**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

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

****

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**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 05 NĂM 2025**

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

TRƯỜNG ĐẠI HỌC BÁCH KHOA Độc lập – Tự do – Hạnh phúc.

-----✩----- -----✩-----

Số: \_\_\_\_\_\_ /BKĐT

Khoa: **Điện – Điện tử**

Bộ Môn: **Điện Tử**

**NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP**

1. HỌ VÀ TÊN : LÊ VIỆT CƯỜNG MSSV: 2110878

NGUYỄN PHÚ CƯỜNG MSSV: 2112715

1. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG** LỚP : DD21DV1
2. Đề tài: Implementing an interrupt handler for RISC-V ISA
3. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):

Tìm hiểu và thiết kế MCU với tập lệnh RISC-V32I, sử dụng APB bus với ngoại vi giao tiếp bên ngoài là UART và TIMER.

Thiết kế chương trình ngắt dựa theo chuẩn của RISCV.

1. Ngày giao nhiệm vụ luận văn: ...9/2/2025............................
2. Ngày hoàn thành nhiệm vụ: ...................................
3. Họ và tên người hướng dẫn: Phần hướng dẫn

...........TS Trần Hoàng Linh.................... .....................................

................................................................. .....................................

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

*Tp.HCM, ngày…... tháng….. năm 20*

**CHỦ NHIỆM BỘ MÔN NGƯỜI HƯỚNG DẪN CHÍNH**

**PHẦN DÀNH CHO KHOA, BỘ MÔN:**

Người duyệt (chấm sơ bộ):.......................

Đơn vị:......................................................

Ngày bảo vệ : ...........................................

Điểm tổng kết: .........................................

Nơi lưu trữ luận văn: ..............................

Lời cảm ơn GVHD: TS. Trần Hoàng Linh

***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 tháng 5 năm 2025 .*

**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 có nhiệm vụ tiếp nhận và xử lý các yêu cầu ngắt từ nhiều nguồn khác nhau trong hệ thống MCU, bao gồm ngắt từ bộ đếm thời gian (timer) và các thiết bị ngoại vi như UART.

Trong thiết kế này, CPU RISC-V 32I được triển khai với bộ xử lý ngắt cơ bản và sử dụng PLIC (Platform-Level Interrupt Controller) để quản lý các ngắt từ thiết bị ngoại vi. Tuy nhiên, ngắt từ bộ đếm thời gian (timer) không được kết nối thông qua PLIC mà được đưa trực tiếp vào CPU dưới dạng một tín hiệu ngắt riêng biệt. Cách tiếp cận này đơn giản hóa logic xử lý ngắt định kỳ, cho phép CPU phản hồi nhanh chóng các sự kiện thời gian thực mà không cần thông qua cơ chế phân ưu tiên của PLIC.

Bộ đếm thời gian (timer) được thiết kế để tạo ra các tín hiệu ngắt định kỳ. Khi đạt đến giá trị đếm giới hạn, timer sẽ gửi tín hiệu ngắt trực tiếp đến CPU. CPU sẽ nhận và xử lý ngắt này nhằm phục vụ các tác vụ định kỳ như cập nhật thời gian hệ thống, kiểm tra thời gian timeout, hoặc thực hiện các nhiệm vụ thời gian thực khác.

Đối với các thiết bị ngoại vi như UART, PLIC được sử dụng để tiếp nhận các tín hiệu ngắt, phân loại và gán mức ưu tiên trước khi chuyển yêu cầu ngắt đến CPU. Việc sử dụng PLIC giúp hệ thống dễ dàng mở rộng và hỗ trợ nhiều nguồn ngắt ngoại vi một cách hiệu quả.

CPU RISC-V 32I trong hệ thống được thiết kế với kiến trúc pipeline 5 tầng, giúp tăng hiệu suất xử lý lệnh và giảm độ trễ khi xảy ra ngắt. Ngoài ra, hệ thống cũng tích hợp bộ dự đoán nhánh để cải thiện số lượng lệnh thực thi trên mỗi chu kỳ (IPC) và tối ưu hóa dòng điều khiển.

**MỤC LỤC**

[1. GIỚI THIỆU 8](#_Toc196369504)

[2. LÝ THUYẾT 10](#_Toc196369505)

[2.1. Tập lệnh RISC-V 10](#_Toc196369506)

[2.2 2-bit prediction 11](#_Toc196369507)

[2.3. Interrupt Controller 12](#_Toc196369508)

[3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG 14](#_Toc196369509)

[3.1. Ý tưởng thiết kế CPU 14](#_Toc196369510)

[3.2. Thiết kế interrupt controller 16](#_Toc196369511)

[3.3 Thiết kế IP Timer 24](#_Toc196369512)

[3.4 THIẾT KẾ IP UART 28](#_Toc196369513)

[3.5 THIẾT KẾ HỆ THỐNG NGẮT 38](#_Toc196369514)

[4. XÁC MINH 46](#_Toc196369515)

[4.1. CPU 46](#_Toc196369516)

[4.2. Interrupt controller 46](#_Toc196369517)

[4.3. IP TIMER 50](#_Toc196369518)

[4.4. IP UART 52](#_Toc196369519)

[4.5. HỆ THỐNG NGẮT 56](#_Toc196369520)

[5. ĐÁNH GIÁ THIẾT KẾ 58](#_Toc196369521)

[5.1 LOGIC SYNTHESIS 58](#_Toc196369522)

[5.2 STATIC TIMING ANALYSIS 59](#_Toc196369523)

[5.2.1 Khái niệm về STA 59](#_Toc196369524)

[5.2.2 Mục đích của STA 60](#_Toc196369525)

[5.2.3 Một số thông số quan trọng 61](#_Toc196369526)

[5.3 KẾT QUẢ ĐÁNH GIÁ THIẾT KẾ 62](#_Toc196369527)

[5.3.1 IP TIMER 62](#_Toc196369528)

[5.3.2 IP UART 64](#_Toc196369529)

[5.3.3 CPU 65](#_Toc196369530)

[5.3.4 CPU tích hợp với các IP 68](#_Toc196369531)

[5.4 MỘT SỐ KỸ THUẬT GIÚP CỦA THIỆN PPA 71](#_Toc196369532)

[6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 73](#_Toc196369533)

[6.1 Kết luận 73](#_Toc196369534)

[6.2 Hướng phát triển 73](#_Toc196369535)

[7. TÀI LIỆU THAM KHẢO 74](#_Toc196369536)

[8. PHỤ LỤC 75](#_Toc196369537)

**DANH SÁCH HÌNH MINH HỌA**

[Hình 2- 1 Định dạng các lệnh cơ bản 10](#_Toc196344898)

[Hình 2- 2 Sơ đồ khối chung của interrupt controller 13](#_Toc196344899)

[Hình 3- 1 Sơ đồ khối của mạch 2-bit prediction với RISC-V………………………………….…………..………………15](file:///D:\0.%20Đồ%20án%202\Báo%20cáo%20đồ%20án%20tốt%20nghiệp.docx#_Toc196344953)

[Hình 3- 2 Sơ đồ khối của APB\_RW 16](#_Toc196344954)

[Hình 3- 3 Thực thi cách ghi data của thanh ghi ISR 19](#_Toc196344955)

[Hình 3- 4 Sơ đồ khối của mạch phát hiện loại tín hiệu mà ngắt có thể bắt 20](#_Toc196344956)

[Hình 3- 5 Mạch phát hiện ngắt 21](#_Toc196344957)

[Hình 3- 6 Sơ đồ khối của mạch đã phát hiện ngắt và gửi tín hiệu yêu cầu ngắt 21](#_Toc196344958)

[Hình 3- 7 Mạch sắp xếp để chọn ra tín hiệu ngắt nào có độ ưu tiên cao nhất 22](#_Toc196344959)

[Hình 3- 8 Mạch tìm ra ngắt có độ ưu tiên cao nhất đi kèm với đó là địa chỉ của nguồn ngắt 23](#_Toc196344960)

[Hình 3- 9 Sơ đồ khối của IP TIMER 24](#_Toc196344961)

[Hình 3- 10 Các thanh ghi sử dụng trong thiết kế 24](#_Toc196344962)

[Hình 3- 11 Cấu trúc của thanh ghi TCR 25](#_Toc196344963)

[Hình 3- 12 Cấu trúc của thanh ghi TSR 25](#_Toc196344964)

[Hình 3-13 Mô tả hoạt động của bộ đếm Timer 26](#_Toc196344965)

[Hình 3-14Cách thức hoạt động của APB controller 27](#_Toc196344966)

[Hình 3-15 Data Framing Uart 28](#_Toc196344967)

[Hình 3-16 Một số ứng dụng của UART 29](#_Toc196344968)

[Hình 3- 17 UART Top Module 30](#_Toc196344969)

[Hình 3- 18 APB UART block diagram 31](#_Toc196344970)

[Hình 3- 19 UART transmiter block diagram 32](#_Toc196344971)

[Hình 3- 20 UART transmiter FSMD 33](#_Toc196344972)

[Hình 3- 21 UART receiver block diagram 34](#_Toc196344973)

[Hình 3- 22 Vị trí lấy mẫu của 1 bit dữ liệu Uart 35](#_Toc196344974)

[Hình 3- 23 UART Receiver FSMD 35](#_Toc196344975)

[Hình 3- 24 Sơ đồ khối APB Master 39](#_Toc196344976)

[Hình 3- 25 Các lệnh phục vụ cho việc đọc & ghi dữ liệu vào thanh ghi CSR. 39](#_Toc196344977)

[Hình 3- 26 Sơ đồ khối của CSR 40](#_Toc196344978)

[Hình 3- 27 Sơ đồ khối chọn PC để lưu vào mepc 43](#_Toc196344979)

[Hình 3- 28 Các trường hợp cần quan tâm. 43](#_Toc196344980)

[Hình 3- 29 Máy trạng thái hoạt động của CPU 45](#_Toc196344981)

[Hình 4- 1 Các testcase đã được kiểm chứng chính xác……………………………………………………………….…….46](file:///D:\0.%20Đồ%20án%202\Báo%20cáo%20đồ%20án%20tốt%20nghiệp.docx#_Toc196345024)

[Hình 4- 2 Sơ đồ của môi trường testbench interrupt controller 47](#_Toc196345026)

[Hình 4- 3 Kết quả sau khi test trường hợp ghi và đọc data cơ bản 47](#_Toc196345027)

[Hình 4- 4 Kết quả sau khi đã test 5 testcase cơ bản. 48](#_Toc196345028)

[Hình 4- 5 Kết quả sau khi test nguồn ngắt là loại cạnh xuống. 49](#_Toc196345029)

[Hình 4- 6 Đánh giá tài nguyên Interrupt Controller với Quartus II 49](#_Toc196345030)

[Hình 4- 7 Kết quả với trường hợp đếm lên với hệ số clock là 2 50](#_Toc196345031)

[Hình 4- 8 Kết quả với trường hợp đếm xuống với hệ số clock là 2. 51](#_Toc196345032)

[Hình 4- 9 Đánh giá tài nguyên Timer với Quartus 52](#_Toc196345033)

[Hình 4- 10 Waveform lý thuyết 52](#_Toc196345034)

[Hình 4- 11 Phân tích thời gian của 1 bit UART ở khối Transmitter 53](#_Toc196345035)

[Hình 4- 12 Phân tích chế độ truyền Uart 53](#_Toc196345036)

[Hình 4- 13 Phân tích Output ở khối Transmitter 53](#_Toc196345037)

[Hình 4- 14 Phân tích 1 bit ở khối Receiver 54](#_Toc196345038)

[Hình 4- 15 Phân tích chế độ nhận của Uart 54](#_Toc196345039)

[Hình 4- 16 Phân tích output ở khối Receiver 55](#_Toc196345040)

[Hình 4- 17 Đánh giá tài nguyên IP Uart với Quartus 55](#_Toc196345041)

[Hình 4- 18 Ngắt Timer 56](#_Toc196345042)

[Hình 4- 19 Chương trình mẫu 56](#_Toc196345043)

[Hình 4- 20 PC của CPU nhảy theo sự điều khiển của máy trạng thái 57](#_Toc196345044)

[Hình 4-21 CPU trả lại địa chỉ chương trình hiện tại để thực thi tiếp chương trình đã bị ngắt 57](#_Toc196345045)

[Hình 4- 22 Tích hợp tất cả các IP: UART, TIMER, PLIC 57](#_Toc196345046)

[Hình 5- 1 Minh họa về Synthesis………………………………………………………………………………………………………58](#_Toc196369193)

[Hình 5- 2 Tổng quan về Synthesis 59](#_Toc196369194)

[Hình 5- 3 Synthesis Flow Steps 59](#_Toc196369195)

[Hình 5- 4 Ví dụ về path delay 60](#_Toc196369196)

[Hình 5- 5 Minh họa về kĩ thuật Cell Swap 60](#_Toc196369197)

[Hình 5- 6 Minh họa về cách tính Slack với trường hợp Setup Check 62](#_Toc196369198)

[Hình 5- 7 Schematic của IP TIMER sau khi Synthesis 62](#_Toc196369199)

[Hình 5- 8 Thông tin về timing của IP TIMER 63](#_Toc196369200)

[Hình 5- 9 Thông tin về power của IP TIMER 63](#_Toc196369201)

[Hình 5- 10 Thông tin về area của IP Timer 63](#_Toc196369202)

[Hình 5- 11 Schematic của IP UART sau khi Synthesis 64](#_Toc196369203)

[Hình 5- 12 Thông tin về timing của IP UART 64](#_Toc196369204)

[Hình 5- 13 Thông tin về power của IP UART 65](#_Toc196369205)

[Hình 5- 14 Thông tin về area của IP UART 65](#_Toc196369206)

[Hình 5- 15 Schematic của CPU sau khi Synthesis 66](#_Toc196369207)

[Hình 5- 16 Thông tin về timing của CPU 66](#_Toc196369208)

[Hình 5- 17 Thông tin về power của CPU 67](#_Toc196369209)

[Hình 5- 18 Thông tin về area của CPU 67](#_Toc196369210)

[Hình 5- 19 Schematic của CPU tích hợp các IP sau khi Synthesis 68](#_Toc196369211)

[Hình 5- 20 Thông tin về timing của CPU tích hợp các IP sau khi Synthesis 68](#_Toc196369212)

[Hình 5- 21 Thông tin về power của CPU tích hợp các IP sau khi Synthesis 69](#_Toc196369213)

[Hình 5- 22 Thông tin về area của CPU tích hợp các IP sau khi Synthesis 70](#_Toc196369214)

**DANH SÁCH BẢNG MINH HỌA**

[Bảng 1 Các tín hiệu của Interrupt Controller 14](#_Toc196345325)

[Bảng 2 Các tín hiệu đầu vào và ra của APB\_RW 18](#_Toc196345326)

[Bảng 3 Nội dung của các bit trong thanh ghi SYSCR 18](#_Toc196345327)

[Bảng 4 Nội dung của các bit trong thanh ghi IPR 18](#_Toc196345328)

[Bảng 5 Nội dung của các bit trong thanh ghi ISCRn 18](#_Toc196345329)

[Bảng 6 Các trường hợp của cặp bit trong thanh ghi ISCR 19](#_Toc196345330)

[Bảng 7 Nội dung của các bit trong thanh ghi SYSCR 19](#_Toc196345331)

[Bảng 8 Các tín hiệu đầu vào và ra của edge\_detection 21](#_Toc196345332)

[Bảng 9 Bảng giá trị của mạch phát hiện yêu cầu ngắt 22](#_Toc196345333)

[Bảng 10 Các tín hiệu của mạch Prio\_deter 23](#_Toc196345334)

[Bảng 11 Signal description APB UART 32](#_Toc196345335)

[Bảng 12 UART Transmiter signal description 33](#_Toc196345336)

[Bảng 13 UART Receiver signal description 34](#_Toc196345337)

[Bảng 14 Signal description of baud clock generator module 36](#_Toc196345338)

[Bảng 15 Control Register 36](#_Toc196345339)

[Bảng 16 Các tín hiệu Enable 37](#_Toc196345340)

[Bảng 17 Các tín hiệu ngắt có trong thiết kế 37](#_Toc196345341)

[Bảng 18 Các tín hiệu của CSR 42](#_Toc196345342)

[Bảng 19 Các thanh ghi phục vụ chương trình ngắt 42](#_Toc196345343)

[Bảng 20 Thống kê các trường hợp lưu lại địa chỉ khi có ngắt/ exception 44](#_Toc196345344)

# 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: RISC-V 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 dựa trên chuẩn APB, cho phép CPU giao tiếp với các ngoại vi như Timer, UART, GPIO một cách hiệu quả và linh hoạt.

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 IP Uart

* Tìm hiểu về nguyên lý hoạt động của IP Timer và UART, bao gồm cơ chế truyền nhận dữ liệu và xử lý ngắt.
* Xây dựng IP Timer và UART đảm bảo tương thích với yêu cầu của MCU, hỗ trợ các chế độ hoạt động cần thiết và tối ưu hóa hiệu suất.
* Thực hiện đánh giá thiết kế về diện tích phần cứng (area), thời gian đáp ứng (timing) và mức tiêu thụ công suất (power) nhằm đảm bảo hiệu quả sử dụng tài nguyên.

