|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  **SCHOOL OF ELECTRICAL AND ELECTRONIC**  logo_128  **GRADUATION PROJECT**  **Subject:**  **APB PROTOCOL DESIGN AND SIMULATION USING UART COMMUNICATION STANDARD BY VERILOG**   |  |  | | --- | --- | | Student: | Pham Hoang Long | | Instructor: | Dr. Nguyen Anh Quang |   Hanoi, 2-2024 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  **SCHOOL OF ELECTRICAL AND ELECTRONIC**  logo_128  **GRADUATION PROJECT**  **Subject:**  **APB PROTOCOL DESIGN AND SIMULATION USING UART COMMUNICATION STANDARD BY VERILOG**   |  |  | | --- | --- | | Student: | Pham Hoang Long | | Instructor: | Dr. Nguyen Anh Quang | | Counter-argument teacher: |  |   s  Hanoi, 2-2024 |

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

**SCHOOL OF ELECTRICAL AND ELECTRONIC**

**ASSESSMENT OF THE GRADUATION PROJECT**

**(FOR COUNTER-ARGUMENT TEACHER)**

Name of project: APB protocol design and simulation using UART communication standard by verilog

Name of student: Pham Hoang Long Student ID: 20182628

Name of teacher:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Criteria**  (Maximum point) | **Criteria Evaluation Guide** | **Criteria Point** |
| 1 | **Report’s technical writing skill (4 points)** | Present correctly the prescribed form, logical and reasonable layout of chapters: Table, clear images, with titles, numbered sequentially and explained or mentioned in the project, with margins, makings spaces after periods, commas, chapter introduction and chapter conclusion, list of references and citations, etc. |  |
| Expression, analysis, explanation, reasoning skills: Clear sentence structure, scientific style, logical and well-founded arguments, appropriate specialized terminology, etc. |
| 2 | **Content and result**  **(5.5 points)** | Clearly state the urgency, scientific and practical significance of the topic, problem and hypotheses, and scope of application of the topic. Fully implement the research process: problem setting, set objectives, research method/problem solving, achieve results, evaluation and conclusion. |  |
| The content and result are presented in a logical and logical manner, analyzed and evaluated satisfactorily. Arguing and analyzing simulation/software/experimental result, comparing the obtained results with relevant previous results. |
| Indicate the fit between the results achieved and the initial objectives set and provide arguments to suggest possible solutions in the future. High scientific content/complexity, novelty/creativity in project content and results. |
| 3 | **Achievement’s**  **point (1 point)** | Having scientific articles published or accepted for publication/winning the prize of scientific research student of the 3rd prize at the university level/ National and international scientific prizes of 3rd prize or higher/ Having registered a patent. **(1 point)** |  |
| Reported at the University Council in the conference of Scientific Research Student but did not win the 3rd prize or higher high performance, requires a large amount of execution. **(0.5 points)** |
|  | | **Total point of above criteria:** |  |
| **Counter argument’s point:** |  |

|  |  |
| --- | --- |
|  | **Counter-argument teacher**  (Sign and write full name) |

**ABSTRACT**

The Universal Asynchronous Receiver-Transmitter (UART) is a fundamental component in modern digital communication systems, facilitating the serial transmission of data between devices. This project focuses on the design and implementation of a UART communication protocol within a Very Large Scale Integration (VLSI) framework.

The project begins with a comprehensive study of UART specifications, protocols, and functionalities. Utilizing Verilog, the UART architecture is modeled and synthesized to fit within the constraints of VLSI design, including area, power, and timing considerations.

Key components of the design include baud rate generation, data framing, error detection, and flow control mechanisms. Emphasis is placed on optimizing the design for efficient data transfer rates while minimizing resource utilization.

The project also involves simulation and verification stages to ensure the correctness and robustness of the UART implementation. Advanced verification techniques such as formal verification and hardware emulation may be employed to validate the design against various test scenarios and corner cases.

**CONTENTS**

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT 1](#_Toc129391768)

[DANH MỤC HÌNH VẼ 2](#_Toc129391769)

[DANH MỤC BẢNG BIỂU 4](#_Toc129391770)

[TÓM TẮT ĐỒ ÁN 5](#_Toc129391771)

[CHƯƠNG 1. QUY TRÌNH THIẾT KẾ ASIC 7](#_Toc129391772)

[1.1 ASIC và quy trình thiết kế ASIC 7](#_Toc129391773)

[1.1.1 Phân tích thiết kế (Design Analyzation) 8](#_Toc129391774)

[1.1.2 Kiểm tra và sửa lỗi (Verification and Debug) 8](#_Toc129391775)

[1.1.3 Thiết kế vật lý (Physical Design) 10](#_Toc129391776)

[1.1.4 Chế tạo sản phẩm 10](#_Toc129391777)

[1.2 Kết luận chương 11](#_Toc129391778)

