TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN THIẾT KẾ HỆ THỐNG SỐ VỚI HDL

CHỦ ĐỀ:

GIAO TIẾP DỮ LIỆU SONG PHƯƠNG UART GIỮA FPGA VÀ MÁY TÍNH – THIẾT KẾ HƯỚNG ỨNG DỤNG THỰC TẾ TRÊN KIT DE2

Nhóm thực hiện: Anh Quân, Thiên Quý, Thanh Phương

Lớp: CE213.P21

Giảng viên phụ trách: Hồ Ngọc Diễm

Tháng 05 năm 2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

THÔNG TIN NHÓM THỰC HIỆN

- Nhóm: Anh Quân, Thiên Quý, Thanh Phương

- Lớp: CE213.P21

STT	Họ và tên	MSSV	Nhiệm vụ	Đóng góp	Hoàn thành
1	Trương Thiên Quý	23521321	Viết code, viết báo cáo, làm slide, kiểm tra thực tế	33.33%	100%
2	Ngô Đỗ Anh Quân	23521257	Viết code, viết test bench, kiểm tra thực tế, chỉnh sửa video	33.33%	100%
3	Lê Huỳnh Thanh Phương	23521242	Mô phỏng, kiểm tra thực tế, viết code, viết báo cáo	33.33%	100%

I. Tổng quan đề tài

1.1. Lý do chọn đề tài

Trong bối cảnh công nghệ số ngày càng phát triển, nhu cầu kết nối và trao đổi dữ liệu giữa các thiết bị phần cứng và phần mềm đóng vai trò then chốt trong nhiều lĩnh vực như tự động hóa, điều khiển nhúng, giám sát từ xa và các **hệ thống IoT**. Đặc biệt, giao tiếp nối tiếp theo chuẩn **UART (Universal Asynchronous Receiver/Transmitter)** là một trong những chuẩn truyền thông phổ biến nhất nhờ cấu trúc đơn giản, dễ triển khai và tính linh hoạt cao trong việc truyền dữ liệu không đồng bộ.

Bên cạnh đó, **FPGA (Field Programmable Gate Array)** đang ngày càng được ứng dụng rộng rãi nhờ khả năng tùy biến phần cứng mạnh mẽ, hiệu năng cao và tính tái cấu hình. Tuy nhiên, để phát huy được toàn diện tiềm năng của **FPGA** trong các hệ thống nhúng, việc tích hợp giao tiếp dữ liệu với môi trường bên ngoài là điều kiện cần thiết. Giao tiếp song phương **UART giữa FPGA** và máy tính chính là một ví dụ điển hình cho khả năng mở rộng, tương tác và điều khiển giữa phần cứng với phần mềm trong thời gian thực.

Việc lựa chọn kit **DE2** làm nền tảng hiện thực giúp sinh viên không chỉ nắm vững kiến thức lý thuyết về kiến trúc số, thiết kế hệ thống trên chip (SoC), mà còn được tiếp cận với môi trường phát triển chuyên nghiệp, phù hợp với các yêu cầu kỹ thuật trong thực tế sản xuất và nghiên cứu.

Từ những lý do trên, nhóm thực hiện chọn đề tài **"Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính – Thiết kế hướng ứng dụng thực tế trên kit DE2"** với mong muốn xây dựng một hệ thống truyền thông hiệu quả, có khả năng ứng dụng thực tiễn, đồng thời nâng cao kỹ năng thiết kế phần cứng, giao tiếp ngoại vi và tư duy tích hợp hệ thống – những yếu tố quan trọng trong lĩnh vực kỹ thuật điều khiển, điện tử và tự động hóa hiện đại.

1.2. Mục tiêu của đề tài

Đề tài **"Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính – Thiết kế hướng ứng dụng thực tế trên kit DE2"** được thực hiện với mục tiêu tổng quát là nghiên cứu, thiết kế và hiện thực một hệ thống giao tiếp nối tiếp hai chiều giữa FPGA và máy tính dựa trên chuẩn UART, từ đó làm nền tảng cho việc phát triển các ứng dụng truyền thông nhúng trong thực tế. Cụ thể, đề tài hướng đến việc đạt được các mục tiêu sau:

1. Nắm vững lý thuyết và cấu trúc chuẩn truyền thông UART

- Tìm hiểu chi tiết về nguyên lý hoạt động của UART, bao gồm khung dữ liệu (start bit, data bits, optional parity bit, stop bit), tốc độ truyền (baud rate), và các kỹ thuật phát hiện lỗi (parity check, framing error).
- Phân tích cách thức đồng bộ hóa dữ liệu trong truyền thông không đồng bộ, cơ chế truyền đơn hướng và song hướng, cũng như các yếu tố ảnh hưởng đến hiệu suất giao tiếp.

2. Thiết kế và hiện thực hệ thống truyền thông UART song phương trên FPGA

- Phát triển mô-đun UART Transmitter và UART Receiver bằng ngôn ngữ mô tả phần cứng Verilog.
- Hiện thực chức năng truyền và nhận dữ liệu song song (full-duplex) giữa kit Altera DE2 và máy tính thông qua cổng RS232 hoặc USB, bảo đảm tốc độ ổn định và độ tin cậy cao.
- Tích hợp bộ điều khiển giao tiếp UART vào kiến trúc hệ thống số tổng thể trên FPGA.

3. Hiển thị và giám sát dữ liệu truyền nhận theo thời gian thực

- Thiết kế cơ chế hiển thị dữ liệu đầu vào và đầu ra thông qua các thiết bị ngoại vi có sẵn trên kit DE2 như: LED đơn (LEDR, LEDG) và LED 7 đoạn (HEX0–HEX7).

- Cho phép người dùng dễ dàng quan sát và đánh giá trực quan quá trình truyền thông dữ liệu.

4. Hỗ trợ cấu hình động tốc độ truyền UART thông qua phần cứng

- Cho phép thay đổi giá trị baud rate trong thời gian thực thông qua các công tắc gạt (SW) và nút nhấn (KEY) trên kit FPGA.
- Hỗ trợ các mức tốc độ phổ biến như 9600, 19200, 38400 bps,… giúp thử nghiệm tính ổn định của hệ thống dưới các điều kiện khác nhau.

5. Kiểm thử và đánh giá hiệu năng truyền thông UART

- Sử dụng phần mềm mô phỏng cổng Serial như **PuTTY**, **Tera Term** hoặc ứng dụng PC tự phát triển để kiểm thử khả năng gửi/nhận dữ liệu giữa FPGA và máy tính.

- Đánh giá độ trễ, tỷ lệ lỗi truyền, và độ tin cậy của hệ thống trong điều kiện hoạt động thực tế.

Thông qua việc hiện thực đầy đủ các mục tiêu trên, đề tài không chỉ giúp sinh viên làm chủ kỹ thuật truyền thông UART từ tầng phần cứng, mà còn rèn luyện kỹ năng thiết kế hệ thống số, tư duy tích hợp phần cứng – phần mềm, đồng thời mở rộng khả năng ứng dụng kiến thức vào các hệ thống nhúng và điều khiển hiện đại.

1.3. Phạm vi đề tài

Đề tài **"Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính – Thiết kế hướng ứng dụng thực tế trên kit DE2"** tập trung triển khai trong phạm vi cụ thể, phù hợp với thời lượng thực hiện, năng lực sinh viên, và tài nguyên phần cứng hiện có. Các giới hạn và phạm vi chính của đề tài bao gồm:

1. Về phần cứng:

- Sử dụng **kit phát triển DE2** của hãng Altera (Intel), tích hợp FPGA Cyclone II, làm nền tảng để thiết kế và hiện thực toàn bộ hệ thống.
- Giao tiếp giữa FPGA và máy tính được thực hiện qua **chuẩn UART thông qua cổng RS232** có sẵn trên kit, hoặc **USB-to-Serial** (khi máy tính không hỗ trợ RS232 vật lý).
- Hiển thị dữ liệu truyền nhận và tín hiệu trạng thái qua **LED đơn (LEDR, LEDG)** và **LED 7 đoạn (HEX0–HEX7)** trên kit.
- Tốc độ baud có thể được **cấu hình thủ công** bằng cách sử dụng **công tắc SW** trên kit DE2.

2. Về phần mềm và lập trình:

- Toàn bộ hệ thống UART được thiết kế **thuần túy bằng Verilog HDL**, không sử dụng lõi IP có sẵn để đảm bảo hiểu sâu và làm chủ từng thành phần truyền thông.
- Không tích hợp bộ xử lý mềm như Nios II, không sử dụng hệ điều hành nhúng, nhằm giữ đề tài ở mức phần cứng cơ bản và thuần túy logic số.
- Sử dụng phần mềm mô phỏng và kiểm thử UART trên máy tính như **PuTTY**, **Tera Term** để giao tiếp và giám sát dữ liệu với FPGA.

3. Về chức năng hệ thống:

- Hệ thống cho phép **truyền và nhận dữ liệu dạng ký tự ASCII** giữa máy tính và FPGA (ví dụ: gửi chữ, số, hoặc ký hiệu).
- Hỗ trợ **truyền thông hai chiều (song phương)**, nhưng **không thực hiện đồng thời nhiều luồng dữ liệu** hoặc buffer phức tạp.
- Không triển khai các chức năng cao cấp như giao tiếp đa thiết bị (multi-drop), FIFO mở rộng, hoặc CRC nâng cao.

4. Về phạm vi nghiên cứu:

- Tập trung nghiên cứu **chuẩn giao tiếp UART** ở mức phần cứng (hardware level) – không đi sâu vào các tầng giao thức truyền thông cấp cao như Modbus, CAN, hoặc TCP/IP.
- Không triển khai ứng dụng điều khiển thiết bị thực tế từ máy tính (như robot, động cơ, cảm biến) – tuy nhiên đề tài có thể được **mở rộng về sau** để phục vụ các hệ thống điều khiển từ xa hoặc IoT.

5. Về giới hạn thời gian và nhân lực:

- Đề tài được triển khai bởi nhóm sinh viên với **thời gian thực hiện giới hạn trong một học kỳ** (hoặc tương đương), do đó chỉ tập trung xây dựng một hệ thống UART **ổn định, tối giản nhưng có khả năng ứng dụng thực tế**.
- Không tích hợp các thuật toán phức tạp về mã hóa, nén, hoặc điều khiển lỗi nâng cao.

1.4. Điểm nổi bật của đề tài

Đề tài **"Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính – Thiết kế hướng ứng dụng thực tế trên kit DE2"** không chỉ đơn thuần dừng lại ở việc mô phỏng lý thuyết, mà còn tập trung vào quá trình hiện thực hóa toàn bộ hệ thống giao tiếp dữ liệu ở cấp độ phần cứng. Các điểm nổi bật chính của đề tài có thể được tóm tắt như sau:

1. Hiện thực chuẩn UART từ đầu bằng ngôn ngữ mô tả phần cứng

- Không sử dụng các IP core có sẵn, đề tài xây dựng **toàn bộ các khối UART Transmitter và Receiver từ đầu bằng Verilog HDL**.
- Việc này giúp người thực hiện **nắm vững nguyên lý hoạt động chi tiết của UART**: từ tạo xung baud, định dạng khung truyền, đến quá trình đồng bộ và kiểm tra lỗi.

- Cách tiếp cận này rèn luyện kỹ năng thiết kế logic số và quản lý trạng thái (FSM) trong thực tế.

2. Hệ thống hỗ trợ truyền thông song phương ổn định

- Đề tài đảm bảo **khả năng truyền và nhận dữ liệu hai chiều giữa FPGA và máy tính** một cách đáng tin cậy.
- Tốc độ truyền có thể thay đổi linh hoạt, nhưng hệ thống vẫn hoạt động ổn định với **tỷ lệ lỗi truyền rất thấp**, nhờ vào thiết kế chính xác bộ tạo baud và đồng bộ dữ liệu.

3. Hỗ trợ cấu hình động tốc độ baud

- Thay vì cố định một tốc độ truyền, hệ thống cho phép người dùng **tự thay đổi tốc độ baud theo thời gian thực** bằng các công tắc SW và nút nhấn KEY trên kit DE2.
- Đây là điểm mới và thực tiễn, bởi tốc độ truyền cần linh hoạt tùy vào ứng dụng cụ thể hoặc môi trường giao tiếp.

4. Tích hợp giao diện hiển thị trực quan trên kit FPGA

- Kết quả truyền nhận được thể hiện **trực tiếp qua LED đơn và LED 7 đoạn** giúp việc kiểm tra và theo dõi dữ liệu dễ dàng, không cần thiết bị đo chuyên dụng.
- Thiết kế này **phù hợp với bối cảnh giáo dục**, giúp sinh viên quan sát dữ liệu một cách trực quan và kiểm tra lỗi tức thời.

5. Hướng đến tính ứng dụng thực tế

- Đề tài không chỉ phục vụ mục tiêu học tập mà còn hướng đến các ứng dụng thực tiễn như:
    - Giao tiếp giữa vi điều khiển và máy tính.
    - Truyền lệnh điều khiển thiết bị ngoại vi từ máy tính đến hệ nhúng.
    - Ứng dụng trong các hệ thống giám sát, thu thập dữ liệu, hoặc giao tiếp IoT.
- Kiến trúc hệ thống có thể **mở rộng** cho các dự án sau này như: điều khiển thiết bị qua giao tiếp Bluetooth (UART-based), hoặc truyền dữ liệu giữa các hệ thống nhúng khác nhau.

1.5. Kết quả mong đợi

Khi hoàn thành đề tài **"Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính – Thiết kế hướng ứng dụng thực tế trên kit DE2"**, nhóm nghiên cứu kỳ vọng đạt được các kết quả sau:

1. Hiện thực thành công hệ thống UART truyền nhận dữ liệu hai chiều

- Thiết kế và triển khai hoàn chỉnh các module UART Transmitter và Receiver bằng Verilog, đảm bảo khả năng truyền và nhận dữ liệu liên tục, ổn định giữa FPGA và máy tính.
- Hệ thống vận hành chính xác theo chuẩn UART phổ biến với các tham số mặc định như tốc độ baud, số bit dữ liệu, bit dừng, không lỗi truyền nhận.

2. Hỗ trợ điều chỉnh tốc độ truyền động (baud rate) linh hoạt

- Người dùng có thể thay đổi tốc độ truyền thông qua các công tắc và nút nhấn trên kit DE2 mà không cần thay đổi mã nguồn hoặc lập trình lại FPGA.
- Hệ thống vẫn duy trì được sự ổn định và độ chính xác trong việc truyền nhận ở các tốc độ baud khác nhau, từ mức thấp đến mức cao.

3. Giao diện hiển thị trực quan và tiện lợi

- Dữ liệu nhận được từ máy tính được thể hiện rõ ràng trên các LED 7 đoạn và LED đơn, giúp người sử dụng dễ dàng quan sát trạng thái và nội dung truyền nhận.
- Các tín hiệu trạng thái như tín hiệu bắt đầu truyền, tín hiệu lỗi (nếu có) cũng được hiển thị rõ ràng, hỗ trợ quá trình kiểm tra và bảo trì hệ thống.

4. Đánh giá hiệu năng và tính ổn định của hệ thống

- Qua quá trình thử nghiệm sử dụng phần mềm mô phỏng giao tiếp Serial như PuTTY hoặc Tera Term, hệ thống chứng minh được khả năng truyền nhận dữ liệu ổn định trong điều kiện thực tế.
- Tỷ lệ lỗi truyền nhận thấp, hệ thống có khả năng phục hồi nhanh khi gặp lỗi truyền dẫn.

5. Cơ sở để phát triển các ứng dụng thực tế

- Hệ thống giao tiếp UART được xây dựng có thể làm nền tảng cho các dự án mở rộng trong tương lai như: điều khiển thiết bị ngoại vi, truyền dữ liệu cảm biến, hệ thống giám sát từ xa.
- Thiết kế mô-đun giúp dễ dàng tích hợp vào các hệ thống nhúng phức tạp hơn, tăng tính ứng dụng thực tế của đề tài.

6. Nâng cao kiến thức và kỹ năng thực tế cho người thực hiện

- Người thực hiện nắm vững nguyên lý hoạt động chuẩn UART, kỹ thuật thiết kế logic số và kỹ năng lập trình Verilog.
- Rèn luyện khả năng phân tích, thiết kế và kiểm thử hệ thống truyền thông trong môi trường FPGA thực tế.

II. Cơ sở lý thuyết

2.1. Tổng quan về giao tiếp nối tiếp

1. Khái niệm giao tiếp nối tiếp

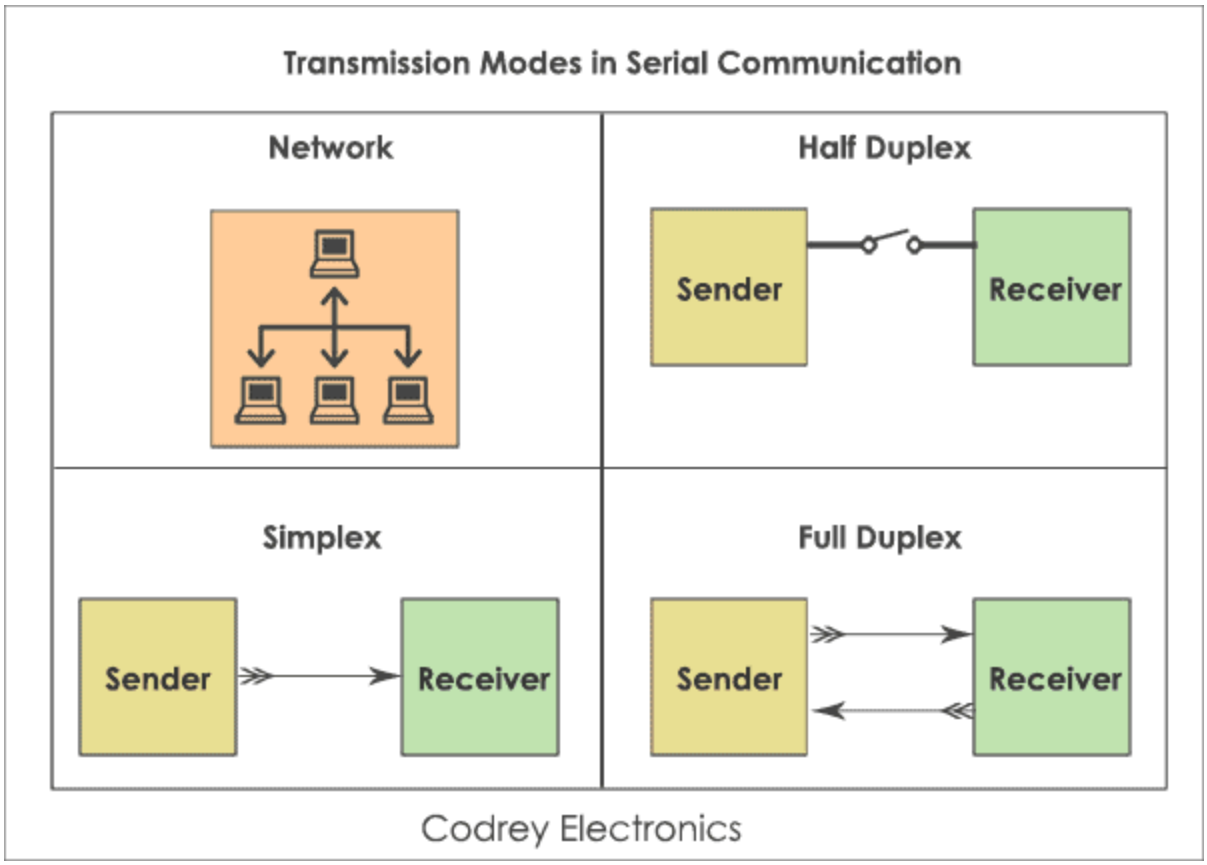
Giao tiếp nối tiếp (Serial Communication) là phương pháp truyền dữ liệu theo từng bit một, tuần tự qua một kênh truyền duy nhất hoặc một cặp dây truyền. Thay vì truyền đồng thời nhiều bit như giao tiếp song song, giao tiếp nối tiếp truyền từng bit một theo trình tự, từ bit đầu tiên đến bit cuối cùng, giúp giảm số lượng dây dẫn và đơn giản hóa kết nối vật lý.