Nội dung 5: Đánh giá và thực hiện trên FPGA

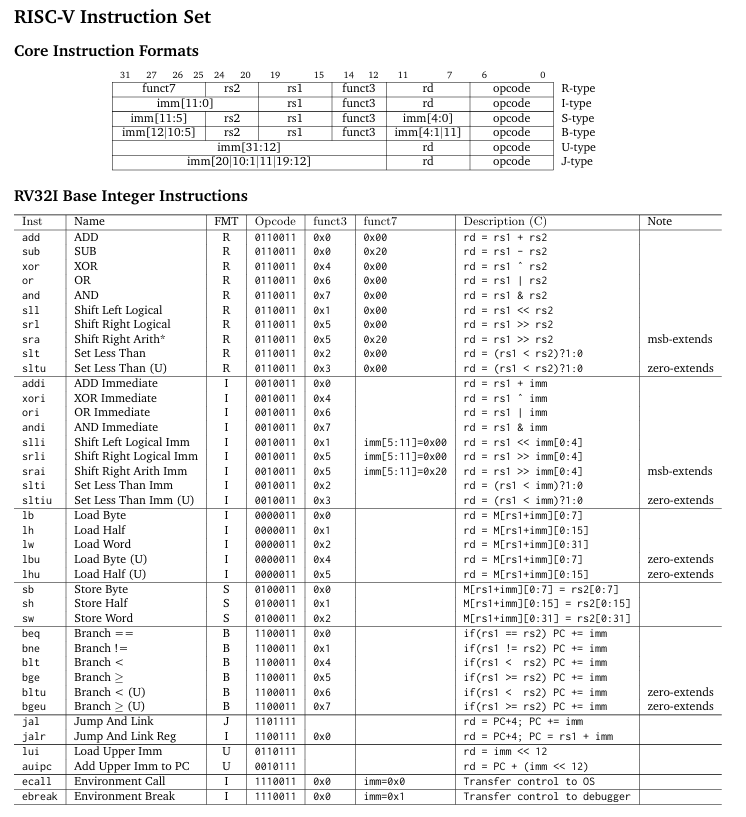
* Nghiên cứu và triển khai chương trình ngắt trên Kit DE2, bao gồm các IP đã thiết kế.
* Thực hiện các bài kiểm tra và đánh giá hiệu năng thiết kế trên FPGA, phân tích độ trễ xử lý, mức sử dụng tài nguyên phần cứng và khả năng mở rộng của hệ thống.

**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 | Phú Cường |
| IP Timer | Việt Cường |
| IP Uart | Việt Cường |
| Đánh giá và thực hiện trên FPGA | Việt Cường, Phú Cường |

# 2. LÝ THUYẾT

## 2.1. Tập lệnh RISC-V



Hình 2- Định dạng các lệnh cơ bản

## 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).

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:

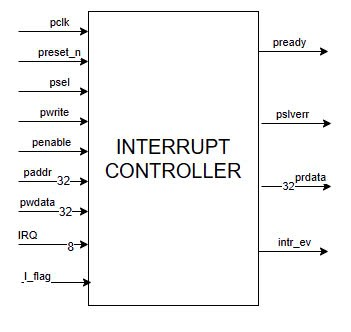
1. Các giá trị trong bộ đếm chương trình (PC), thanh ghi mã điều kiện (CCR), và thanh ghi mở rộng (EXR) được lưu vào ngăn xếp.
2. Bit mask ngắt được cập nhật và bit T được xóa về 0.
3. Một địa chỉ vector tương ứng với nguồn ngắt được tạo ra, địa chỉ bắt đầu được nạp từ bảng vector vào PC, và chương trình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

**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- 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 |
| IRQ | 8 | Input | Ngắt ngoại có thể bị mask |
| i\_bit | 1 | Input | Bit được gửi từ CPU cho biết interrupt source đó có được xử lý hay không? |
| 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 Các tín hiệu của Interrupt Controller

# 3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

## 3.1. Ý tưởng thiết kế CPU

**Các tầng của pipeline:**

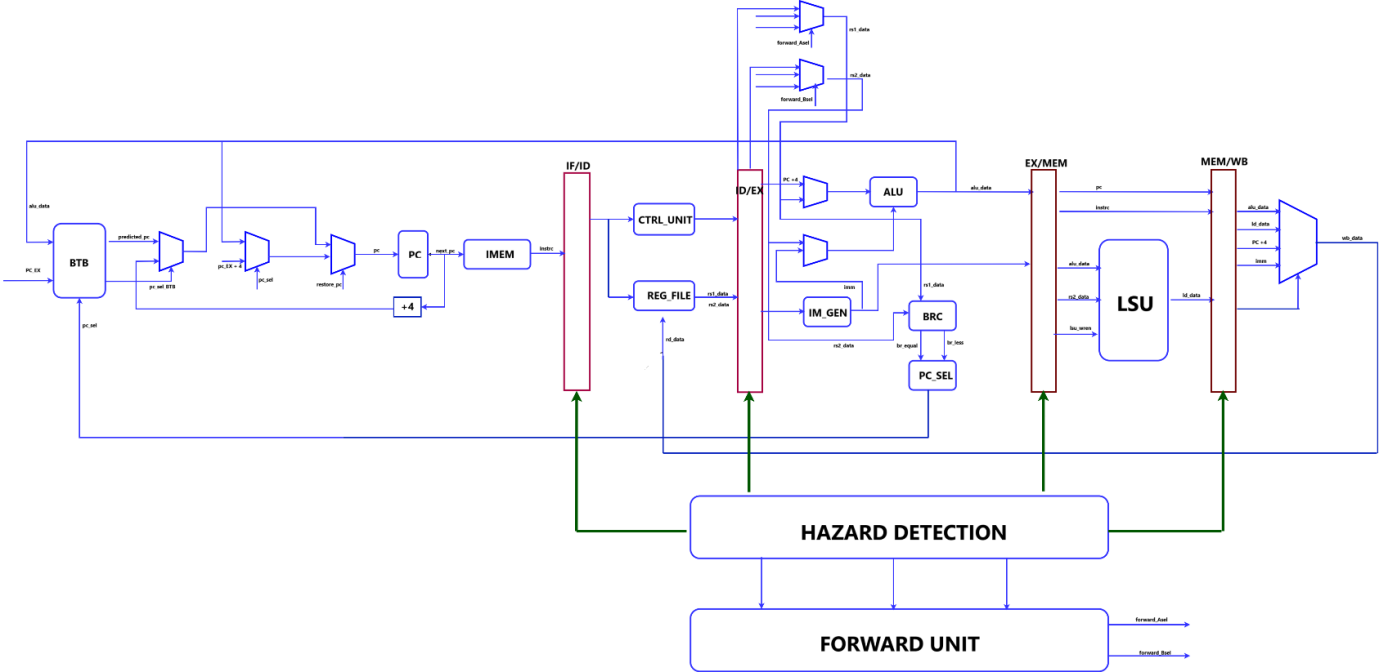
Nhóm sẽ thiết kế một bộ xử lý theo đường ống năm giai đoạn bao gồm:

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.

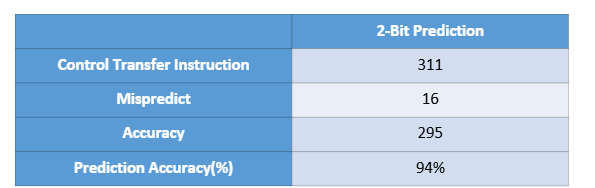
5. WB (Write-back): Ghi kết quả trở lại các thanh ghi

Hình 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:

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



## 3.2. Thiết kế interrupt controller

Interrupt controller bao gồm:

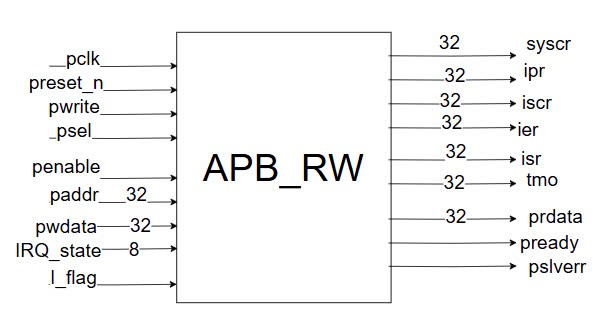
* Có giao thức APB bus interface của ARM bus
* Có thể được set mode cho SYSCR(NMI\_eg)
* Độ ưu tiên được set bởi IPRA,IPRB,IPRC,IPRD.

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- 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 | 32 | Input | Địa chỉ APB bus. Có thể lên tới 32 bit và được lái bởi peripheral bus bridge unit |
| Pwdata | 32 | Input | Lái bởi peripheral bus bridge unit trong chu kỳ ghi khi pwrite = 1 |
| Prdata | 32 | 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 | 32 | Output | Xác định loại ngắt NMI nào được bắt và báo tín hiệu timeout và slverr |
| Ipr | 32 | Output | Set mức độ ưu tiên của tín hiệu ngắt(ngoại trừ NMI) |
| Iscr | 32 | 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 |
| Ier | 32 | Output | Điều khiển sự cho phép của ngắt nội, ngắt ngoại |
| Isr | 32 | Output | Chỉ rõ trạng thái của các tín hiệu ngắt(ngắt nội và ngoại) |
| Tmo | 32 | Output | Thanh ghi đếm xuống khi tín hiệu ngắt xảy ra |
| IRQ\_state | 32 | Output | Trạng thái ngắt ngoại |

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

A1. System control register (SYSCR)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Tên | - | - | - | - | SLVERR\_EN | TMO\_EN | NMIEG[1] | NMIEG[0] |

Bảng Nội dung của các bit trong thanh ghi SYSCR

A2. Interrupt priority register (IPR)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Tên | - | IPR6 | IPR5 | IPR4 | - | IPR2 | IPR1 | IPR0 |

Bảng Nội dung của các bit trong thanh ghi IPR

Các cặp 3 bit liên tiếp này sẽ đại diện cho độ lớn của mức độ ưu tiên của một nguồn ngắt. Như vậy, cứ 1 thanh ghi sẽ đại diện cho 2 mức nguồn ngắt. Vậy có 4 thanh ghi sẽ đại diện cho 1 thanh ghi có 8 bit về nguồn ngắt. Trong phạm vi đồ án này, nhóm sẽ xài chung mức độ ưu tiên cho nguồn ngắt nội và ngoại.

A3. IRQ enable register: Ứng với 1 bit sẽ là 1 công tắc để cho phép nguồn ngắt đó có được ngắt hay không.

A4. IRQ Sense Control register

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Tên | IRQ7 | IRQ7 | IRQ6 | IRQ6 | IRQ5 | IRQ5 | IRQ4 | IRQ4 |

Bảng Nội dung của các bit trong thanh ghi ISCRn

|  |  |  |
| --- | --- | --- |
| IRQ[2n+1] | IRQ[2n] | Mô tả |
| 0 | 0 | Yêu cầu ngắt được tạo ra bởi IRQn hoặc Int\_IRQn với đầu vào là mức 0 |
| 0 | 1 | Yêu cầu ngắt được tạo ra bởi IRQn hoặc Int\_IRQn với đầu vào là cạnh xuống |
| 1 | 0 | Yêu cầu ngắt được tạo ra bởi IRQn hoặc Int\_IRQn với đầu vào là cạnh lên |
| 1 | 1 | Yêu cầu ngắt được tạo ra bởi IRQn hoặc Int\_IRQn với đầu vào là cạnh |

Bảng Các trường hợp của cặp bit trong thanh ghi ISCR

A5. IRQ Status register

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Tên | IRQ7F | IRQ6F | IRQ5F | IRQ4F | IRQ3F | IRQ2F | IRQ1F | IRQ0F |
|  | R/W\* | R/W\* | R/W\* | R/W\* | R/W\* | R/W\* | R/W\* | R/W\* |

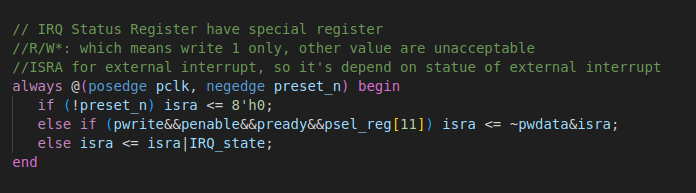
Bảng Nội dung của các bit trong thanh ghi SYSCR

**Chú ý:**

Thanh ghi ISR:

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 ISCR xảy ra thì các trạng thái này của thanh ghi sẽ được tự động kích hoạt lên 1.



Hình 3- Thực thi cách ghi data của thanh ghi ISR

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

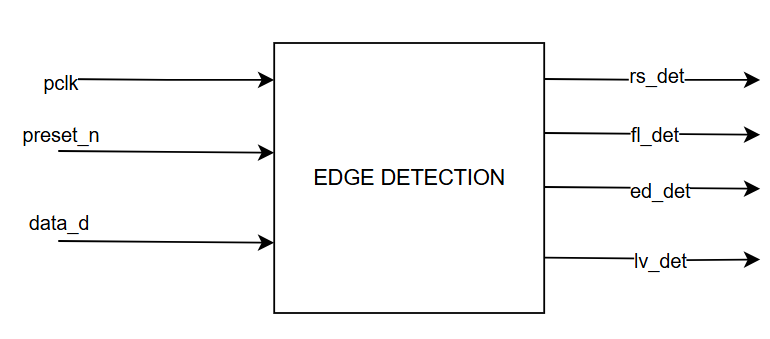
Các bit của thanh ghi được xóa bởi việc ta ghi bit 1 vào thanh ghi ấy.

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).

A6. TIME OUT register (TMO): Đây là thanh ghi 32 bit có giá trị mặc định là 32’hFFFFFFFF. Khi TMO\_EN của thanh ghi SYSCR = 1 thì nó sẽ bắt đầu đếm xuống đại diện cho quá trình TIME OUT. Nếu sau thời gian đó cờ I\_flag và UI\_flag không thay đổi thì chương trình sẽ bỏ qua ngắt đó.

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



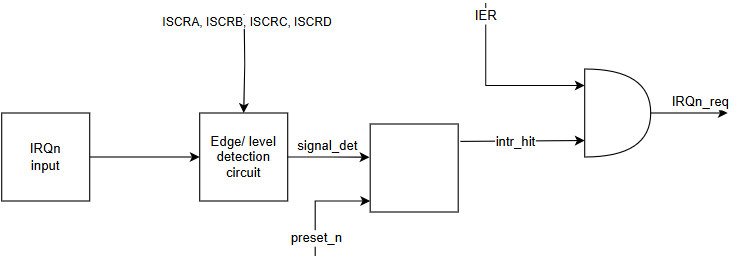
Hình 3- Sơ đồ khối của mạch phát hiện loại tín hiệu mà ngắt có thể bắt

|  |  |  |  |
| --- | --- | --- | --- |
| 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 Các tín hiệu đầu vào và ra của edge\_detection

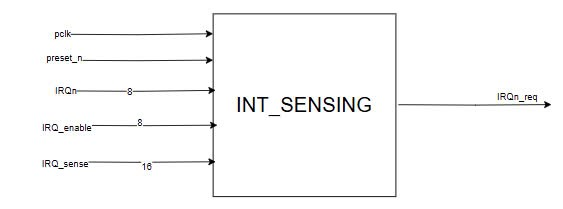
1. **Khối yêu cầu các tín hiệu ngắt: (int\_sensing)**

Khối tín hiệu ngắt theo mô hình chung:



Hình 3- 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:



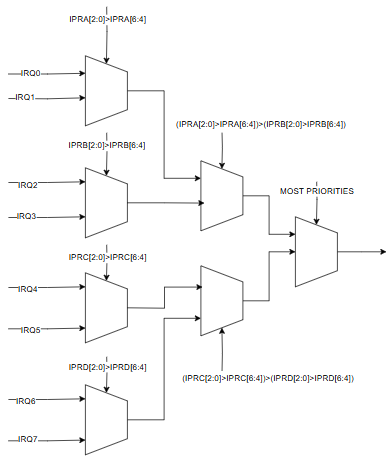
Hình 3- Sơ đồ khối của mạch đã phát hiện ngắt và gửi tín hiệu yêu cầu ngắt

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| IRQn | 8 | Input | Ngắt ngoại có thể bị mask |
| IRQ\_enable | 8 | Input | Cho phép ngắt ngoại(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 |
| IRQn\_req | 1 | Output | Yêu cầu ngắt của ngắt ngoại |