[CHƯƠNG 2. GIỚI THIỆU LÕI IP AHB TO AXI 12](#_Toc129391779)

[2.1 Chuẩn giao tiếp AHB 12](#_Toc129391780)

[2.1.1 Giới thiệu chung về chuẩn giao tiếp AHB 12](#_Toc129391781)

[2.1.2 Giao dịch ghi – write transfers 13](#_Toc129391782)

[2.1.3 Giao dịch đọc – read transfers 14](#_Toc129391783)

[2.2 Chuẩn giao tiếp AXI 16](#_Toc129391784)

[2.2.1 Giới thiệu chung về chuẩn giao tiếp AXI4 16](#_Toc129391785)

[2.2.2 Cơ chế bắt tay 17](#_Toc129391786)

[2.3 Giới thiệu lõi IP AHB to AXI 18](#_Toc129391787)

[2.3.1 Đặc tính kỹ thuật và chức năng chính của lõi IP AHB to AXI 19](#_Toc129391788)

[2.3.2 Thanh ghi cấu hình 20](#_Toc129391789)

[2.4 Kết luận chương 21](#_Toc129391790)

[CHƯƠNG 3. THIẾT KẾ AHB VERIFICATION IP 22](#_Toc129391791)

[3.1 Môi trường kiểm thử trên SystemVerilog 22](#_Toc129391792)

[3.1.1 Vai trò của môi trường kiểm thử 22](#_Toc129391793)

[3.1.2 Các thành phần của SystemVerilog Testbench 23](#_Toc129391794)

[3.2 Môi trường kiểm thử UVM 24](#_Toc129391795)

[3.2.1 Sự cần thiết của môi trường kiểm thử UVM 24](#_Toc129391796)

[3.2.2 Lịch sử phát triển 24](#_Toc129391797)

[3.2.3 Đặc điểm cơ bản của UVM 25](#_Toc129391798)

[3.2.4 Kiến trúc môi trường kiểm thử UVM 27](#_Toc129391799)

[3.3 Thiết kế AHB Verification IP 28](#_Toc129391800)

[3.3.1 Kế hoạch kiểm thử - Verification plan 29](#_Toc129391801)

[3.3.2 Xây dựng khung kiến trúc VIP 31](#_Toc129391802)

[3.3.3 Xây dựng kiến trúc VIP 31](#_Toc129391803)

[3.3.4 Xây dựng các thành phần trong VIP 35](#_Toc129391804)

[3.4 Kết luận chương 41](#_Toc129391805)

[CHƯƠNG 4. KIỂM THỬ LÕI IP AHB TO AXI 42](#_Toc129391806)

[4.1 Giới thiệu phần mềm QuestaSim 42](#_Toc129391807)

[4.2 Kiến trúc testbench 44](#_Toc129391808)

[4.3 Xây dựng testcase 46](#_Toc129391809)

[4.4 Xây dựng testplan 47](#_Toc129391810)

[4.5 Kết luận chương 49](#_Toc129391811)

[CHƯƠNG 5. KẾT QUẢ VÀ ĐÁNH GIÁ 50](#_Toc129391812)

[5.1 Kết quả tổng hợp và mô phỏng VIP 50](#_Toc129391813)

[5.1.1 Tổng hợp môi trường 50](#_Toc129391814)

[5.1.2 Kết quả và đánh giá quá trình truyền dữ liệu của driver 52](#_Toc129391815)

[5.1.3 Đánh giá waveform 52](#_Toc129391816)

[5.1.4 Kết quả và đánh giá memory model trong scoreboard 54](#_Toc129391817)

[5.2 Kết quả kiểm thử lõi IP AHB to AXI 54](#_Toc129391818)

[5.2.1 Kết quả sau khi chạy mô phỏng các testcase 55](#_Toc129391819)

[5.2.2 Đánh giá functional coverage và SVA 56](#_Toc129391820)

[5.2.3 Đánh giá độ bao phủ code – Code coverage 58](#_Toc129391821)

[5.3 Đánh giá quá trình kiểm thử 59](#_Toc129391822)

[5.4 Kết luận chương 60](#_Toc129391823)

[KẾT LUẬN 61](#_Toc129391824)

[Kết luận chung 61](#_Toc129391825)

[Hướng phát triển 61](#_Toc129391826)

[TÀI LIỆU THAM KHẢO 62](#_Toc129391827)

[PHỤ LỤC 63](#_Toc129391828)

[Phụ lục 1. Testplan lõi IP AHB to AXI 63](#_Toc129391829)

# LIST OF SIGNS AND ABBREVIATIONS

|  |  |
| --- | --- |
| **APB** | Advanced High-performance Bus |
| **UART** | Universal asynchronous receiver-transmitter |
| **IP** | Intellectual Property |
| **ASIC** | Application Specific Integrated Circuit |

# LIST OF FIGURES

[Figure 1.1 Data Framing UART 6](#_Toc159335080)

