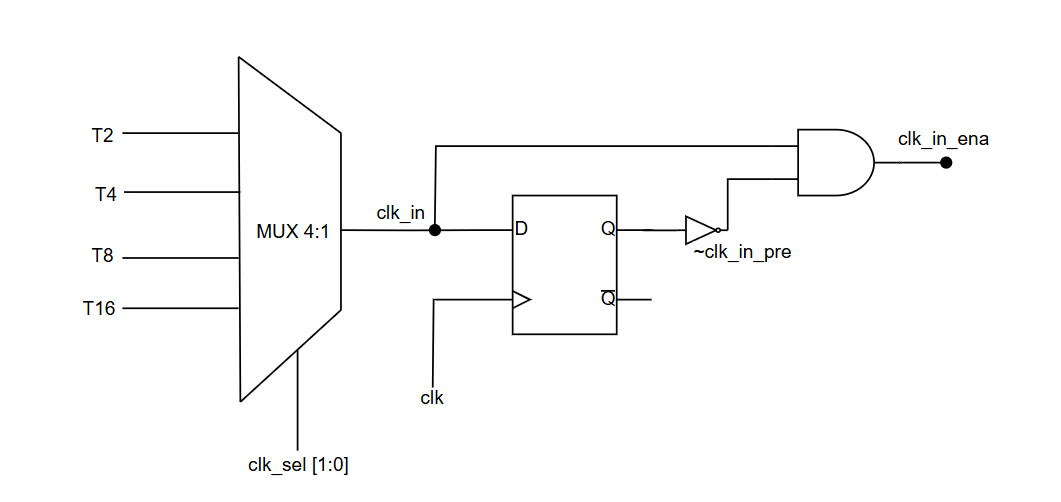
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit position** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit function** |  |  |  |  |  |  | **Underflow** | **Overflow** |

Hình 3- 13 Cấu trúc của thanh ghi TSR

* **Overflow detection(bit 0):** Nếu giá trị của **TCNT** chuyển từ 255 về 0, overflow flag sẽ được bật.
* **Underflow detection(bit 1):** Nếu giá trị của **TCNT** chuyển từ 0 lên 255, underflow flag sẽ được bật.

Phản ánh trạng thái của bộ đếm thời gian, bao gồm các cờ: overflow và underflow. Các cờ này có thể được xóa hoặc đọc dựa trên logic điều khiển cụ thể.

**3.3.3 Clock Selection**

****

Hình 3- 14 Mô tả hoạt động của Clock Selection

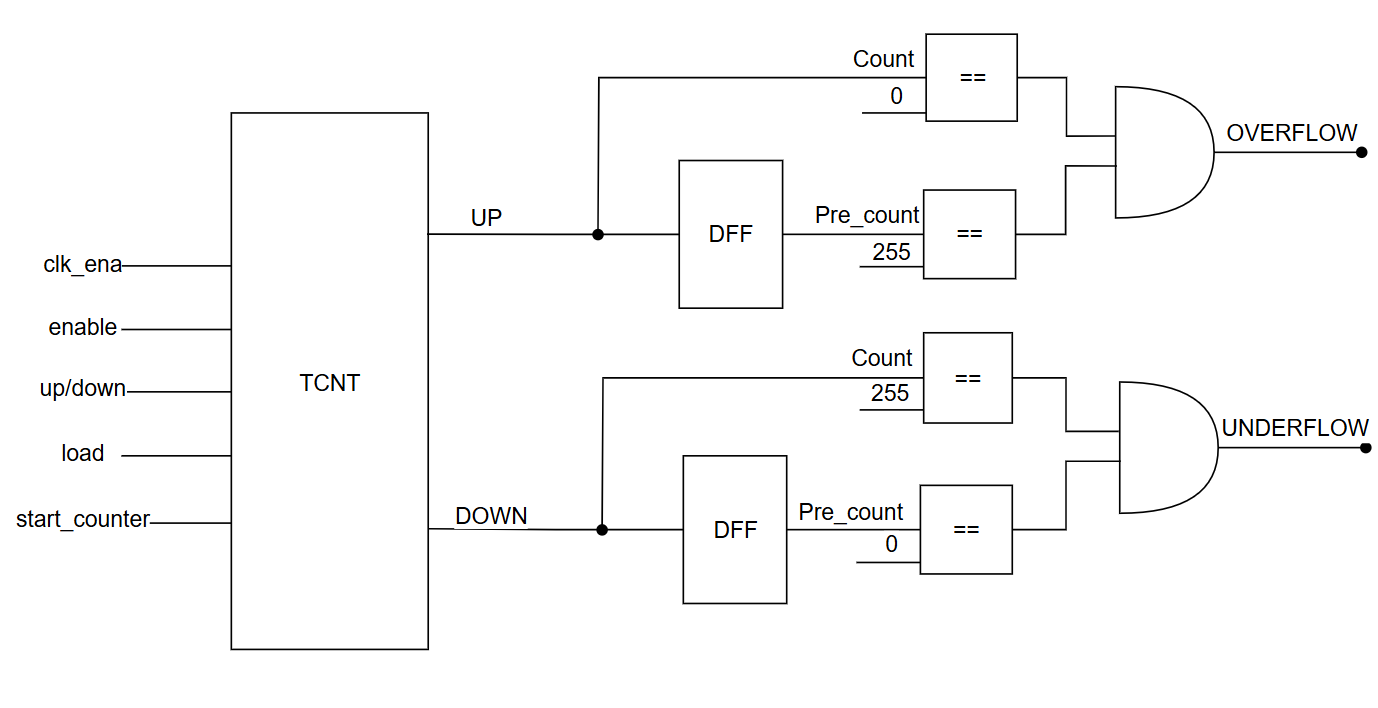
**Ngõ vào:**

* **clk\_sel [1:0]:** Tín hiệu chọn 2-bit được sử dụng để chọn xung nhịp nào từ **clk\_in** sẽ được kích hoạt.
* **clk\_in [3:0]:** Tín hiệu đầu vào 4-bit đại diện cho bốn xung nhịp khác nhau.

**Ngõ ra:**

* **clk\_ena:** Tín hiệu xung nhịp được kích hoạt, được xác định bởi giá trị của **clk\_sel**
* Khi **clk\_sel = 0**, **clk\_ena** nhận giá trị từ **clk\_in\_ena[0]**.
* Khi **clk\_sel = 1**, **clk\_ena** nhận giá trị từ **clk\_in\_ena[1]**.
* Khi **clk\_sel = 2**, **clk\_ena** nhận giá trị từ **clk\_in\_ena[2]**.
* Khi **clk\_sel = 3**, **clk\_ena** nhận giá trị từ **clk\_in\_ena[3]**.
* Nếu **clk\_sel** không hợp lệ, **clk\_ena** sẽ được đặt bằng 0

**3.3.4 Timer Counting Logic**

****

Hình 3- 15 Mô tả hoạt động của bộ đếm Timer

**Ngõ vào:**

* **clk\_ena:** Tín hiệu cho phép clock.
* **start\_counter:** Giá trị khởi tạo cho bộ đếm (8-bit).
* **up\_down:** Tín hiệu xác định đếm lên hay đếm xuống, tăng (1) hoặc giảm (0).
* **load:** Tín hiệu nạp giá trị vào bộ đếm.
* **enable:** Tín hiệu kích hoạt bộ đếm.

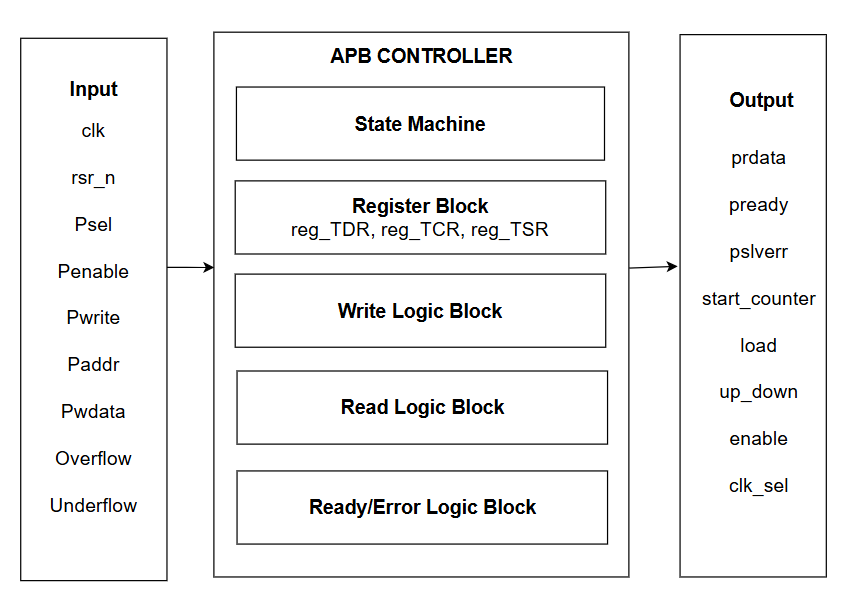
**Ngõ ra:**

* **overflow:** Tín hiệu tràn số (1-bit).
* **underflow:** Tín hiệu hụt số (1-bit).