Bảng Bảng giá trị của mạch phát hiện yêu cầu ngắt

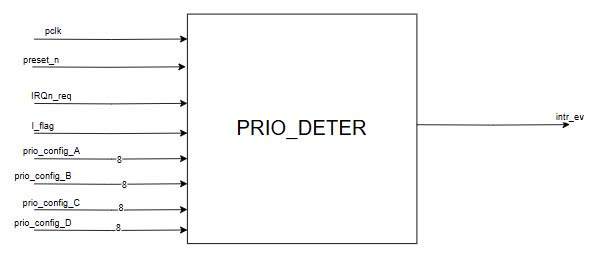
1. **Khối xác định mức độ ưu tiên của ngắt: (prio\_deter)**

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- 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:



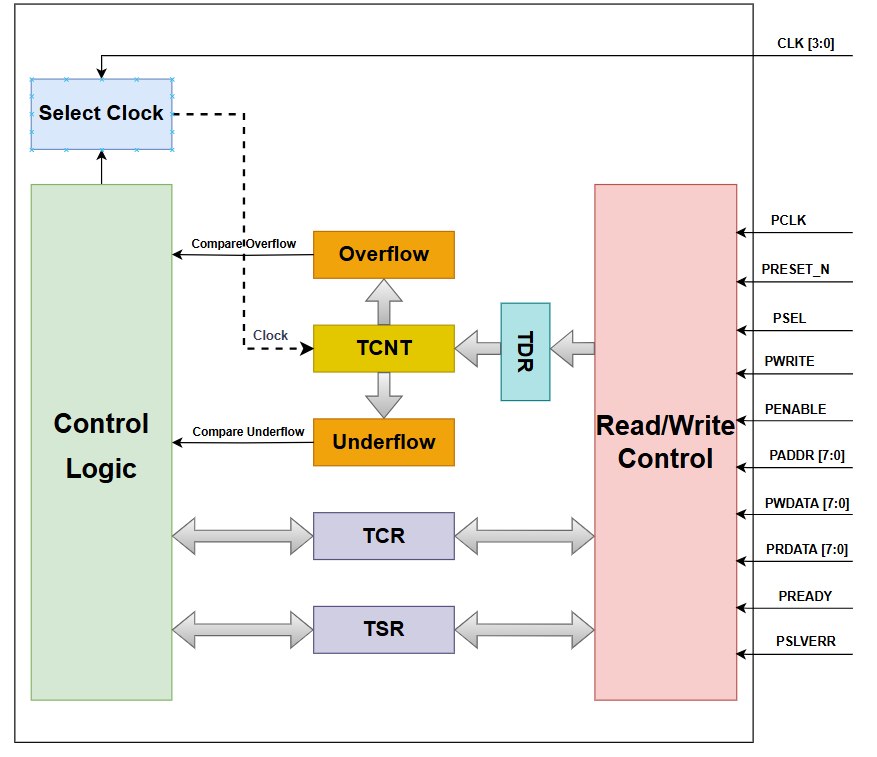
Hình 3- Mạch tìm ra ngắt có độ ưu tiên cao nhất đi kèm với đó là địa chỉ của nguồn ngắt

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Direction | Function |
| IRQn\_req | 1 | Input | Yêu cầu ngắt ngoại |
| I\_flag | 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) |
| Intr\_ev | 1 | Output | Tín hiệu ngắt cuối cùng gửi tới CPU |

Bảng Các tín hiệu của mạch Prio\_deter

## 3.3 Thiết kế IP Timer

**3.3.1 Sơ đồ khối**

****

Hình 3- Sơ đồ khối của IP TIMER

**3.3.2 Các thanh ghi được sử dụng**

TDR (32-bit)

Timer Data Register

TCNT (32-bit)

Timer Counter

TSR (32-bit)

Timer Status Register

TCR (32-bit)

Timer Control Register

Hình 3- Các 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- 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)**

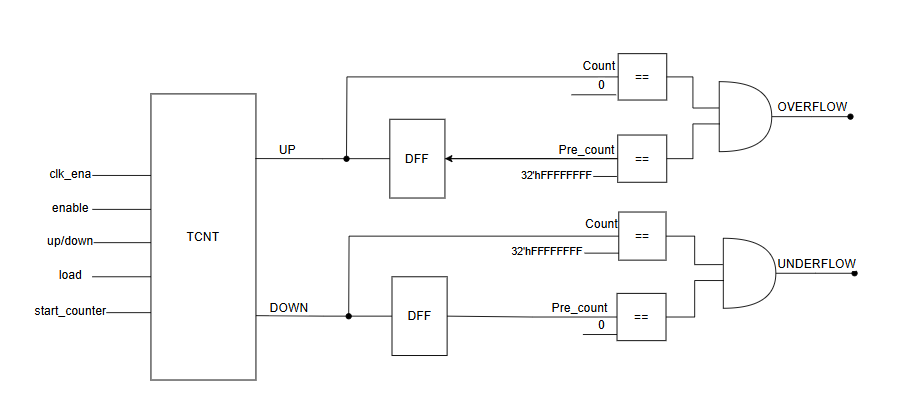
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit position** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit function** |  |  |  |  |  |  | **Underflow** | **Overflow** |

Hình 3- 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.

**3.3.4 Timer Counting Logic**



Hình 3- 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.

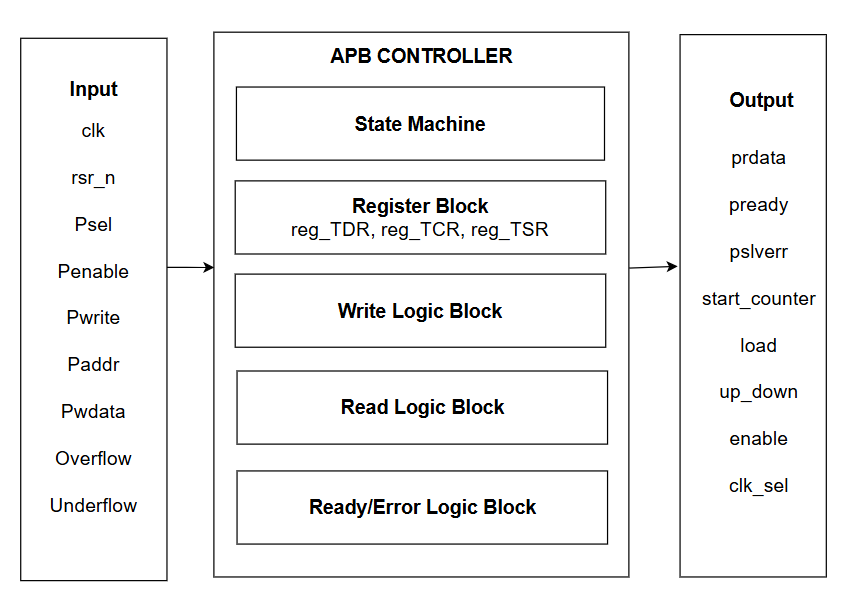
**Ngõ ra:**

* **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 0, **count** được tăng thêm 1.
  + Nếu **up\_down** là 1, **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-Cách thức hoạt động của APB controller

**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[3:0] = 4'h0**, giá trị của **pwdata** sẽ được ghi vào thanh ghi **reg\_TCR**.
* Nếu **paddr[3:0] = 4'h4**, giá trị của **pwdata** sẽ được ghi vào thanh ghi **reg\_TDR**.
* Nếu **paddr[3:0] = 4'h8**, 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.

## 3.4 THIẾT KẾ IP 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- 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- 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.

**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).

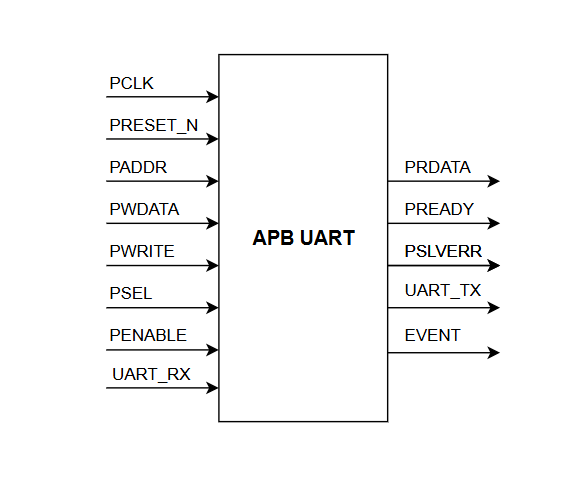
**Mô tả hoạt động:**

* 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.

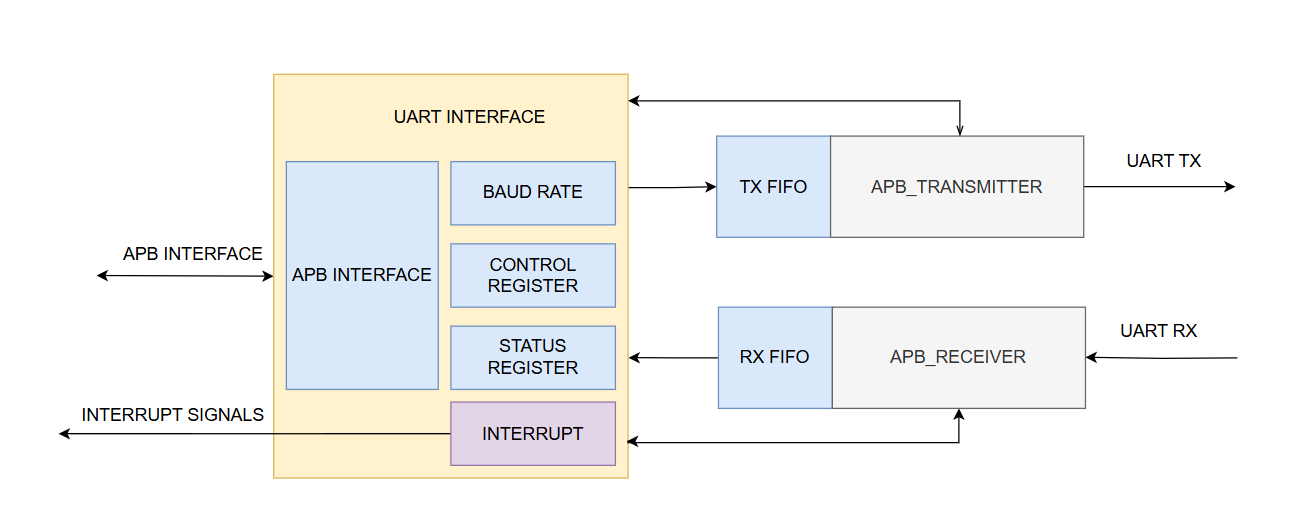
**Thông số kỹ thuật chi tiết:**

* **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- UART Top Module

****

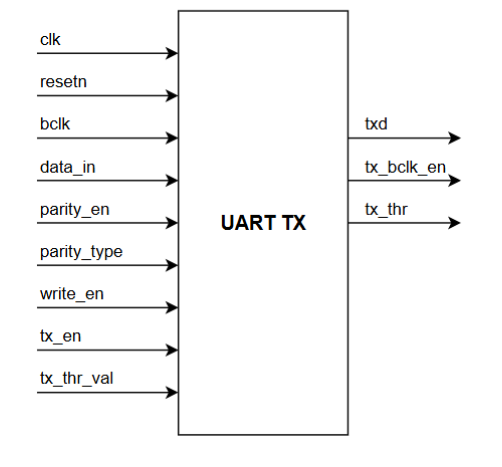
Hình 3- APB UART block diagram

|  |  |  |  |
| --- | --- | --- | --- |
| **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 Signal description APB UART

**3.4.4 UART Transmiter**

**Block diagram**

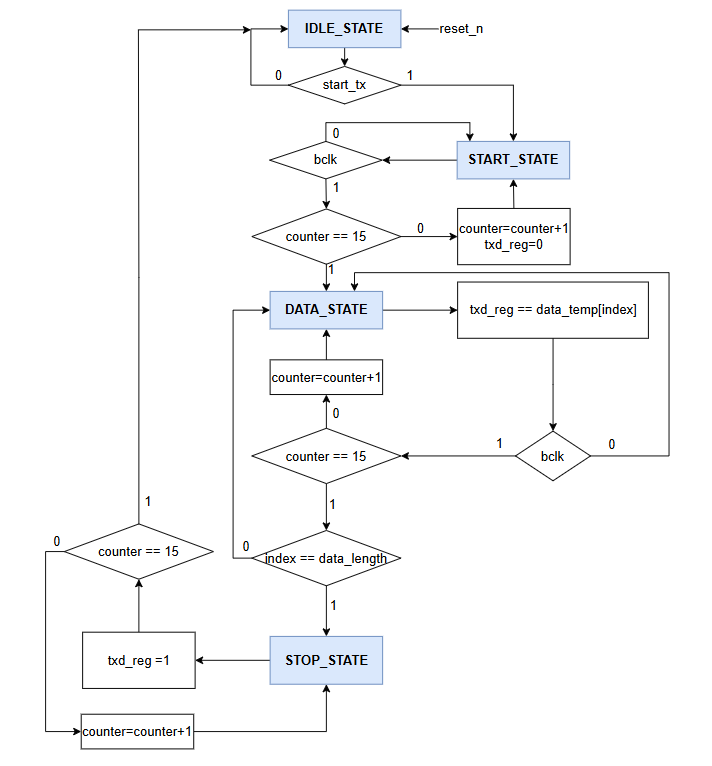
****

Hình 3- 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. |

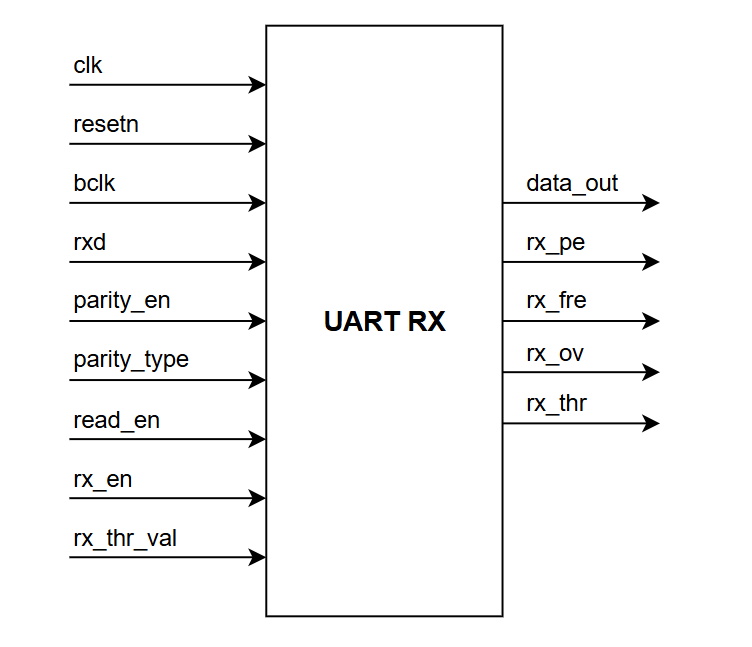
Bảng UART Transmiter signal description

****

Hình 3- UART transmiter FSMD

**3.4.5 UART Receiver**

**Block diagram**

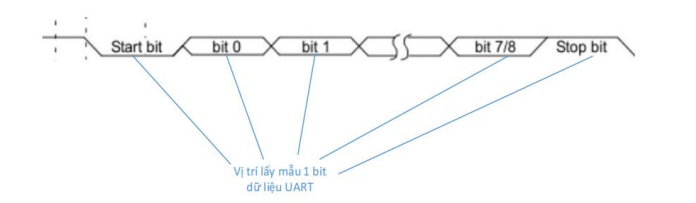
****

Hình 3- UART receiver block diagram

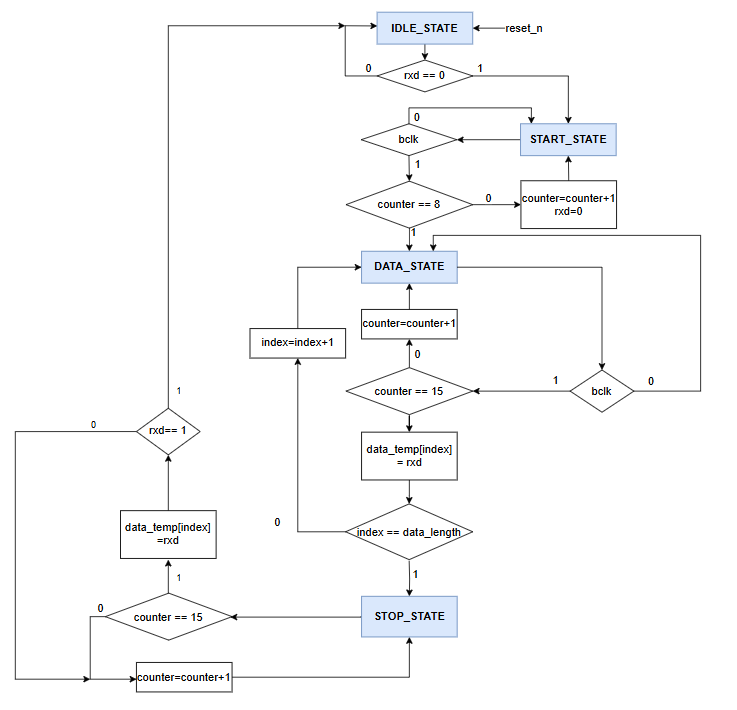
**Signal description**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 UART Receiver signal description