[Figure 1.2 Some applications of UART 7](#_Toc159335081)

[Figure 1.3 Combined circuit diagram 7](#_Toc159335082)

[Figure 1.4 Series circuit diagram 8](#_Toc159335083)

[Figure 1.5 Moore and Mealy state transition diagram 10](#_Toc159335084)

[Figure 1.6 Migration activity of a single node in ASM 11](#_Toc159335085)

[Figure 1.7 Description of an ASM block 11](#_Toc159335086)

[Figure 1.8 Example state transition diagram 12](#_Toc159335087)

[Figure 1.9 FSMD blocks 14](#_Toc159335088)

[Figure 1.10 A state in FSMD 15](#_Toc159335089)

[Figure 1.11 APB signal descriptions 16](#_Toc159335090)

[Figure 1.12 APB signal descriptions 17](#_Toc159335091)

[Figure 2.1 UART data framing 18](#_Toc159335092)

[Figure 2.2 APB UART top module 19](#_Toc159335093)

[Figure 2.3 APB UART block diagram 19](#_Toc159335094)

[Figure 2.4 UART FIFO block diagram 21](#_Toc159335095)

[Figure 2.5 Operating mechanism of FIFO 22](#_Toc159335096)

[Figure 2.6 UART transmiter block diagram 23](#_Toc159335097)

[Figure 2.7 UART transmiter FSMD 26](#_Toc159335098)

[Figure 2.8 UART receiver block diagram 27](#_Toc159335099)

[Figure 2.9 UART receiver FSMD 30](#_Toc159335100)

[Figure 2.10 Register block block diagram 31](#_Toc159335101)

# LIST OF TABLES

[Table 2.1 Signal description APB UART 20](#_Toc159335102)

[Table 2.2 UART FIFO signal description 21](#_Toc159335103)

[Table 2.3 UART transmiter signal description 24](#_Toc159335104)

[Table 2.4 Register summary 32](#_Toc159335105)

[Table 2.5 rbr field descriptions 32](#_Toc159335106)

[Table 2.6 thr field descriptions 33](#_Toc159335107)

[Table 2.7 ier field descriptions 33](#_Toc159335108)

[Table 2.8 iir field descriptions 34](#_Toc159335109)

[Table 2.9 fcr field descriptions 35](#_Toc159335110)

[Table 2.10 lcr field descriptions 36](#_Toc159335111)

[Table 2.11 mcr field descriptions 37](#_Toc159335112)

[Table 2.12 lsr field descriptions 38](#_Toc159335113)

[Table 2.13 msr field descriptions 40](#_Toc159335114)

[Table 2.14 scr field descriptions 41](#_Toc159335115)

[Table 2.15 dll field descriptions 41](#_Toc159335116)

[Table 2.16 dlh field descriptions 41](#_Toc159335117)

[Table 2.17 mdr field descriptions 42](#_Toc159335118)

[Table 2.18 stt field descriptions 42](#_Toc159335119)

# SPECIFICATION

Input: Data frames, each data frame contains 8 bits of data Output: Data frames received from the transmitter

Operation description: The transmitter side receives 8 bits of data from the buffer and creates 1 parity bit to send to the receiver side. The receiving side receives 8 data bits and 1 parity bit from the transmitter and also creates a parity bit from those 8 received data bits, then compares the generated parity bit with the parity bit from the transmitter. sent. If those 2 bits are the same, push the received data to the buffer; otherwise, do not push the data to the buffer.

Technical specifications:

• Baud rate: 19200 bit/s

• Number of bits in 1 data frame: 8

• Parity bit: 1 parity bit

• Stop bit: 1 (Transmission time 16 x baudrate)

• Transmission time of 1 bit: 16 x baudrate

The following figure depicts the transmission diagram of a data frame from the transmitter to the receiver: The order of transmission is 8 data bits followed by parity bits.

# PROJECT SUMMARY

In this project report, I divide the report content into 3 chapters:

Chapter 1: I present the UART communication standard, UART transmission structure and UART applications in electronics. In addition, I also present how to design some electronic circuits, FSM, FSMD using verilog to serve the process of implementing the UART standard using verilog in chapter 2.

Chapter 2: I present the specs of the UART standard that I implemented and designed. The block diagram design project of the UART set includes 4 modules: Receiver, transmitter, baud rate generator and buffer block. In each module there is a detailed description of how the module operates, detailed design and signal table to implement the verilog programming process.

Chapter 3: I present the results of the UART simulation that I performed.

# GENERAL INTRODUCTION

In this chapter, I introduce the UART communication standard, applications of UART and theoretical basis, design methods for combinational circuits, sequence circuits, state machines (FSM), state machines with data lines (FSMD) using verilog.