**Cách thức hoạt động:**

* Khi **load** được kích hoạt, **count** được nạp giá trị từ **start\_counter**.
* Khi **enable** và **clk\_ena** được kích hoạt:
  + Nếu **up\_down** là 1, **count** được tăng thêm 1.
  + Nếu **up\_down** là 0, **count** được giảm đi 1.
* Giá trị của **prev\_count** được cập nhật từ **count**.
* **Overflow:** Kích hoạt khi **count = 0** và **prev\_count = 255**.
* **Underflow :** Kích hoạt khi **count = 255** và **prev\_count = 0**.

**3.3.5 APB Controller**

****

Hình 3- 16Cách thức hoạt động của APB controller

**Ngõ vào**

 **clk:** Tín hiệu clock.

 **rst\_n:** Tín hiệu reset.

 **psel:** Tín hiệu chọn ngoại vi.

 **penable:** Tín hiệu cho phép hoạt động.

 **pwrite:** Tín hiệu xác định hoạt động ghi dữ liệu.

 **paddr [7:0]:** Địa chỉ của ngoại vi.

 **pwdata [7:0]:** Dữ liệu được ghi vào ngoại vi.

 **overflow:** Tín hiệu báo hiệu dữ liệu bị overflow

 **underflow:** Tín hiệu báo hiệu dữ liệu bị underflow

**Máy trạng thái:**

**state**: Biến lưu trữ trạng thái hiện tại (IDLE, SETUP, ACCESS).

**next\_state**: Biến lưu trữ trạng thái tiếp theo dựa trên các điều kiện.

**IDLE**:

* Nếu cả psel bằng 1 và penable bằng 0, máy trạng thái chuyển sang trạng thái **SETUP**.
* Nếu không, máy trạng thái vẫn ở trạng thái **IDLE**.

**SETUP**:

* Nếu penable bằng 1, bộ máy trạng thái chuyển sang trạng thái **ACCESS**.
* Nếu không, nó quay lại trạng thái **IDLE**.

**ACCESS**: Sau khi hoàn thành thao tác truy cập, bộ máy trạng thái quay lại trạng thái **IDLE**.

**Thanh ghi:**

* **reg\_TDR:** Thanh ghi Dữ liệu Bộ đếm (Timer Data Register).
* **reg\_TCR:** Thanh ghi Điều khiển Bộ đếm (Timer Control Register).
* **reg\_TSR:** Thanh ghi Trạng thái Bộ đếm (Timer Status Register).
* **Khối này lưu trữ và quản lý các giá trị cần thiết dựa trên các thao tác đọc/ghi.**

**Write Logic Block:**

Nếu hệ thống ở trạng thái **ACCESS** và các tín hiệu **psel**, **penable**, và **pwrite** đều được kích hoạt, thì sẽ xảy ra thao tác ghi dữ liệu.

**paddr** xác định thanh ghi nào sẽ nhận giá trị từ **pwdata**:

* Nếu **paddr = 8'h0**, giá trị của **pwdata** sẽ được ghi vào thanh ghi **reg\_TCR**.
* Nếu **paddr = 8'h1**, giá trị của **pwdata** sẽ được ghi vào thanh ghi **reg\_TDR**.
* Nếu **paddr = 8'h2**, giá trị của **pwdata** sẽ được ghi vào thanh ghi **reg\_TSR**.

Nếu xảy ra hiện tượng **overflow** hoặc **underflow**, giá trị của **reg\_TSR** sẽ được cập nhật với các bit tương ứng.

**Read Logic Block:**

Nếu hệ thống ở trạng thái **ACCESS**, và cả hai tín hiệu **psel** và **penable** đều được kích hoạt, trong khi **pwrite** không được kích hoạt (0), thao tác **đọc** sẽ xảy ra.

Tín hiệu **paddr** xác định thanh ghi nào sẽ được đọc:

* Nếu **paddr = 0**, giá trị của thanh ghi **reg\_TDR** sẽ được đưa vào **prdata**.
* Nếu **paddr = 1**, giá trị của thanh ghi **reg\_TCR** sẽ được đưa vào **prdata**.
* Nếu **paddr = 2**, giá trị của thanh ghi **reg\_TSR** sẽ được đưa vào **prdata**.
* Nếu **paddr** có địa chỉ không hợp lệ, giá trị của **prdata** sẽ được đặt lại về 0.

**Ready/Error Logic Block:**

**PREADY:** Khi ở trạng thái **ACCESS**, tín hiệu **pready** sẽ được đặt thành 1; nếu không, nó sẽ được đặt thành 0.

**PSLVERR (Phát hiện lỗi):**

* Nếu hệ thống ở trạng thái **ACCESS**, và các tín hiệu **psel** và **penable** đều được kích hoạt, và địa chỉ **paddr** lớn hơn 7, tín hiệu lỗi **pslverr** sẽ được đặt thành 1.
* Nếu các điều kiện này không được thỏa mãn, **pslverr** sẽ được đặt thành 0.

**Ngõ ra:**

* **prdata [7:0]:** Dữ liệu đọc từ thiết bị ngoại vi.
* **pready:** Tín hiệu sẵn sàng (ready).
* **pslverr:** Tín hiệu lỗi (error).
* **start\_counter [7:0]:** Giá trị của thanh ghi **reg\_TDR**.
* **load:** Tín hiệu tải từ bit 7 của **reg\_TCR**.
* **up\_down:** Tín hiệu điều khiển tăng/giảm từ bit 5 của **reg\_TCR**.
* **enable:** Tín hiệu kích hoạt từ bit 4 của **reg\_TCR**.
* **clk\_sel [1:0]:** Tín hiệu chọn xung nhịp từ các bit [1:0] của **reg\_TCR**.

## 3.4 THIẾT KẾ UART

**3.4.1 Giới thiệu chuẩn giao thức UART**

Giao thức UART (Universal Asynchronous Receiver and Transmitter) là một chuẩn giao tiếp nối tiếp bất đồng bộ. Vào năm 1960, hiệp hội điện tử ETA đã phát triển một chuẩn giao tiếp, RS-232, được sử dụng cho truyền dữ liệu nối tiếp. Ngoài RS-232, còn có các chuẩn phổ biến khác như RS-422 và RS-485 được áp dụng cho mạch UART. Những chuẩn này mang lại tính thực tiễn cao hơn nhiều so với RS-232.

Một khung dữ liệu trong UART bao gồm: 1 start bit, 5-8 data bit, parity bit và 1 stop bit.Dữ liệu được truyền đi từng bit trong các khung, với bit đầu tiên được truyền là start bit, tiếp theo là các data bit, có thể bao gồm parity bit, và cuối cùng là stop bit.

A row of black text

Description automatically generated

Hình 3- 17 Data Framing Uart

Hình ảnh trên minh họa cấu trúc truyền dữ liệu của UART khi truyền 8 bit trong mỗi khung dữ liệu, không sử dụng bit chẵn lẻ và sử dụng một bit dừng.

UART là một trong những kỹ thuật giao tiếp nối tiếp đơn giản nhất và được sử dụng phổ biến nhất. UART xuất hiện trong hầu hết các vi điều khiển để đáp ứng các yêu cầu về độ chính xác. Ngày nay, UART được sử dụng trong nhiều ứng dụng như máy thu GPS, mô-đun Bluetooth, mô-đun GSM và GPRS, hệ thống giao tiếp không dây, và các ứng dụng dựa trên RFID.

A diagram of a microcontroller

Description automatically generated

Hình 3- 18 Một số ứng dụng của UART

**3.4.2 Thiết kế APB UART**

**Các khái niệm quan trọng**

***Baudrate:*** Số bit truyền được trong 1s, ở truyền nhận không đồng bộ thì ở các bên truyền và nhận phải thống nhất Baudrate. Các thông số tốc độ Baudrate thường hay sử dụng để giao tiếp với máy tính là 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200.

***Frame:*** Ngoài việc giống nhau của tốc độ baud 2 thiết bị truyền nhận thì khung truyền của 2 bên cũng được cấu hình giống nhau. Khung truyền quy định số bit trong mỗi lần truyền, bit bắt đầu “Start bit”, các bit kết thúc (Stop bit), bit kiểm tra tính chẵn lẻ (Parity), ngoài ra số bit quy định trong một gói dữ liệu cũng được quy định bởi khung truyền. Có thể thấy, khung truyền đóng một vai trò rất quan trọng trong việc truyền thành công dữ liệu.