****

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



Hình 3- 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 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 | 0x00000100 | Register temporarily stores data |
| REG\_BCLK | Read/Write | 11 bits | 0x00000101 | Register stores BCLK value |
| REG\_EN | Read/Write | 8 bits | 0x00000110 | Register stores enable signals |
| REG\_THR | Read/Write | 4 bits | 0x00000111 | Register stores threshold for transmitter and receiver |

Bảng 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 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 Các tín hiệu ngắt có trong thiết kế

# 

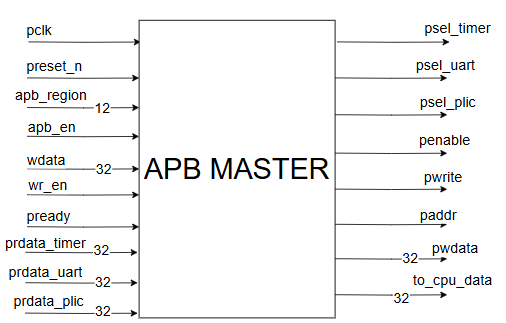
## 3.5 THIẾT KẾ HỆ THỐNG NGẮT

Giao tiếp giữa CPU và các IP:

|  |  |
| --- | --- |
| **Address Range** | **Description** |
| 0x7820 -- 0xFFFF | Reserved |
| 0x7810 -- 0x781F | Buttons |
| 0x7800 -- 0x780F | Switches (required) |
| 0x7040 -- 0x70FF | Reserved |
| 0x7030 -- 0x703F | LCD Control Registers |
| 0x7020 -- 0x7027 | Seven-segment LEDs |
| 0x7010 -- 0x701F | Green LEDs (required) |
| 0x7000 -- 0x700F | Red LEDs (required) |
| 0x4000 -- 0x6FFF | Reserved |
| 0x2000 -- 0x3FFF | Data Memory (8KiB using SDRAM) (required) |
| 0x0000 -- 0x1FFF | APB Region |

###### LSU Memory Mapping

Để giao tiếp với các IP qua APB Protocol, ta có thể sử dụng phương pháp ghi data bằng APB thông qua khối LSU với vùng địa chỉ được định sẵn. Ở đây, nhóm quyết định sử dụng vùng 0x0000 – 0x1FFFF là vùng đọc ghi data thông qua APB protocol.

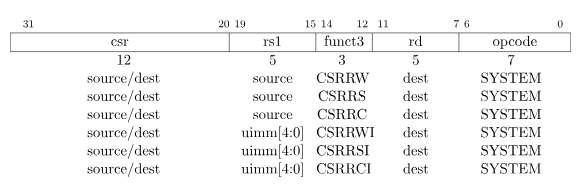


Hình 3- Sơ đồ khối APB Master

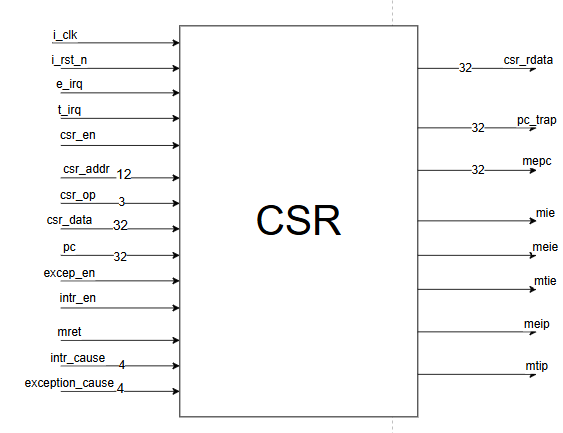
**Ngắt trong CPU:**

Để xử lý được ngắt, việc đầu tiên, CPU cần phải có các quy định về hoạt động của ngắt cũng như là các thanh ghi đảm nhiệm cho ngắt. Để làm được điều đó, nhóm đã xây dựng thanh ghi CSR phục vụ cho việc lưu trạng thái của CPU và máy trạng thái cho các tình huống xảy ra của ngắt.

Thanh ghi CSR:



Hình 3- Các lệnh phục vụ cho việc đọc & ghi dữ liệu vào thanh ghi CSR.



Hình 3- Sơ đồ khối của CSR

|  |  |  |  |
| --- | --- | --- | --- |
| Tín hiệu | Độ rộng | Direction | Chức năng |
| i\_clk | 1 | Input | Clock hệ thống |
| I\_rst\_n | 1 | Input | Reset hệ thống |
| E\_irq, t\_irq | 1 | Input | Các tín hiệu ngắt |
| Csr\_en | 1 | Input | Tín hiệu cho phép hoạt động CSR |
| Csr\_addr | 12 | Input | Thanh ghi cần ghi vào |
| Csr\_op | 3 | Input | Loại lệnh csr |
| Csr\_data | 32 | Input | Data đầu vào của csr |
| pc | 32 | Input | Địa chỉ lưu vào khi có ngắt |
| Excep\_en | 1 | Input | Kích hoạt nếu có exception xảy ra |
| Intr\_en | 1 | Input | Kích hoạt nếu có ngắt xảy ra |
| Mret | 1 | Input | Tín hiệu trở về khi thực hiện xong chương trình ngắt |
| Intr\_cause | 4 | Input | Nguyên nhân nguồn ngắt |
| Exception\_cause | 4 | Input | Nguyên nhân nguồn exception |
| Csr\_rdata | 32 | Output | Data đọc ra từ thanh ghi CSR |
| Pc\_trap | 32 | Output | Địa chỉ ngắt/exception(thường là địa chỉ tập hợp mọi chương trình ngắt) |
| Mepc | 32 | Output | Địa chỉ hiện tại của chương trình chính được lưu lại khi có ngắt xảy ra |
| Mie | 1 | Output | Cho phép ngắt chung |
| Meie | 1 | Output | Cho phép ngắt các nguồn ngắt external |
| Mtie | 1 | Output | Cho phép ngắt các nguồn ngắt timer |
| Meip | 1 | Output | Tín hiệu chờ khi có ngắt external xảy ra |
| Mtip | 1 | Output | Tín hiệu chờ khi có ngắt timer xảy ra |

Bảng Các tín hiệu của CSR

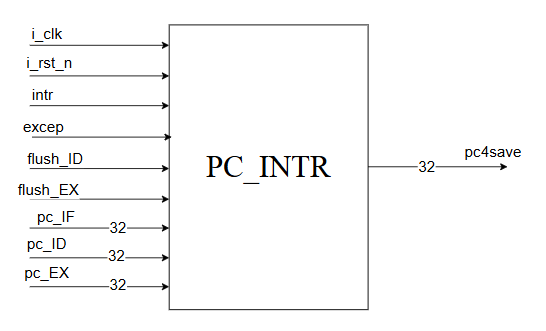
Việc thiết kế các câu lệnh này giúp ta dễ dàng hơn trong việc xác định nguồn ngắt và địa chỉ ngắt của các nguồn ngắt có thể xảy ra.

Các thanh ghi phục vụ cho chương trình ngắt:

|  |  |  |  |
| --- | --- | --- | --- |
| **Thanh ghi** | **Cấu trúc** | **Địa chỉ** | **Chức năng** |
| Mstatus | {24’b0,mpie, 3’b0, mie, 3’b0} | 0x300 | Bật/ tắt nguồn ngắt(mie) và lưu trạng thái mie(mpie) trước khi xảy ra trap |
| Mcause | Mcause | 0x342 | Lưu nguyên nhân gây ra trap |
| Mepc | {mepc\_base, 2’b00} | 0x341 | Lưu địa chỉ của lệnh bị trap |
| Mie | {20’b0, meie, 3’b0, mtie, 7’b0} | 0x304 | Chứa các bit cho phép enable từng loại ngắt riêng biệt |
| Mtvec | {mtvec\_base, mtvec\_mode} | 0x305 | lưu địa chỉ của handler xử lý trap(exception và interrupt) |
| Mip | {20’b0, meip, 3’b0, mtip, 7’b0} | 0x344 | Lưu trạng thái của các ngắt đang chờ xử lý |

Bảng Các thanh ghi phục vụ chương trình ngắt

Chọn địa chỉ PC để lưu khi có ngắt xảy ra:

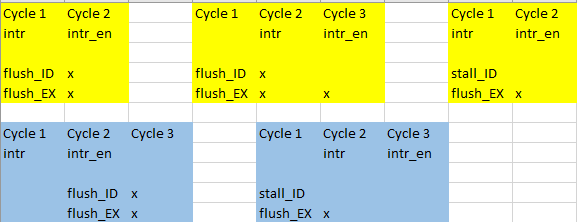


Hình 3- Sơ đồ khối chọn PC để lưu vào mepc

**Các trường hợp chọn PC:**

Thông thường, pipeline sẽ có 5 tầng, và có 3 tầng ảnh hưởng rất lớn nếu như có hazard xảy ra: IF stage, ID stage và EX stage.

Các data trọng yếu thường được cập nhật ở tầng MEM(với lệnh load hoặc store) và WB(với các lệnh liên quan tới register file). Chính vì vậy, để đảm bảo không có data nào bị bỏ lỡ, nhóm quyết định chọn pc ở tầng EX làm địa chỉ lưu nếu có interrupt xảy ra. Tuy nhiên, liệu có phải lúc nào pc ở tầng EX cũng chính xác?



Hình 3- Các trường hợp cần quan tâm.

Với các hình màu vàng, đây là các trường hợp ta không thể theo lý thuyết đặt ra. Với mô phỏng chương trình 1, ta có thể thấy, ngắt cần phải trải qua 2 cycle kể từ khi có ngắt chính thức để có thể ghi pc vào mepc. Chính vì vậy, nếu ngắt xảy ra đúng lúc có hazard xảy ra, nếu cứ theo lý thuyết lưu PC ở tầng EX, có thể sẽ dẫn tới sai địa chỉ(quay về địa chỉ ban đầu của chương trình). Ở tầng này, ta nên lưu PC ở tầng IF.

Tương tự với các trường hợp còn lại.

Với màu xanh, đây không phải là trường hợp đáng ngại, ta vẫn có thể thực hiện theo lý thuyết nhóm đề ra.

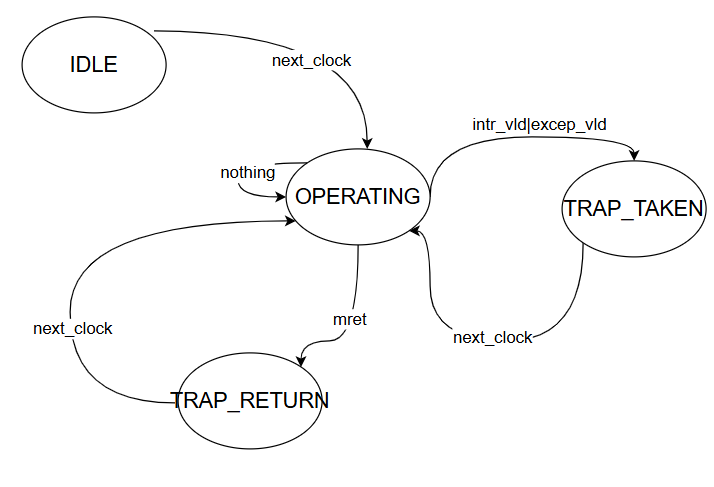
Với trường hợp có exception(cụ thể là lệnh sai), lệnh sẽ đi từ IF-ID-EX-MEM-WB, nhưng đến tầng ID, ta sẽ giải mã ra đây là lệnh nào. Nếu có sai sót về lệnh, exception\_vld sẽ được bật lên, dẫn tới phát hiện exception. Như vậy, nếu cứ theo lý thuyết là lấy pc\_EX làm địa chỉ lưu lại, lệnh sẽ fetch lại từ lệnh trước khi lệnh sai đó được fetch, sẽ tạo 1 vòng lặp không hồi kết. Vì thế, để tránh trường hợp đó, nhóm quyết định với trường hợp exception, ta sẽ chọn pc\_IF làm địa chỉ để lưu lại khi có exception xảy ra.

Tổng kết:

|  |  |
| --- | --- |
| Địa chỉ lưu | Điều kiện |
| Pc\_IF | Nếu có ngắt xảy ra đúng lúc có hazard  Có lệnh sai |
| Pc\_ID | 2 hình màu vàng ở góc trái ở hình trên |
| Pc\_EX | Các trường hợp còn lại |

Bảng Thống kê các trường hợp lưu lại địa chỉ khi có ngắt/ exception

Máy trạng thái hoạt động của CPU:



Hình 3- Máy trạng thái hoạt động của CPU

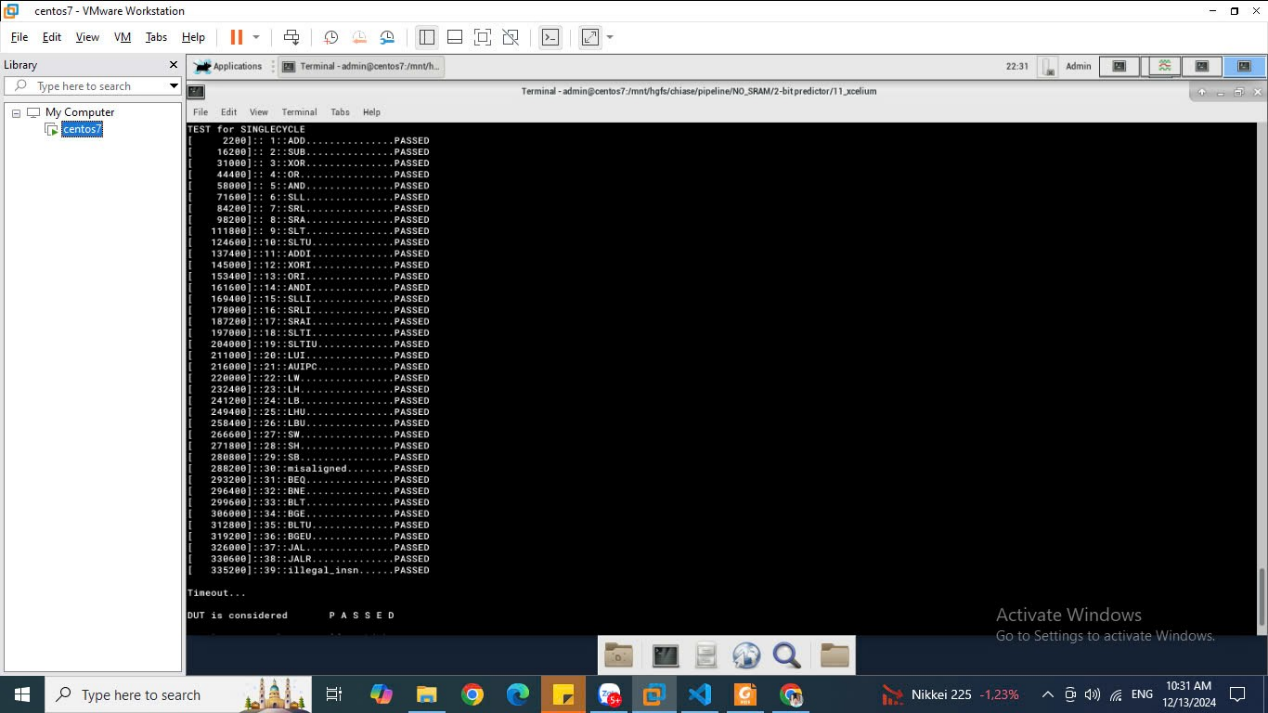
Các trạng thái của CPU:

* IDLE: Trạng thái ban đầu của CPU(khi ta khởi động lại).
* OPERATING: Trạng thái hoạt động của CPU(pc sẽ luôn cập nhật theo dự đoán của branch prediction).
* TRAP\_TAKEN: Trạng thái khi CPU phát hiện ra trap và cho phép nhảy vào chương trình ngắt để xử lý(ISR).
* TRAP\_RETURN: Trạng thái khi CPU đã xử lý xong chương trình ngắt.