## Introduction to the UART communication standard

### General introduction

The UART (Universal Asynchronous Receiver and Transmitter) communication standard is an asynchronous serial communication protocol. In 1960, the electronics association ETA developed a communication standard, RS - 232, which was used for serial data communication. Besides RS - 232, there are also popular RS - 422 and RS - 485 standards applied to UART circuits. These standards offer much more practicality in communication than RS -232.

A data frame in UART consists of 1 start bit, 5-8 data bits, parity bit (optional) and stop bit. The stop bit can be 1, 1.5 or 2 bits long.

Data is transmitted bit by bit in frames, the first bit transmitted is the start bit (logic level 0), followed by data bits, possibly a parity bit, and finally the stop bit (logic level 1).

A row of black text

Description automatically generated

Figure 1.1 Data Framing UART

The image above depicts the data transmission structure of a UART transmitting 8 bits per data frame, without parity bits and using a stop bit.

### Application of UART

UART is one of the simplest and most commonly used Serial Communication techniques. UART appears in most microcontrollers for precision requirements. Nowadays, UART is being used in many applications like GPS Receivers, Bluetooth Modules, GSM and GPRS Modules, Wireless Communication Systems, RFID based applications, etc

A diagram of a microcontroller

Description automatically generated

Figure 1.2 Some applications of UART

## Design digital systems using verilog

### Combination circuit

#### General theory

A combinational circuit is a digital circuit without memory elements. Corresponding to each input of the circuit changing at a time, the combinational circuit will give the corresponding output. The drawing of the combinational circuit is shown below.

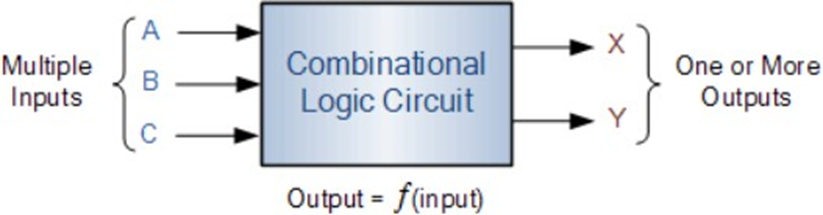


Figure 1.3 Combined circuit diagram

#### Structure of combinational circuit formation in Verilog

We consider a combinational circuit as a black box. This blackboard will react to each input. In verilog, we use the syntax always @(sensitivity list). The always syntax will create a black box of the combinational circuit. That black box will respond to input signals in the (sensitivity list). To describe the logical operation of a combinational circuit, we start and end the always block with the syntax: begin, end and describe the logical relationship of the output corresponding to the input in the begin, end paragraph.

In addition, during the circuit design process, to facilitate changing parameters, we use the syntax: localparam (signal name) = (Value you want to assign); Or #(parameter) (Global variable name) = (value you want to assign)) before the module declaration.

### Series circuit

Unlike combinational circuits, series circuits are memory circuits. The output of a sequence circuit depends on the input and the internal state of the circuit at that time. Sequence circuits are divided into three types: regular sequence circuits, state machines (FSM), and state machines with data streams (FSMD). The block diagram of the array circuit is shown below.

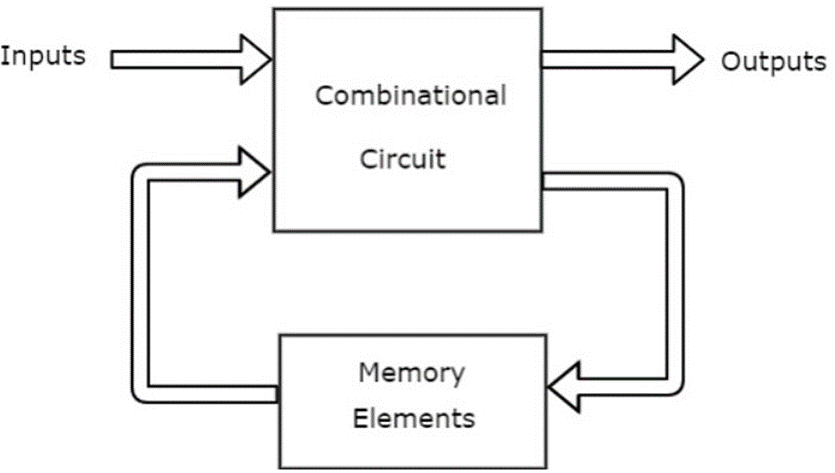


Figure 1.4 Series circuit diagram

In verilog programming, we will also divide it into parts as shown above: Part describing memory components (registers), part describing the next state performed via combinational circuit (next state logic). Details of the sequence circuit template are shown below.

Module ( khai báo đầu vào, đầu ra)

// Khai báo tín hiệu bên trong Reg a = …;

// Phân thân

// Khai báo phần tử nhớ

Always @(posedge (tín hiệu clk..))

Begin

….

end

// Khai báo trạng thái tiếp theo

Always @\* Begin

….