***Idle frame:*** Đường truyền UART ở mức “1” để xác nhận hiện tại đường truyền dữ liệu trống, không có frame nào đang được truyền đi.

***Break frame:*** Đường truyền UART ở mức “0” để xác nhận hiện tại trên đường truyền đang truyền dữ liệu, có frame đang được truyền đi.

**Specification**

**Input: Khung dữ liệu (Data Frames):** Mỗi khung chứa 8 bit dữ liệu được truyền từ bộ Transmitter.

**Output: Khung dữ liệu(8 bit)** được nhận thành công từ bộ Receiver (sau khi kiểm tra bit chẵn lẻ).

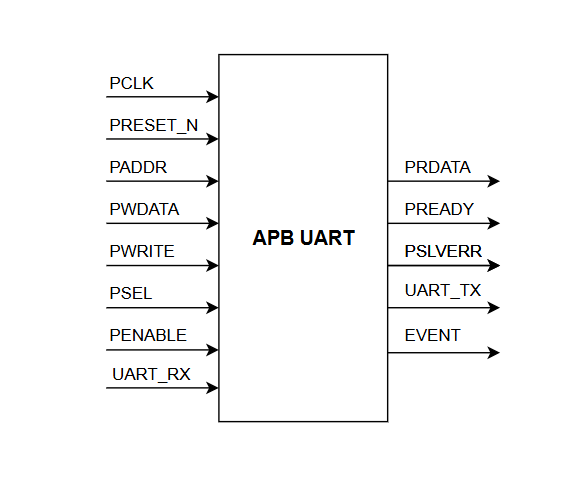
**Mô tả hoạt động:**

* Phía **bộ Transmitter** nhận 8 bit dữ liệu từ bộ đệm và tạo ra 1 **bit chẵn lẻ (parity bit)** để gửi đến phía bộ **Receiver**.
* Phía **bộ Receiver** nhận 8 bit dữ liệu và 1 **bit chẵn lẻ** từ bộ phát, sau đó cũng tạo ra 1 **bit chẵn lẻ** từ 8 bit dữ liệu đã nhận, rồi so sánh **bit chẵn lẻ** tạo ra với **bit chẵn lẻ** được gửi từ bộ **Transmitter**.
  + Nếu 2 bit chẵn lẻ giống nhau, dữ liệu nhận được sẽ được đẩy vào bộ đệm.
  + Nếu 2 bit chẵn lẻ khác nhau, dữ liệu sẽ không được đẩy vào bộ đệm.

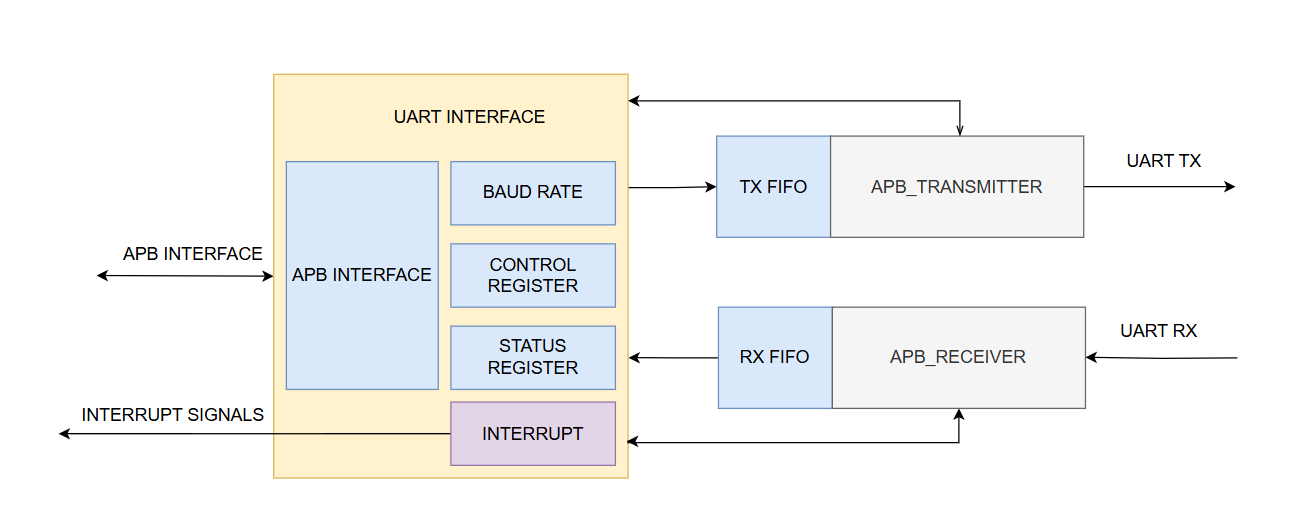
**Thông số kỹ thuật chi tiết:**

* **Baud rate**: 112000 bit/s.
* **Số bit trong mỗi khung dữ liệu**: 5, 6, 7, hoặc 8.
* **Bit chẵn lẻ**: 1 bit chẵn lẻ.
* **Bit dừng**: 1 sop bit (Thời gian truyền: 16 x baud rate).
* **Thời gian truyền 1 bit**: 16 x baud rate

**3.4.3 UART Top Module**

****

Hình 3- 19 UART Top Module

****

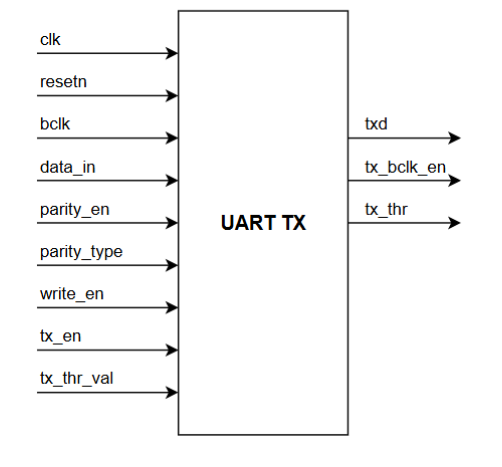
Hình 3- 20 APB UART block diagram

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| PCLK | 1 | Input | Clock. |
| PRESETN | 11 | Input | The APB reset signal is active LOW. This signal is normally connected directly to the system bus reset signal. |
| PADDR | 12 | Input | This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit. |
| PWDATA | 32 | Input | This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH. This bus can be up to 32 bits wide. |
| PWRITE | 1 | Input | Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW. |
| PSEL | 1 | Input | The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. |
| PENABLE | 1 | Input | This signal indicates the second and subsequent cycles of an APB transfer. |
| PRDATA | 32 | Input | The selected slave drives this bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide. |
| PREADY | 1 | Input | The slave uses this signal to extend an APB transfer. |
| PSLVERR | 1 | Output | This signal indicates a transfer failure. |
| UART\_RX | 1 | Input | Receiver input |
| UART\_TX | 1 | Output | Transmitter output |
| EVENT | 1 | Output | Interrupt/event output |

Bảng 13 Signal description APB UART

**3.4.4 UART Transmiter**

**Block diagram**

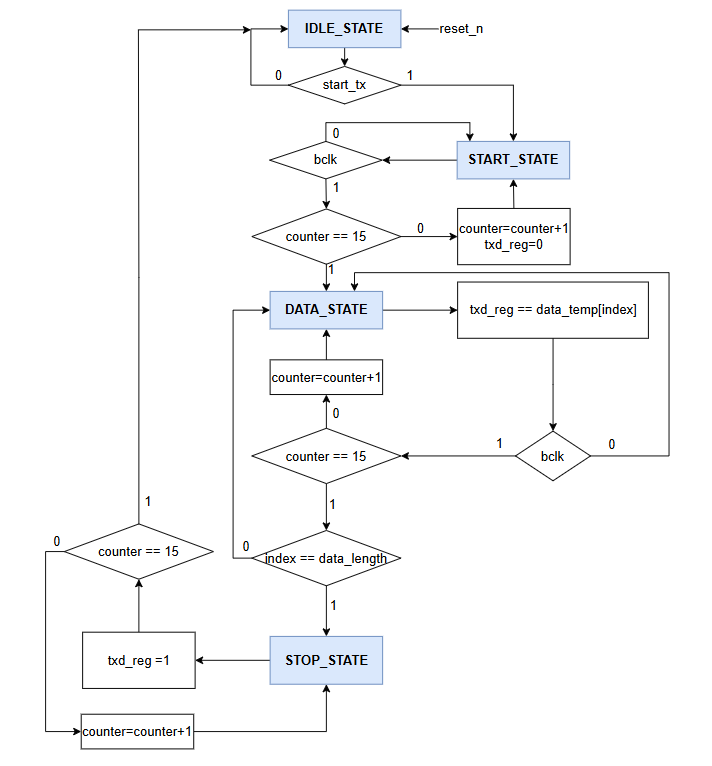
****

Hình 3- 21 UART transmiter block diagram

**Signal description**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| clk | 1 | Input | System clock signal for synchronous operations.. |
| resetn | 1 | Input | Active-low reset signal to initialize the module. |
| bclk | 1 | Input | Baud rate clock signal used for timing transmission. |
| data\_in | 8 | Input | Parallel data input to the UART transmitter. |
| parity\_en | 1 | Input | Enable parity bit generation for data transmission. |
| parity\_type | 1 | Input | Selects parity type: 1 for odd parity, 0 for even parity. |
| write\_en | 1 | Input | Write enable signal to add data to the FIFO buffer. |
| tx\_en | 1 | Input | Transmitter enable signal. |
| tx\_thr\_val | 2 | Input | Configures the TX FIFO threshold to control the tx\_thr flag. |
| txd | 1 | Output | Serial output data. |
| tx\_bclk\_en | 1 | Output | Indicates whether the transmitter is active and using the baud clock. |
| tx\_thr | 1 | Output | FIFO threshold flag. High when the data in FIFO is below the threshold configured by tx\_thr\_val. |