# 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.

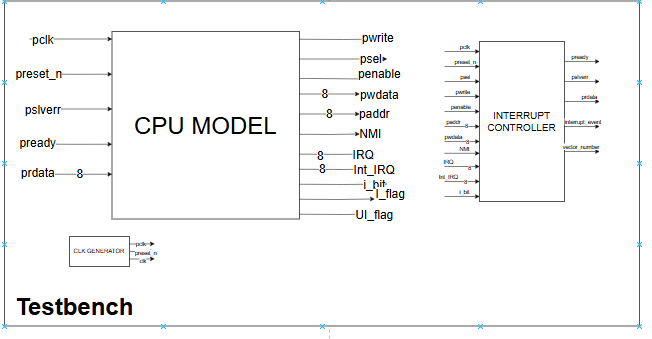


Hình 4- Các testcase đã được kiểm chứng chính xác

Hình 4- Các testcase đã được kiểm chứng chính xác

## 4.2. Interrupt controller

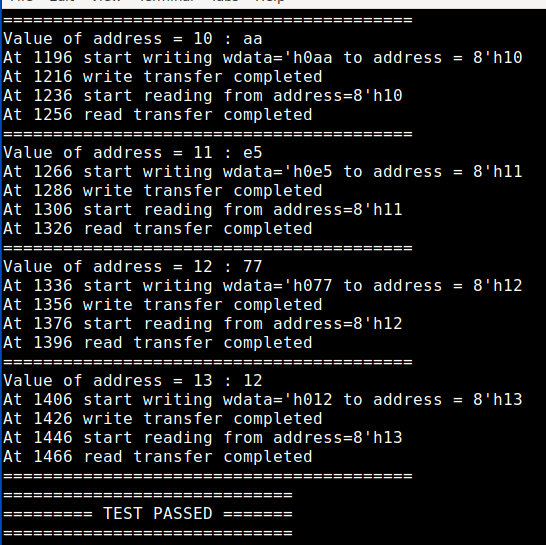
Nhóm sẽ viết môi trường testbench đơn giản để kiểm tra interrupt controller với việc ghi data vào các thanh ghi.



Hình 4- 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 tạo ra các tín hiệu cần cho interrupt controller để mô phỏng việc ghi và đọc data thông qua giao thức APB cơ bản.



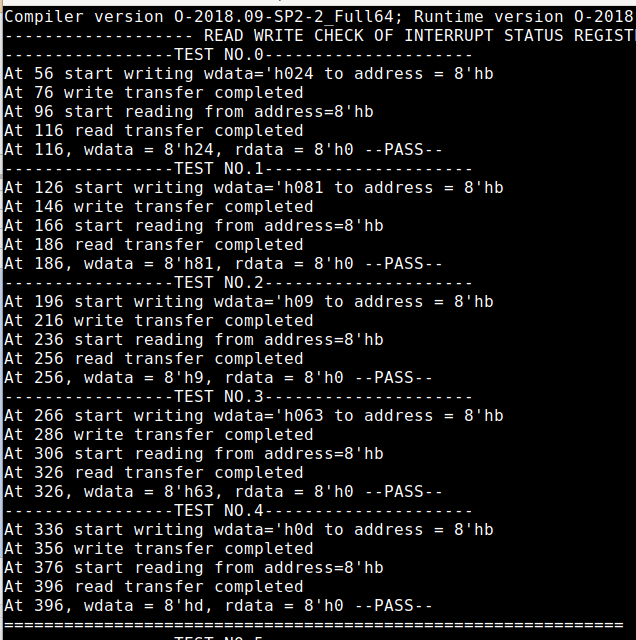
Hình 4- Kết quả sau khi test trường hợp ghi và đọc data cơ bản

1. **Basic Interrupt Testcase:**

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

1. Configure Interrupt Enable Register (IER): Chọn các tín hiệu cho phép ngắt của các nguồn ngắt
2. 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.
3. Trigger Interrupt Event: Kích tín hiệu intr\_ev thông qua bộ Controller.
4. 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).
5. ISR Execution: Clear the interrupt flag to acknowledge the event.: Chờ cờ ngắt được xóa.
6. 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.

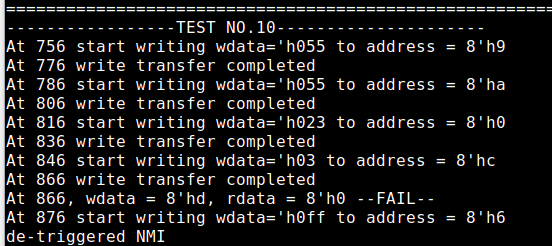
Trước hết, ta sẽ kiểm tra trạng thái R/W\* của ISRA, ISRB

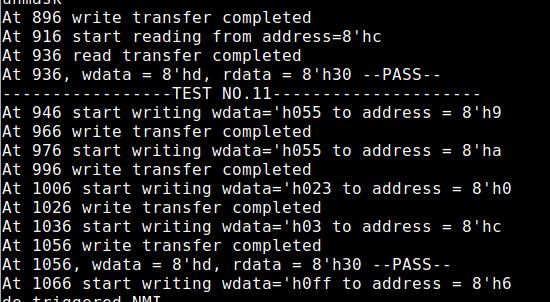


Hình 4- Kết quả sau khi đã test 5 testcase cơ bản.

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. .

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

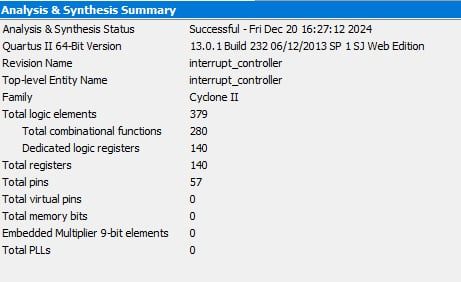




Hình 4- Kết quả sau khi test nguồn ngắt là loại cạnh xuống.

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.

Nhóm cũng tiến hành đánh giá tài nguyên interrupt controller trên FPGA

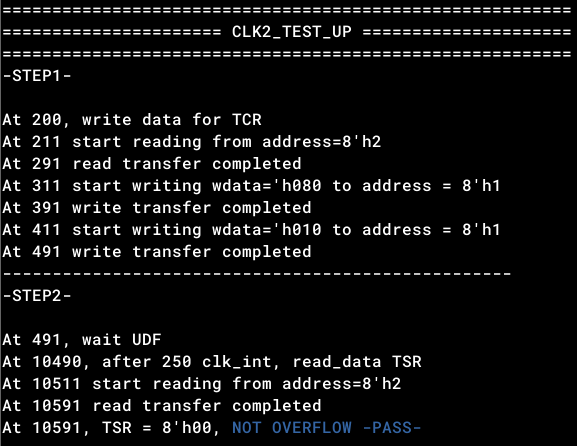


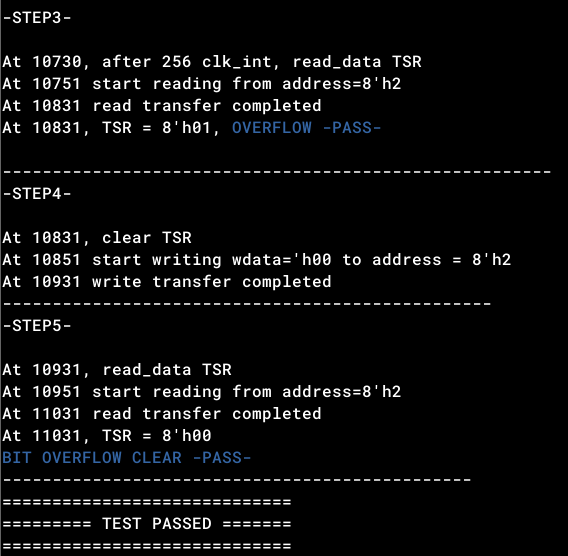
Hình 4- Đánh giá tài nguyên Interrupt Controller với Quartus II

## 4.3. IP TIMER

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:

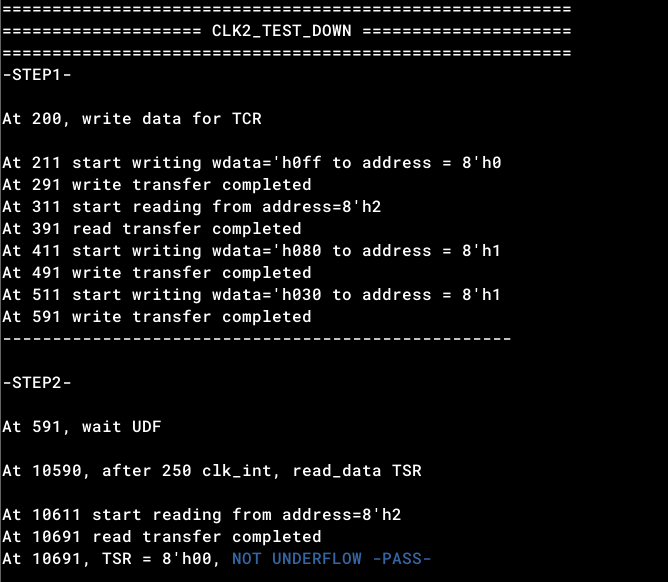
Với trường hợp đếm lên

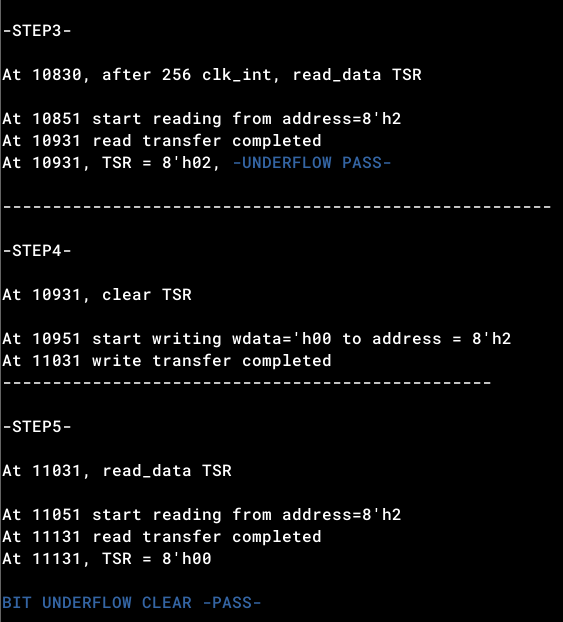




Hình 4- Kết quả với trường hợp đếm lên với hệ số clock là 2

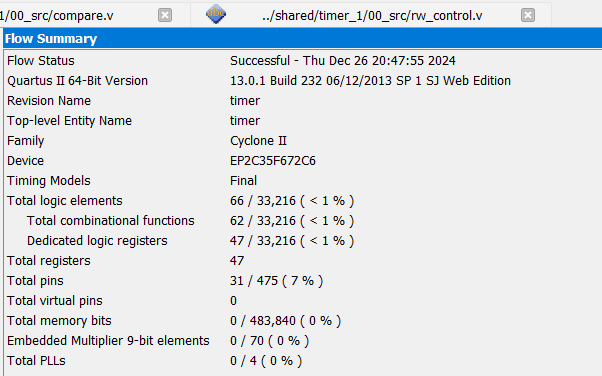
Với trường hợp đếm xuống:





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

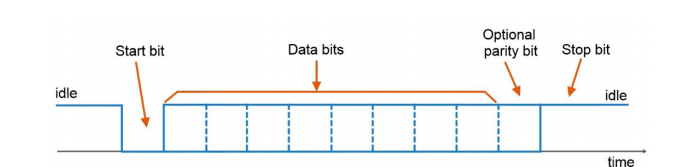
Có thể thấy thiết kế đã pass các test case cơ bản, sau đó nhóm tiến hành đánh giá tài nguyên thiết kế với phần mềm Quartus II:



Hình 4- Đánh giá tài nguyên Timer với Quartus

## 4.4. IP UART

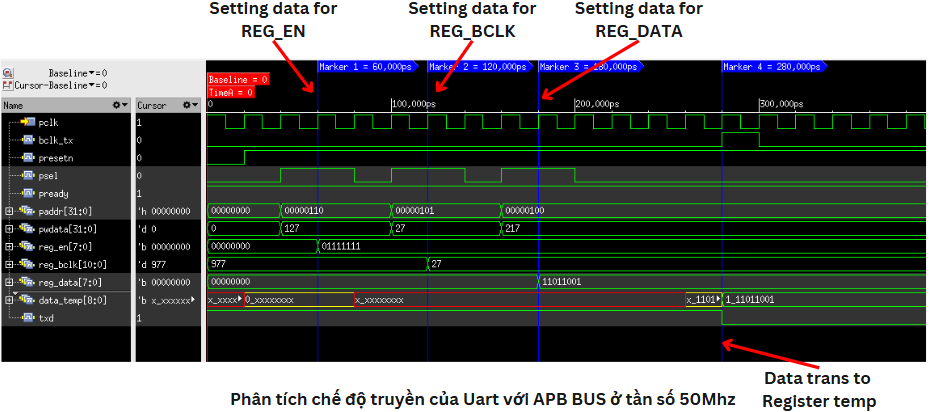
Nhóm tiến hành kiểm tra hoạt động của module Uart Transmitter và Uart Reciever so với lý thuyết cũng như ý tưởng thiết kế.

****

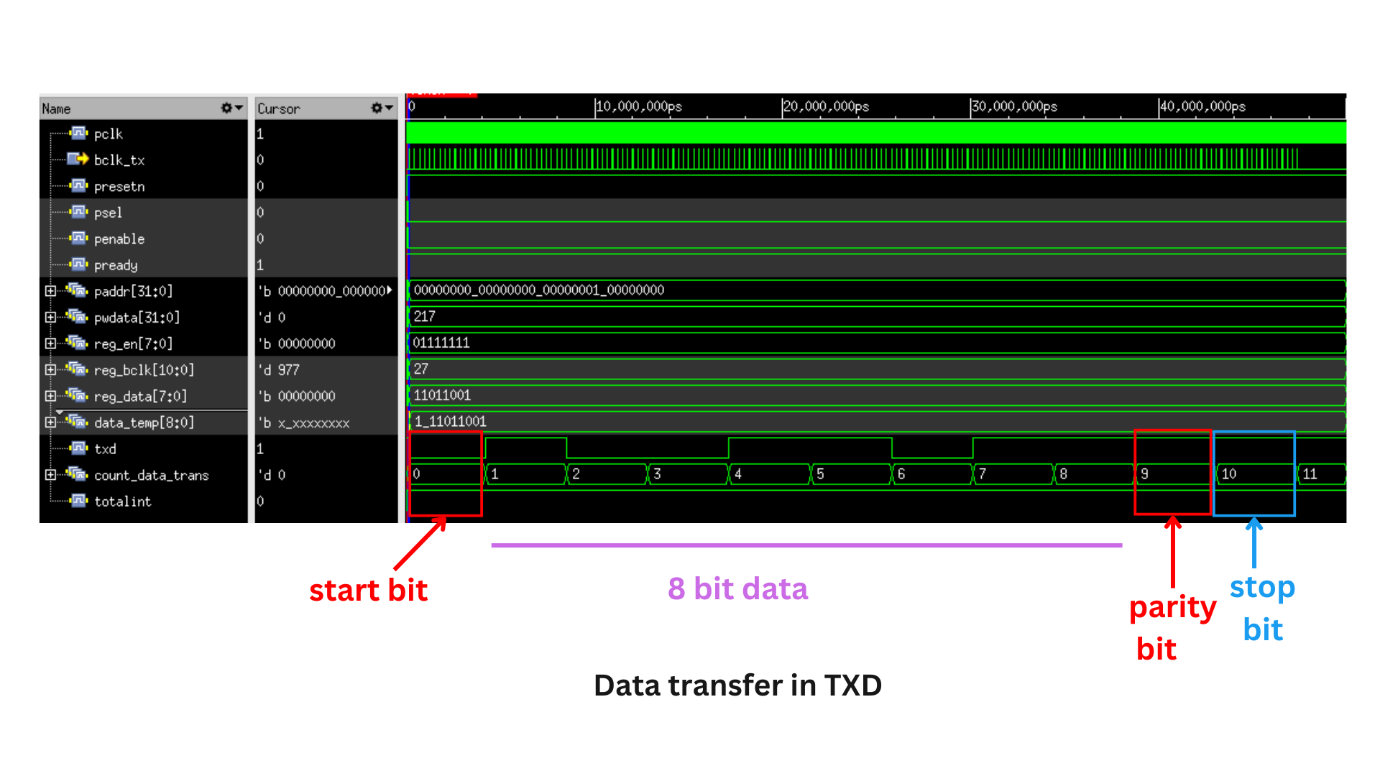
Hình 4- Waveform lý thuyết

****

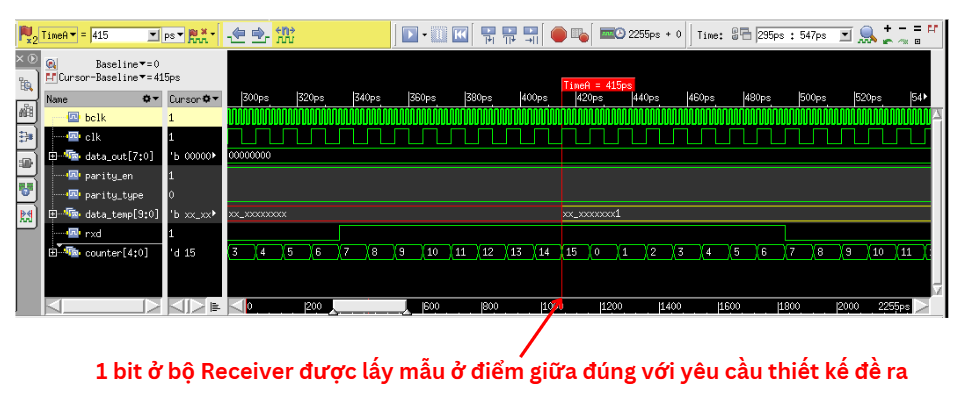
Hình 4- Phân tích thời gian của 1 bit UART ở khối Transmitter