R\_next = …;

End

// logic đầu ra

….

endmodule

### Common series circuit

For regular series circuits, the state transition of the circuit has a "normal" rule, meaning that the next state of the circuit always has a repeating general rule such as adding up to 1, subtracting 1 or shifting by 1. bit, 2 bits... Ordinary sequence circuits can be seen through examples such as adders, translators... The code formation structure of ordinary sequence circuits is similar to the template in the above section.

### FSM

Unlike sequence circuits, normally, state machines (FSMs) have next states that do not follow any rules but must rely on the current state and inputs other than the clock pulse. The operation of the FSM is represented by a State Transition Diagram consisting of moore and mealy diagrams.

A diagram of a state

Description automatically generated A diagram of a state

Description automatically generated

Figure 1.5 Moore and Mealy state transition diagram

The picture above shows two circuit models designed according to the two state transition models Moore and Mealy. The moore diagram has state-dependent output. The mealy diagram depends on input and state.

To facilitate programming, people have proposed an algorithm diagram based on FSM called ASM state machine charts (Algorithmics state machine charts). This diagram captures input, output, states, and state transitions in a single representation. These two representations together provide a state transition operation, with the FSM being a more compact and simple representation, and the ASM a more descriptive and diagram-like representation of the transition conditions. and complex operations.

A state diagram consists of nodes representing states drawn as circles and transitions between those states. The figure below shows a single node and its shifting operation. The shifting operation is performed when the conditional expression evaluates to true.

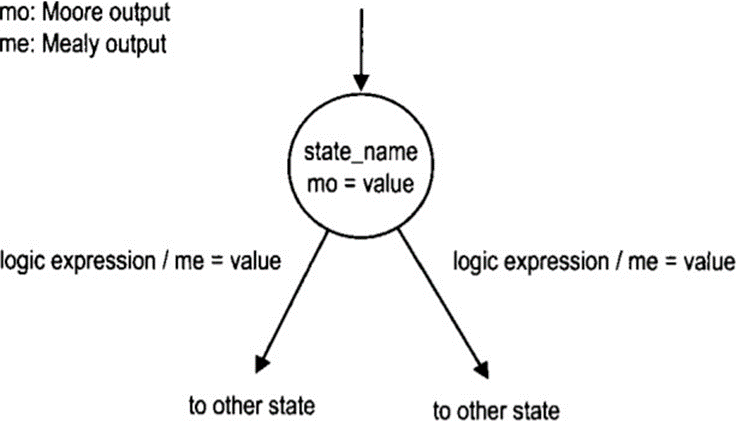


Figure 1.6 Migration activity of a single node in ASM

The output value of the Moore diagram is placed in a circle because it depends only on the current state. The output value of the Mealy diagram is tied to the condition of the shift operation because it depends on the current state and the input. To reduce clutter in the diagram, only confirmed output values are listed. The output signal may include other default (i.e. unassigned) values.

The ASM diagram consists of many ASM blocks. Each ASM block includes a status box, a condition check box and a conditional output signal determination box as shown in the figure below.

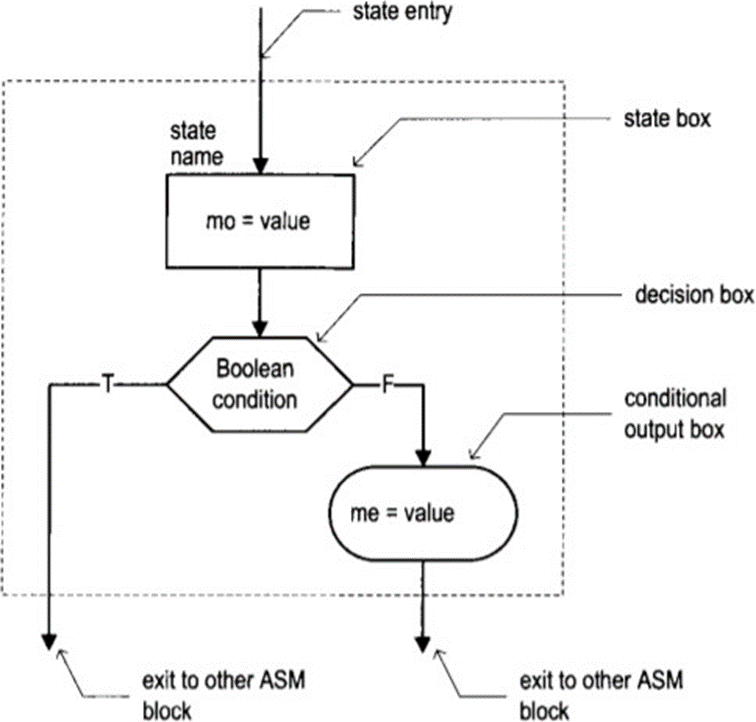


Figure 1.7 Description of an ASM block

A state box represents a state in the FSM and assigns outputs according to the Moore diagram inside the box and it has only one output. The condition checking block checks the input and determines the output afterwards. It will have 2 outputs, labeled T and F corresponding to the True and False values of the condition test expression. The output determination meeting is used to determine the output signal actions according to the Mealy diagram following the tested conditions. It represents an output signal that is activated only when the condition in the decision box is met.