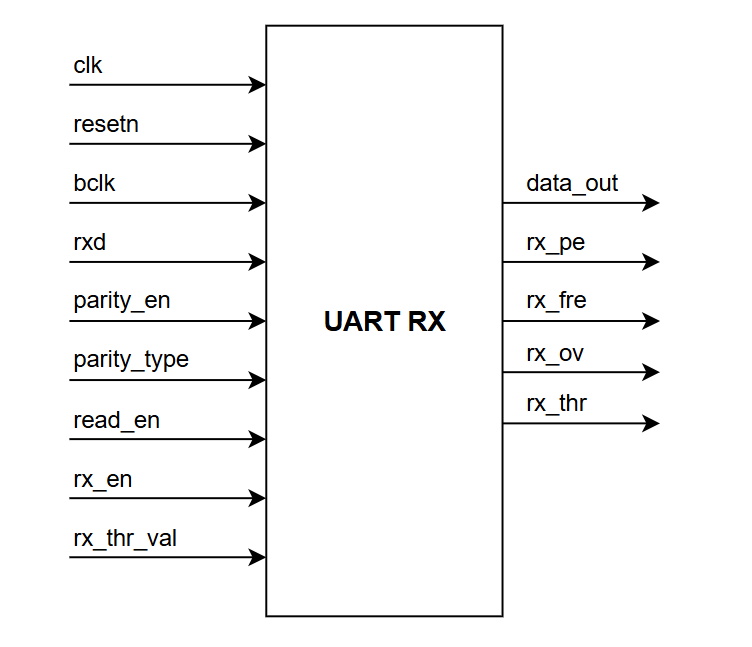
Bảng 14 UART Transmiter signal description

****

Hình 3- 22 UART transmiter FSMD

**3.4.5 UART Receiver**

**Block diagram**

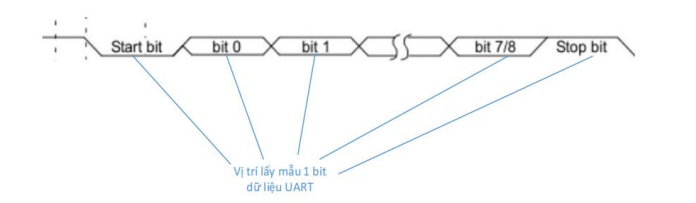
****

Hình 3- 23 UART receiver block diagram

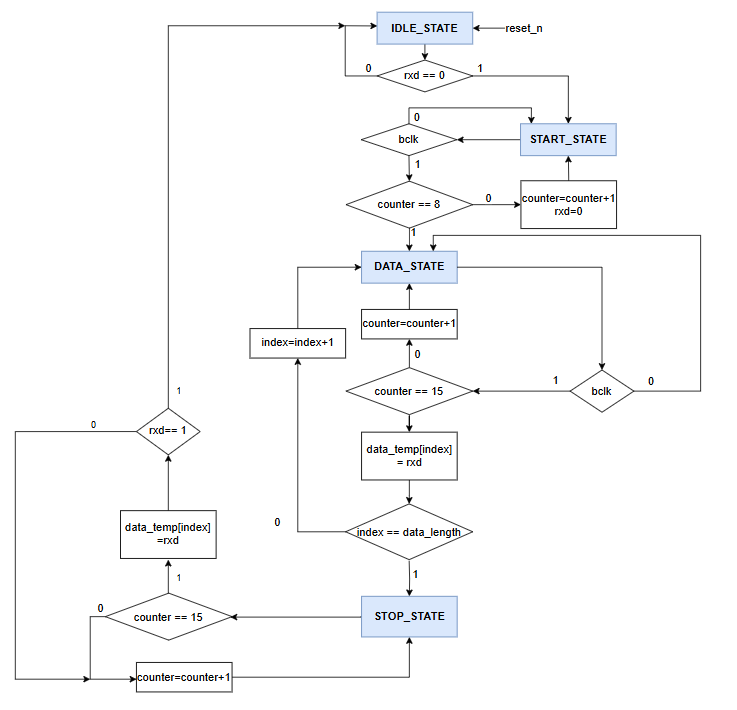
**Signal description**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| clk | 1 | Input | System clock signal for synchronous operations.. |
| resetn | 1 | Input | Active-low reset signal to initialize the module. |
| bclk | 1 | Input | Baud rate clock signal used for timing transmission. |
| rxd | 1 | Input | Serial data input to the UART receiver. |
| parity\_en | 1 | Input | Enable signal for parity checking functionality.. |
| parity\_type | 1 | Input | Selects parity type: 1 for odd parity, 0 for even parity. |
| read\_en | 1 | Input | Read enable signal to fetch data from the FIFO. |
| rx\_en | 1 | Input | Enable signal to activate the UART receiver. |
| rx\_thr\_val | 2 | Input | Sets the FIFO threshold level for the rx\_thr signal. |
| data\_out | 8 | Output | Parallel data output from the FIFO. |
| rx\_fre | 1 | Output | Frame error indicator: 1 if the stop bit is invalid. |
| rx\_pe | 1 | Output | Parity error indicator: 1 if a parity mismatch occurs. |
| rx\_ov | 1 | Output | FIFO overflow indicator: 1 if an attempt is made to write to a full FIFO. |
| rx\_thr | 1 | Output | Threshold indicator: 1 if the FIFO reaches or exceeds the configured threshold. |

Bảng 15 UART Receiver signal description

****

Hình 3- 24 Vị trí lấy mẫu của 1 bit dữ liệu Uart



Hình 3- 25 UART Receiver FSMD

**3.4.6 UART Interface**

**Baud clock generator**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| pclk | 1 | Input | System clock signal for synchronous operations.. |
| presetn | 1 | Input | Active-low reset signal to initialize the module. |
| div\_val | 11 | Input | Clock divider value to adjust the output clock frequencies. |
| tx\_bclk\_en | 1 | Input | Enable signal for generating the TX bit clock. |
| rx\_bclk\_en | 1 | Input | Enable signal for generating the RX bit clock. |
| bclk\_rx | 1 | Output | Generated RX bit clock with frequency based on div\_val. |
| bclk\_tx | 1 | Output | Generated TX bit clock with frequency based on div\_val. |

Bảng 16 Signal description of baud clock generator module

**Cách thức hoạt động:**

* + - Bộ đếm tăng lên mỗi chu kỳ của pclk khi tín hiệu rx\_bclk\_en hoặc tx\_bclk\_en được bật.
    - Khi đạt giá trị div\_val, bộ đếm được đặt lại về 1.
    - Tín hiệu bclk\_rx và bclk\_tx được kích hoạt (mức cao) khi các bộ đếm tương ứng đạt giá trị div\_val.
    - Module tạo các tín hiệu bit clock với tần số điều chỉnh dựa trên div\_val, được sử dụng riêng cho khối RX và TX.