Ở mức cơ bản, dữ liệu số được mã hóa thành các tín hiệu điện, trong đó:

- Mức điện cao (logic 1) và mức điện thấp (logic 0) đại diện cho các bit dữ liệu.
- Dữ liệu được đóng gói thành từng khung truyền (frame), mỗi khung bao gồm: bit bắt đầu (start bit), các bit dữ liệu (data bits), bit chẵn lẻ (parity bit, tùy chọn), và bit dừng (stop bit).

Việc truyền nối tiếp giúp giảm thiểu nhiễu và suy hao tín hiệu khi truyền xa, rất phù hợp cho các hệ thống nhúng, vi điều khiển, FPGA hoặc giao tiếp máy tính với thiết bị ngoại vi.

2. Các chế độ truyền dữ liệu trong giao tiếp nối tiếp

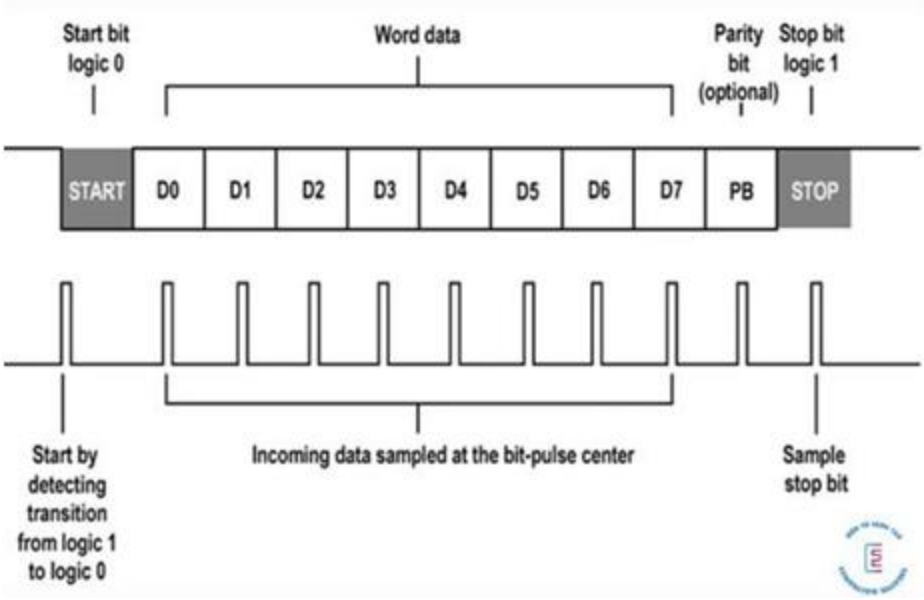


Hình 2.1. Transmission Modes – Serial Communication

Giao tiếp nối tiếp có ba chế độ cơ bản:

- **Simplex:** Truyền dữ liệu một chiều duy nhất từ thiết bị gửi đến thiết bị nhận. Ví dụ như truyền hình, radio.
  Half Duplex: Hai thiết bị có thể gửi và nhận dữ liệu nhưng không đồng thời, phải chờ nhau để gửi. Ví dụ như bộ đàm.
- **Full Duplex:** Hai thiết bị có thể truyền và nhận dữ liệu đồng thời trên hai kênh riêng biệt, nâng cao hiệu quả truyền thông. Đây là chế độ thường dùng trong các giao tiếp UART giữa FPGA và máy tính.

3. Các thành phần và cấu trúc khung dữ liệu

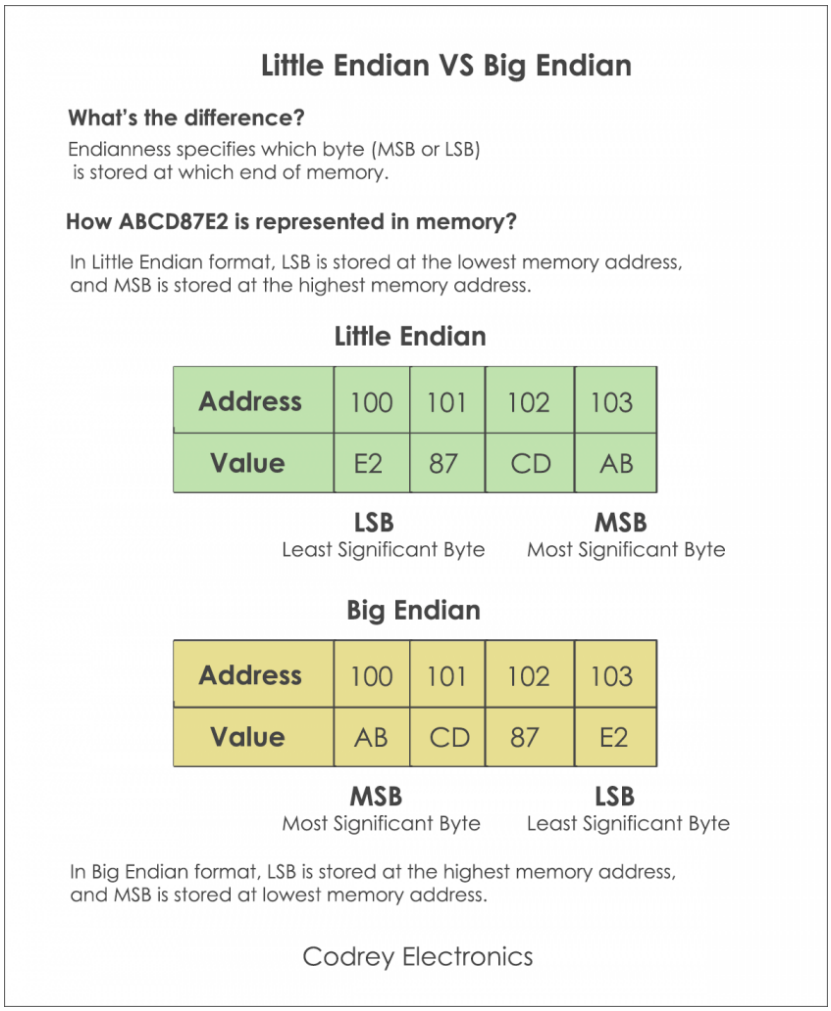


Hình 2.2 Cấu trúc khung truyền

Khung dữ liệu trong giao tiếp nối tiếp thường bao gồm:

- **Start bit (bit bắt đầu):** Tín hiệu báo hiệu bắt đầu truyền một khung dữ liệu, thường là mức logic 0 kéo dài 1 bit thời gian.
- **Data bits (bit dữ liệu):** Chuỗi các bit thể hiện thông tin truyền, thường từ 5 đến 9 bit tùy chuẩn giao tiếp.
- **Parity bit (bit chẵn lẻ):** Một bit kiểm tra lỗi tùy chọn, giúp phát hiện lỗi bit lẻ trong khung dữ liệu (có thể là parity chẵn hoặc parity lẻ).
  Stop bit (bit dừng): Tín hiệu báo hiệu kết thúc khung dữ liệu, thường là mức logic 1 kéo dài 1 hoặc 2 bit thời gian, giúp thiết bị nhận biết kết thúc khung.

4. Endianness – Quy ước sắp xếp byte trong truyền dữ liệu



Hình 2.3 Little Endian vs Big Endian

Khi giao tiếp dữ liệu, nhất là khi truyền dữ liệu đa byte (ví dụ: số nguyên 16-bit, 32-bit, hoặc các kiểu dữ liệu phức tạp), cách sắp xếp thứ tự byte trong dữ liệu gọi là **Endianness** có vai trò quan trọng, đảm bảo thiết bị gửi và thiết bị nhận hiểu đúng dữ liệu.

- **Big Endian**:
  Byte quan trọng nhất (Most Significant Byte – MSB) được truyền và lưu trữ trước (ở địa chỉ thấp hơn). Nói cách khác, thứ tự truyền byte là từ phần dữ liệu cao nhất xuống thấp nhất.
  Ví dụ, số 0x12345678 sẽ được truyền theo thứ tự: 0x12 → 0x34 → 0x56 → 0x78.
- **Little Endian**:
  Byte ít quan trọng nhất (Least Significant Byte – LSB) được truyền và lưu trữ trước.

Thứ tự truyền byte ngược lại so với Big Endian.
Ví dụ, số 0x12345678 sẽ được truyền theo thứ tự: 0x78 → 0x56 → 0x34 → 0x12.

Tại sao Endianness quan trọng trong giao tiếp nối tiếp?

- Giao tiếp nối tiếp thường truyền từng byte một, nếu thiết bị gửi và nhận không đồng bộ về thứ tự byte, dữ liệu đa byte có thể bị hiểu sai hoặc lỗi.
- Các thiết bị FPGA, vi điều khiển hoặc máy tính có thể dùng các chuẩn endianness khác nhau, do đó cần thống nhất trước khi truyền.
- Trong giao tiếp UART, thông thường dữ liệu được truyền byte theo thứ tự, nên ứng dụng cần quy ước endianness cụ thể để giải mã đúng.

Ví dụ minh họa:

Giả sử truyền số 16-bit có giá trị thập phân 4660 (0x1234 trong Hex):

Endianness	Thứ tự truyền byte	Giải thích
Big Endian	0x12 (MSB) → 0x34 (LSB)	Truyền byte quan trọng trước
Little Endian	0x34 (LSB) → 0x12 (MSB)	Truyền byte ít quan trọng trước

Bảng 2.1 Ví dụ về Big Endian và Little Endian

Nếu thiết bị nhận hiểu sai thứ tự này sẽ dẫn đến dữ liệu sai lệch, ví dụ nhận 0x3412 thay vì 0x1234.

4. Ưu điểm và nhược điểm của giao tiếp nối tiếp

Ưu điểm:

- **Giảm số lượng dây dẫn:** Chỉ cần 1 hoặc 2 dây truyền dữ liệu (TX và RX), tiết kiệm không gian, chi phí vật liệu và độ phức tạp.
- **Truyền được khoảng cách xa hơn:** Ít dây dẫn giúp giảm nhiễu và suy hao tín hiệu, thuận lợi cho các ứng dụng công nghiệp, truyền thông.
- **Dễ dàng triển khai trên FPGA, vi điều khiển:** Chuẩn giao tiếp phổ biến, có nhiều module, IP core hỗ trợ.

Nhược điểm:

- **Tốc độ truyền thấp hơn giao tiếp song song:** Vì truyền tuần tự từng bit một nên băng thông bị giới hạn.
- **Độ trễ cao khi truyền khối dữ liệu lớn:** Dữ liệu lớn phải chờ truyền lần lượt từng bit.
- **Yêu cầu đồng bộ chính xác:** Thiết bị gửi và nhận phải đồng bộ tốc độ baud, chuẩn hóa khung dữ liệu để tránh lỗi truyền.

5. Các chuẩn giao tiếp nối tiếp phổ biến

- **UART (Universal Asynchronous Receiver/Transmitter):**
  Chuẩn giao tiếp nối tiếp không đồng bộ phổ biến, không sử dụng tín hiệu clock ngoài mà dựa vào bit bắt đầu và dừng để đồng bộ dữ liệu. Tốc độ baud có thể điều chỉnh tùy ý (ví dụ 9600, 115200 bps). Được sử dụng rộng rãi trong giao tiếp máy tính – vi điều khiển – FPGA.
- **RS-232:**
  Chuẩn truyền nối tiếp không đồng bộ, sử dụng mức điện áp ±12V để tăng khả năng chống nhiễu. Được dùng nhiều trong các thiết bị máy tính cũ và các thiết bị công nghiệp.
- **RS-485:**
  Chuẩn truyền nối tiếp đa điểm, hỗ trợ truyền trên khoảng cách xa và chống nhiễu tốt nhờ phương pháp truyền khác biệt (differential signaling).
- **SPI (Serial Peripheral Interface):**
  Giao tiếp đồng bộ tốc độ cao sử dụng tín hiệu đồng hồ, cho phép truyền đồng thời hai chiều (full duplex). Thường dùng trong giao tiếp giữa vi điều khiển và các thiết bị ngoại vi như bộ nhớ flash, cảm biến.
- **I2C (Inter-Integrated Circuit):**
  Giao tiếp hai dây đồng bộ, hỗ trợ nhiều thiết bị trên cùng một bus với địa chỉ riêng. Tốc độ thấp hơn SPI nhưng đơn giản hơn và phổ biến trong các module cảm biến.

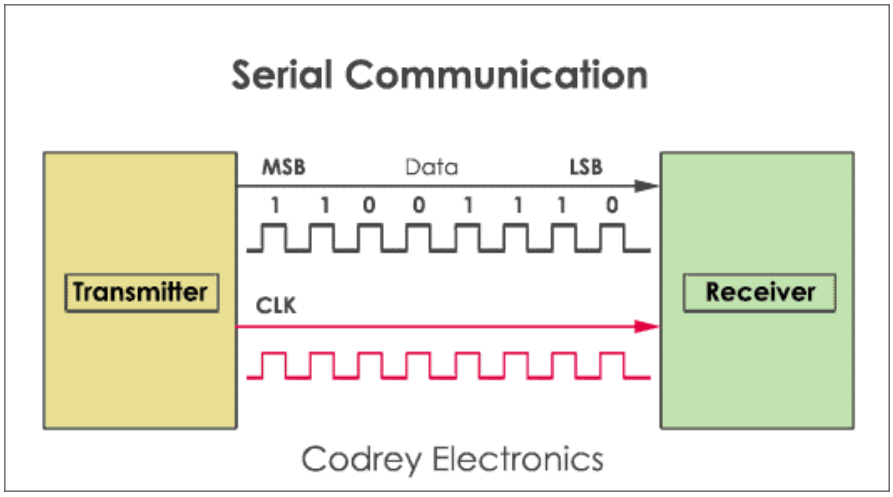
6. So sánh giao tiếp nối tiếp và song song

1. **Ví dụ về truyền dữ liệu nối tiếp**

Ví dụ: Giả sử ta muốn truyền một dữ liệu nhị phân 8 bit là 11001110 từ bộ phát (transmitter) đến bộ thu (receiver). Vậy bit nào sẽ được truyền đi trước – **bit quan trọng nhất** (MSB – bit thứ 7) hay **bit ít quan trọng nhất** (LSB – bit thứ 0)?

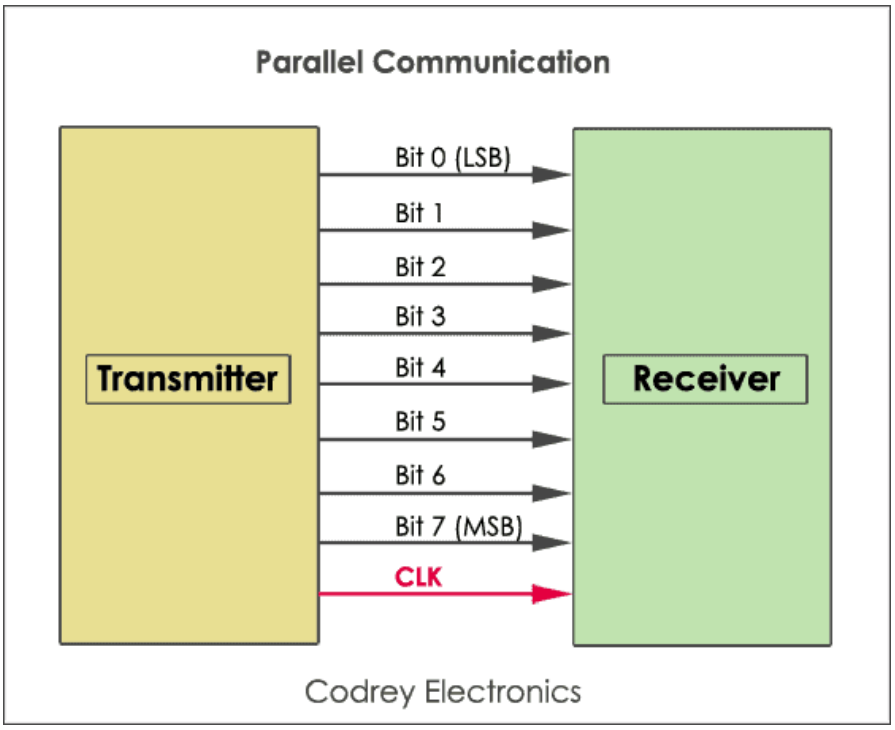
Câu trả lời là: **không thể khẳng định một cách tuyệt đối nếu không biết rõ chuẩn giao tiếp đang dùng**, vì điều này phụ thuộc vào thiết kế hệ thống truyền nhận. Tuy nhiên, **trong chuẩn UART, theo quy ước phổ biến, bit LSB thường được truyền trước**. Điều này tương ứng với cách hoạt động của kiểu **Little Endian** tại cấp độ bit.

trong giao tiếp nối tiếp, **dữ liệu được truyền từng bit một**, tức là mỗi chu kỳ xung clock sẽ truyền đi một bit dữ liệu từ bộ phát đến bộ thu. Giao tiếp nối tiếp có ưu điểm lớn về mặt tiết kiệm dây truyền, giảm chi phí và đơn giản hóa thiết kế phần cứng.



Hình 2.4 Giao tiếp nối tiếp

Ngược lại, **giao tiếp song song** (Parallel Communication) truyền cùng lúc nhiều bit (thường là 8, 16 hoặc 32 bit) thông qua nhiều đường truyền riêng biệt. Vì thế, nó có thể nhanh hơn trong các ứng dụng cần tốc độ cao, chẳng hạn như trong **máy in, máy photocopy**, v.v.