In verilog programming, the FSM circuit has a program framework for execution. Suppose verilog programming is performed for the following state transition diagram.

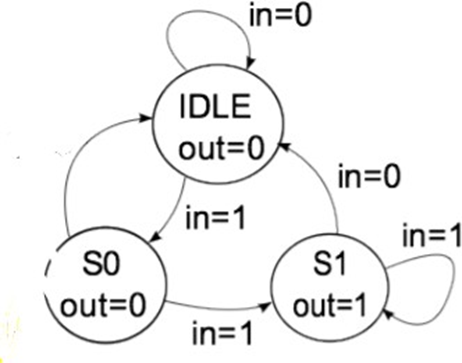


Figure 1.8 Example state transition diagram

The program framework is implemented as follows:

Module FSM1(clk, rst, in, out);

Input clk, rst;

Input in;

Output out;

// Khai báo trạng thái

ParameterIDLE=2’b00, S0=2’b01, S1=2’b10;

// Khai báo tín hiệu nội tại

Reg out;

Reg [1:0] present\_state, next\_state;

// Tệp thanh ghi

Always @(posedge clk)

If (rst)

present\_state <= IDLE;

Else

present\_state <= next\_state;

//Khối định nghĩa trạng thái tiếp theo

Always @(present\_state or in)

Case (present\_state)

//Định nghĩa trạng thái tiếp theo đối với từng input

IDLE: begin

Out = 1’b0;

If (in==1’b1) next\_state = S0;

Else next\_state= IDLE;

End

S0: begin

Out =1’b0;

If(in==1’’b1) next\_state=S1;

Else next\_state=IDLE;

End

S1: begin

Out=1’b1;

If(in==1’b1) next\_state =S1;

Else next\_state=IDLE;

End

Defautl: begin

Next\_state=IDLE;

Out=1’b0;

End

Endcase

Endmodule

### FSMD

FSMD is defined as a state machine (FSM) and a data flow (Datapath). The state machine is considered a control path. The control block will take control signals from the outside and internal status signals to produce data flow control signals, or in other words, common sequence circuits. FSMD is used to describe systems designed using the RT(register transfer) method. The RT system operates based on shifting and transforming data between registers. An image depicting FSMD is shown below.

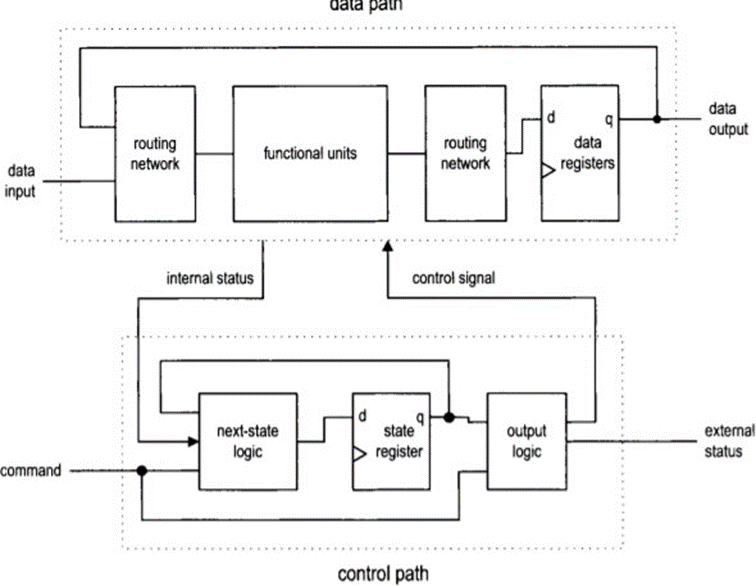


Figure 1.9 FSMD blocks

To design the UART communication standard, we need to do it through the ASMD diagram. The ASMD diagram, also known as the state machine and data flow algorithm diagram, describes how an FSMD block works. The image below depicts the structure of an ASMD diagram block.

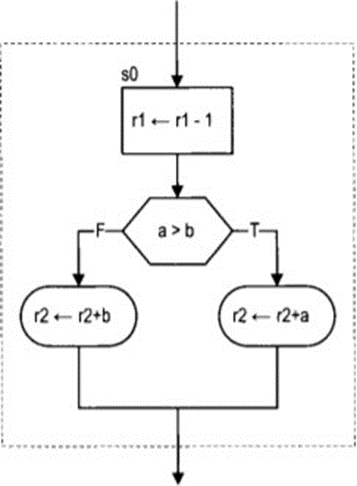


Figure 1.10 A state in FSMD

The dashed line in the block above represents a state. Each state specifies the operations that need to be performed between registers. The way to design this diagram is not much different from ASM diagrams or regular algorithm diagrams. The only difference is that the results of the operations are only updated when exiting that state, that is, at the next positive edge of the clock cycle.