**Control register**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name register** | **Type** | **Length** | **Address** | **Description** |
| REG\_DATA | Read/Write | 8 bits | 0x00000000 | Register temporarily stores data |
| REG\_BCLK | Read/Write | 11 bits | 0x00000004 | Register stores BCLK value |
| REG\_EN | Read/Write | 8 bits | 0x00000008 | Register stores enable signals |
| REG\_THR | Read/Write | 4 bits | 0x0000000C | Register stores threshold for transmitter and receiver |

Bảng 17 Control Register

**Trong đó, có 8 tín hiệu enable ở thành ghi REG\_EN:**

|  |  |  |
| --- | --- | --- |
| **Name Register** | **Bit position in REG\_EN[7:0]** | **Description** |
| TXTHR\_EN | REG\_EN[0] | Enable threshold interrupt signal in Transmitter |
| RXTHR\_EN | REG\_EN[1] | Enable threshold interrupt signal in Receiver |
| RXOV\_EN | REG\_EN[2] | Enable overflow interrupt signal in Receiver |
| PE\_EN | REG\_EN[3] | Enable parity error interrupt signal in Receiver |
| FE\_EN | REG\_EN[4] | Enable frame error interrupt signal in Receiver |
| IP\_EN | REG\_EN[5] | Enable ip active |
| PARITY\_EN | REG\_EN[6] | Enable trans and recive data with parity |
| PARITY\_TYPE | REG\_EN[7] | Type of parity, 1 is odd parity and 0 is even parity |

Bảng 18 Các tín hiệu Enable

**Interrupt Signals**

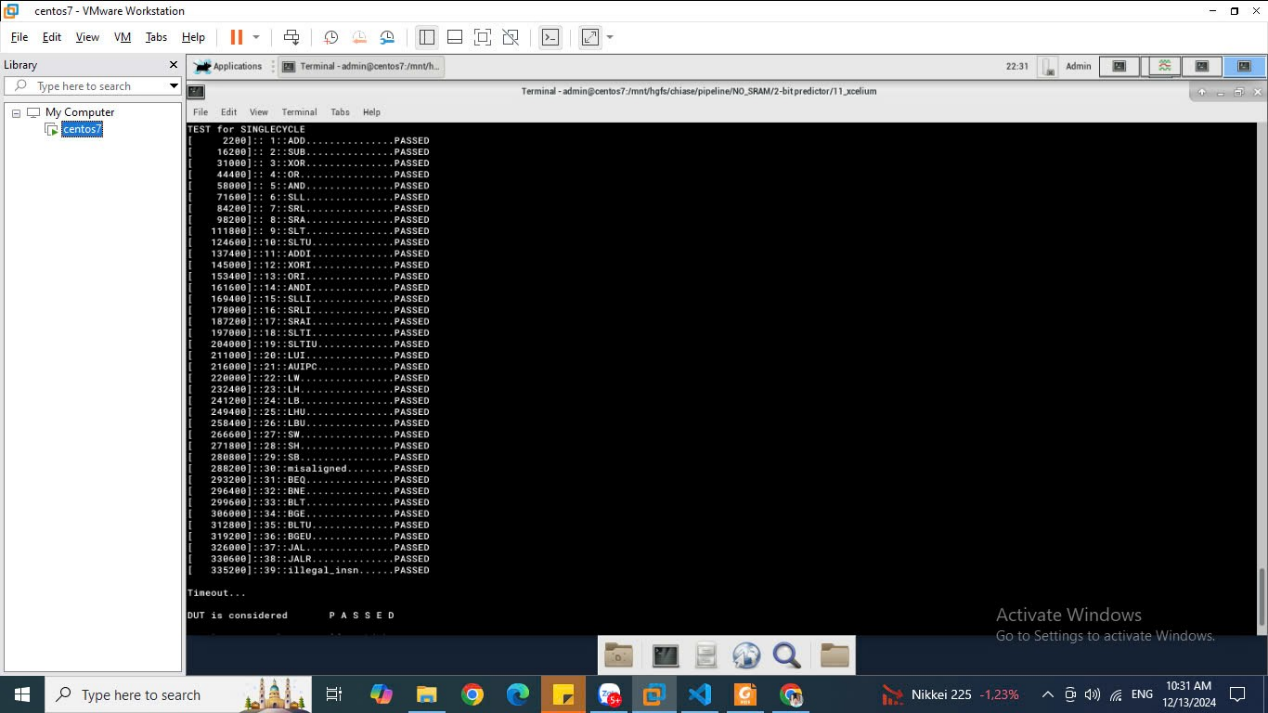
|  |  |
| --- | --- |
| **Name** | **Description** |
| I\_RXOV | Overflow interrupt |
| I\_TXTHR | Threshold of Transmitter interrupt |
| I\_RXTHR | Threshold of Receiver interrupt |
| I\_FE | Frame error of UART data interrupt |
| I\_PE | Parity error of UART data interrupt |
| I\_TOTAL | Total interrupt, high when any other interrupt signals high |

Bảng 19 Các tín hiệu ngắt có trong thiết kế

# 4. XÁC MINH

## 4.1. CPU

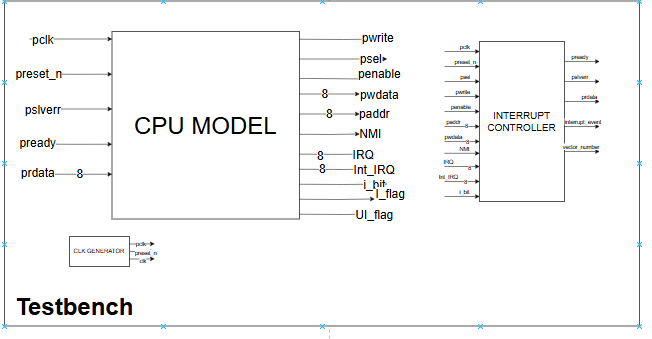
Để verify được CPU sẽ rất khó, vì khá nhiều tín hiệu và các trường hợp corner case khác. Nhưng để đảm bảo CPU của nhóm được thực thi tốt, nhóm quyết định sử dụng testcase mà anh Hải đã yêu cầu trong môn học cấu trúc máy tính kỳ 241 và đã chạy đúng với tất cả các lệnh cơ bản.



Hình 4- 1 Các testcase đã được kiểm chứng chính xác

## 4.2. Interrupt controller

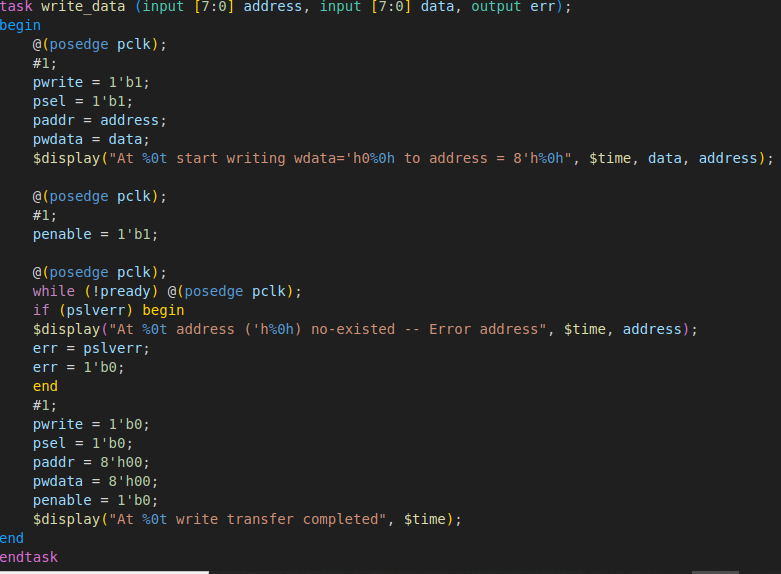
Nhóm sẽ viết môi trường testbench đơn giản để kiểm tra interrupt controller với việc ghi data vào các thanh ghi.



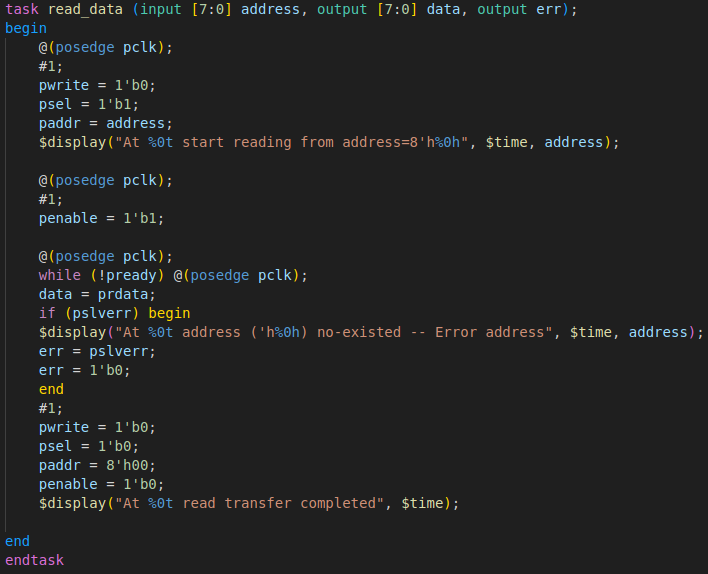
Hình 4- 2 Sơ đồ của môi trường testbench interrupt controller

1. **Register test:**

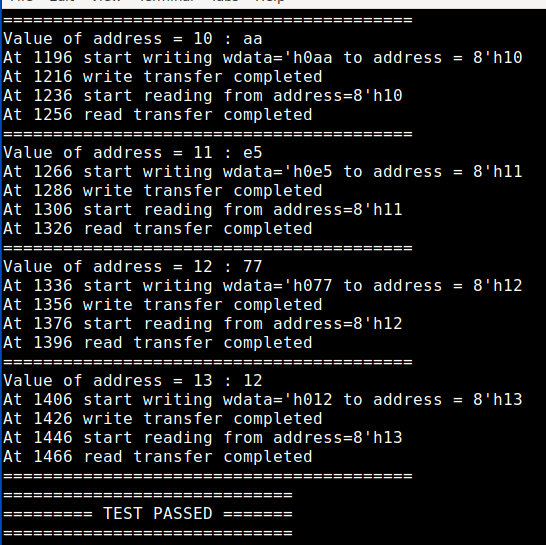
Với trường hợp này, nhóm sẽ giả định các thanh ghi được xuất phát từ CPU model(RISC-V cho đồ án tốt nghiệp sau này) tạo ra các tín hiệu cần cho interrupt controller để mô phỏng việc ghi và đọc data thông qua giao thức APB cơ bản.