Hình 2.5 Giao tiếp song song

2. So sánh giữa truyền nối tiếp và truyền song song

Tiêu chí	Giao tiếp nối tiếp	Giao tiếp song song
Số dây tín hiệu	1 hoặc 2 dây	Nhiều dây (8, 16, 32…)
Tốc độ truyền	Thấp hơn (tuần tự từng bit)	Cao hơn (truyền đồng thời nhiều bit)
Khoảng cách truyền	Xa hơn, ít bị nhiễu	Ngắn hơn, dễ nhiễu
Độ phức tạp phần cứng	Thấp, đơn giản	Cao, đòi hỏi nhiều dây và mạch phức tạp
Chi phí	Thấp	Cao

Bảng 2.2 So sánh Giao tiếp nối tiếp và song song

2.2. Giao tiếp UART là gì?

Khái niệm tổng quan

UART (Universal Asynchronous Receiver/Transmitter) là một chuẩn giao tiếp nối tiếp không đồng bộ, đóng vai trò trung gian trong việc **truyền và nhận dữ liệu nhị phân** giữa hai thiết bị kỹ thuật số. UART **không yêu cầu tín hiệu xung clock đồng bộ** giữa bên phát và bên nhận, điều này làm cho nó đơn giản và phổ biến trong các ứng dụng giao tiếp tầm gần, đặc biệt là giữa vi điều khiển, FPGA và máy tính cá nhân.

Nguyên lý hoạt động

Khi một thiết bị muốn truyền dữ liệu thông qua UART, dữ liệu sẽ được **đóng gói thành một khung truyền (data frame)**, bao gồm các thành phần sau:

Thành phần	Mô tả
Start bit	Là bit đầu tiên trong khung, có giá trị logic **0**. Nó báo hiệu cho bộ thu rằng một khung dữ liệu sắp được gửi đến.
Data bits	Dữ liệu nhị phân thực tế, thường là **7 hoặc 8 bit**, truyền theo thứ tự **LSB trước – Little Endian** (tùy vào cấu hình).
Parity bit	Tùy chọn, dùng để **phát hiện lỗi truyền**. Có thể là chẵn (even), lẻ (odd) hoặc không có (none).
Stop bit(s)	Một hoặc hai bit có giá trị logic **1**, để xác định kết thúc của một khung dữ liệu. Nó giúp bộ thu phân biệt giữa các khung kế tiếp.

Bảng 2.3 Khung truyền (data frame)

Ví dụ với cấu hình phổ biến **"8-N-1"**, một khung truyền sẽ gồm:

1 bit start + 8 bit data + 0 bit parity + 1 bit stop = 10 bit tổng cộng.

Cách truyền dữ liệu – Thứ tự bit (Endian)

Trong UART, dữ liệu nhị phân được gửi theo từng bit, **bắt đầu từ bit ít quan trọng nhất (LSB – bit 0)** đến bit quan trọng nhất (MSB – bit 7). Đây là phương pháp truyền **Little Endian ở cấp độ bit**.

Ví dụ: Truyền chuỗi dữ liệu 11001110 (dạng 8-bit):

- Bit được gửi đi theo thứ tự: **0 → 1 → 1 → 1 → 0 → 0 → 1 → 1**
- Tức là 01110011 theo chiều thời gian (LSB đi trước).

Đặc điểm nổi bật của UART

- **Không cần đồng bộ clock:** UART hoạt động **hoàn toàn dựa trên thời gian (baud rate)**, không yêu cầu tín hiệu xung đồng hồ chung giữa hai thiết bị.
- **Cấu hình đơn giản:** Có thể điều chỉnh được các tham số như **baud rate, số bit dữ liệu, parity và stop bit**, phù hợp với nhiều ứng dụng thực tế.
- **Tích hợp dễ dàng:** Hầu hết các vi điều khiển và FPGA đều có module UART tích hợp sẵn, hoặc có thể thiết kế module truyền nhận đơn giản trong HDL (Verilog/VHDL).
- **Ứng dụng rộng rãi:** Dùng trong giao tiếp **vi điều khiển – máy tính**, kết nối với module **Bluetooth HC-05**, giao tiếp qua **cổng RS232**, hoặc qua **USB-UART** với các phần mềm như **PuTTY**, **Tera Term**, v.v.

Hạn chế của UART

- **Chỉ hỗ trợ truyền 2 thiết bị** (point-to-point).
- **Không đồng bộ**, nên dễ gây lỗi nếu baud rate không khớp.
- **Tốc độ không cao bằng SPI/I2C** trong các hệ thống thời gian thực**.**

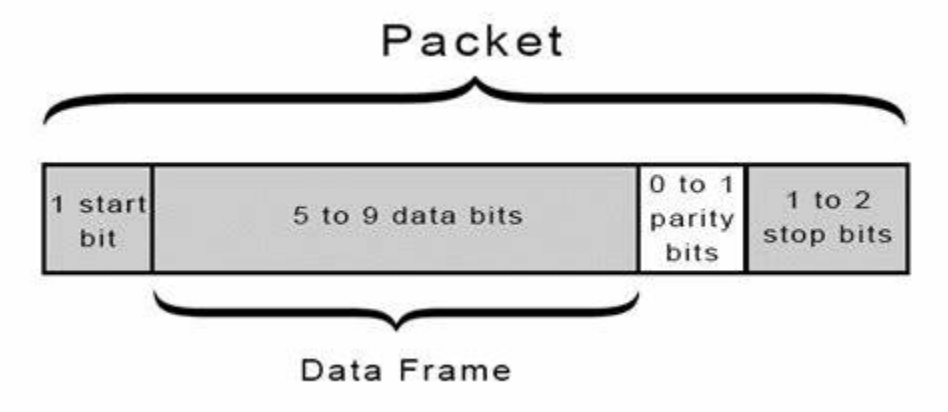
2.3. Cấu trúc khung truyền UART

Giới thiệu chung

Trong giao tiếp nối tiếp UART (Universal Asynchronous Receiver and Transmitter), dữ liệu được truyền từ thiết bị gửi đến thiết bị nhận theo từng khung (frame) nối tiếp. Không giống như các giao thức đồng bộ (synchronous), UART không dùng xung clock chung giữa hai thiết bị, mà thay vào đó đồng bộ dựa trên **bit khởi đầu (start bit)** và tốc độ baud được cấu hình sẵn.

Mỗi **khung truyền UART** được xây dựng với cấu trúc cụ thể bao gồm: bit bắt đầu (start bit), chuỗi bit dữ liệu (data bits), bit kiểm tra chẵn lẻ (parity bit – tùy chọn), và bit kết thúc (stop bit). Các thành phần này kết hợp lại thành một gói tin giúp bộ nhận phân tích đúng chuỗi dữ liệu được gửi đi.

Cấu trúc tổng quát của khung truyền UART



Hình 2.6 Cấu trúc truyền dữ liệu Uart

Một khung truyền UART bao gồm các thành phần theo thứ tự sau:

1. **Start bit (1 bit)**
    - Mức logic: LOW (0)
    - Dùng để **đồng bộ hóa** khởi đầu truyền.
    - Đánh dấu sự bắt đầu của một khung truyền.
    - Thiết bị nhận nhận diện một tín hiệu giảm từ mức cao (idle - 1) xuống thấp (0) như là một **trigger** để bắt đầu tiếp nhận dữ liệu.
2. **Data bits (5 đến 9 bits)**
    - Chứa dữ liệu chính cần truyền (thường là 7 hoặc 8 bit).
    - Thứ tự truyền: thường theo **Little Endian** (LSB đi trước), nhưng có thể cấu hình theo yêu cầu.
    - Bit dữ liệu có thể đại diện cho mã ASCII, giá trị số, ký tự hoặc lệnh điều khiển.
3. **Parity bit (0 hoặc 1 bit – tùy chọn)**
    - Dùng để **kiểm tra lỗi đơn giản** trong quá trình truyền.
    - Có ba chế độ:

        Even Parity: Tổng số bit 1 (bao gồm cả parity) là **chẵn**.

        Odd Parity: Tổng số bit 1 là **lẻ**.

        No Parity: Không dùng kiểm tra parity.

4. **Stop bit(s) (1 hoặc 2 bit)**

- Mức logic: HIGH (1)
- Đánh dấu **kết thúc** khung dữ liệu.
- Giúp thiết bị nhận có thời gian xử lý dữ liệu trước khi nhận khung kế tiếp.

Ví dụ minh họa cụ thể

Giả sử muốn truyền một byte dữ liệu 11001110 (hex: 0xCE) với cấu hình UART phổ biến **8-N-1** (8 data bits, No parity, 1 stop bit):

- **Start bit**: 0
- **Data bits** (LSB → MSB): 0 1 1 1 0 0 1 1
- **Stop bit**: 1

➞ **Khung truyền sẽ là**: 0 0 1 1 1 0 0 1 1 1 (tổng cộng 10 bit)

Lưu ý:
Trong trường hợp truyền dữ liệu dạng số nguyên nhiều byte (như int, float), thứ tự byte giữa thiết bị gửi và nhận cần được đồng bộ theo chuẩn **Little Endian** (LSB trước) hoặc **Big Endian** (MSB trước).
UART thường chỉ truyền từng byte nên thứ tự giữa các byte phải được phần mềm xử lý đúng.

2.4. Baudrate và cách tính

1. Khái niệm Baudrate

Baudrate, hay còn gọi là **tốc độ baud**, là đại lượng biểu thị **số tín hiệu thay đổi trên đường truyền trong một giây**. Trong giao tiếp UART, mỗi tín hiệu thay đổi thường tương ứng với một bit dữ liệu, do đó:

$$\text{Baudrate} \approx \text{Bits per second (bps)}$$

Tốc độ này quy định **tốc độ truyền thông giữa hai thiết bị UART**. Cả bộ phát (transmitter) và bộ thu (receiver) cần được cấu hình cùng một baudrate để đảm bảo dữ liệu được truyền và nhận chính xác, tránh sai lệch bit hoặc lỗi khung (frame error).

2. Cách truyền dữ liệu theo baudrate

UART truyền dữ liệu tuần tự từng bit theo một khung truyền. **Mỗi bit được truyền trong một khoảng thời gian nhất định**, gọi là bit time. Bit time được tính theo công thức:

$$\text{Bit time} = \frac{1}{\text{Baudrate}} \text{ (giây/bit)}$$

Hình 2.7 Công thức tính Bit time

Ví dụ: Với baudrate 9600 bps, mỗi bit mất:

$$\frac{1}{9600} = 104.167\,\mu s$$

Nghĩa là, cứ sau 104.167 micro giây, bộ phát sẽ gửi ra một bit mới, và bộ thu cũng phải đọc đúng theo tốc độ này.

3. Tính toán Baudrate trên FPGA

Khác với vi điều khiển đã có UART tích hợp sẵn, FPGA yêu cầu người dùng **thiết kế thủ công mạch tạo xung baud** (baudrate generator) từ xung clock hệ thống (ví dụ 50 MHz trên kit DE2).

Công thức tính hệ số chia:

$$\text{Divisor} = \frac{F_{\text{clock}}}{\text{Baudrate} \times \text{Oversampling}}$$

Hình 2.8 Công thức hệ số chia

- **Fclock:** Tần số clock hệ thống (ví dụ: 50 MHz)
- **Oversampling**: số lần lấy mẫu mỗi bit (thường là 1 hoặc 16 để tăng độ chính xác)
- **Baudrate**: tốc độ truyền mong muốn

Nếu **oversampling = 1**:

$$\text{Divisor} = \frac{50,000,000}{9600} \approx 5208.33$$

⇒ Bộ chia phải tạo xung đều đặn mỗi **5208 chu kỳ clock**, để sinh ra xung baudrate 9600 Hz.

4. Ví dụ bảng chia baudrate (với Clock = 50 MHz, oversampling = 1)

Baudrate (bps)	Divisor cần chia	Sai số thực tế
2400	20833	~0.0%
4800	10417	~0.0%
9600	5208	~0.0%
19200	2604	~0.0%
38400	1302	~0.0%
57600	868	~0.0%
115200	434	~0.0%

Bảng 2.4 Ví dụ bảng chia BaudRate

Trong thực tế, FPGA thường chia clock với sai số nhỏ chấp nhận được (<3%). Nếu sai số quá lớn, cần cân nhắc tăng oversampling hoặc dùng thuật toán điều chỉnh tốc độ chính xác hơn.

5. Vai trò của Oversampling

Một kỹ thuật phổ biến trong UART là **oversampling**, tức là lấy mẫu nhiều lần trên mỗi bit (thường là 16 lần) để xác định chính xác biên dữ liệu (ví dụ như trung điểm của bit). Khi dùng oversampling:

$$\text{Divisor} = \frac{F_{\text{clock}}}{\text{Baudrate} \times 16}$$

Hình 2.9 Công thức chia OverSampling

⇒ Với 9600 bps và clock 50 MHz:

$$\text{Divisor} = \frac{50,000,000}{9600 \times 16} = 325.52 \approx 326$$

=> Cần mạch chia clock chính xác để tạo xung tick mỗi 326 chu kỳ clock.

6. Cấu hình baudrate linh hoạt

Trên FPGA, việc cấu hình baudrate có thể được điều khiển linh hoạt qua **công tắc SW** hoặc **nút nhấn KEY**, giúp thay đổi tốc độ truyền động mà không cần nạp lại chương trình.

Ví dụ Verilog:

```
case(SW[1:0])
  2'b00: divisor = 5208;  // 9600 bps
  2'b01: divisor = 2604;  // 19200 bps
  2'b10: divisor = 1302;  // 38400 bps
  2'b11: divisor = 868;   // 57600 bps
endcase
```

2.5. Hoạt động của UART Transmitter

UART Transmitter là bộ phận chịu trách nhiệm **gửi dữ liệu dạng số (bit)** từ thiết bị phát (ở đây là FPGA) đến thiết bị nhận (máy tính) theo dạng **giao tiếp nối tiếp (serial)**. Nói cách khác, nó biến đổi dữ liệu song song (ví dụ 8 bit) thành chuỗi các bit được gửi ra từng bit một, tuần tự theo thời gian.

1. Dữ liệu được gửi như thế nào?

Giả sử bạn có một byte dữ liệu 8 bit cần gửi, ví dụ: 11001110. Bộ phát UART sẽ thực hiện các bước:

- **Bước 1: Thêm Start bit**
  Trước khi gửi dữ liệu, bộ phát sẽ gửi một bit bắt đầu (start bit), có giá trị luôn bằng 0. Start bit giúp thiết bị nhận biết rằng một chuỗi dữ liệu mới sắp bắt đầu.

- **Bước 2: Gửi từng bit dữ liệu**
  Các bit dữ liệu sẽ được gửi ra lần lượt từng bit một. Thông thường, truyền từ **bit thấp nhất (LSB) trước**, rồi đến bit cao hơn. Với ví dụ 11001110, bit LSB là 0, bit MSB là 1. Vì vậy chuỗi bit gửi sẽ là: 0 1 1 1 0 0 1 1.
- **Bước 3: (Tùy chọn) Gửi bit kiểm tra parity**
  Parity bit là một bit dùng để kiểm tra lỗi khi truyền dữ liệu, có thể là chẵn hoặc lẻ, tùy thiết lập. Nếu không dùng thì bước này sẽ bị bỏ qua.
- **Bước 4: Gửi Stop bit(s)**
  Sau khi dữ liệu được gửi xong, bộ phát gửi 1 hoặc 2 bit dừng (stop bit), có giá trị luôn là 1. Stop bit giúp thiết bị nhận biết dữ liệu đã kết thúc.

2. Thời gian gửi dữ liệu

Mỗi bit được gửi đi trong một khoảng thời gian xác định gọi là **chu kỳ baud** (baud rate). Ví dụ, nếu tốc độ baud là 9600 bps, thì mỗi bit được gửi trong khoảng 104,17 micro giây.

Toàn bộ một khung dữ liệu gồm start bit, 8 bit dữ liệu, parity bit (nếu có) và stop bit(s) sẽ được gửi liên tục, bit này nối tiếp bit kia.

3. Quy trình hoạt động của UART Transmitter

- **Chờ tín hiệu yêu cầu truyền dữ liệu từ mạch điều khiển**.
- **Khi nhận được dữ liệu**, transmitter sẽ chuẩn bị khung dữ liệu theo cấu trúc đã nói.
- **Gửi start bit** đầu tiên.
- **Gửi từng bit dữ liệu từ LSB đến MSB**.
- **Gửi bit parity nếu được kích hoạt**.
- **Gửi stop bit(s)**.
- **Quay lại trạng thái chờ**, sẵn sàng nhận dữ liệu tiếp theo.

4. Ví dụ minh họa

Giả sử dữ liệu cần gửi là 11001110 (byte 0xCE). Với cấu hình:

- 1 start bit (0)
- 8 data bits (gửi LSB trước)
- Không dùng parity

- 1 stop bit (1)

Thì trên đường truyền, bit sẽ được gửi theo thứ tự: 0 (start) - 0 - 1 - 1 - 1 - 0 - 0 - 1 - 1 (data bits) - 1 (stop)

Mỗi bit sẽ giữ nguyên mức điện áp tương ứng trong thời gian quy định bởi baudrate.

2.6. Hoạt động của UART Receiver

UART Receiver là thành phần chịu trách nhiệm **nhận dữ liệu nối tiếp** được truyền từ thiết bị phát thông qua đường truyền UART, sau đó **giải mã và tái tạo lại dữ liệu song song** ban đầu để xử lý hoặc hiển thị.

1. Nguyên lý hoạt động tổng quát

Bộ thu UART (Receiver) cần hoạt động đồng bộ với tốc độ truyền dữ liệu của bộ phát (Transmitter), vì vậy tốc độ baud của cả hai bên phải **được thiết lập giống nhau** để dữ liệu nhận đúng thời điểm và chính xác.

UART Receiver thường được thiết kế dựa trên **Finite State Machine (FSM)** gồm các trạng thái cơ bản như: Chờ (Idle), Bắt đầu (Start), Nhận dữ liệu (Data), Kiểm tra lỗi (Parity – nếu có), và Kết thúc (Stop).

2. Các bước hoạt động chi tiết

Bước 1: Chờ Start Bit (Idle → Start)

- Ở trạng thái ban đầu (Idle), đường truyền UART luôn ở mức logic cao (1).
- Khi xuất hiện một mức thấp (0) kéo dài đúng bằng 1 chu kỳ bit, bộ thu xác định đó là **start bit** → đây là tín hiệu báo hiệu dữ liệu sắp đến.
- Khi phát hiện start bit, Receiver bắt đầu lấy mẫu tín hiệu theo chuẩn 16x oversampling (hoặc 8x) để tăng độ chính xác.

Bước 2: Nhận Dữ Liệu (Data Bits)

- Sau start bit, UART Receiver lần lượt **lấy mẫu từng bit dữ liệu** ở thời điểm giữa mỗi bit (để tránh sai lệch biên).

- Dữ liệu được nhận **theo đúng thứ tự như bộ phát truyền**: thường là từ **LSB đến MSB** (tùy theo quy ước endianness).
- Bộ thu tiếp tục ghi nhận đủ số lượng bit (thường là 8 bit).

Bước 3: Kiểm tra Parity (nếu dùng)

- Nếu chế độ kiểm tra chẵn lẻ (Parity) được bật, Receiver sẽ nhận thêm một bit parity và **kiểm tra tính đúng đắn của dữ liệu** (số bit 1 là chẵn hoặc lẻ).
- Nếu xảy ra lỗi, một **cờ báo lỗi parity** sẽ được kích hoạt.

Bước 4: Nhận Stop Bit

- Sau khi nhận xong dữ liệu (và parity nếu có), Receiver chờ nhận **1 hoặc 2 bit stop** có mức logic cao (1).
- Nếu bit stop không có giá trị đúng (tức là không ở mức 1), hệ thống sẽ kích hoạt **cờ lỗi framing** (Framing Error), báo hiệu việc truyền bị lỗi.