****

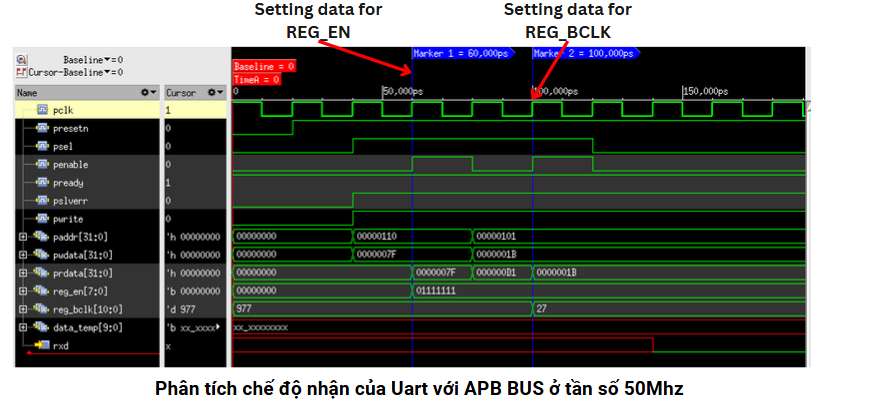
Hình 4- Phân tích chế độ truyền Uart



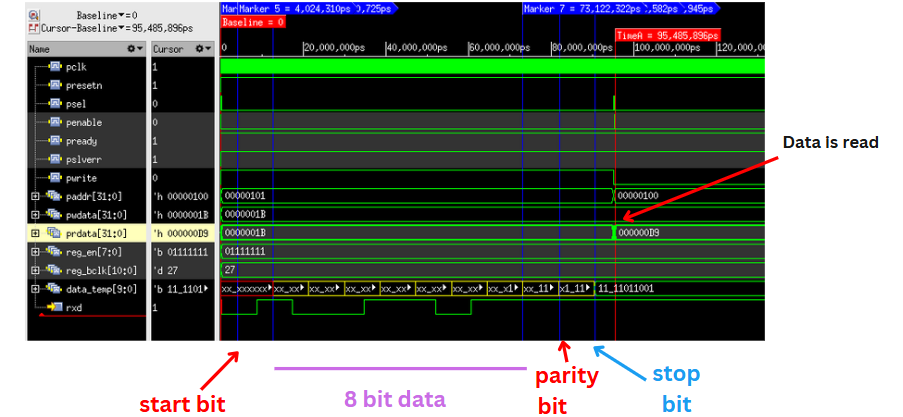
Hình 4- Phân tích Output ở khối Transmitter



Hình 4- Phân tích 1 bit ở khối Receiver

****

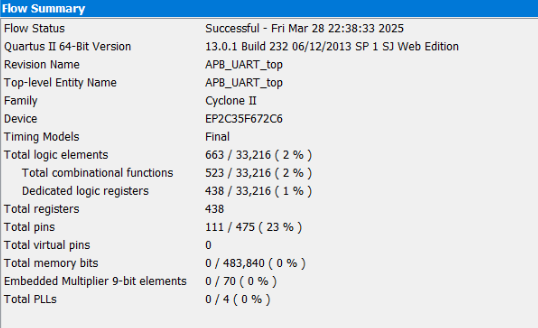
Hình 4- Phân tích chế độ nhận của Uart



Hình 4- Phân tích output ở khối Receiver

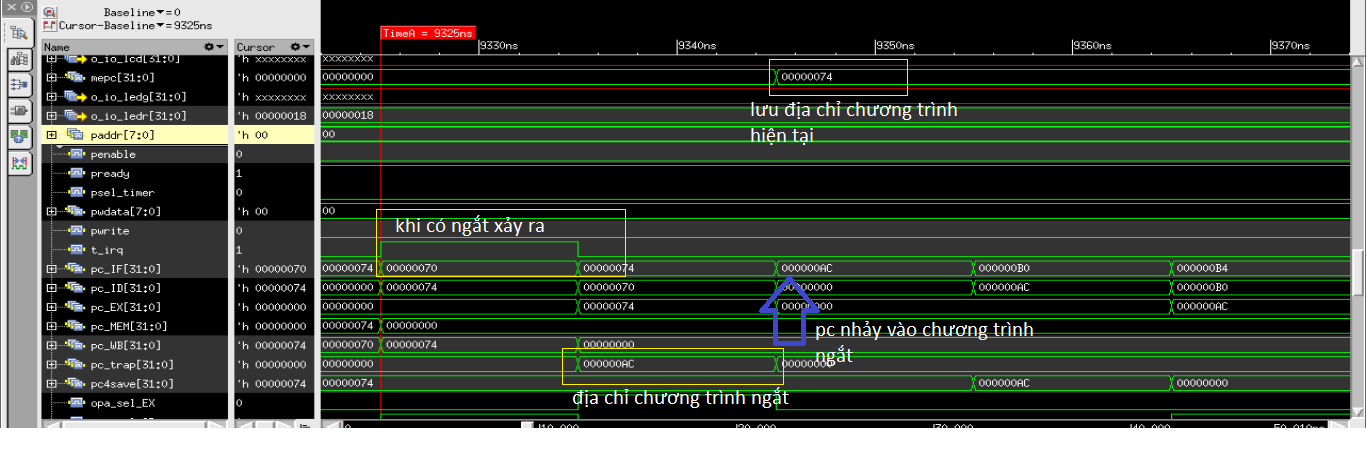
Từ Waveform lý thuyết và Waveform của thiết kế, chúng ta có thể thấy Waveform phù hợp với lý thuyết đã được đề ra.

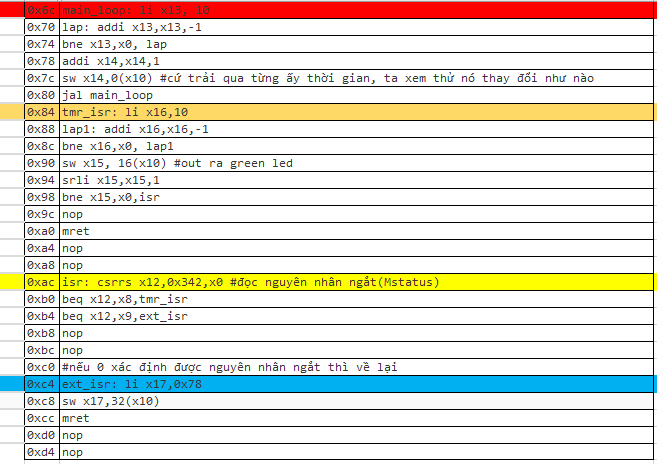
Sau đó nhóm tiến hành đánh giá tài nguyên với phần mềm Quartus



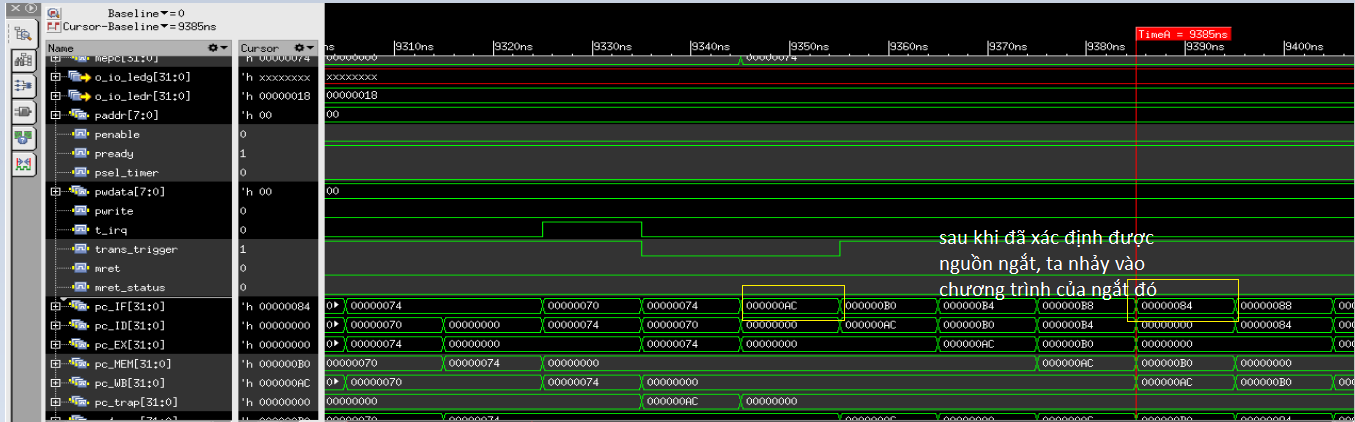
Hình 4- Đánh giá tài nguyên IP Uart với Quartus

## 4.5. HỆ THỐNG NGẮT

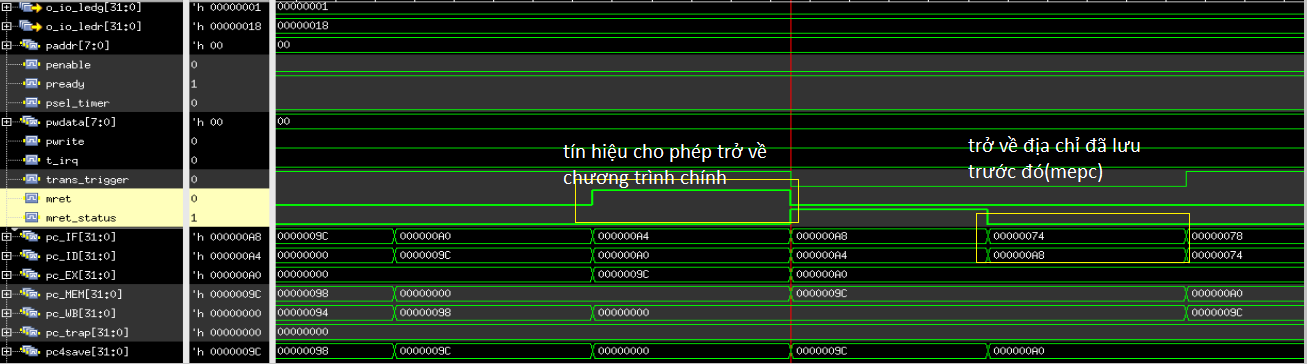
Hình 4- Ngắt Timer



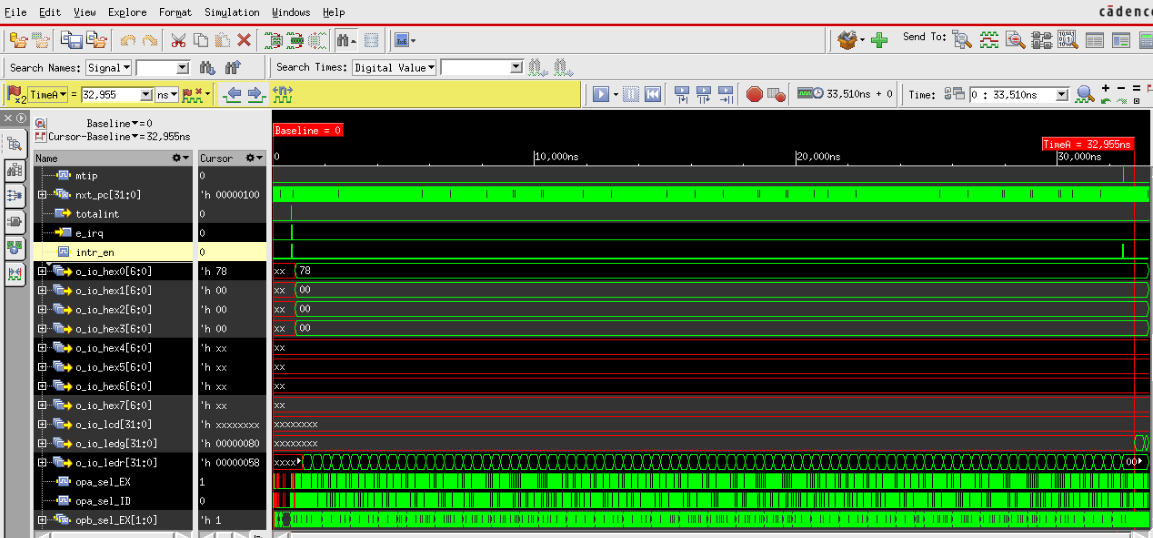
Hình 4- Chương trình mẫu



Hình 4- PC của CPU nhảy theo sự điều khiển của máy trạng thái



Hình 4- Sau khi xử lý chương trình ngắt, CPU trả lại địa chỉ chương trình hiện tại để thực thi tiếp chương trình đã bị ngắt



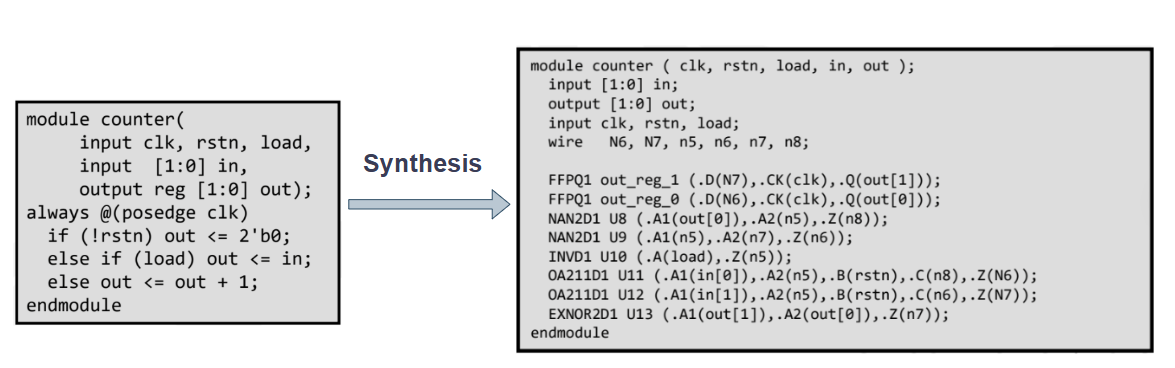
Hình 4- Tích hợp tất cả các IP: UART, TIMER, PLIC

# 5. ĐÁNH GIÁ THIẾT KẾ

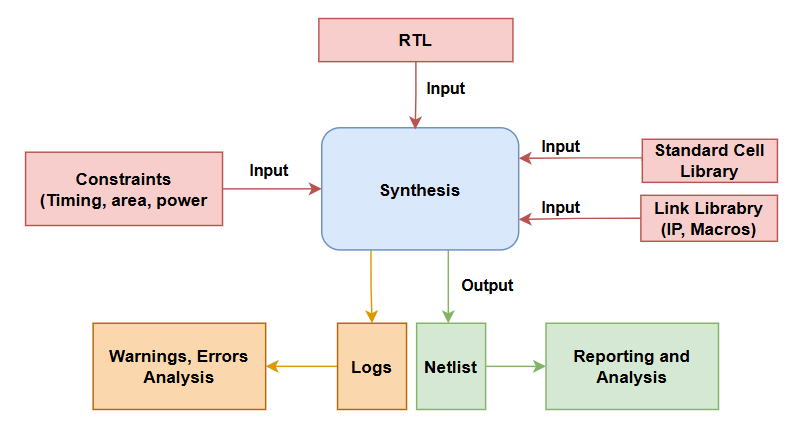
Để đánh giá thiết kế thì nhóm sử dụng công cụ Cadence Genus với thư viện SkyWater chu trình 130nm từ Openlane, cụ thể gồm 2 bước chính là Synthesis và đánh giá PPA.

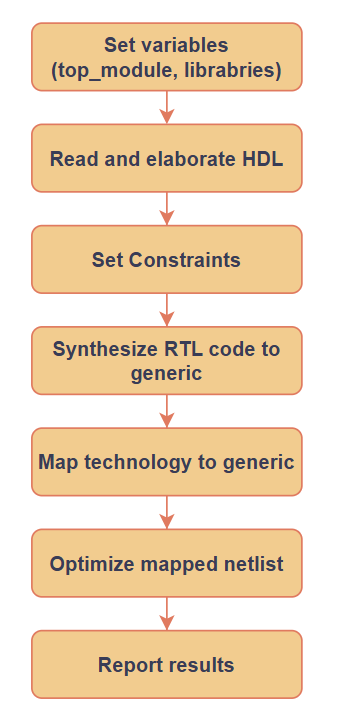
## ****5.1 LOGIC SYNTHESIS****

**Logic synthesis** (tổng hợp logic) là quá trình chuyển đổi thiết kế phần cứng ở cấp độ **Register Transfer Level (RTL)** sang một **gate-level netlist** được biểu diễn bằng các phần tử trong thư viện cell tiêu chuẩn của công nghệ mục tiêu. Mục tiêu của quá trình này là tối ưu hóa thiết kế theo các ràng buộc kỹ thuật nhằm đạt được hiệu suất cao, diện tích nhỏ và mức tiêu thụ điện năng thấp.