Hình 4- 3 Hàm ghi data vào thanh ghi thông qua giao thức APB



Hình 4- 4 Hàm đọc data vào thanh ghi thông qua giao thức APB



Hình 4- 5 Kết quả sau khi test trường hợp ghi và đọc data cơ bản

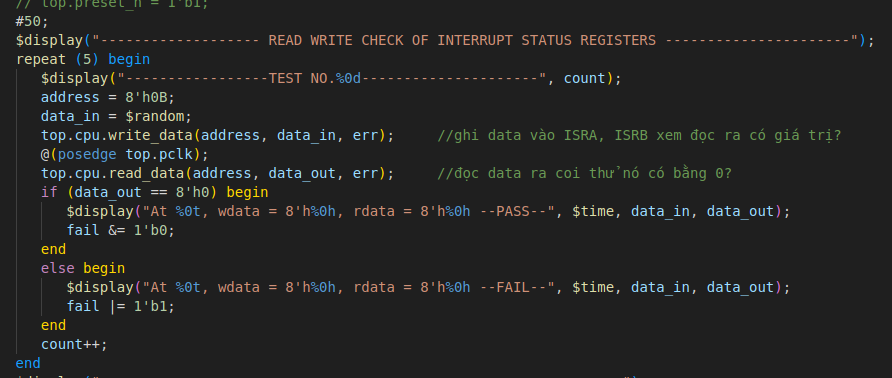
Với trường hợp ghi data vào thanh ghi cơ bản đã cho kết quả đúng hết.

1. **Basic Interrupt Testcase:**

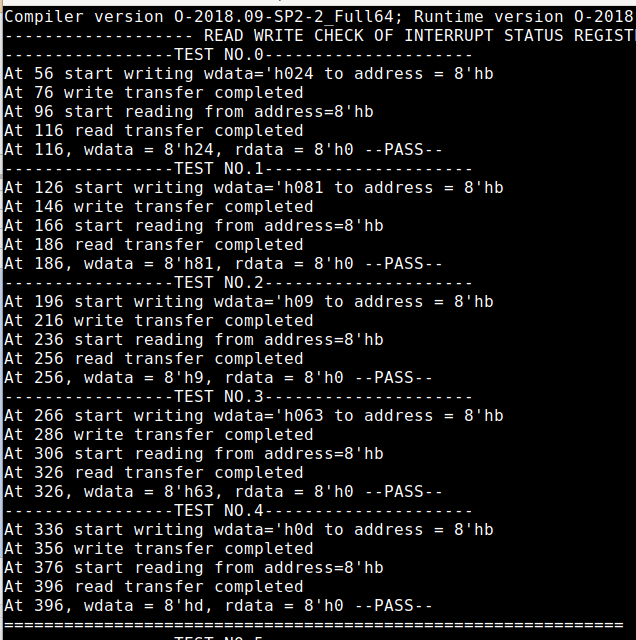
Với các trường hợp ngắt này, ta sẽ có các bước cơ bản sau:

* Configure Interrupt Enable Register (IER): Chọn các tín hiệu cho phép ngắt của các nguồn ngắt
* Set Interrupt Trigger Condition: configure the source of the interrupt (internal or external), Set the interrupt trigger (rising edge, falling edge, or level-triggered): Chọn giá trị cho các tín hiệu ngắt(ngắt nội, ngắt ngoại). Sau đó là cấu hình loại ngắt mà bộ điều khiển sẽ bắt.
* Trigger Interrupt Event: Kích tín hiệu intr\_ev thông qua bộ Controller.
* Detect Interrupt: Wait for the system to detect the interrupt, The processor halts its current execution and jumps to the predefined Interrupt Service Routine (ISR): Chờ hệ thống phát hiện ngắt. RISC-V sẽ ngừng trạng thái hiện tại và nhảy vào chương trình ngắt(ISR).
* ISR Execution: Clear the interrupt flag to acknowledge the event.: Chờ cờ ngắt được xóa.
* Return from ISR: After completing the interrupt handling, the system exits the ISR and resumes its normal program execution: Trở lại chương trình ngắt sau khi đã thoát khỏi ISR.

Trước hết, ta sẽ kiểm tra trạng thái R/W\* của ISRA, ISRB



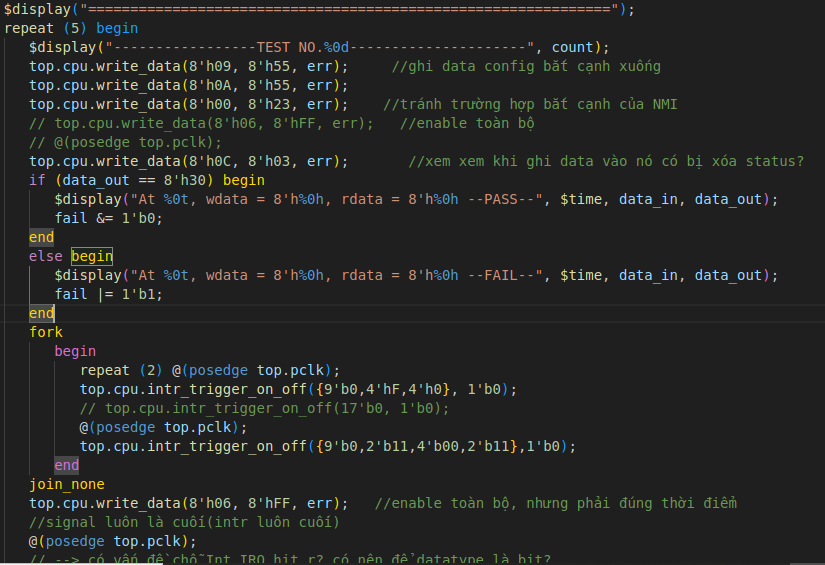
Hình 4- 6 Test cơ bản ghi data vào thanh ghi trạng thái



Hình 4- 7 Kết quả sau khi đã test 5 testcase cơ bản.

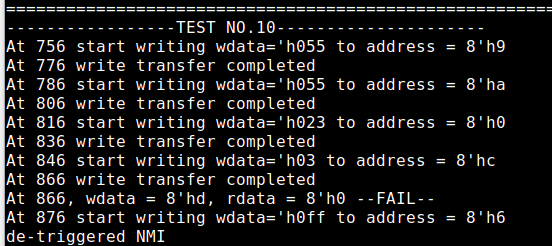
Tương tự với trường hợp Int\_IRQ(nguồn ngắt nội).

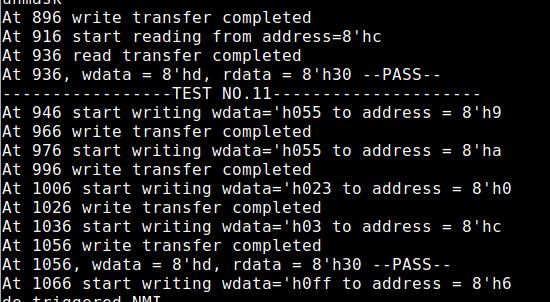
Sau khi đã kiểm tra tính chất ghi data vào thanh ghi trạng thái, ta sẽ kiểm tra với các trường hợp cơ bản của các loại ngắt.



Hình 4- 8 Trường hợp nguồn ngắt là loại cạnh xuống và giả định cho phép toàn bộ ngắt đều được thông qua.

Ta thấy ở thời điểm khởi đầu, thanh ghi chưa ổn định, nên data đọc ra vẫn chưa đáp ứng được yêu cầu.