Bước 5: Hoàn tất và tái tạo dữ liệu

- Sau khi khung dữ liệu hoàn tất, các bit thu được sẽ được **chuyển đổi thành dữ liệu song song (8-bit)** và lưu vào thanh ghi hoặc buffer.
- Bộ thu trở lại trạng thái chờ để tiếp tục nhận dữ liệu kế tiếp.

3. Minh họa bằng sơ đồ khung nhận

Giả sử truyền dữ liệu 11001110 (0xCE), khung truyền gồm:

- **Start bit**: 0
- **Dữ liệu (LSB trước)**: 0 1 1 1 0 0 1 1
- **Stop bit**: 1

Trên đường truyền, UART Receiver sẽ đọc tuần tự: [0] - [0 1 1 1 0 0 1 1] - [1]

4. Các lỗi thường gặp trong quá trình nhận

Loại lỗi	Mô tả
Framing Error	Không nhận được stop bit hoặc bit stop bị nhiễu → sai định dạng khung
Parity Error	Số bit 1 trong dữ liệu không khớp với bit parity đã truyền
Overrun Error	Bộ thu chưa kịp xử lý dữ liệu trước đó, dữ liệu mới đến bị ghi đè

Bảng 2.5 Các lỗi thường gặp trong quá trình nhận

5. Vai trò trong hệ thống

- UART Receiver trên kit DE2 giúp hệ thống **nhận dữ liệu từ máy tính** để điều khiển hoạt động của FPGA hoặc để phục vụ ứng dụng cụ thể như giao tiếp, giám sát, cấu hình.
- Khi kết hợp với UART Transmitter, hệ thống có thể thực hiện **giao tiếp hai chiều** hiệu quả và tin cậy.

2.7. Ứng dụng thực tế của UART

Giao tiếp UART là một trong những phương thức truyền thông nối tiếp đơn giản và phổ biến nhất trong lĩnh vực điện tử và nhúng. Với thiết kế dễ triển khai, chi phí thấp và hiệu quả truyền thông ổn định, UART được ứng dụng rộng rãi trong nhiều hệ thống nhúng và thiết bị điện tử hiện đại.

1. Giao tiếp giữa vi điều khiển và máy tính

- Một trong những ứng dụng phổ biến nhất của UART là **truyền dữ liệu giữa vi điều khiển (MCU/FPGA) và máy tính cá nhân (PC)**.
  Giao tiếp này thường được thực hiện thông qua **cổng COM ảo qua USB**, với sự hỗ trợ của chip chuyển đổi như **FT232, CP2102 hoặc PL2303**.
- Ví dụ: truyền dữ liệu đo nhiệt độ, cấu hình thông số từ máy tính, hoặc điều khiển thiết bị từ giao diện phần mềm như PuTTY hoặc TeraTerm.

2. Giao tiếp giữa các thiết bị nhúng

- UART thường được sử dụng để kết nối các vi điều khiển với nhau hoặc với các ngoại vi như:
    - **GPS module**
    - **GSM/GPRS module**
    - **WiFi (ESP8266, ESP32)**
    - **Bluetooth (HC-05, HC-06)**
    - **LoRa modules**
- Ví dụ: một hệ thống định vị có thể sử dụng UART để truyền dữ liệu tọa độ GPS từ module về bộ vi điều khiển để xử lý.

3. Cập nhật firmware và lập trình thiết bị

- Nhiều thiết bị nhúng hỗ trợ **nạp chương trình (firmware) thông qua UART**, đặc biệt là trong các hệ thống không có khả năng giao tiếp qua JTAG hoặc USB.
- Ví dụ: một số dòng vi điều khiển STM32 có **bootloader UART**, cho phép cập nhật chương trình thông qua giao tiếp nối tiếp.

4. Hệ thống giám sát và điều khiển công nghiệp

- Trong môi trường công nghiệp, UART được tích hợp vào hệ thống PLC, cảm biến, bộ điều khiển để:
    - Gửi dữ liệu trạng thái, cảnh báo
    - Cấu hình ngưỡng cảnh báo, chế độ hoạt động
- Ưu điểm là **kết nối đơn giản, tin cậy, dễ triển khai trên đường truyền dài với hỗ trợ của chuẩn RS232 hoặc RS485**.

5. Ứng dụng trong hệ thống nhúng học tập và nghiên cứu

- Trong các dự án sinh viên, đề tài học thuật, UART là **chuẩn truyền thông lý tưởng để giao tiếp giữa FPGA/vi điều khiển và PC**.
- Ví dụ trong đề tài này: thiết kế hệ thống UART truyền dữ liệu từ FPGA lên máy tính và hiển thị dữ liệu trên PuTTY, đồng thời phản hồi dữ liệu lại để điều khiển LED, bảy đoạn.

6. Thiết bị tiêu dùng thông minh (IoT)

- UART được tích hợp trong các sản phẩm IoT như:
    - Đồng hồ thông minh, vòng đeo tay sức khỏe
    - Hệ thống nhà thông minh (smart home)
    - Robot, xe tự hành nhỏ
- Những thiết bị này sử dụng UART để truyền dữ liệu về server hoặc thiết bị điều khiển trung tâm.

7. Giao tiếp với mạch mở rộng

- Khi thiết kế hệ thống cần mở rộng chức năng, UART thường được sử dụng để giao tiếp giữa **bo mạch chính và module mở rộng** như:
    - Bộ nhớ ngoài
    - Màn hình LCD thông minh
    - Module cảm biến chuyên dụng

2.8. So sánh UART với SPI, I2C

Trong lĩnh vực hệ thống nhúng, việc lựa chọn giao thức truyền thông phù hợp giữa các thiết bị là yếu tố then chốt ảnh hưởng đến hiệu quả và độ tin cậy của toàn bộ hệ thống. Ba chuẩn truyền thông nối tiếp phổ biến nhất hiện nay là **UART (Universal Asynchronous Receiver/Transmitter)**, **SPI (Serial Peripheral Interface)** và **I2C (Inter-Integrated Circuit)**. Mỗi chuẩn giao tiếp mang đặc điểm riêng, được thiết kế để phục vụ các mục tiêu khác nhau về tốc độ, chi phí phần cứng, số lượng thiết bị kết nối và độ phức tạp của giao thức.

1. Bảng so sánh tổng quát

Tiêu chí	UART	SPI	I2C
Loại truyền thông	Nối tiếp không đồng bộ	Nối tiếp đồng bộ	Nối tiếp đồng bộ
Dây kết nối cần thiết	2 dây: TX, RX	4 dây: MOSI, MISO, SCLK, SS	2 dây: SDA, SCL
Đồng bộ xung nhịp	Không – cần cấu hình baudrate	Có – dùng tín hiệu SCLK	Có – dùng tín hiệu SCL
Hình thức truyền dữ liệu	Full-duplex	Full-duplex	Half-duplex
Cấu trúc mạng thiết bị	1:1 (không hỗ trợ đa thiết bị trực tiếp)	1 Master – N Slave (mỗi slave cần chân SS)	1 Master – N Slave (sử dụng địa chỉ)
Tốc độ truyền	Trung bình (tối đa ~1 Mbps)	Cao (lên đến hàng chục Mbps)	Trung bình – Cao (tối đa 3.4 Mbps)
Chi phí phần cứng	Thấp – đơn giản	Cao hơn – cần nhiều chân điều khiển	Thấp – tiết kiệm dây kết nối
Độ phức tạp giao thức	Thấp – không có khung truyền phức tạp	Trung bình – yêu cầu xử lý clock và SS	Cao – có START, STOP, ACK, địa chỉ, v.v.
Địa chỉ hóa thiết bị	Không	Không	Có – địa chỉ 7 hoặc 10 bit
Khả năng mở rộng hệ thống	Kém – chỉ 1 thiết bị truyền/nhận	Tốt – mở rộng nhiều slave với chân SS riêng	Rất tốt – dễ dàng mở rộng qua địa chỉ hóa
Ứng dụng điển hình	Giao tiếp máy tính, module GPS, Bluetooth	Giao tiếp LCD, EEPROM tốc độ cao, cảm biến	Giao tiếp cảm biến, đồng hồ RTC, LCD nhỏ

Bảng 2.6 So sánh UART, SPI và I2C

2. Phân tích chi tiết

UART (Universal Asynchronous Receiver/Transmitter)

- **Đặc điểm nổi bật**: Là giao thức nối tiếp không đồng bộ, không cần tín hiệu xung clock dùng chung. Hai thiết bị truyền và nhận phải thống nhất baudrate từ trước để đảm bảo dữ liệu được giải mã đúng.
- **Ưu điểm**:
    - Cấu trúc đơn giản, chỉ cần 2 dây.
    - Rất phổ biến trong giao tiếp máy tính – vi điều khiển thông qua cổng RS232 hoặc USB-UART.

- **Hạn chế**:
    - Không hỗ trợ đa thiết bị.
    - Phụ thuộc vào độ chính xác baudrate, dễ gây lỗi nếu không đồng bộ.

SPI (Serial Peripheral Interface)

- **Đặc điểm nổi bật**: Giao tiếp đồng bộ tốc độ cao, sử dụng một thiết bị master để phát tín hiệu xung nhịp đến các slave. Mỗi slave cần một chân chọn riêng (SS).
- **Ưu điểm**:
    - Truyền dữ liệu tốc độ cao và song song hai chiều (full-duplex).
    - Giao tiếp đơn giản giữa các IC tốc độ cao.
- **Hạn chế**:
    - Tốn nhiều chân nếu giao tiếp với nhiều thiết bị.
    - Không có chuẩn chung cho cấu trúc khung truyền, phụ thuộc vào thiết kế.

I2C (Inter-Integrated Circuit)

- **Đặc điểm nổi bật**: Giao tiếp đồng bộ với chỉ 2 dây chia sẻ cho tất cả các thiết bị. Mỗi thiết bị được nhận diện qua địa chỉ riêng biệt (7 hoặc 10 bit).
- **Ưu điểm**:
    - Phù hợp khi cần giao tiếp với nhiều thiết bị trên một bus duy nhất.
    - Cấu trúc bus nhỏ gọn, tiết kiệm dây và chân I/O.
- **Hạn chế**:
    - Tốc độ thấp hơn SPI.
    - Cấu trúc giao tiếp phức tạp hơn UART và SPI (START, STOP, ACK/NACK...).

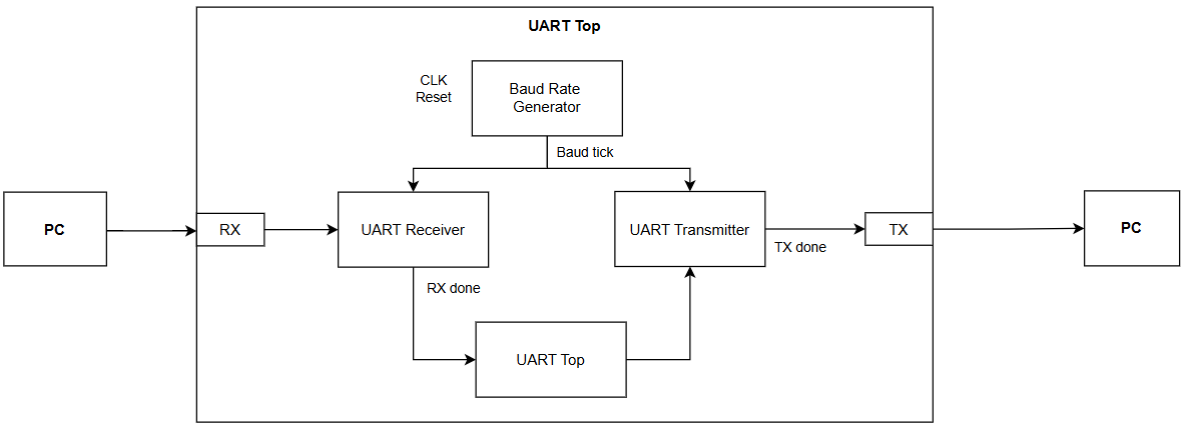
3. Khi nào chọn UART, SPI hay I2C?

Tình huống ứng dụng	Giao thức phù hợp
Truyền dữ liệu giữa máy tính và vi điều khiển/FPGA	UART
Truyền dữ liệu tốc độ cao giữa vi điều khiển và bộ nhớ	SPI
Giao tiếp với nhiều cảm biến hoặc thiết bị trên cùng bus	I2C

Bảng 2.7 Khi nào nên chọn UART, SPI hay I2C

III. Thiết kế hệ thống

3.1. Sơ đồ khối tổng thể của hệ thống



Hình 3.1 Sơ đồ khối tổng thể hệ thống

Giải thích luồng dữ liệu:

1. **PC → RX → uart_rx → uart_top**:
   - Dữ liệu nối tiếp từ PC được gửi qua cổng RX đến module uart_rx. Module uart_rx nhận dữ liệu, chuyển đổi từ định dạng nối tiếp thành dữ liệu song song (thường là 8 bit), và gửi đến uart_top cùng với tín hiệu rx_done để báo hiệu quá trình nhận hoàn tất. uart_top nhận dữ liệu này để xử lý hoặc chuyển tiếp.
2. **uart_top → uart_tx → TX → PC**:
   - Module uart_top nhận dữ liệu song song từ uart_rx hoặc từ hệ thống bên ngoài qua cổng data_in (ví dụ: từ CPU). Khi nhận tín hiệu tx_start, uart_top gửi dữ liệu đến uart_tx. Module uart_tx chuyển đổi dữ liệu song song thành dữ liệu nối tiếp, gửi qua cổng TX về PC, và báo hiệu hoàn tất bằng tx_done.
3. **baudrate_generator**:
   - Module baudrate_generator nhận tín hiệu xung nhịp hệ thống (clk) và tạo tín hiệu baud_tick với tần số phù hợp (thường là 16x baud rate, ví dụ: 9600 hoặc 115200 baud). Tín hiệu này được cung cấp cho cả uart_rx và uart_tx để đảm bảo truyền và nhận đồng bộ với tốc độ baud mong muốn.

Tín hiệu điều khiển và giao tiếp:

- Hệ thống sử dụng các tín hiệu chung như clk (xung nhịp), rst (reset) để điều khiển toàn bộ hệ thống.
- uart_top quản lý luồng dữ liệu giữa các module và giao tiếp với hệ thống bên ngoài qua cổng data_in (dữ liệu đầu vào) và data_out (dữ liệu đầu ra), đảm bảo gửi/nhận độc lập theo yêu cầu.

3.2. Mô tả chức năng từng module

3.2.1. Baudrate Generator

Chức năng:

Module BaudRate Generator chịu trách nhiệm tạo ra các tín hiệu xung đồng hồ (clock) có tần số phù hợp với các tốc độ baudrate chuẩn trong giao tiếp UART. Hai tín hiệu được sinh ra gồm:

- tx_clk: xung vuông có tần số đúng bằng baudrate, dùng làm clock truyền dữ liệu (transmit clock).
- rx_clk: xung vuông có tần số gấp 16 lần baudrate, dùng làm clock thu dữ liệu (receive clock) với kỹ thuật oversampling.

Thông số đầu vào:

- clk: tín hiệu clock hệ thống chính, tốc độ 50 MHz.
- reset_n: tín hiệu reset kích hoạt mức thấp (active-low reset), dùng để khởi động lại module.
- baud_select: tín hiệu 2-bit dùng để chọn tốc độ baudrate mong muốn, gồm 4 mức lựa chọn:
    - 00: 9600 bps
    - 01: 14400 bps
    - 10: 19200 bps
    - 11: 115200 bps

Thông số đầu ra:

- tx_clk: xung clock đồng bộ với tốc độ baudrate được chọn, dùng cho bộ truyền UART.
- rx_clk: xung clock có tần số gấp 16 lần baudrate, dùng cho bộ thu UART thực hiện oversampling nhằm tăng độ chính xác khi nhận dữ liệu.

Cách hoạt động:

- Module sử dụng các thanh ghi đếm counter_tx và counter_rx để tạo ra tín hiệu xung vuông dựa trên giá trị chia tần (divider) tương ứng với tốc độ baudrate được chọn.
- Giá trị divider_tx và divider_rx được thiết lập tương ứng với từng mức baudrate theo công thức tính toán dựa trên tần số clock hệ thống 50 MHz:

$$\text{divider\_tx} = \frac{50\,000\,000}{2 \times \text{baudrate}}, \quad \text{divider\_rx} = \frac{50\,000\,000}{2 \times \text{baudrate} \times 16}$$

Hình 3.2 Công thức chia TX

- Khi bộ đếm đạt giá trị divider, tín hiệu xung sẽ đảo trạng thái (toggle), tạo ra dạng sóng vuông với chu kỳ tương ứng.
- Tín hiệu tx_clk có tần số bằng baudrate, còn rx_clk có tần số gấp 16 lần, hỗ trợ kỹ thuật oversampling để tăng độ tin cậy thu nhận dữ liệu.

Ý nghĩa:

Module này là thành phần nền tảng quan trọng trong hệ thống UART, giúp đồng bộ hóa tốc độ truyền và nhận dữ liệu giữa FPGA và thiết bị bên ngoài (máy tính, cảm biến, module khác). Việc có rx_clk gấp 16 lần baudrate giúp bộ thu UART có thể lấy mẫu tín hiệu nhiều lần trong một bit dữ liệu, giúp phát hiện lỗi và cải thiện độ chính xác trong truyền nhận.

Giải thích chi tiết code:

- **Phần chọn giá trị chia tần (divider):**

```
// Chọn hệ số chia tương ứng
always @(*) begin
    case (baud_select)
        2'b00: begin // 9600
            divider_tx = 2604;   // Half-period: 50_000_000 / (2 * 9600)
            divider_rx = 163;    // Half-period: 50_000_000 / (2 * 9600 * 16)
        end
        2'b01: begin // 14400
            divider_tx = 1736;   // Half-period
            divider_rx = 109;
        end
        2'b10: begin // 19200
            divider_tx = 1302;
            divider_rx = 82;
        end
        2'b11: begin // 115200
            divider_tx = 217;
            divider_rx = 13;
        end
        default: begin
            divider_tx = 2604;
            divider_rx = 163;
        end
    endcase
end
```

Hình 3.3 Bộ chia tần số BaudRate

Phần này sử dụng case để chọn các giá trị divider_tx và divider_rx tương ứng với tốc độ baudrate được chọn qua baud_select. Các giá trị này được tính dựa trên tần số clock 50 MHz.

Tại mỗi mức baudrate, giá trị divider_tx và divider_rx được tính toán để chia tần số clock 50 MHz thành tần số xung tương ứng.

Công thức chung:

- divider_tx = 50MHz / (2 * baudrate) vì xung vuông có chu kỳ gồm 2 lần đếm (lên 1 lần và xuống 1 lần).
- divider_rx = 50MHz / (2 * baudrate * 16) vì oversampling bộ thu là 16 lần baudrate.