Hình 5- Minh họa về Synthesis



Hình 5- Tổng quan về Synthesis

Hình 5- Synthesis Flow Steps

## 5.2 STATIC TIMING ANALYSIS

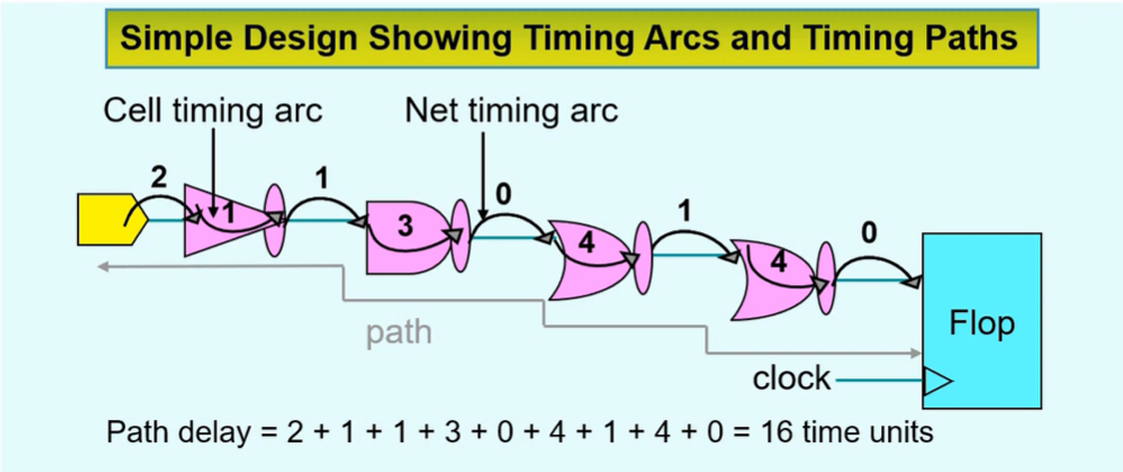
Sau bước Synthesis, nhóm tiến hành đánh giá thiết kế thông qua các reports và 1 trong những thông số quan trọng nhất là về timing. Với bước này, nhóm sử dụng phương pháp Static Timing analysis

### 5.2.1 Khái niệm về STA

**Static Timing Analysis (STA)** là một kỹ thuật **phân tích thời gian** trong mạch số **mà không cần mô phỏng toàn bộ mạch** .Thay vào đó, STA tính toán:

* **Độ trễ của các cổng logic (cell delays)**
* **Độ trễ của các đường dây nối (net delays)**

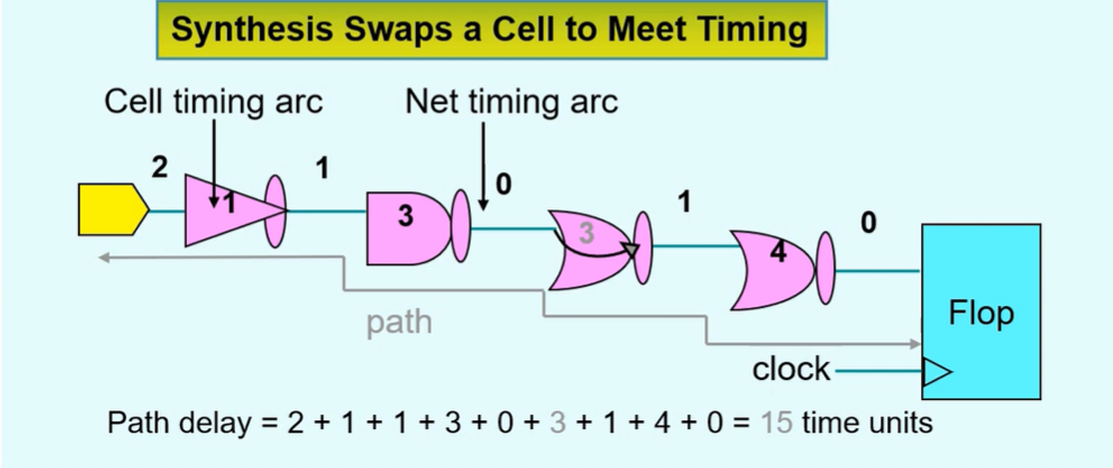
→ Sau đó cộng lại thành **tổng độ trễ trên từng đường truyền tín hiệu (timing path)** và **so sánh với yêu cầu thời gian** (timing constraints), như clock period.

Hình 5- Ví dụ về path delay

### 5.2.2 Mục đích của STA

**1. Tối ưu thiết kế (optimization)**: STA tính toán độ trễ của tất cả các đường tín hiệu quan trọng → giúp các công cụ tổng hợp (synthesis) và đặt-rút dây (place & route) **chọn thành phần logic phù hợp** từ thư viện (library) để đáp ứng yêu cầu thời gian (timing requirements).

**2. Xác minh thiết kế có hoạt động đúng tần số yêu cầu không (timing verification)**: Sau khi tổng hợp và layout, STA sẽ kiểm tra toàn bộ mạch để đảm bảo **không có đường tín hiệu nào bị vi phạm thời gian (timing violations)**.Điều này rất quan trọng để đảm bảo mạch hoạt động chính xác khi được đưa lên silicon thực tế hoặc FPGA.



Hình 5- Minh họa về kĩ thuật Cell Swap

### 5.2.3 Một số thông số quan trọng

**Combinational Cell**: Là các cổng logic thuộc mạch tổ hợp như AND, OR, XOR, NOT,... được sử dụng trong quá trình tổng hợp (synthesis) để thực hiện các phép toán logic không có liên quan đến xung clock.

**Sequential Cell**: Là các phần tử mạch tuần tự như latch (SR, D) và flip-flop (D, JK, SR, T), được sử dụng để lưu trữ dữ liệu đồng bộ theo xung clock trong quá trình synthesis.

**Buffer/Inverter (Buf/Inv)**: Được chèn vào mạng phân phối clock (clock routing) nhằm mục đích cân bằng độ trễ giữa các nhánh clock đến các phần tử tuần tự. Việc này giúp giảm thiểu sự sai lệch xung clock giữa các phần tử (clock skew), từ đó đảm bảo độ chính xác trong hoạt động của toàn mạch.

**Setup Time**: Là khoảng thời gian yêu cầu mà dữ liệu tại đầu vào của flip-flop cần phải **ổn định trước cạnh kích hoạt của xung clock** (posedge hoặc negedge) để đảm bảo dữ liệu được chốt đúng.

**Hold Time**: Là khoảng thời gian yêu cầu mà dữ liệu phải được **giữ ổn định sau cạnh kích hoạt của xung clock** để tránh việc dữ liệu mới ghi đè dữ liệu cũ, đảm bảo capture FF lưu đúng dữ liệu.

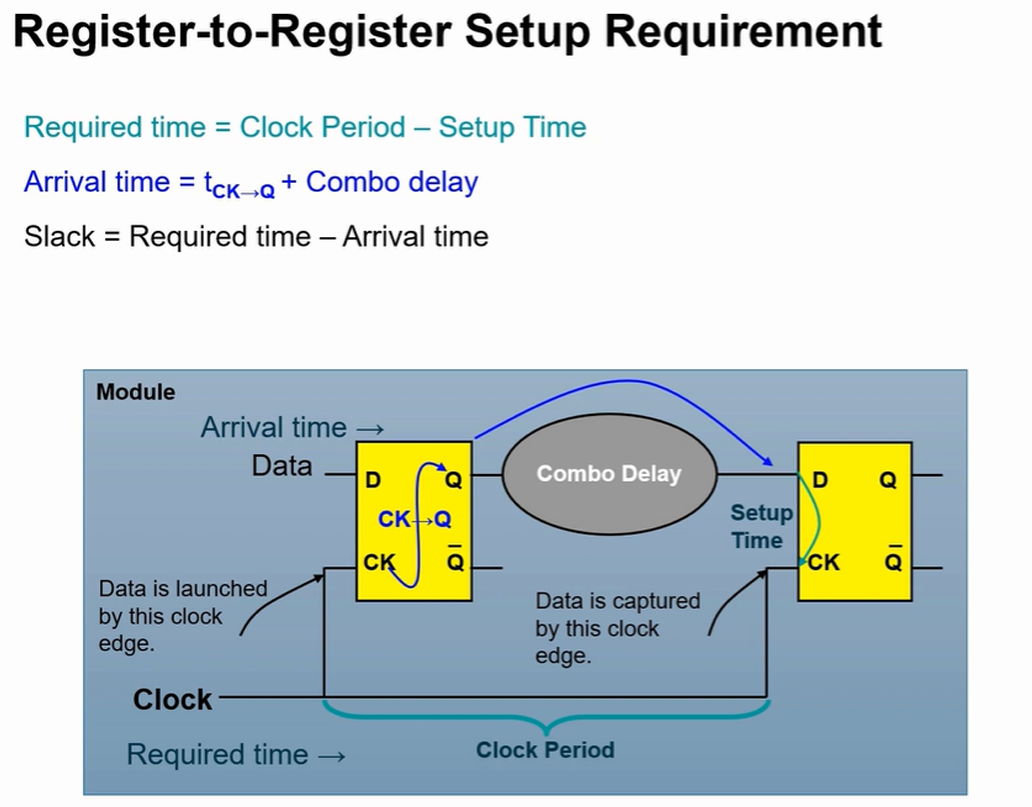
**Clock Network Delay**: Là tổng độ trễ phát sinh từ các phần tử buffer/inverter trong mạng phân phối clock từ nguồn đến các phần tử tuần tự như flip-flop hoặc latch.

**Data Required Time**: Là thời điểm yêu cầu mà dữ liệu phải xuất hiện tại đầu vào của phần tử tuần tự kế tiếp để đảm bảo hoạt động đúng logic. Thời điểm này được xác định dựa vào thông số setup hoặc hold time và độ trễ mạng clock.

**Data Arrival Time**: Là thời điểm thực tế mà dữ liệu đến tại đầu vào của phần tử tuần tự tiếp theo, tính từ đầu ra của flip-flop trước đó qua toàn bộ logic tổ hợp trung gian.

**Slack**: Là hiệu số giữa Required Time và Arrival Time. Slack thể hiện thời gian mà dữ liệu có thể đến trễ hoặc sớm hơn mà vẫn đảm bảo hoạt động đúng:

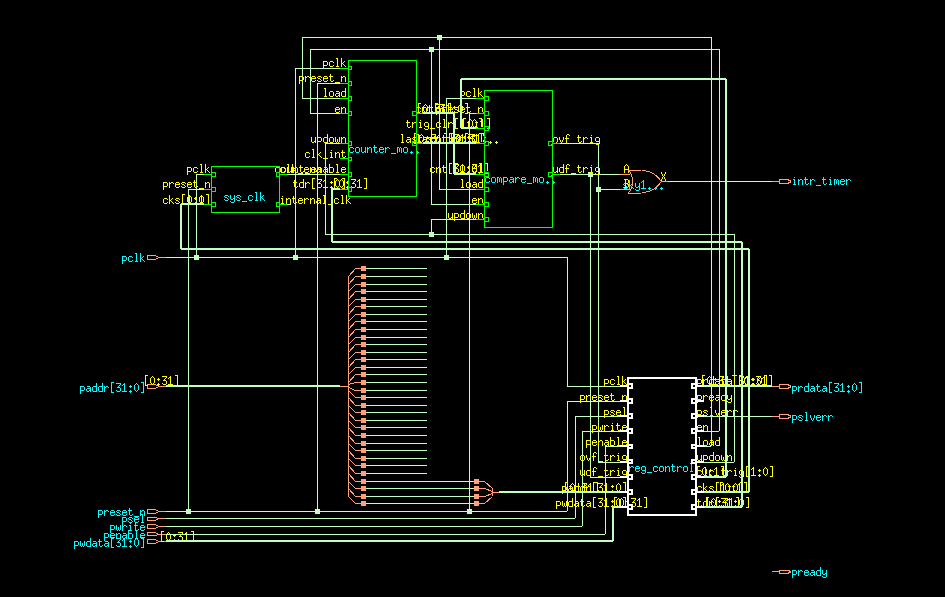
* **Slack > 0** → Đáp ứng yêu cầu thời gian (MET)
* **Slack < 0** → Vi phạm yêu cầu thời gian (VIOLATED)



Hình 5- Minh họa về cách tính Slack với trường hợp Setup Check

## 5.3 KẾT QUẢ ĐÁNH GIÁ THIẾT KẾ

### 5.3.1 IP TIMER

****

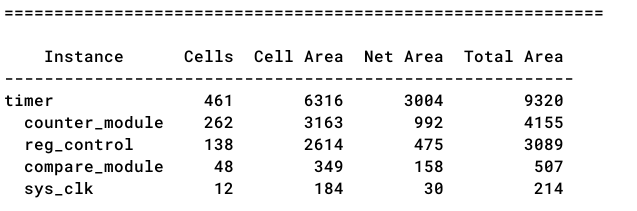
Hình 5- Schematic của IP TIMER sau khi Synthesis



Hình 5- Thông tin về timing của IP TIMER



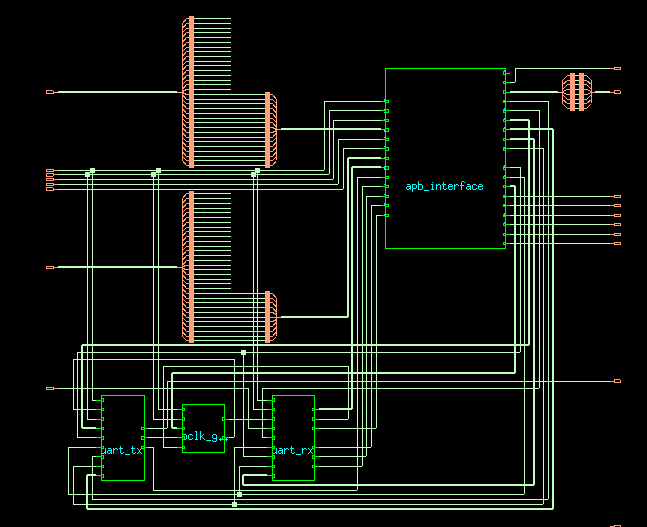
Hình 5- Thông tin về power của IP TIMER



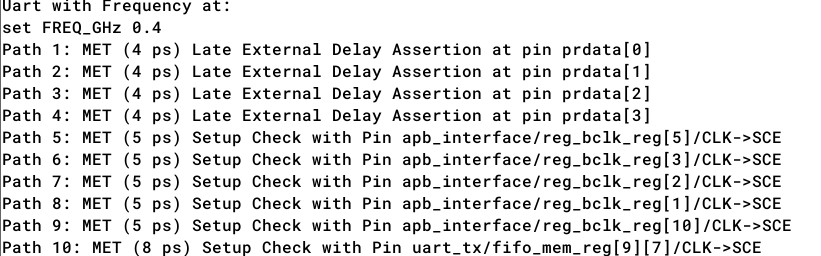
Hình 5- Thông tin về area của IP Timer

**Nhận xét:** Thiết kế IP Timer sau khi synthesis bằng Cadence Genus cho thấy kết quả tốt với tổng diện tích 9320 µm² và công suất tiêu thụ khoảng 3.85 mW, trong đó các thành phần counter\_module và reg\_control chiếm phần lớn tài nguyên. Timing đạt yêu cầu với độ lệch chỉ 4 ps tại tần số 300 MHz, phản ánh độ ổn định cao và khả năng đáp ứng thời gian tốt. Tổng thể, thiết kế đạt được sự cân bằng giữa hiệu năng, diện tích và công suất, phù hợp cho các ứng dụng nhúng hoặc hệ thống yêu cầu thời gian thực.