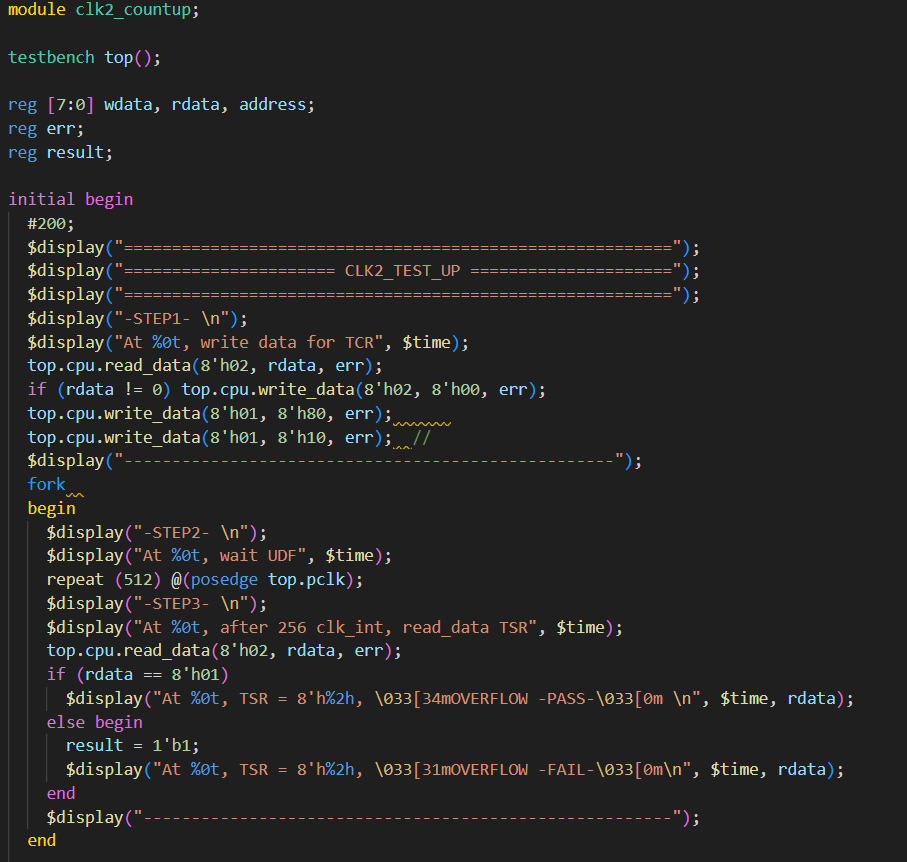
Hình 4- 9 Kết quả sau khi test nguồn ngắt là loại cạnh xuống.

Sau khi test trường hợp nào, chúng ta phải xóa ngay trạng thái cờ ở sau để tránh bị ảnh hưởng các data của thanh ghi về sau.

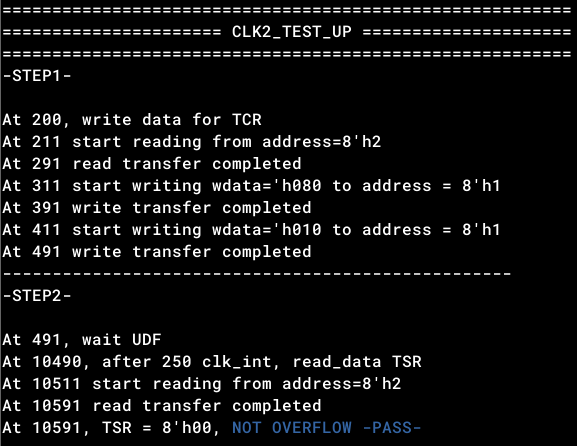
## 4.3. TIMER 8 BIT

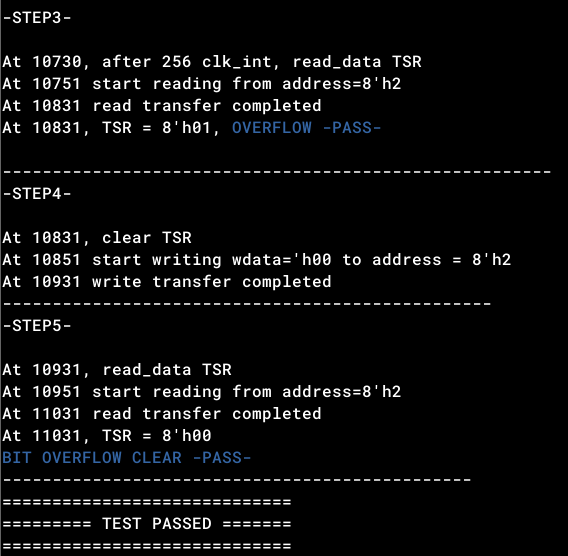
Đối với Timer, nhóm sẽ tiến hành thiết kế testbench với trường hợp đếm lên và đếm xuống với các hệ số clock khác nhau:

* Với trường hợp đếm lên



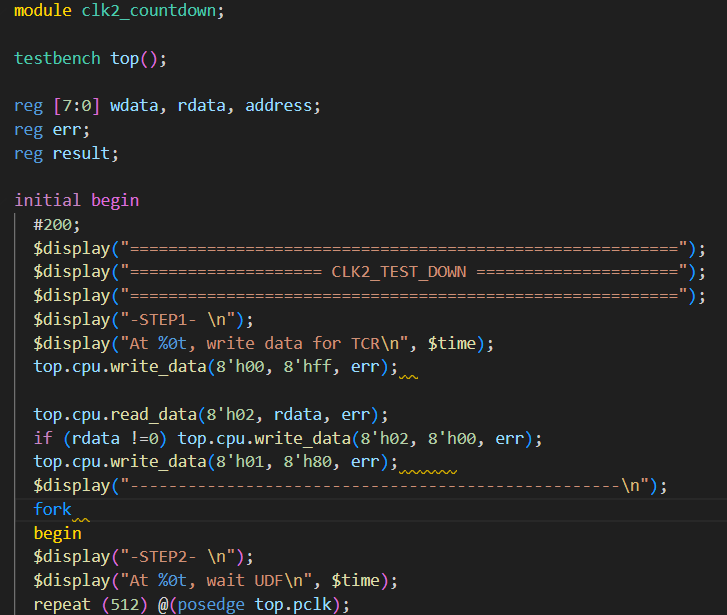
Hình 4- 10 Testbench đếm lên với hệ số clock là 2



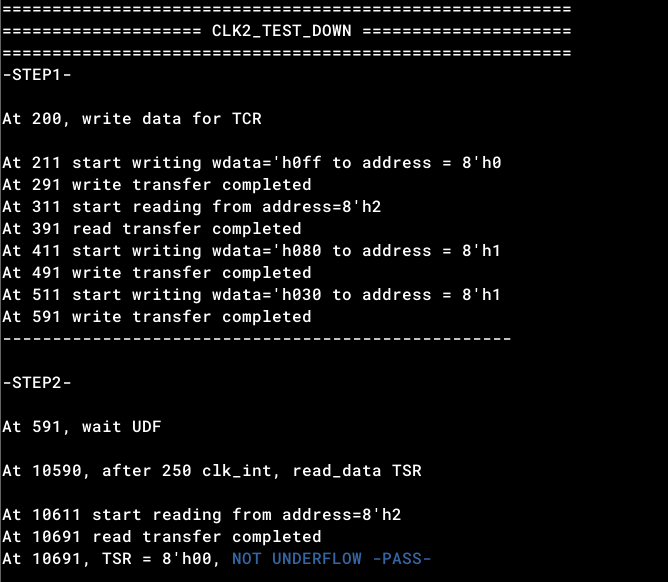


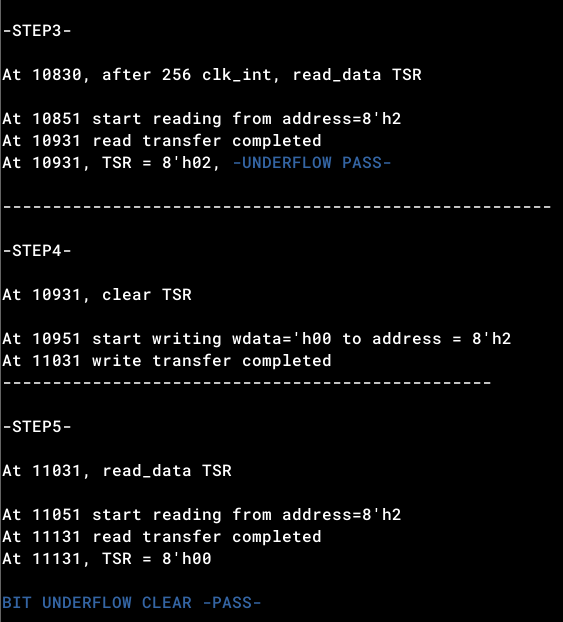
Hình 4- 11Kết quả với trường hợp đếm lên với hệ số clock là 2