- **Phần tạo xung truyền (tx_clk):**

```
// Tạo tx_clk (dạng xung vuông)
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        counter_tx <= 0;
        tx_clk <= 0;
    end else begin
        if (counter_tx >= divider_tx) begin
            counter_tx <= 0;
            tx_clk <= ~tx_clk;
        end else begin
            counter_tx <= counter_tx + 1;
        end
    end
end
```

Hình 3.4 Bộ tạo xung truyền

Phần này tạo xung vuông đồng hồ truyền tx_clk bằng cách dùng bộ đếm counter_tx. Khi bộ đếm đạt giá trị divider_tx, tx_clk được đảo ngược (toggle), tạo ra tín hiệu xung vuông với chu kỳ tương ứng baudrate.

- Bộ đếm counter_tx đếm từng chu kỳ clock 50 MHz.
- Khi đếm đủ divider_tx chu kỳ, tx_clk đảo trạng thái, tạo ra 1 nửa chu kỳ xung vuông.
- Như vậy, chu kỳ đầy đủ xung vuông tx_clk sẽ là 2 * divider_tx clock 50 MHz, tương đương baudrate cần thiết.
- Nếu reset được kích hoạt (reset_n = 0), bộ đếm và xung được khởi tạo về 0.

- **Phần tạo xung nhận (rx_clk):**

```
// Tạo rx_clk (dạng xung vuông - oversample x16)
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        counter_rx <= 0;
        rx_clk <= 0;
    end else begin
        if (counter_rx >= divider_rx) begin
            counter_rx <= 0;
            rx_clk <= ~rx_clk;
        end else begin
            counter_rx <= counter_rx + 1;
        end
    end
end
```

Hình 3.5 Bộ tạo xung nhận

Phần này tương tự như phần trên nhưng dùng bộ đếm counter_rx với giá trị chia nhỏ hơn (chia cho 16 lần baudrate) để tạo ra xung rx_clk hỗ trợ kỹ thuật oversampling cho bộ thu UART.

Ý nghĩa oversampling 16x trong UART

- Trong UART, bộ thu không nhận trực tiếp theo xung baudrate mà thường nhận mẫu dữ liệu nhiều lần trong mỗi bit.
- Oversampling 16x có nghĩa là lấy mẫu tín hiệu 16 lần trong một bit dữ liệu, tăng độ chính xác nhận dạng bit.
- Nhờ vậy giảm sai sót do jitter, nhiễu tín hiệu.

3.2.2. UART Transmitter

a. Mục đích

Module UartTransmitter là bộ truyền dữ liệu nối tiếp theo chuẩn UART được thiết kế bằng ngôn ngữ mô tả phần cứng Verilog. Mục tiêu của module là nhận dữ liệu song song từ hệ thống (8-bit hoặc cấu hình từ 5 đến 9 bit), chuyển đổi thành tín hiệu nối tiếp UART và truyền ra ngoài qua chân TX. Module đảm bảo đúng định dạng khung truyền UART:

Start bit (0) – Data bits (5–9 bit) – Parity bit (nếu bật) – Stop bit (1 hoặc 2 bit)

Chức năng chính của module:

- Truyền dữ liệu UART với cấu hình linh hoạt:
  - Số bit dữ liệu: từ 5 đến 9 bit.
  - Có hoặc không parity bit.
  - 1 hoặc 2 stop bit.
- Tích hợp FIFO 16 phần tử để lưu nhiều byte chờ truyền.
- Báo hiệu trạng thái hoạt động:
  - busy: đang truyền.
  - done: hoàn tất một khung truyền.
- Phát hiện và báo lỗi trong quá trình truyền (ví dụ như mất xung baud).

b. Các tín hiệu và tham số

Bảng tín hiệu vào/ra:

Tên tín hiệu	Kiểu	Mô tả
clk	input	Clock hệ thống
rst_n	input	Reset đồng bộ, active-low
baud_tick	input	Xung baud: 1 xung trên mỗi chu kỳ truyền 1 bit
en	input	Cho phép module hoạt động
start	input	Kích hoạt truyền dữ liệu từ FIFO

in[DATA_BITS-1:0]	input	Dữ liệu song song đầu vào
out	output reg	Tín hiệu UART nối tiếp đầu ra
busy	output reg	Cờ cho biết module đang truyền
done	output reg	Cờ báo hoàn tất một khung truyền
tx_error	output reg	Cờ lỗi timeout hoặc lỗi trong quá trình truyền

Bảng 3.1 Tín hiệu ra/vào UART Transmitter

Các tham số cấu hình:

Tên tham số	**Mô tả**
DATA_BITS	Số bit dữ liệu (5–9)
STOP_BITS	Số stop bit (1 hoặc 2)
ENABLE_PARITY	0: Tắt parity, 1: Bật parity
PARITY_TYPE	0: Even parity, 1: Odd parity

TIMEOUT_CYCLES	Số chu kỳ baud_tick tối đa cho phép giữa các bit truyền

Bảng 3.2 Các tham số cấu hình UART Transmitter

c. Hoạt động FSM (Finite State Machine)

FSM điều khiển quá trình truyền dữ liệu nối tiếp được mô hình hóa qua 5 trạng thái chính:

1. **STATE_IDLE**
   - Module ở trạng thái chờ.
   - Nếu start = 1 và FIFO không rỗng → chuyển sang STATE_START_BIT.
2. **STATE_START_BIT**
   - Phát bit start (logic 0).
   - Chờ 1 xung baud_tick để chuyển sang STATE_DATA.
3. **STATE_DATA**
   - Gửi từng bit dữ liệu, từ LSB đến MSB.
   - Sau khi gửi đủ DATA_BITS, chuyển sang STATE_PARITY nếu bật parity, hoặc sang STATE_STOP.
4. **STATE_PARITY** (nếu ENABLE_PARITY = 1)
   - Tính toán và phát bit parity theo cấu hình even/odd.
   - Sau đó chuyển sang STATE_STOP.
5. **STATE_STOP**
   - Phát 1 hoặc 2 stop bit (logic 1).
   - Sau stop bit cuối cùng → bật cờ done, chuyển về STATE_IDLE.

d. FIFO truyền

Module tích hợp một bộ FIFO 16 phần tử (độ sâu = 16 byte) để lưu trữ dữ liệu song song trước khi truyền.

Cơ chế hoạt động:

- Khi tín hiệu start = 1 và busy = 0, dữ liệu từ FIFO sẽ được đọc ra để truyền.
- FIFO giúp truyền nhiều dữ liệu liên tục mà không bị mất dữ liệu đầu vào.

e. Tính parity

Công thức tính parity:

wire parity_even = ^in_data; // XOR tất cả các bit

wire parity_to_send = (PARITY_TYPE == 1) ? ~parity_even : parity_even;

- ^in_data: Tính parity even bằng XOR toàn bộ dữ liệu.
- Nếu cấu hình PARITY_TYPE = 1 (odd), bit parity là **nghịch đảo** của parity_even.

f. Timeout (giới hạn thời gian)

Trong khi truyền, nếu module **không nhận được xung baud_tick** trong khoảng thời gian xác định bởi TIMEOUT_CYCLES, thì:

- Cờ tx_error được bật lên.
- FSM quay trở lại STATE_IDLE.
- Dữ liệu đang truyền được hủy bỏ để tránh lỗi khung hoặc treo trạng thái.

Ý nghĩa: Timeout giúp phát hiện lỗi phần cứng như mất kết nối baudrate hoặc xung nhịp không đều, đảm bảo module luôn quay về trạng thái ổn định.

g. Phân tích chi tiết code

1. Khai báo module và các ngõ vào/ra

```
module UartTransmitter (
    input clk,                // Xung baud rate (1 clk = 1 bit truyền)
    input enable,            // Nếu enable = 0, reset toàn bộ module
    input tx_start,          // Ghi dữ liệu vào FIFO
    input [7:0] tx_in,       // Dữ liệu đầu vào

    input parity_enable,     // Bật/tắt parity
    input parity_odd_even,   // 0: chẵn, 1: lẻ

    output reg out,          // Tín hiệu UART nối tiếp
    output reg busy,         // Cờ báo đang truyền dữ liệu (1: đang bận)
    output reg done,         // Cờ báo truyền xong 1 chu kì truyền
    output [2:0] current_state  // Trạng thái hiện tại
);
```

Hình 3.5 Code khai báo của UART Transmitter

- Module này nhận dữ liệu song song tx_in và truyền nó nối tiếp ra chân out theo chuẩn UART.
- Các cổng điều khiển như tx_start, busy, done giúp kết nối linh hoạt với hệ thống bên ngoài.

2. Khai báo các trạng thái của FSM và các thanh ghi điều khiển bên trong

```
localparam STATE_IDLE        = 3'd0,
           STATE_START_BIT   = 3'd1,
           STATE_DATA_BIT    = 3'd2,
           STATE_PARITY_BIT  = 3'd3,
           STATE_STOP_BIT    = 3'd4;

reg [2:0] state = STATE_IDLE;
reg [2:0] bit_index = 0;
reg [7:0] tx_shift = 0;
reg [7:0] parity_calc_data = 0;
reg parity_bit;
```

Hình 3.6 Khai báo trạng thái FSM UART Transmitter

- Máy trạng thái hữu hạn (FSM) gồm 5 trạng thái:
  - IDLE: chờ dữ liệu mới.
  - START_BIT: gửi bit start (0).
  - DATA_BIT: gửi 8 bit dữ liệu.
  - PARITY_BIT: gửi parity (nếu được bật).
  - STOP_BIT: kết thúc với bit stop (1).
- tx_shift dùng để dịch từng bit ra ngoài.
- bit_index đếm từ 0 đến 7 khi truyền dữ liệu.
- parity_calc_data dùng để giữ nguyên dữ liệu gốc để tính toán parity, tránh xung đột với tx_shift.

3. FIFO lưu tạm dữ liệu và khối tính toán parity

```
// FIFO 16-byte
reg [7:0] fifo [0:15];
reg [3:0] fifo_wr_ptr = 0;
reg [3:0] fifo_rd_ptr = 0;
reg [4:0] fifo_count = 0;

wire fifo_empty = (fifo_count == 0);
wire fifo_full  = (fifo_count == 16);

assign current_state = state; // Trạng thái hiện tại

// Tính parity
always @(*) begin
    if (parity_enable)
        parity_bit = parity_odd_even ^ (^parity_calc_data);
    else
        parity_bit = 1'b0;
end
```

Hình 3.7 FIFO và khối tính toán Parity của Transmitter

- FIFO hoạt động như bộ đệm tạm: giúp chờ dữ liệu nếu UART đang bận.
- Khi tx_start = 1, dữ liệu được đẩy vào FIFO nếu chưa đầy.
- Khi ở trạng thái IDLE và FIFO không rỗng → lấy ra để truyền.
- Toán tử ^ (XOR reduce): tính tổng chẵn/lẻ các bit trong parity_calc_data.
- Nếu parity_odd_even = 0 → parity chẵn (even), nếu 1 → parity lẻ (odd).
- Ví dụ: ^8'b10110010 = 1'b1 → số bit 1 là lẻ.

7. Khối xử lý FIFO đọc/ghi

```
// FIFO điều khiển ghi/đọc
always @(posedge clk) begin
    if (!enable) begin
        fifo_wr_ptr <= 0;
        fifo_rd_ptr <= 0;
        fifo_count <= 0;
    end else begin
        if (tx_start && !fifo_full) begin
            fifo[fifo_wr_ptr] <= tx_in;
            fifo_wr_ptr <= fifo_wr_ptr + 1;
            fifo_count <= fifo_count + 1;
        end
        if (state == STATE_IDLE && !fifo_empty) begin
            fifo_rd_ptr <= fifo_rd_ptr + 1;
            fifo_count <= fifo_count - 1;
        end
    end
end
```

Hình 3.8 Khối xử lý FIFO đọc/ghi

- Nếu enable = 0: reset FIFO.
- Nếu tx_start = 1 và FIFO chưa đầy: ghi dữ liệu vào FIFO.
- Nếu đang IDLE và FIFO có dữ liệu: lấy ra để chuẩn bị truyền.

8. FSM điều khiển tiến trình truyền UART

```
// FSM UART
always @(posedge clk) begin
    if (!enable) begin
        state <= STATE_IDLE;
        out <= 1'b1;
        busy <= 0;
        done <= 0;
        bit_index <= 0;
        tx_shift <= 0;
    end else begin
      done <= 0;
        case (state)
            STATE_IDLE: begin
                out <= 1'b1;
                busy <= 0;
                if (!fifo_empty) begin
                    tx_shift <= fifo[fifo_rd_ptr];
                    parity_calc_data <= fifo[fifo_rd_ptr];
                    state <= STATE_START_BIT;
                    busy <= 1;
                end
            end

            STATE_START_BIT: begin
                out <= 1'b0;
                bit_index <= 0;
                state <= STATE_DATA_BIT;
            end

            STATE_DATA_BIT: begin
                out <= tx_shift[0];
                tx_shift <= tx_shift >> 1; // Truyền từ LSB trước
                bit_index <= bit_index + 1;
                if (bit_index == 3'd7) begin
                    state <= (parity_enable) ? STATE_PARITY_BIT : STATE_STOP_BIT;
                end
            end

            STATE_PARITY_BIT: begin
                out <= parity_bit;
                state <= STATE_STOP_BIT;
            end

            STATE_STOP_BIT: begin
                out <= 1'b1;
                state <= STATE_IDLE;
                busy <= 0;
                done <= 1;
            end
        endcase
    end
end

endmodule
```

Hình 3.9 FSM điều khiển tiến trình truyền UART Transmitter

- Nếu enable = 0: FSM bị reset về trạng thái IDLE. Các tín hiệu điều khiển được đưa về mặc định:

- out = 1'b1: đường truyền UART thả nổi ở mức HIGH.
- busy = 0: không bận truyền.
- done = 0: chưa hoàn tất truyền.
- bit_index = 0: chuẩn bị đếm bit từ đầu.
- tx_shift = 0: thanh ghi tạm chứa dữ liệu cần truyền được xóa.

- Nếu enable = 1: bắt đầu xử lý FSM. Đặt lại done = 0 để đảm bảo chỉ set khi hoàn tất 1 byte ở trạng thái STATE_STOP_BIT.

Trạng thái STATE_IDLE

Chờ dữ liệu trong FIFO. Nếu có dữ liệu:

- Nạp byte cần truyền vào tx_shift.
- Ghi lại dữ liệu để tính **parity** nếu cần.
- Chuyển sang trạng thái gửi **bit START**.
- Báo hiệu module đang **bận** truyền (busy = 1).

Trạng thái STATE_START_BIT

Gửi **bit START** (mức thấp 0) để báo hiệu bắt đầu truyền 1 byte.

Trạng thái STATE_DATA_BIT

FSM gửi từng bit dữ liệu:

- Gửi bit thấp nhất tx_shift[0] (UART dùng **LSB first**).
- Dịch phải tx_shift để chuẩn bị cho lần gửi tiếp theo.
- Tăng bit_index để đếm đủ 8 bit.
- Sau khi gửi xong 8 bit:
    - Nếu có bật parity (parity_enable) → sang trạng thái gửi **bit parity**.
    - Ngược lại → sang trạng thái gửi **bit STOP**.

Trạng thái STATE_PARITY_BIT

Gửi **bit parity** (odd hoặc even tùy cấu hình trước đó). Sau đó chuyển sang trạng thái gửi **bit STOP**.

Trạng thái STATE_STOP_BIT

Gửi **bit STOP** (luôn là 1). Sau đó:

- Quay lại IDLE.
- Bỏ cờ busy.
- Kích hoạt tín hiệu done trong đúng 1 chu kỳ để báo quá trình truyền byte đã hoàn tất.

3.2.3. UART Receiver

a. Mục đích

Module UartReceiver là khối nhận dữ liệu nối tiếp chuẩn UART, được thiết kế bằng ngôn ngữ Verilog. Nhiệm vụ của module là nhận tín hiệu UART nối tiếp từ bên ngoài (qua chân rx), giải mã khung dữ liệu gồm:

Start bit (0) – Dữ liệu (5–9 bit) – Parity (nếu có) – Stop bit (1 hoặc 2)

Sau đó, dữ liệu được chuyển thành song song và lưu vào FIFO đầu ra. Module đảm bảo quá trình nhận diễn ra ổn định, có báo lỗi và hỗ trợ cấu hình linh hoạt.

Chức năng chính:

- Nhận tín hiệu nối tiếp và giải mã khung UART.
- Hỗ trợ:
    - Tùy chọn số bit dữ liệu (5–9).
    - Parity bit (even/odd hoặc tắt).
    - Số stop bit: 1 hoặc 2.
- Tích hợp FIFO 16 phần tử để lưu dữ liệu nhận được.
- Phát hiện lỗi khung:
    - Lỗi parity.
    - Lỗi framing (sai stop bit).
    - Timeout: mất xung baud hoặc không có tín hiệu đúng lúc.

b. Các tín hiệu và tham số

Bảng tín hiệu vào/ra:

Tên tín hiệu	Kiểu	Mô tả
clk	input	Clock hệ thống
rst_n	input	Reset đồng bộ, active-low
baud_tick	input	Xung baud: 1 xung trên mỗi chu kỳ bit
rx	input	Tín hiệu UART đầu vào
en	input	Cho phép hoạt động
data_out[DATA_BITS-1:0]	output reg	Dữ liệu song song đã nhận
ready	output reg	Báo dữ liệu đã sẵn sàng để đọc
rx_error	output reg	Cờ lỗi khung/parity/timeout

Bảng 3.3 Bảng tín hiệu vào/ra UART Receiver

Tham số cấu hình:

Tên tham số	Mô tả
DATA_BITS	Số bit dữ liệu (5–9)
STOP_BITS	Số stop bit (1 hoặc 2)

ENABLE_PARITY	0: Tắt parity, 1: Bật parity
PARITY_TYPE	0: Even parity, 1: Odd parity
TIMEOUT_CYCLES	Số chu kỳ baud_tick tối đa giữa các bit

Bảng 3.4 Tham số cấu hình UART Receiver

c. Hoạt động FSM

FSM chính điều khiển module UART Receiver gồm các trạng thái sau:

1. **STATE_IDLE**
   - Chờ phát hiện cạnh xuống (bit start) trên rx = 0.
   - Khi phát hiện tín hiệu start → đợi 1/2 chu kỳ baud để lấy mẫu giữa bit.
2. **STATE_START_BIT**
   - Kiểm tra giá trị của bit start.
   - Nếu `rx ≠ 0` → lỗi start bit, quay về IDLE.
   - Nếu hợp lệ → sang STATE_DATA.
3. **STATE_DATA**
   - Ghi nhận từng bit dữ liệu vào thanh ghi đệm, theo baud_tick.
   - Đếm đến đủ DATA_BITS, chuyển tiếp.
4. **STATE_PARITY** (nếu ENABLE_PARITY = 1)
   - Đọc bit parity.
   - Tính toán parity từ dữ liệu nhận được và so sánh với bit parity nhận.
   - Nếu sai → đặt rx_error.
5. **STATE_STOP**
   - Kiểm tra stop bit (logic 1).
   - Nếu `rx ≠ 1` tại bất kỳ bit stop nào → lỗi framing.
   - Nếu hợp lệ → ghi dữ liệu vào FIFO, bật cờ ready.
6. **STATE_DONE/RESET**
   - Reset bộ đếm, chuẩn bị nhận tiếp dữ liệu tiếp theo.

d. FIFO nhận

Module sử dụng một FIFO sâu 16 phần tử để lưu trữ dữ liệu nhận. FIFO giúp giảm nguy cơ mất dữ liệu nếu phía đọc không kịp đọc từng byte ngay sau khi nhận.