### APB Protocol

About the APB protocol The APB protocol is a low-cost interface, optimized for minimal power consumption and reduced interface complexity. The APB interface is not pipelined and is a simple, synchronous protocol. Every transfer takes at least two cycles to complete. The APB interface is designed for accessing the programmable control registers of peripheral devices. APB peripherals are typically connected to the main memory system using an APB bridge. For example, a bridge from AXI to APB could be used to connect a number of APB peripherals to an AXI memory system. APB transfers are initiated by an APB bridge. APB bridges can also be referred to as a Requester. A peripheral interface responds to requests. APB peripherals can also be referred to as a Completer.

AMBA APB signals This section describes the APB interface signals:

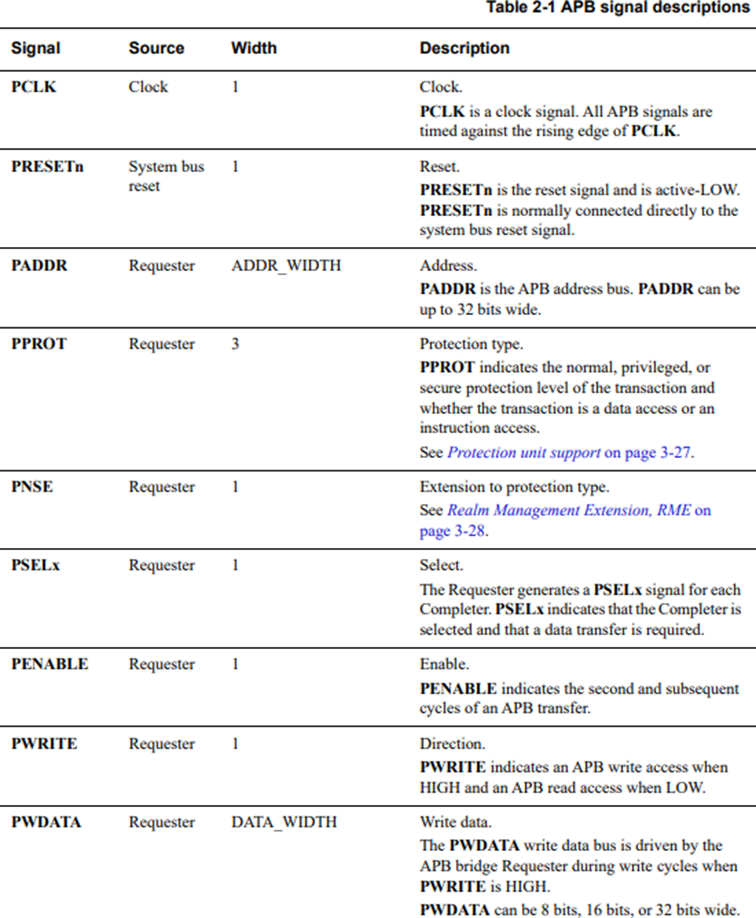


Figure 1.11 APB signal descriptions

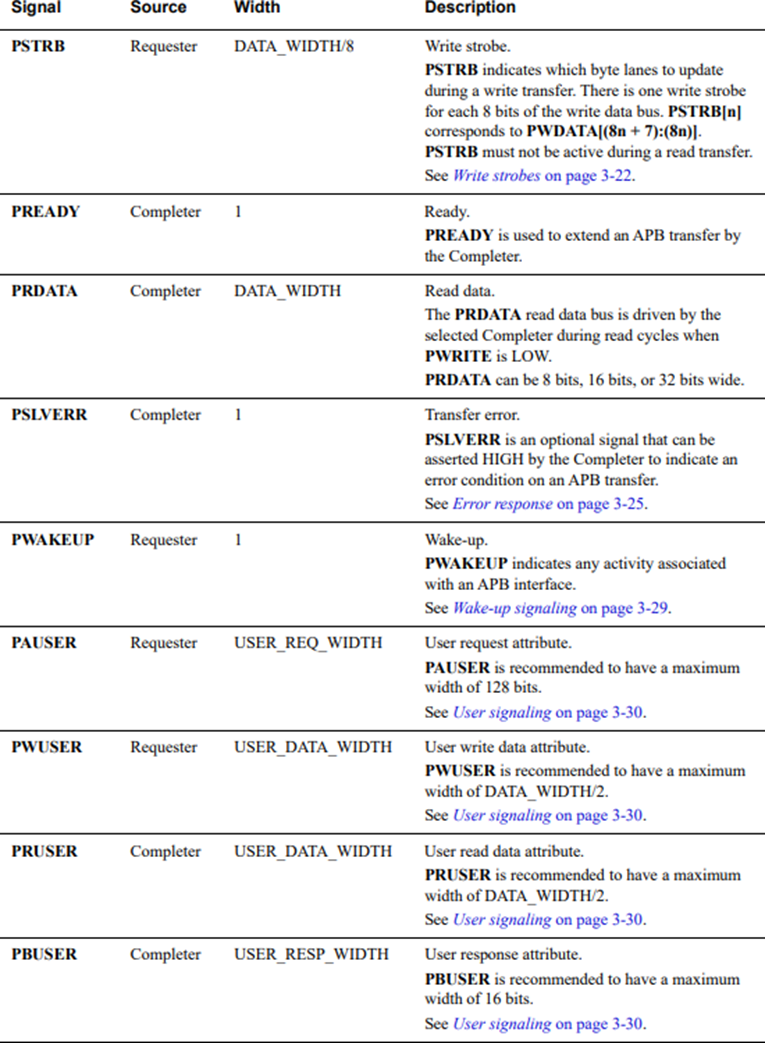


Figure 1.12 APB signal descriptions

# DESIGNING APB UART

In this chapter, I have designed the APB UART communication standard. Before proceeding with the design, I presented the specification and block diagram of the APB UART system I implemented and divided the block diagram into 6 modules for design. In each module there is a detailed description of how the module operates, detailed design and signal table to implement the verilog programming process.