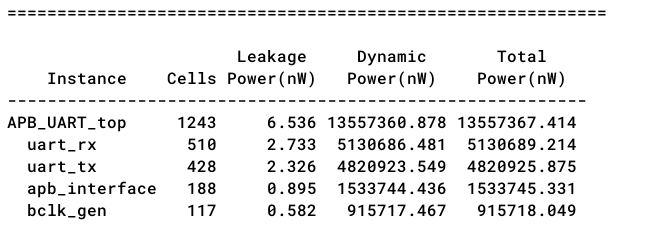
### 5.3.2 IP UART



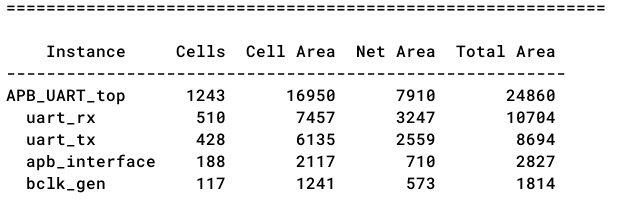
Hình 5- Schematic của IP UART sau khi Synthesis



Hình 5- Thông tin về timing của IP UART



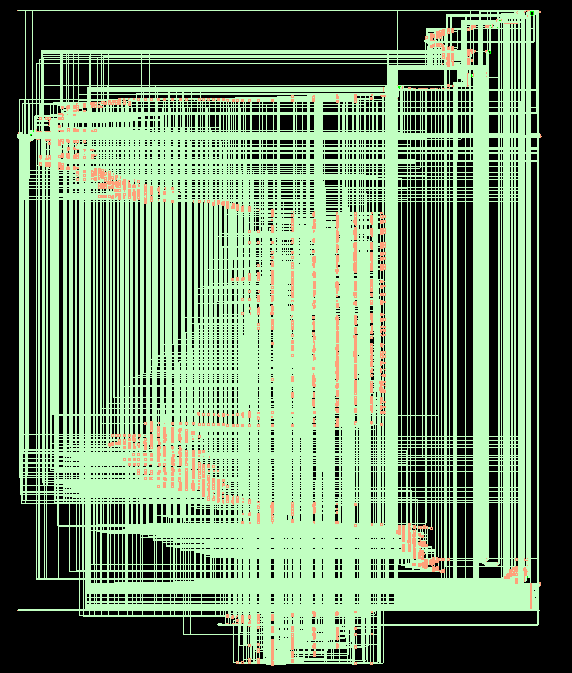
Hình 5- Thông tin về power của IP UART



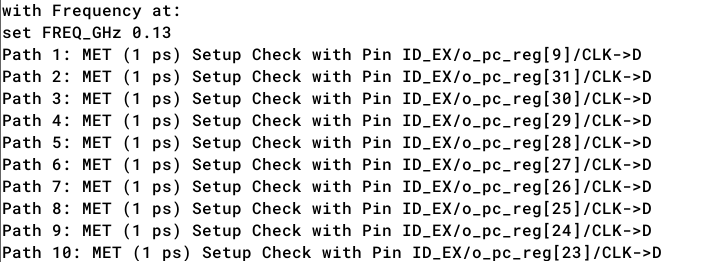
Hình 5- Thông tin về area của IP UART

**Nhận xét:** Thiết kế APB\_UART sau synthesis đạt kết quả tốt với **diện tích tổng cộng 24,860 µm²** và **công suất tiêu thụ khoảng 13.56 mW**, trong đó công suất động chiếm phần lớn. Các thành phần chính như uart\_rx, uart\_tx, và apb\_interface đều có mức tiêu thụ tài nguyên và năng lượng phù hợp với độ phức tạp chức năng tương ứng. Timing tại tần số **400 MHz** đều **đạt yêu cầu (MET)** với độ lệch nhỏ (4–8 ps), thể hiện tính ổn định của thiết kế. Tổng thể, IP APB\_UART đạt được sự cân bằng tốt giữa hiệu suất, tiêu thụ tài nguyên và độ tin cậy về thời gian, sẵn sàng tích hợp trong các SoC yêu cầu giao tiếp nối tiếp hiệu quả và tin cậy.

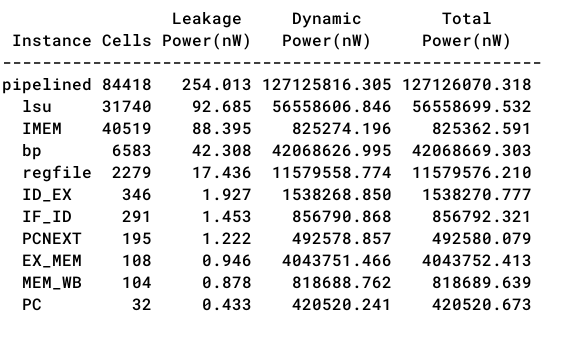
### 5.3.3 CPU



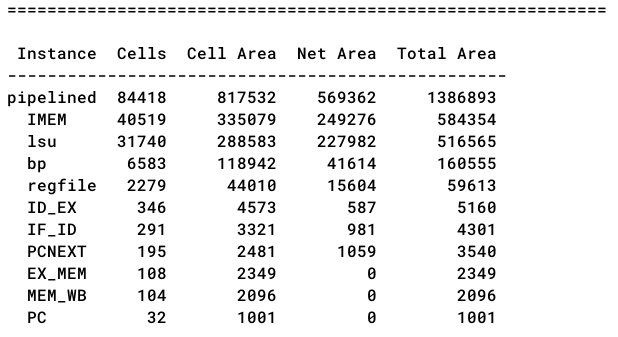
Hình 5- Schematic của CPU sau khi Synthesis

****

Hình 5- Thông tin về timing của CPU

****

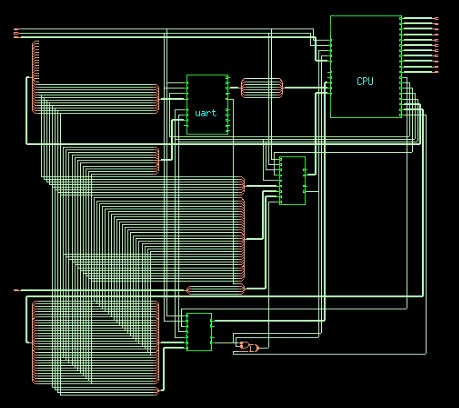
Hình 5- Thông tin về power của CPU

****

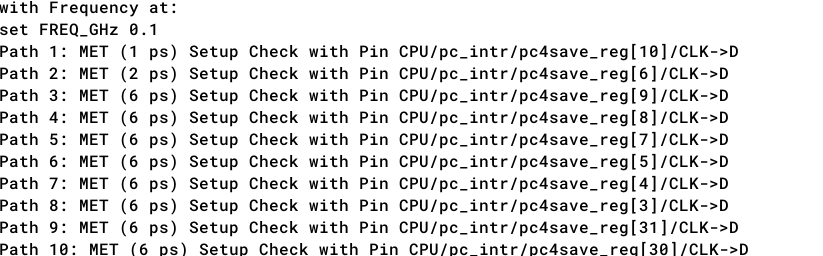
Hình 5- Thông tin về area của CPU

**Nhận xét:** Thiết kế CPU RISCV với kỹ thuật pipeline cũng như tích hợp 2 bit branch prediction đạt yêu cầu timing với tần số hoạt động 0.13 GHz, toàn bộ 10 đường truyền kiểm tra đều đạt (MET) với độ lệch chỉ 1 ps, cho thấy hệ thống định thời tốt và ổn định. Về công suất, tổng tiêu thụ là 127.1 mW, chủ yếu đến từ dynamic power phản ánh hoạt động switching tích cực trong các khối như lsu, bp, và regfile. Leakage power chỉ chiếm một phần nhỏ (~0.25 mW), cho thấy hiệu quả năng lượng ở trạng thái nghỉ. Tổng diện tích thiết kế là 1,386,893 μm², trong đó diện tích lớn nhất thuộc về các khối bộ nhớ và xử lý lệnh như IMEM, lsu, và bp. Net area chiếm khoảng 41% tổng diện tích, thể hiện mức độ kết nối cao giữa các khối logic. Tổng thể, thiết kế đảm bảo timing, tiêu thụ năng lượng và diện tích hợp lý, tuy nhiên vẫn có thể tối ưu thêm về diện tích và năng lượng nếu cần.

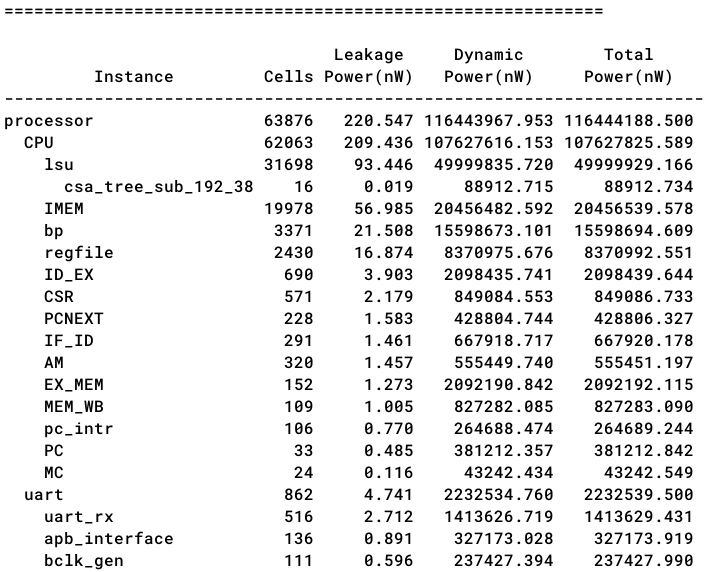
### 5.3.4 CPU tích hợp với các IP



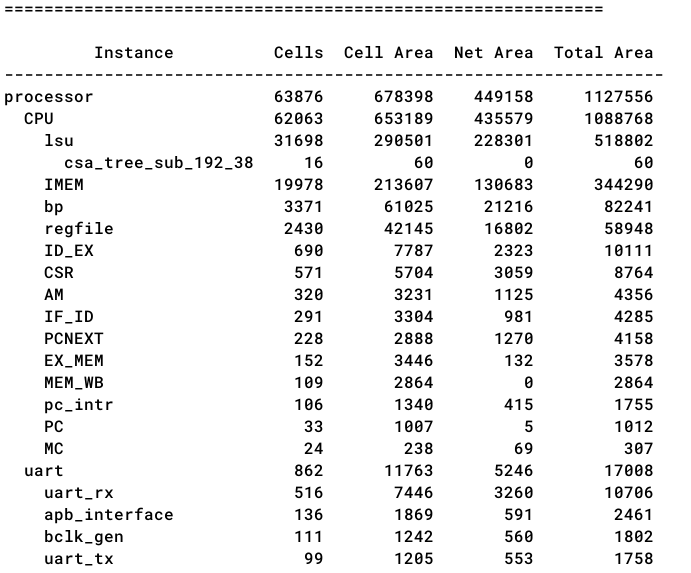
Hình 5- Schematic của CPU tích hợp các IP sau khi Synthesis

****

Hình 5- Thông tin về timing của CPU tích hợp các IP sau khi Synthesis

****

Hình 5- Thông tin về power của CPU tích hợp các IP sau khi Synthesis

****

Hình 5- Thông tin về area của CPU tích hợp các IP sau khi Synthesis

**Nhận xét:** Hệ thống hoạt động ở tần số 0.1 GHz đã đạt yêu cầu thời gian (timing met) ở tất cả các đường truyền chính, với độ trễ setup rất thấp (chỉ 1 ps), cho thấy khả năng đáp ứng tốt về thời gian của thiết kế. Về tiêu thụ công suất, tổng công suất tiêu thụ đạt khoảng 116,444 mW, trong đó công suất động chiếm phần lớn (116,443 mW), còn công suất rò chỉ khoảng 220 nW. Thành phần lsu (Load Store Unit) và regfile (register file) là hai khối tiêu thụ công suất lớn nhất. Điều này phản ánh hoạt động đọc/ghi dữ liệu và truy cập thanh ghi diễn ra thường xuyên trong hệ thống. Về diện tích, tổng diện tích sử dụng là khoảng 1.12 triệu µm², trong đó IMEM và lsu cũng là hai thành phần chiếm diện tích lớn nhất. Số lượng cell toàn hệ thống là hơn 63,000 cho thấy đây là một thiết kế có quy mô vừa phải.

## 5.4 MỘT SỐ KỸ THUẬT GIÚP CỦA THIỆN PPA

Trong thiết kế vi mạch số (Digital Integrated Circuit Design), việc tối ưu đồng thời các yếu tố **Timing**, **Power**, và **Area** là một bài toán then chốt. Tuy nhiên, các yếu tố này thường **mâu thuẫn** với nhau, dẫn đến sự cần thiết của các **trade-off** trong quá trình thiết kế. Thông qua quá trình làm đồ án tốt nghiệp, nhóm cũng đã nghiên cứu và áp dụng một số kỹ thuật phổ biến giúp cải thiện TPA kèm theo phân tích về ảnh hưởng qua lại giữa các yếu tố.

**1. Logic Optimization**

Bao gồm các kỹ thuật như logic restructuring, redundancy removal, Boolean algebra simplification,... nhằm rút ngắn critical path hoặc giảm switching activity.

* **Timing**: Có thể giảm độ trễ logic.
* **Power**: Giảm nếu loại bỏ được các logic dư thừa, giảm switching.
* **Area**: Tùy vào kỹ thuật, có thể giảm hoặc tăng nhẹ.

**Trade-off**: Tối ưu đồng thời nhiều mặt nhưng cần đánh giá chi tiết, vì trong một số trường hợp có thể làm tăng số cổng logic.

**2. Resource Sharing**

Dùng chung các khối phần cứng như ALU, bộ nhân, bộ chia, đặc biệt trong các thiết kế xử lý tuần tự hoặc bán song song.

* **Area**: Giảm do giảm số lượng phần tử phần cứng.
* **Power**: Có thể giảm nếu giảm switching đồng thời.
* **Timing**: Có thể tăng độ trễ do cần thêm mux hoặc logic điều khiển.

**Trade-off**: Thích hợp cho thiết kế tiết kiệm diện tích, nhưng có thể làm giảm throughput hoặc hiệu suất tổng thể.

**3. Pipelining**

Pipelining là kỹ thuật chia nhỏ các đường truyền dữ liệu (data path) thành nhiều giai đoạn, trong đó mỗi giai đoạn được đăng ký hóa bởi một thanh ghi. Mục tiêu chính là rút ngắn độ dài của **critical path** và do đó tăng được **tần số hoạt động (frequency)**.

* **Timing**: Cải thiện rõ rệt. Độ trễ giữa các tầng giảm, cho phép tần số đồng hồ cao hơn.
* **Power**: Tăng nhẹ, chủ yếu do dynamic power từ các flip-flop bổ sung.
* **Area**: Tăng do cần thêm tài nguyên (flip-flop, mux, điều khiển).

**Trade-off**: Đạt được hiệu suất cao hơn về tốc độ, nhưng đánh đổi bằng việc tiêu tốn thêm diện tích và năng lượng.

**4. Clock Gating**

Là kỹ thuật tiết kiệm năng lượng bằng cách **ngắt tín hiệu clock** đến các khối không hoạt động, tránh switching không cần thiết.

* **Power**: Giảm mạnh dynamic power.
* **Area**: Tăng nhẹ do thêm mạch gating (gating cells).
* **Timing**: Ảnh hưởng nhỏ nếu thiết kế tốt, nhưng có thể làm tăng skew nếu bố trí không hợp lý.

**Trade-off**: Cải thiện hiệu suất năng lượng đáng kể, phù hợp với các thiết kế yêu cầu tiêu thụ thấp (low-power designs).

5. **Multi-Vt Design**

Sử dụng transistor với ngưỡng điện áp khác nhau (High-Vt, Low-Vt) để cân bằng giữa tốc độ và rò rỉ điện.

* **Timing**: Cải thiện khi dùng Low-Vt cho các đường critical.
* **Power**: Giảm tổng thể khi dùng High-Vt ở các đường không nhạy thời gian.
* **Area**: Không ảnh hưởng lớn, nhưng thêm lớp mask phức tạp.

**Trade-off**: Thiết kế và sản xuất phức tạp hơn, phù hợp với tiến trình công nghệ tiên tiến.

# 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 6.1 Kết luận

So với nhiệm vụ đã đề ra lúc ban đầu, về cơ bản nhóm đã thiết kế và xác minh CPU RISCV tích hợp pipeline, các core IP với các chức năng cơ bản tuy nhiên còn một số hạn chế sau:

Thứ nhất, Interrupt Controller chưa xử lí được trường hợp ngắt lồng. Ví dụ, giả sử 1 trường hợp ngắt đang xảy ra và có 1 trường hợp ngắt với mức độ ưu tiên cao hơn thì chương trình sẽ tạm dừng chương trình ngắt hiện tại để thực hiện chương trình ngắt có mức độ ưu tiên cao hơn.

Thứ hai, nhóm chưa tối ưu hóa được thiết kế về các thông số như timing, area, power.

## 6.2 Hướng phát triển

1. Hỗ trợ xử lý ngắt lồng (Nested Interrupts): Phát triển cơ chế ưu tiên ngắt, cho phép tạm dừng một chương trình ngắt để xử lý ngắt có mức ưu tiên cao hơn, đồng thời đảm bảo lưu và khôi phục trạng thái đúng cách.

2. Tích hợp giao tiếp với các Core IP qua BUS: Thiết kế và triển khai hệ thống BUS (như AHB, AXI hoặc APB) để CPU có thể giao tiếp hiệu quả với các thiết bị ngoại vi hoặc core IP.

3. Cải thiện hiệu suất thiết kế: Sử dụng các kỹ thuật đã tìm hiểu nhằm tối ưu hóa thiết kế.

# 7. 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.

# 8. PHỤ LỤC