- Dữ liệu nhận hợp lệ được đẩy vào FIFO.
- Tín hiệu ready được bật lên khi có dữ liệu trong FIFO.

e. Kiểm tra parity

Cách tính và so sánh parity:

wire parity_even = ^rx_data; // XOR tất cả bit dữ liệu

wire expected_parity = (PARITY_TYPE == 1) ? ~parity_even : parity_even;

- Bit parity được nhận so sánh với giá trị tính được.
- Nếu sai → bật cờ rx_error.

f. Framing error và timeout

- **Framing error:** xảy ra nếu stop bit không phải là logic '1' như yêu cầu.
- **Timeout:** nếu không nhận được xung baud_tick trong khoảng TIMEOUT_CYCLES, module sẽ:
    - Báo lỗi rx_error.
    - Reset trạng thái FSM.

g. Phân tích chi tiết code

1. Khai báo module và các tín hiệu

```
module UartReceiver (
    input clk,                // Tín hiệu lấy mẫu UART (1 clk = 1/16 bit)
    input enable,             // Reset toàn module khi = 0
    input rx_in,              // Dữ liệu UART nối tiếp

    input parity_enable,      // Bật kiểm tra parity
    input parity_odd_even,    // 0: chẵn, 1: lẻ

    input rx_read,            // Đọc từ FIFO

    output reg [7:0] rx_data, // Dữ liệu nhận được
    output reg fifo_empty,
    output reg fifo_full,
    output reg error,
    output [2:0] current_state
);
```

Hình 3.10 Khai báo module và các tín hiệu Receiver

2. Định nghĩa các trạng thái FSM,các biến trạng thái và bộ đếm

```
localparam STATE_IDLE        = 3'd0,
           STATE_START_BIT   = 3'd1,
           STATE_DATA_BIT    = 3'd2,
           STATE_PARITY_BIT  = 3'd3,
           STATE_STOP_BIT    = 3'd4;

reg [2:0] state = STATE_IDLE;
reg [2:0] bit_index = 0;
reg [7:0] rx_shift = 0;
reg parity_calc = 0;
reg [3:0] tick_count = 0;    // đếm từ 0..15
reg data_ready = 0;
assign current state = state;
```

Hình 3.11 Định nghĩa các trạng thái FSM,các biến trạng thái và bộ đếm Receiver

- Các trạng thái của bộ FSM để giải mã UART:
  - **IDLE**: chờ tín hiệu start bit (thường là 0).
  - **START_BIT**: xác nhận start bit.
  - **DATA_BIT**: nhận dữ liệu 8 bit.
  - **PARITY_BIT**: nhận và kiểm tra parity bit nếu bật.
  - **STOP_BIT**: kiểm tra stop bit (phải là 1).
- state: trạng thái FSM hiện tại.
- bit_index: vị trí bit dữ liệu đang nhận (0 đến 7).
- rx_shift: thanh trượt chứa dữ liệu nhận dần.
- parity_calc: tính parity (dùng XOR dần các bit nhận được).

- tick_count: đếm 16 clock (tương đương 1 bit UART) để lấy mẫu chính xác giữa bit.
- Data_ready: báo dữ liệu đã nhận xong và sẵn sàng lưu vào FIFO.

3. FIFO bộ đệm dữ liệu và trạng thái trống/đầy FIFO

```
// FIFO buffer
reg [7:0] fifo [0:15];
reg [3:0] fifo_wr_ptr = 0;
reg [3:0] fifo_rd_ptr = 0;
reg [4:0] fifo_count = 0;

always @(*) begin
    fifo_empty = (fifo_count == 0);
    fifo_full  = (fifo_count == 16);
end
```

Hình 3.12 FIFO của Receiver

- FIFO có 16 phần tử, dùng để lưu tạm dữ liệu nhận.
- fifo_wr_ptr: vị trí ghi tiếp theo.
- fifo_rd_ptr: vị trí đọc tiếp theo.
- fifo_count: số phần tử hiện có trong FIFO.
- Nếu fifo_count == 0 => FIFO rỗng.
- Nếu fifo_count == 16 => FIFO đầy.
- Hai tín hiệu này luôn được cập nhật khi fifo_count thay đổi.

4. Xử lý ghi vào, đọc ra FIFO (cùng trong clock posedge)

```
always @(posedge clk) begin
    if (!enable) begin
        fifo_wr_ptr <= 0;
        fifo_rd_ptr <= 0;
        fifo_count <= 0;
        rx_data <= 0;
    end else begin
        if (data_ready && !fifo_full) begin
            fifo[fifo_wr_ptr] <= rx_shift;
            fifo_wr_ptr <= fifo_wr_ptr + 1;
            fifo_count <= fifo_count + 1;
        end
        if (rx_read && !fifo_empty) begin
            rx_data <= fifo[fifo_rd_ptr];
            fifo_rd_ptr <= fifo_rd_ptr + 1;
            fifo_count <= fifo_count - 1;
        end
    end
end
```

Hình 3.13 Xử lý ghi vào, đọc ra FIFO của Receiver

- Khi module **enable = 0**, reset các con trỏ và biến FIFO.
- Nếu data_ready (dữ liệu mới nhận xong) và FIFO chưa đầy, ghi rx_shift (byte nhận được) vào FIFO.
- Khi tín hiệu rx_read (bên ngoài đọc dữ liệu) và FIFO không rỗng, lấy dữ liệu ra rx_data và cập nhật con trỏ, số lượng.
- Cơ chế FIFO này giúp nhận dữ liệu liên tục, không bị mất bit khi chưa kịp đọc.

5. FSM điều khiển nhận dữ liệu UART

```
always @(posedge clk) begin
    if (!enable) begin
        state <= STATE_IDLE;
        bit_index <= 0;
        rx_shift <= 0;
        parity_calc <= 0;
        tick_count <= 0;
        data_ready <= 0;
        error <= 0;
    end else begin
        data_ready <= 0;
        case (state)
            STATE_IDLE: begin
                error <= 0;
                tick_count <= 0;
                if (rx_in == 0) begin
                    state <= STATE_START_BIT;
                    tick_count <= 0; // bắt đầu đếm baudtick
                end
            end

            STATE_START_BIT: begin
                tick_count <= tick_count + 1;
                if (tick_count == 7) begin  // Lấy mẫu giữa bit start
                    if (rx_in == 0) begin
                        // chuẩn bị nhận data bits
                        tick_count <= 0;
                        bit_index <= 0;
                        parity_calc <= 0;
                        state <= STATE_DATA_BIT;
                    end else begin
                        state <= STATE_IDLE; // lỗi start bit
                    end
                end
            end

            STATE_DATA_BIT: begin
                tick_count <= tick_count + 1;
                if (tick_count == 15) begin  // Lấy mẫu giữa bit data
                    rx_shift <= {rx_in, rx_shift[7:1]};    // LSB => MSB
                    parity_calc <= parity_calc ^ rx_in;

                    tick_count <= 0;
                    if (bit_index == 7) begin
                        state <= (parity_enable) ? STATE_PARITY_BIT : STATE_STOP_BIT;
                        bit_index <= 0;
                    end else begin
                        bit_index <= bit_index + 1;
                    end
                end
            end

            STATE_PARITY_BIT: begin
                tick_count <= tick_count + 1;
                if (tick_count == 15) begin // lấy mẫu parity bit
                    if ((parity_odd_even && rx_in != ~parity_calc) ||
                        (!parity_odd_even && rx_in != parity_calc))
                        error <= 1;
                    tick_count <= 0;
                    state <= STATE_STOP_BIT;
                end
            end

            STATE_STOP_BIT: begin
                tick_count <= tick_count + 1;
                if (tick_count == 15) begin // lấy mẫu stop bit
```

```
                    if (rx_in == 1) begin
                        if (!error)
                            data_ready <= 1;
                    end else begin
                        error <= 1;
                    end
                    tick_count <= 0;
                    state <= STATE_IDLE;
                end
            end

            default: state <= STATE_IDLE;
        endcase
    end
end

endmodule
```

Hình 3.14 FSM điều khiển nhận dữ liệu UART Receiver

Giải thích chi tiết FSM:

- **STATE_IDLE**:
  - Module chờ tín hiệu **rx_in == 0** (start bit).
  - Reset lỗi, reset bộ đếm.
  - Nếu thấy start bit, chuyển sang trạng thái STATE_START_BIT.
- **STATE_START_BIT**:
  - Đếm tick_count lên, đếm đến giữa bit (tick 7/15).
  - Lấy mẫu giữa bit start để tránh nhiễu hoặc sai lệch.
  - Nếu vẫn là 0, start bit hợp lệ, chuẩn bị nhận 8 bit data.
  - Nếu không, quay về IDLE.
- **STATE_DATA_BIT**:
  - Đếm 16 tick cho mỗi bit data.
  - Lấy mẫu tại tick 15 (giữa bit data).
  - Dữ liệu nhận được được shift vào rx_shift, bit mới nằm bên trái (MSB).
  - Cập nhật parity tính bằng XOR dần các bit data.
  - Khi nhận đủ 8 bit, nếu parity bật thì qua trạng thái kiểm tra parity, ngược lại chuyển thẳng stop bit.
- **STATE_PARITY_BIT**:
  - Lấy mẫu parity bit.
  - So sánh bit nhận với giá trị parity tính được (theo parity chẵn/lẻ).
  - Nếu sai, báo lỗi parity.
- **STATE_STOP_BIT**:
  - Lấy mẫu stop bit.

- Nếu stop bit không đúng (không phải 1), báo lỗi.

- Nếu đúng và không lỗi, đánh dấu dữ liệu nhận xong data_ready.
- Quay lại trạng thái IDLE chờ nhận byte tiếp theo.

3.2.4. UART Module-Top

3.2.4.1. UART Top-Transmitter

Chức năng:

Đây là module top-level dùng để tích hợp toàn bộ hệ thống truyền dữ liệu UART, bao gồm bộ tạo xung baudrate (BaudRateGenerator), bộ truyền dữ liệu (UartTransmitter), và mạch giải mã 7 đoạn (SevenSegmentDecoder) để hiển thị dữ liệu truyền. Module này nhận dữ liệu từ hệ thống bên ngoài, xử lý cấu hình truyền như parity và baudrate, sau đó truyền dữ liệu qua giao tiếp UART.

Chức năng chi tiết các module con:

1. **BaudRateGenerator**
   - Chức năng: Tạo xung clock tốc độ thấp hơn từ clock hệ thống 50 MHz, tương ứng với baudrate được chọn.
   - Tín hiệu đầu vào:
     - clk: Clock hệ thống (50 MHz).
     - reset_n: Tín hiệu reset active-low.
     - baud_select: 2-bit để chọn baudrate (00: 9600, 01: 14400, 10: 19200, 11: 115200).
   - Tín hiệu đầu ra:
     - tx_clk: Clock baudrate dùng cho truyền UART.
2. **UartTransmitter**
   - Chức năng: Truyền một byte dữ liệu UART theo cấu hình parity và baudrate.
   - Tín hiệu đầu vào:
     - clk: Clock baudrate từ module BaudRateGenerator.
     - enable: Tín hiệu cho phép hoạt động, kết nối với reset_n.
     - tx_start: Tín hiệu bắt đầu truyền (đảo ngược với ~tx_start do logic active-high của transmitter).

        - tx_in: Byte dữ liệu 8 bit cần truyền.
        - parity_enable: Cho phép truyền bit parity.
        - parity_odd_even: Cấu hình parity chẵn/lẻ (0: even, 1: odd).
    - Tín hiệu đầu ra:
        - out: Tín hiệu UART output (tx).
        - busy: Cho biết transmitter đang truyền (logic high khi bận).
        - done: Báo hiệu truyền xong (1 chu kỳ).
3. **SevenSegmentDecoder**
    - Chức năng: Giải mã dữ liệu truyền để hiển thị trên 2 LED 7 đoạn (HEX1, HEX0).
    - tx_data_in[3:0] được giải mã hiển thị trên HEX0, biểu diễn 4 bit thấp.
    - tx_data_in[7:4] được giải mã hiển thị trên HEX1, biểu diễn 4 bit cao.

Bảng 3.2.4.1. Tín hiệu đầu vào/ra của UartTopTransmitter:

Tên tín hiệu	I/O	Mô tả
clk	input	Clock hệ thống 50 MHz từ KIT DE2
reset_n	input	Reset active-low
baud_select	input	2-bit chọn baudrate
tx_start	input	Tín hiệu bắt đầu truyền dữ liệu (1 xung cao)
tx_data_in	input	Byte dữ liệu cần truyền (8 bit)
parity_enable	input	Cho phép dùng bit parity (1: có parity)
parity_odd_even	input	0: parity chẵn (even), 1: parity lẻ (odd)

tx	output	Tín hiệu truyền UART ra ngoài
busy	output	Báo transmitter đang truyền dữ liệu
done	output	Báo hiệu truyền xong một byte dữ liệu
HEX0, HEX1	output	Hiển thị dữ liệu truyền trên LED 7 đoạn

Bảng 3.5 Tín hiệu đầu vào/ra của UartTopTransmitter

3.2.4.2. UART Top-Receiver

Chức năng:

Đây là module top-level tích hợp toàn bộ hệ thống nhận dữ liệu UART, bao gồm:

- Bộ tạo xung baudrate cho nhận (BaudRateGenerator)
- Bộ thu UART có hỗ trợ parity, FIFO và báo lỗi (UartReceiver)
- Mạch giải mã LED 7 đoạn để hiển thị dữ liệu nhận được.

Module này nhận dữ liệu UART từ bên ngoài (ví dụ từ PuTTY), kiểm tra tính hợp lệ của dữ liệu (parity, khung...), lưu trữ dữ liệu qua FIFO, cung cấp cơ chế đọc dữ liệu ra và hiển thị giá trị nhận được trực tiếp trên KIT FPGA thông qua LED 7 đoạn.

Chức năng chi tiết các module con:

1. **BaudRateGenerator**
    - Chức năng: Tạo xung clock tốc độ thấp hơn từ clock 50 MHz, phù hợp với tốc độ truyền được cấu hình.
    - Tín hiệu đầu vào:
        - clk: Clock hệ thống 50 MHz.
        - reset_n: Tín hiệu reset active-low.
        - baud_select: 2-bit chọn baudrate (00: 9600, 01: 14400, 10: 19200, 11: 115200).
    - Tín hiệu đầu ra:

        - rx_clk: Clock dùng cho module nhận UART.
2. **UartReceiver**
    - Chức năng: Nhận dữ liệu UART từ PC, kiểm tra lỗi parity, khung truyền, sau đó lưu trữ vào FIFO nếu hợp lệ. Hỗ trợ FIFO có tín hiệu báo đầy (fifo_full), rỗng (fifo_empty) và lỗi (error).
    - Tín hiệu đầu vào:
        - clk: Clock baudrate từ BaudRateGenerator.
        - enable: Tín hiệu cho phép hoạt động, kết nối với reset_n.
        - rx_in: Tín hiệu UART từ ngoài vào.
        - parity_enable: Cho phép kiểm tra parity.
        - parity_odd_even: Cấu hình parity chẵn/lẻ.
        - rx_read: Tín hiệu đọc dữ liệu từ FIFO (đảo ngược ~rx_read vì logic active-high).
    - Tín hiệu đầu ra:
        - rx_data: Byte dữ liệu đã nhận.
        - fifo_empty, fifo_full: Báo trạng thái FIFO.
        - error: Cảnh báo nếu dữ liệu nhận có lỗi parity hoặc framing.
        - current_state: Trạng thái hiện tại của FSM trong bộ thu UART (dùng để debug hoặc hiển thị).
3. **SevenSegmentDecoder**

    - Chức năng: Giải mã dữ liệu 8-bit nhận được và hiển thị trực tiếp trên LED 7 đoạn:
        - rx_data[3:0] hiển thị trên HEX0.
        - rx_data[7:4] hiển thị trên HEX1.

Bảng 3.2.4.2. Tín hiệu đầu vào/ra của UartTopReceiver:

Tên tín hiệu	I/O	Mô tả
clk	input	Clock hệ thống 50 MHz từ KIT

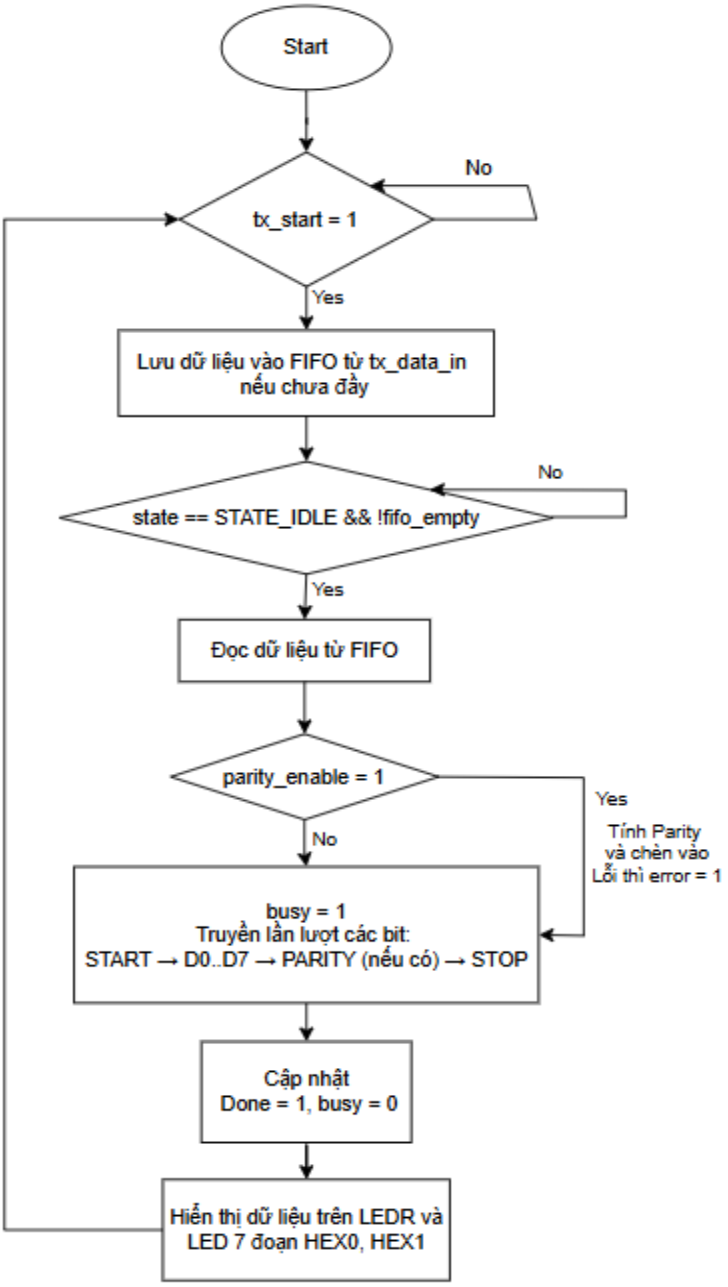
reset_n	input	Tín hiệu reset (active low)
baud_select	input	2-bit chọn tốc độ baudrate nhận
rx_in	input	Dữ liệu UART từ PC (ví dụ từ PuTTY)
parity_enable	input	Cho phép kiểm tra parity
parity_odd_even	input	0: parity chẵn (even), 1: parity lẻ (odd)
rx_read	input	Tín hiệu đọc dữ liệu từ FIFO (active-high)
rx_data	output	Dữ liệu 8-bit đã nhận thành công
fifo_empty	output	FIFO rỗng, không có dữ liệu để đọc
fifo_full	output	FIFO đầy, không nhận thêm được nữa
error	output	Cảnh báo lỗi nhận data(parity/frame error)
current_state	output	Trạng thái hiện tại bộ thu UART (để kiểm tra/debug)
HEX0, HEX1	output	Hiển thị dữ liệu nhận trên LED 7 đoạn HEX0, HEX1