* Với trường hợp đếm xuống:



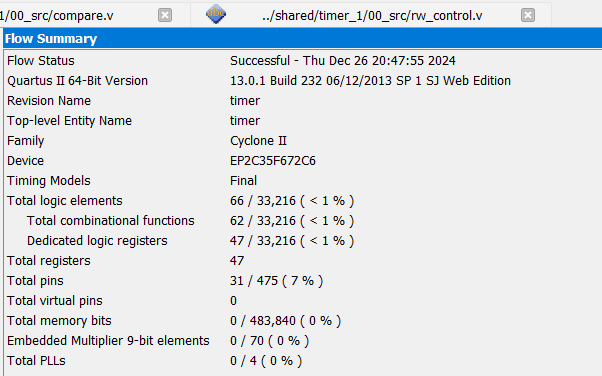
Hình 4- 12 Testbench đếm xuống với hệ số clock là 2





Hình 4- 13 Kết quả với trường hợp đếm xuống với hệ số clock là 2

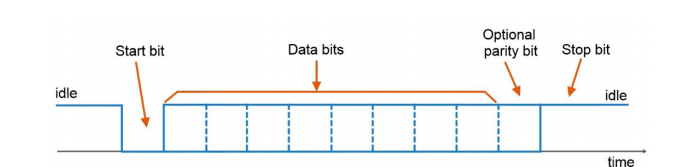
Có thể thấy thiết kế đã pass các test case cơ bản, sau đó nhóm tiến hành đánh giá tài nguyên thiết kế với phần mềm Quartus II:



Hình 4- 14 Đánh giá tài nguyên Timer với Quartus

## 4.4. UART

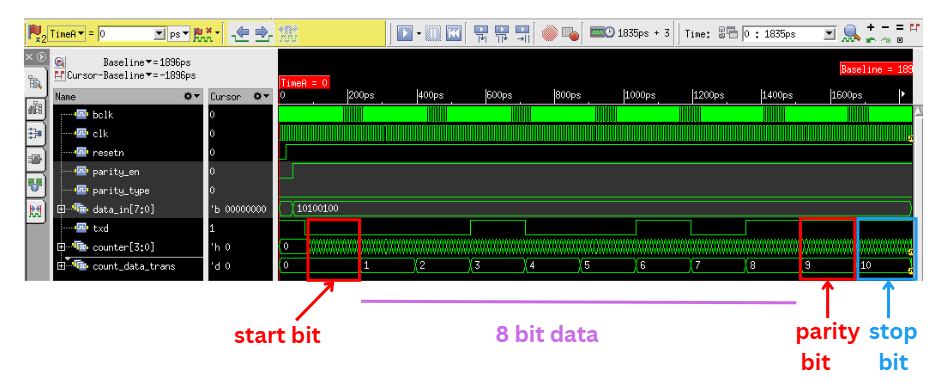
Nhóm tiến hành kiểm tra hoạt động của module Uart Transmitter và Uart Reciever so với lý thuyết cũng như ý tưởng thiết kế.

****

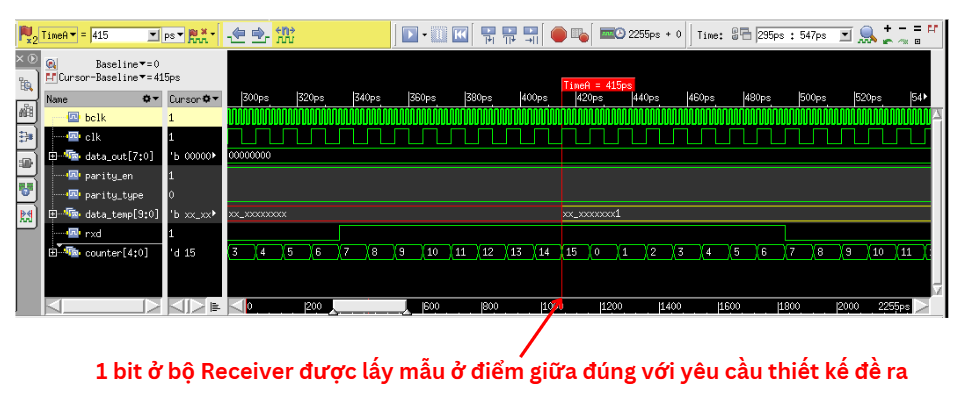
Hình 4- 15 Waveform lý thuyết

****

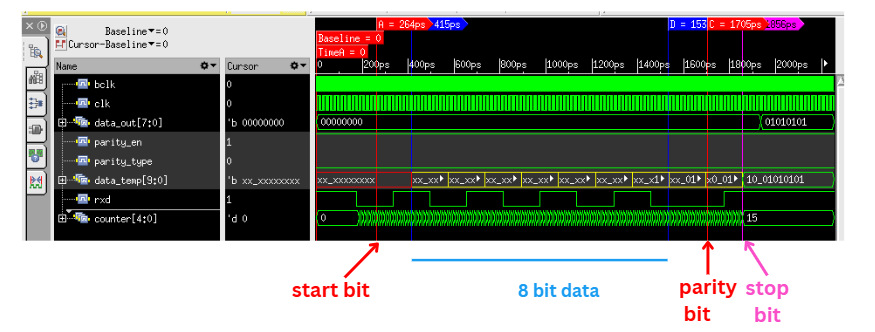
Hình 4- 16 Phân tích thời gian của 1 bit UART ở khối Transmitter

****

Hình 4- 17 Phân tích output ở khối Transmitter



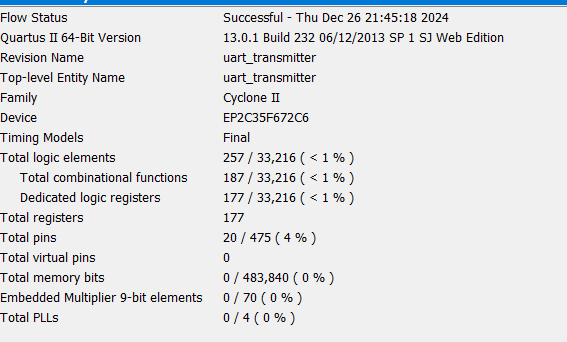
Hình 4- 18 Phân tích 1 bit ở khối Receiver

****

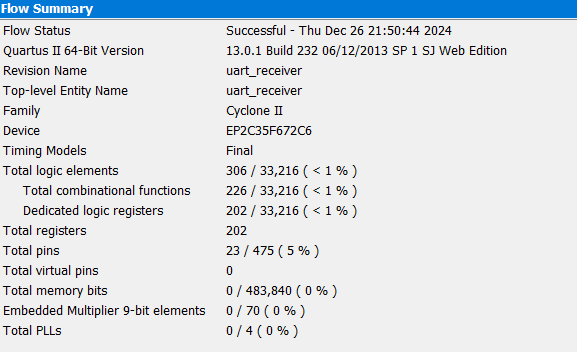
Hình 4- 19 Phân tích output ở khối Receiver

Từ Waveform lý thuyết và Waveform của thiết kế, chúng ta có thể thấy Waveform phù hợp với lý thuyết đã được đề ra.

Sau đó nhóm tiến hành đánh giá tài nguyên với phần mềm Quartus



Hình 4- 20 Đánh giá tài nguyên bộ Uart Transmitter



Hình 4- 21 Đánh giá tài nguyên bộ Uart Receiver

# TÀI LIỆU THAM KHẢO

[1] Patterson, D. A., & Hennessy, J. L., "Computer Organization and Design RISC-V Edition: The Hardware Software Interface," 1st ed., Morgan Kaufmann, 2017.

[2] L. Poli, S. Saha, X. Zhai and K. D. Mcdonald-Maier, "Design and Implementation of a RISC V Processor on FPGA," *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, Exeter, United Kingdom, 2021, pp. 161-166.

[3] D. Thilakarathna *et al*., "RV32IMF Five-Stage Pipeline Implementation with Interrupt and Random Number Generation Units," *2023 IEEE 17th International Conference on Industrial and Information Systems (ICIIS)*, Peradeniya, Sri Lanka, 2023, pp. 1-6.

[4] F. Hussain and S. Sarkar, "Design and FPGA Implementation of Five Stage Pipelined RISC -V Processor," *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, Pune, India, 2024, pp. 1-6.

[5] J. Saussereau, C. Jego, C. Leroux and J. -B. Begueret, "Design and Implementation of a RISC-V core with a Flexible Pipeline for Design Space Exploration," *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Istanbul, Turkiye, 2023, pp. 1-5.

[6] Patterson, D. A., & Waterman, A. (2017). *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon LLC.

[7] Waterman, A., Lee, Y., Patterson, D., & Asanović, K. (2011). *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0*. EECS Department, University of California, Berkeley.