Bảng 3.6 Tín hiệu đầu vào/ra của UartTopReceiver

3.3. Lưu đồ hoạt động trong hệ thống

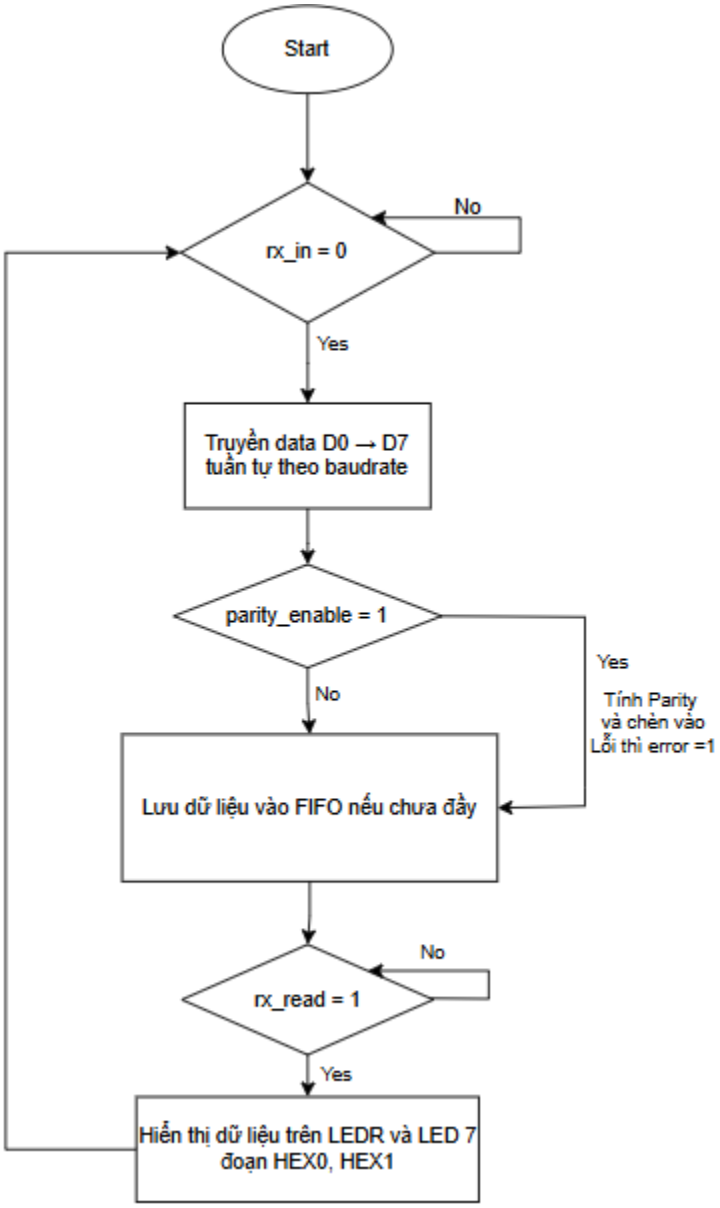
Hệ thống giao tiếp UART được thiết kế với hai khối chính: **Truyền (Transmitter)** và **Nhận (Receiver)**. Cả hai khối đều hoạt động đồng thời và có cấu trúc độc lập, nhưng sử dụng chung bộ tạo xung baudrate để đảm bảo đồng bộ dữ liệu. Hoạt động của hệ thống được thể hiện thông qua hai lưu đồ sau:

3.3.1. Lưu đồ hoạt động khối UART Transmitter



Hình 3.15 Lưu đồ hoạt động khối UART Transmitter

3.3.2. Lưu đồ hoạt động khối UART Receiver



Hình 3.16 Lưu đồ hoạt động khối UART Receiver

3.3.3. Mô tả tổng quát hệ thống

- Bộ tạo xung BaudRateGenerator nhận tín hiệu xung clk = 50MHz từ KIT và chia xung thành baudrate mong muốn (như 9600, 115200...).

- Dữ liệu cần truyền từ người dùng (bấm nút hoặc gửi từ máy tính) sẽ đi qua module UartTransmitter, mã hóa thành tín hiệu UART chuẩn và đưa ra chân tx.

- Ở phía nhận, tín hiệu rx_in từ máy tính được giải mã bởi UartReceiver, kiểm tra parity, sau đó lưu vào FIFO và đưa ra LED 7 đoạn.

- Hệ thống hỗ trợ chọn parity (even/odd), bật/tắt parity, lựa chọn baudrate và tương thích với các phần mềm giao tiếp như PuTTY.

3.4. Kết nối trên KIT DE2

Để hiện thực giao tiếp UART giữa KIT FPGA và máy tính, hệ thống cần cấu hình đúng các **chân tín hiệu (I/O pin)** và thiết lập **kết nối vật lý** giữa phần cứng và phần mềm PuTTY. Việc giao tiếp chủ yếu thực hiện qua các chân UART tiêu chuẩn (tx, rx) và một số tín hiệu điều khiển phụ trợ.

3.4.1. Sơ đồ kết nối UART

Thiết bị	Tín hiệu UART	Kết nối đến
KIT FPGA	tx (Transmit)	Cổng RX của USB-UART (PC)
KIT FPGA	rx (Receive)	Cổng TX của USB-UART (PC)
KIT FPGA	clk	Clock 50 MHz từ hệ thống
KIT FPGA	reset_n	Nút reset trên KIT (active low)
KIT FPGA	baud_select	SW[15:14] – Switch chọn tốc độ truyền
KIT FPGA	tx_start	Nút nhấn (KEY0) để gửi dữ liệu

KIT FPGA	tx_data_in	SW[7:0] – chọn dữ liệu truyền (8-bit)
KIT FPGA	rx_data	Hiển thị bằng LED 7 đoạn Và hiển thị LEDR[7:0]
KIT FPGA	parity_enable	SW[17] – bật/tắt parity
KIT FPGA	parity_odd_even	SW[16] – chọn kiểu parity
KIT FPGA	rx_read	KEY1 – nút đọc dữ liệu từ FIFO

Bảng 3.7 Sơ đồ kết nối UART

3.4.2. Mô tả chân tín hiệu chính

Tín hiệu	I/O	Mô tả chức năng
clk	In	Clock hệ thống 50 MHz
reset_n	In	Reset toàn bộ hệ thống (tích cực mức thấp) (KEY[0])
tx	Out	Tín hiệu UART truyền từ FPGA đến máy tính
rx_in	In	Tín hiệu UART từ máy tính gửi đến FPGA
tx_start	In	Bắt đầu truyền dữ liệu (1 xung mức cao) KEY[2]
tx_data_in	In	Dữ liệu 8-bit cần truyền (SW[7:0])

rx_data	Out	Dữ liệu 8-bit nhận được Hiển thị bằng LED 7 đoạn và LEDR[7:0]
baud_select	In	Lựa chọn tốc độ truyền (SW[15:14])
parity_enable	In	Bật/Tắt chế độ parity (SW[17])
parity_odd_even	In	Chọn parity: Even (0), Odd (1) (SW[16])
rx_read	In	Đọc dữ liệu từ FIFO khi có dữ liệu (KEY1)
HEX0, HEX1	Out	Hiển thị dữ liệu truyền/nhận dạng số thập lục phân
busy	Out	Báo hiệu đang truyền (trạng thái của UartTransmitter)(LEDG6 nếu cần)
done	Out	Báo hiệu truyền xong một frame dữ liệu (LEDG7 nếu cần)
fifo_empty	Out	Báo FIFO rỗng (không có dữ liệu nhận)(LEDG0)
fifo_full	Out	Báo FIFO đầy (không thể nhận thêm dữ liệu)(LEDG1)
error	Out	Báo lỗi parity hoặc lỗi giao tiếp (LEDG[2])
current_state	Out	Trạng thái FSM của UartReceiver để debug

Bảng 3.8 Mô tả chân tín hiệu chính

3.4.3. Kết nối vật lý

- **Kết nối với máy tính**:
  - Sử dụng **cáp USB to UART** (USB To RS232 D9 CH340G) để kết nối chân tx và rx giữa KIT FPGA và cổng COM máy tính.
  - Sơ đồ nối dây:
    - FPGA_tx → USB-UART_RX
    - FPGA_rx ← USB-UART_TX
    - GND ↔ GND
- **Kết nối phần mềm**:
  - Trên máy tính sử dụng phần mềm **PuTTY** để mở cổng COM tương ứng.
  - Cấu hình trên PuTTY:
    - Baudrate: 9600 hoặc 115200 (tùy baud_select)
    - Data bits: 8
    - Stop bits: 1
    - Parity: None / Even / Odd tùy cấu hình
    - Flow control: None

3.4.4. Mô tả hoạt dộng phần cứng

- Người dùng chọn dữ liệu 8-bit cần truyền qua các công tắc SW[7:0].
- Nhấn nút KEY2 để bắt đầu truyền.
- Trạng thái truyền được hiển thị qua LED 7 đoạn (HEX0, HEX1) và cờ busy, done.
- Dữ liệu nhận được từ PC cũng hiển thị trên LED 7 đoạn khi rx_read được nhấn (KEY1).
- Hệ thống kiểm tra lỗi parity tự động nếu parity_enable được bật.

3.5. Thông số cấu hình

Hệ thống UART được thiết kế với khả năng cấu hình linh hoạt, nhằm hỗ trợ nhiều chuẩn truyền thông khác nhau và phù hợp với các phần mềm/thiết bị ngoại vi như PuTTY, TeraTerm, hoặc các vi điều khiển giao tiếp UART. Các thông số có thể cấu hình bao gồm tốc độ truyền (baudrate), chế độ parity, và hỗ trợ oversampling để tăng độ chính xác trong truyền nhận.

1. Baudrate

Module BaudRateGenerator cho phép lựa chọn một trong bốn tốc độ truyền khác nhau thông qua tín hiệu điều khiển baud_select (2-bit). Các cấu hình baudrate cụ thể như sau:

baud_select	Baudrate (bps)
2'b00	9600
2'b01	14400
2'b10	19200
2'b11	115200

Bảng 3.9 Tín hiệu BaudRate

Tín hiệu tx_clk và rx_clk được tạo ra tương ứng để sử dụng trong các module UartTransmitter và UartReceiver.

2. Stop bit

Hệ thống UART được thiết kế sử dụng **1 stop bit** cố định cho mỗi khung truyền dữ liệu (frame), phù hợp với chuẩn UART phổ biến. Việc dùng 1 stop bit giúp tiết kiệm băng thông, trong khi vẫn đảm bảo độ tin cậy khi kết hợp với kiểm tra parity.

3. Parity

Hệ thống hỗ trợ sử dụng bit parity để kiểm tra lỗi đơn trong quá trình truyền. Có thể bật/tắt kiểm tra parity, cũng như chọn kiểu parity (even/odd) thông qua 2 tín hiệu điều khiển:

Tín hiệu	Mô tả
parity_enable	Bật (1) / Tắt (0) chế độ kiểm tra parity

parity_odd_even	Chọn kiểu parity: 0 → Even parity, 1 → Odd parity

Bảng 3.10 Tín hiệu Parity

Khi parity_enable = 1, hệ thống sẽ tính toán parity bit dựa trên dữ liệu 8-bit đầu vào và chèn thêm một bit parity vào khung truyền. Module UartReceiver cũng kiểm tra parity tương ứng, và nếu phát hiện sai lệch sẽ bật cờ lỗi error.

4. Oversampling

Trong module UartReceiver, tín hiệu UART được lấy mẫu với tốc độ **16 lần mỗi bit (16x oversampling)** để đảm bảo độ chính xác và khả năng chống nhiễu khi thu tín hiệu. Đây là phương pháp tiêu chuẩn trong UART nhằm cải thiện độ tin cậy khi xác định ranh giới bit và đọc giá trị chính xác của bit dữ liệu.

5. Độ dài dữ liệu

- Mỗi khung truyền gồm:
  1 start bit + 8 data bits + (optional) 1 parity bit + 1 stop bit
- Tổng độ dài: 10 đến 11 bit/frame tùy cấu hình parity.

6. FIFO (Receiver)

Trong module UartReceiver, dữ liệu nhận được được lưu vào một **FIFO buffer**. Điều này cho phép hệ thống xử lý các dữ liệu đến liên tục, đồng thời giảm khả năng mất dữ liệu nếu hệ thống đọc dữ liệu chậm hơn tốc độ nhận.

Tín hiệu FIFO	Mô tả
rx_data	Dữ liệu đầu ra 8-bit từ FIFO
fifo_empty	Cờ báo FIFO rỗng

fifo_full	Cờ báo FIFO đầy
rx_read	Tín hiệu yêu cầu đọc dữ liệu từ FIFO (active low)

Bảng 3.11 Tín hiệu FIFO

7. Hiển thị dữ liệu trên LED 7 đoạn

Trong cả module UartTopTransmitter và UartTopReceiver, dữ liệu đang truyền/nhận được giải mã và hiển thị theo từng nibble (4-bit) qua hai LED 7 đoạn:

LED	**Dữ liệu hiển thị**
HEX1	4 bit cao (bit 7–4)
HEX0	4 bit thấp (bit 3–0)

Bảng 3.12 Tín hiệu LED 7 đoạn

Điều này hỗ trợ trực quan hóa dữ liệu UART ngay trên KIT DE2.

Thông số	**Giá trị cấu hình**
Baudrate	9600 bps, 14400 bps, 19200 bps, 115200 bps
Stop bit	1
Parity	Có hỗ trợ parity, có thể bật/tắt

Oversampling	16x
Dữ liệu truyền/nhận	8-bit
FIFO	Có
Hiển thị	LED 7 đoạn (HEX0, HEX1)

Bảng 3.13 Thông số cấu hình

IV. Mô phỏng và kiểm thử

4.1. Môi trường mô phỏng (ModelSim)

Để kiểm tra tính đúng đắn của các module UART được thiết kế bằng ngôn ngữ Verilog, nhóm sử dụng phần mềm **ModelSim - Intel FPGA Edition** làm môi trường mô phỏng. Đây là công cụ phổ biến đi kèm với Quartus, hỗ trợ mô phỏng và kiểm tra các module phần cứng số (HDL) trước khi nạp xuống KIT FPGA.

Thông tin môi trường mô phỏng:

- **Tên phần mềm**: ModelSim - Intel FPGA Starter Edition
- **Phiên bản**: v10.5b
- **Hệ điều hành**: Windows 11
- **Tích hợp với Quartus**: Dùng chung thư mục project, dễ tích hợp mô phỏng với các file thiết kế Verilog

Mục đích mô phỏng:

- Kiểm tra hoạt động của các module:
    - BaudRateGenerator
    - UartTransmitter
    - UartReceiver
- Đảm bảo logic thiết kế chính xác trước khi tiến hành lập trình và thực nghiệm trên KIT DE2.
- Quan sát sóng tín hiệu như tx, rx_in, busy, done, rx_data, error trong waveform.

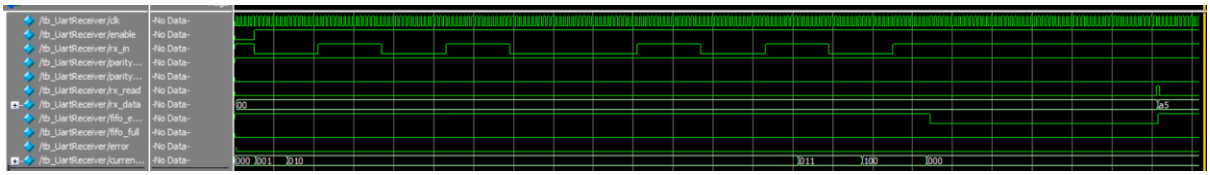
4.2. Testbench thiết kế và kết quả mô phỏng từng module

```
`timescale 1ns / 1ps

module tb_UartReceiver;

    reg clk;
    reg enable;
    reg rx_in;
    reg parity_enable;
    reg parity_odd_even;
    reg rx_read;
    wire [7:0] rx_data;
    wire fifo_empty, fifo_full, error;
    wire [2:0] current_state;

    // Instantiate the UartReceiver module
    UartReceiver uut (
        .clk(clk),
        .enable(enable),
        .rx_in(rx_in),
        .parity_enable(parity_enable),
        .parity_odd_even(parity_odd_even),
        .rx_read(rx_read),
        .rx_data(rx_data),
        .fifo_empty(fifo_empty),
        .fifo_full(fifo_full),
        .error(error),
        .current_state(current_state)
    );

    // Clock generation (16x baud rate)
    initial clk = 0;
    always #10 clk = ~clk; // 20ns period => 50MHz clock

    // Task to send a UART frame
    task send_uart_frame;
        input [7:0] data;
        input parity_en;
        input parity_odd;
        reg parity_bit;
        integer i;
        begin
```

```
            // Start bit
            rx_in = 0;
            #(16 * 20); // 16 clock ticks

            // Data bits (LSB first)
            parity_bit = 0;
            for (i = 0; i < 8; i = i + 1) begin
                rx_in = data[i];
                parity_bit = parity_bit ^ data[i];
                #(16 * 20);
            end

            // Parity bit
            if (parity_en) begin
                if (parity_odd)
                    rx_in = ~parity_bit; // odd parity
                else
                    rx_in = parity_bit;  // even parity
                #(16 * 20);
            end

            // Stop bit
            rx_in = 1;
            #(16 * 20);
        end
    endtask

    initial begin
        // Initial values
        enable = 0;
        rx_in = 1; // idle state
        parity_enable = 1;
        parity_odd_even = 0; // even parity
        rx_read = 0;

        // Reset and enable
        #100;
        enable = 1;

        // Send one frame: 0xA5 = 8'b10100101
        // Even parity: 1^0^1^0^0^1^0^1 = 0 => parity bit = 0
```

Hình 4.1 Testbench thiết kế Receiver

- Clock: Tạo xung với chu kỳ 20ns (tương ứng với tốc độ gấp 16 lần baud rate UART).
- Task send_uart_frame: Mô phỏng việc gửi khung dữ liệu UART với start, data, parity (nếu bật), và stop bit.
- Test case:
    - Gửi byte 0xA5 với even parity
    - Sau khi khung được nhận xong, assert tín hiệu rx_read để đọc dữ liệu từ FIFO.



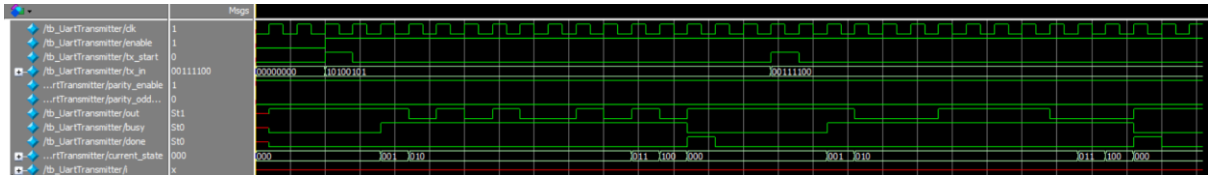
Hình 4.2 Kết quả mô phỏng uart_rx

```
`timescale 1ns / 1ps

module tb_UartTransmitter;

    reg clk;
    reg enable;
    reg tx_start;
    reg [7:0] tx_in;
    reg parity_enable;
    reg parity_odd_even;

    wire out;
    wire busy;
    wire done;
    wire [2:0] current_state;

    // Instantiate the module
    UartTransmitter uut (
        .clk(clk),
        .enable(enable),
        .tx_start(tx_start),
        .tx_in(tx_in),
        .parity_enable(parity_enable),
        .parity_odd_even(parity_odd_even),
        .out(out),
        .busy(busy),
        .done(done),
        .current_state(current_state)
    );

    // Clock generator (baudtick rate)
    initial clk = 0;
    always #20 clk = ~clk; // 25MHz => 40ns period, simulates baud tick

    // Task để gửi một byte
    task transmit_byte;
        input [7:0] byte;
        begin
            wait (!busy); // đợi transmitter rảnh
            @(posedge clk);
            tx_in <= byte;
```

```
            tx_start <= 1;
            @(posedge clk);
            tx_start <= 0;
        end
    endtask

    integer i;

    initial begin
        // Init
        clk = 0;
        enable = 0;
        tx_start = 0;
        tx_in = 8'h00;
        parity_enable = 1;
        parity_odd_even = 0; // even parity

        // Bật module
        #100;
        enable = 1;

        // Gửi dữ liệu 0xA5 = 8'b10100101 (parity even)
        transmit_byte(8'hA5);

        // Đợi truyền xong
        wait (done);
        #100;

        // Gửi thêm một byte khác (0x3C)
        transmit_byte(8'h3C);
        wait (done);
        #100;

        $display("Transmission done.");
        $stop;
    end

endmodule
```

Hình 4.3 Testbench thiết kế Transmitter

- Clock chạy mỗi 40ns, tượng trưng cho baudtick (mỗi tick = 1 bit truyền)
- Dùng tx_start để ghi byte mới vào FIFO
- tx_in là byte cần truyền.
- Mỗi lần byte được ghi vào FIFO, module sẽ tự động lấy ra và truyền đi tuần tự: start bit → 8 data bits → parity bit (nếu bật) → stop bit.



Hình 4.4 Kết quả mô phỏng uart_tx

```
`timescale 1ns / 1ps

module BaudRateGenerator_tb;

    reg clk;
    reg reset_n;
    reg [1:0] baud_select;
    wire tx_clk;
    wire rx_clk;

    // Instantiate DUT
    BaudRateGenerator uut (
        .clk(clk),
        .reset_n(reset_n),
        .baud_select(baud_select),
        .tx_clk(tx_clk),
        .rx_clk(rx_clk)
    );

    // Generate 50 MHz clock (20 ns period)
    initial clk = 0;
    always #10 clk = ~clk;

    // Monitor các xung đầu ra
    integer tx_edge_count = 0;
    integer rx_edge_count = 0;

    reg tx_clk_prev, rx_clk_prev;

    initial begin
        // Initialize
        reset_n = 0;
        baud_select = 2'b00; // 9600
        #100;
        reset_n = 1;

        // Theo dõi trong 5ms mỗi chế độ baud_select
        repeat (4) begin
            tx_edge_count = 0;
            rx_edge_count = 0;

            $display("Testing baud_select = %b", baud_select);
            #5000_000; // 5ms at 1ns resolution

            $display("tx_clk toggles: %0d", tx_edge_count);
            $display("rx_clk toggles: %0d", rx_edge_count);

            baud_select = baud_select + 1;
            #1000_000; // Wait 1ms before next test
        end

        $display("Test completed.");
        $stop;
    end

    // Đếm số lần tx_clk và rx_clk đổi trạng thái
    always @(posedge clk) begin
        tx_clk_prev <= tx_clk;
        rx_clk_prev <= rx_clk;

        if (tx_clk_prev != tx_clk) tx_edge_count <= tx_edge_count + 1;
        if (rx_clk_prev != rx_clk) rx_edge_count <= rx_edge_count + 1;
    end

endmodule
```

Hình 4.5 Testbench thiết kế BaudRate

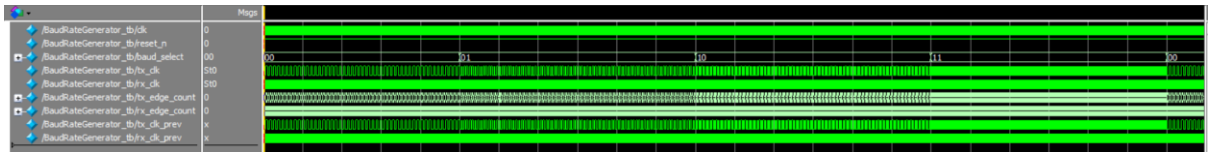
Trong mỗi giai đoạn:

- **tx_clk phải dao động với tần số tương ứng baudrate.**
- **rx_clk dao động gấp 16 lần tx_clk.**
- **Thấy số lần toggles trong 5ms giúp xác minh tần số:**

4.3. **Ví dụ cho 9600 baud, trong 5ms:**

4.3.1. **tx_clk toggle ~ 48 lần → 1 chu kỳ ≈ 104.1µs (2 toggles = 1 chu kỳ) → ~9600Hz.**

4.3.2. **rx_clk toggle ~ 768 lần → tần số ~153.6kHz (9600 × 16).**



```
SIM 10> run -all
# Testing baud_select = 00
# tx_clk toggles: 95
# rx_clk toggles: 1524
# Testing baud_select = 01
# tx_clk toggles: 144
# rx_clk toggles: 2272
# Testing baud_select = 10
# tx_clk toggles: 192
# rx_clk toggles: 3012
# Testing baud_select = 11
# tx_clk toggles: 1147
# rx_clk toggles: 17857
# Test completed.
```

Hình 4.6 Kết quả mô phỏng BaudRate

V. Thực nghiệm trên KIT DE2

5.1. Giới thiệu KIT DE2

Kit DE2 là một bo mạch phát triển FPGA do hãng **Terasic** sản xuất, sử dụng chip FPGA **Cyclone II EP2C35F672C6** của hãng **Intel (trước đây là Altera)**. Đây là một nền tảng phổ biến trong giảng dạy và nghiên cứu về thiết kế hệ thống số, lập trình phần cứng (HDL), xử lý tín hiệu số và các giao tiếp ngoại vi.

Các thành phần chính trên KIT DE2:

- **FPGA:** Cyclone II EP2C35F672C6 – 33.216 LEs (Logic Elements)
- **Bộ nhớ:**
  - SDRAM 8MB
  - SRAM 512KB
  - Flash 4MB
- **Ngoại vi vào/ra:**
  - 18 công tắc gạt (SW)
  - 4 nút nhấn (KEY)
  - 18 đèn LED đỏ (LEDR)
  - 8 LED 7 đoạn (HEX0–HEX7)
  - LCD 16x2 ký tự
  - VGA, PS/2, Audio Codec
  - 2 cổng UART RS232
  - 2 cổng USB (Host & Device)
  - Cổng Ethernet (LAN)

Khả năng kết nối với UART:

KIT DE2 có sẵn **cổng RS-232** để giao tiếp UART truyền thống với máy tính hoặc các thiết bị khác. Giao tiếp này có thể được kết nối trực tiếp với các module UART được thiết kế trên FPGA thông qua các chân **GPIO hoặc chân UART tích hợp**.

Ứng dụng trong đề tài:

Trong đề tài này, KIT DE2 được sử dụng để:

- Nạp và kiểm thử hệ thống UART truyền và nhận.
- Hiển thị dữ liệu nhận được lên LED 7 đoạn hoặc LCD.
- Giao tiếp với máy tính thông qua phần mềm mô phỏng cổng COM như Tera Term hoặc RealTerm.
- Kiểm tra chức năng của các tín hiệu điều khiển như parity, FIFO, lỗi truyền nhận.

5.2. Kết nối máy tính với KIT (qua UART/USB)

Tín hiệu	I/O	Mô tả chức năng
clk	In	Clock hệ thống 50 MHz
reset_n	In	Reset toàn bộ hệ thống (tích cực mức thấp) (KEY[0])
tx	Out	Tín hiệu UART truyền từ FPGA đến máy tính
rx_in	In	Tín hiệu UART từ máy tính gửi đến FPGA
tx_start	In	Bắt đầu truyền dữ liệu (1 xung mức cao) KEY[2]
tx_data_in	In	Dữ liệu 8-bit cần truyền (SW[7:0])
rx_data	Out	Dữ liệu 8-bit nhận được Hiển thị bằng LED 7 đoạn và LEDR[7:0]
baud_select	In	Lựa chọn tốc độ truyền (SW[15:14])
parity_enable	In	Bật/Tắt chế độ parity (SW[17])

parity_odd_even	In	Chọn parity: Even (0), Odd (1) (SW[16])
rx_read	In	Đọc dữ liệu từ FIFO khi có dữ liệu (KEY1)
HEX0, HEX1	Out	Hiển thị dữ liệu truyền/nhận dạng số thập lục phân
busy	Out	Báo hiệu đang truyền (trạng thái của UartTransmitter)(LEDG6 nếu cần)
done	Out	Báo hiệu truyền xong một frame dữ liệu (LEDG7 nếu cần)
fifo_empty	Out	Báo FIFO rỗng (không có dữ liệu nhận)(LEDG0)
fifo_full	Out	Báo FIFO đầy (không thể nhận thêm dữ liệu)(LEDG1)
error	Out	Báo lỗi parity hoặc lỗi giao tiếp (LEDG[2])
current_state	Out	Trạng thái FSM của UartReceiver để debug

Bảng 5.1 Tín hiệu Kết nối máy tính với KIT (qua UART/USB)

5.3. Mô phỏng thực tế và quan sát kết quả trên KIT và terminal (Putty)

Link video: https://drive.google.com/file/d/1qyV_4H-sbSfBDFI28mMMRn243OtQ3O-t/view?usp=drive_link

5.4. Nhận xét kết quả

Sau quá trình thiết kế, mô phỏng và thực nghiệm hệ thống UART trên KIT DE2, nhóm nhận thấy hệ thống hoạt động **ổn định và chính xác**, đáp ứng đầy đủ các yêu cầu đặt ra ban đầu. Một số nhận xét cụ thể như sau:

1. Về chức năng truyền và nhận:

- **UART Transmitter** hoạt động đúng theo thiết kế FSM: truyền đầy đủ các bit START, DATA, PARITY (nếu bật), và STOP.
- **UART Receiver** nhận dữ liệu chính xác, kiểm tra được bit parity, bit stop và phát hiện lỗi hiệu quả.
- Dữ liệu được **lưu trữ vào FIFO** trước khi truyền và sau khi nhận, đảm bảo không mất dữ liệu trong quá trình xử lý.

2. Về khả năng cấu hình:

- Hệ thống hỗ trợ **bật/tắt kiểm tra parity** và chọn **parity chẵn/lẻ**, giúp tăng tính linh hoạt.
- Có thể cấu hình số bit dừng, tốc độ baud thông qua module BaudRateGenerator (nếu được tích hợp).

3. Về giao tiếp với máy tính:

- Khi kết nối KIT DE2 với máy tính qua cổng RS232, dữ liệu truyền/nhận được hiển thị đúng trên phần mềm terminal (Tera Term).
- Khi gửi từ máy tính xuống KIT, dữ liệu được hiển thị chính xác trên **LED 7 đoạn HEX** hoặc LCD tùy theo cấu hình phần cứng.

4. Về mô phỏng và kiểm thử:

Quá trình mô phỏng trên ModelSim cho thấy các tín hiệu truyền nhận đều tuân thủ thời gian và định dạng UART chuẩn.

Các testbench viết cho từng module đều cho kết quả chính xác, không phát hiện lỗi logic trong thiết kế.

VI. Kết luận và hướng phát triển

6.1. Tóm tắt mục tiêu và phạm vi dự án

Đồ án "Giao tiếp dữ liệu song phương UART giữa FPGA và máy tính trên kit DE2" được thực hiện với mục tiêu xây dựng một hệ thống truyền nhận dữ liệu số hiệu quả, ổn định, hỗ trợ đa tốc độ baud phổ biến (9600, 14400, 19200, 115200 bps) và tính năng kiểm tra parity chẵn/lẻ nhằm nâng cao độ tin cậy trong giao tiếp. Hệ thống được thiết kế theo kiến trúc module hóa, dễ dàng tích hợp và mở rộng cho nhiều ứng dụng thực tế trong lĩnh vực nhúng và điều khiển.

6.2. Đánh giá tổng quan và điểm mạnh của giải pháp

Hai mô-đun top UartTopTransmitter và UartTopReceiver đã được thiết kế và kiểm chứng hoạt động chính xác, tương thích với các chuẩn UART thông dụng. Việc tạo clock baudrate riêng biệt cho truyền và nhận, cùng với khả năng lựa chọn tốc độ và chế độ parity linh hoạt, giúp hệ thống đáp ứng đa dạng yêu cầu ứng dụng. Màn hình LED 7 đoạn tích hợp trực tiếp trên kit DE2 hỗ trợ giám sát dữ liệu truyền nhận theo thời gian thực, tăng tính minh bạch và thuận tiện cho quá trình thử nghiệm, vận hành.

6.3. Hiệu suất và kết quả thực nghiệm

Trong quá trình thử nghiệm với phần mềm mô phỏng PuTTY trên máy tính, hệ thống thể hiện hiệu năng truyền nhận dữ liệu ổn định với tỷ lệ lỗi truyền thấp, đảm bảo tính chính xác và liên tục trong giao tiếp. Bộ đệm FIFO tích hợp ở module nhận có khả năng lưu trữ dữ liệu tạm thời, giúp giảm thiểu mất mát dữ liệu khi tốc độ xử lý giữa các thành phần không đồng bộ. Các tín hiệu trạng thái như busy, done (truyền) và fifo_empty, fifo_full, error (nhận) được báo cáo chính xác, hỗ trợ giám sát trạng thái hoạt động và xử lý lỗi kịp thời.

6.4. Những hạn chế, lỗi gặp phải và biện pháp khắc phục

- **Dao động cơ học của nút nhấn (chattering):** Tín hiệu đầu vào từ nút nhấn bị nhiễu và gây sai lệch trạng thái bắt đầu truyền hoặc reset module. Nhóm đã áp dụng phương pháp debounce bằng bộ đếm thời gian nhằm ổn định tín hiệu, từ đó nâng cao độ tin cậy trong việc điều khiển luồng dữ liệu.
- **Đọc dữ liệu FIFO liên tục gây in lặp:** Khi tín hiệu đọc dữ liệu giữ lâu, dữ liệu được in ra nhiều lần không mong muốn. Để khắc phục, nhóm đã thiết kế tín hiệu rx_read thành

xung ngắn và kết hợp kiểm tra trạng thái fifo_empty để chỉ đọc khi có dữ liệu, tránh in thừa và giảm tải xử lý.

- **Đồng bộ clock và xử lý lỗi parity, framing:** Nhóm đã sử dụng clock baudrate chính xác với tần số cao (16x baudrate) để đảm bảo đồng bộ và giảm sai số. Việc bật/tắt kiểm tra parity theo yêu cầu giúp hệ thống linh hoạt, đồng thời tín hiệu báo lỗi được tích hợp để dễ dàng phát hiện và xử lý các trường hợp lỗi truyền nhận.

6.5. Phương hướng cải tiến và phát triển mở rộng trong tương lai

Để hoàn thiện và nâng cao hệ thống, một số hướng phát triển sau đây được đề xuất:

- **Hỗ trợ đa kênh UART:** Thiết kế giao tiếp đồng thời nhiều kênh truyền nhận giúp mở rộng phạm vi ứng dụng trong hệ thống đa thiết bị.
- **Bộ đệm FIFO cấu hình động:** Cho phép tùy chỉnh dung lượng bộ đệm phù hợp với từng ứng dụng, tối ưu tài nguyên phần cứng.
- **Hỗ trợ các chuẩn dữ liệu đa dạng:** Thêm các tùy chọn dữ liệu 7-bit, 9-bit, cấu hình số bit dừng khác nhau để tăng tính tương thích.
- **Kiểm soát lỗi nâng cao:** Tích hợp các phương pháp phát hiện và sửa lỗi tự động, cảnh báo người dùng ngay lập tức khi xảy ra sự cố.
- **Giao diện giám sát hiện đại:** Phát triển giao diện LCD hoặc OLED, hoặc giao diện máy tính thân thiện để theo dõi trạng thái truyền nhận trực quan và chi tiết hơn.
- **Giao tiếp không dây:** Mở rộng sang các giao thức không dây như Bluetooth hoặc USB-UART để tăng tính linh hoạt trong ứng dụng thực tế.
- **Phần mềm điều khiển đa năng:** Phát triển phần mềm quản lý giao tiếp UART có giao diện đồ họa giúp người dùng dễ dàng điều khiển và giám sát hệ thống.

6.6. Ý nghĩa thực tiễn và bài học kinh nghiệm thu được

Hệ thống giao tiếp UART song phương đã chứng minh được tính khả thi và hiệu quả trong việc truyền nhận dữ liệu giữa FPGA và máy tính, đóng góp quan trọng vào các ứng dụng nhúng và công nghiệp tự động hóa. Qua quá trình thực hiện, nhóm đã tích lũy được nhiều kinh nghiệm quý giá về thiết kế FPGA, xử lý tín hiệu số, đồng bộ clock, lập trình Verilog, cũng như kỹ năng giải quyết các vấn đề thực tế liên quan đến phần cứng và phần mềm. Đây là nền tảng vững chắc cho các dự án phát triển tiếp theo trong lĩnh vực kỹ thuật số và hệ thống nhúng.

VII. Tài liệu tham khảo

[1] Xilinx, “UART Communication IP Core User Guide,” 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/uart.pdf

[2] M. Barr, *Programming Embedded Systems: With C and GNU Development Tools*, Sebastopol, CA: O'Reilly Media, 2006.

[3] Altera Corporation, “UART Reference Design for FPGA,” 2018. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_uart.pdf

[4] J. Bhasker, *A Verilog HDL Primer*, Star Galaxy Publishing, 1999.

[5] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Francisco, CA: Morgan Kaufmann, 2013.

[6] FPGA4student, “FPGA UART Tutorial,” 2023. [Online]. Available: https://www.fpga4student.com/2017/08/uart-verilog-code.html

[7] FPGA Developer, “UART Communication Using FPGA,” 2023. [Online]. Available: https://www.fpgadeveloper.com/2019/05/uart-communication-using-fpga.html