## Technical requirements

* Input: Data frames, each data frame contains 8 bits of data
* Output: 8bit data frames received from the transmitter

Activity description:

The transmitter side receives 8 bits of data from the buffer and creates 1 parity bit to send to the receiver side. The receiving side receives 8 data bits and 1 parity bit from the transmitter and also creates a parity bit from those 8 received data bits, then compares the generated parity bit with the parity bit from the transmitter. sent. If those 2 bits are the same, push the received data to the buffer; otherwise, do not push the data to the buffer.

The figure below depicts the transmission diagram of a data frame from the transmitter to the receiver: The order of transmission is 8 data bits followed by parity bits.

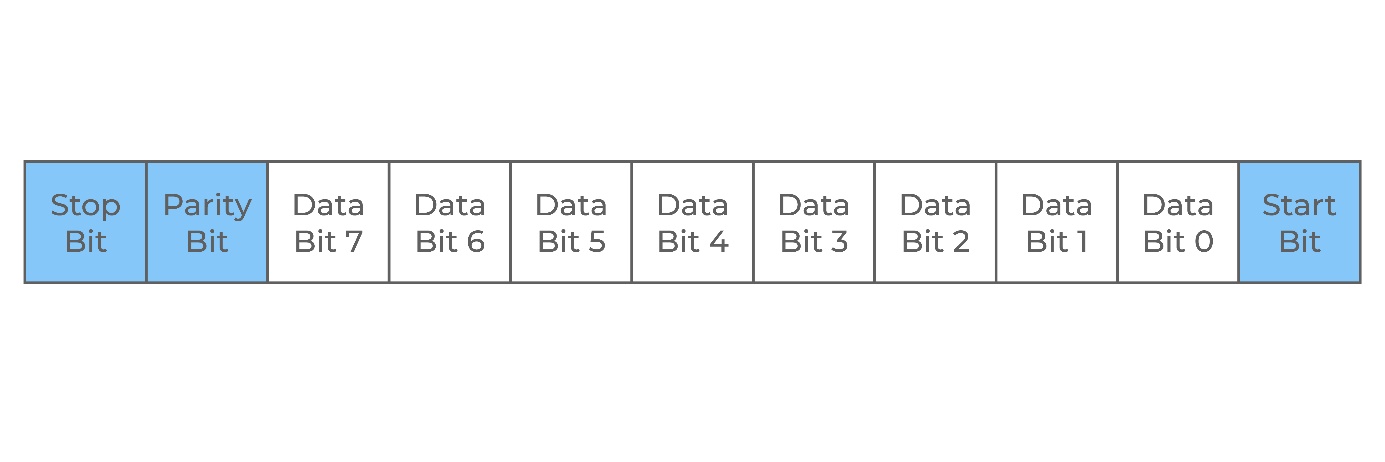


Figure 2.1 UART data framing

Technical specifications:

* Baud rate: 19200 bit/s
* Number of bits in 1 data frame: 5, 6, 7, 8
* Parity bit: 1 parity bit
* Stop bit: 1, 2 (Transmission time 16 x baudrate)
* Transmission time of 1 bit: 16x baudrate, 8x baudrate

## Design the APB UART system

### Top module

A white rectangular object with black text

Description automatically generated

Figure 2.2 APB UART top module

A black and white rectangular object with text

Description automatically generated

Figure 2.3 APB UART block diagram

From the block diagram above, I divided it into 4 modules to perform the design: Band rate block, transmitter block, receiver block, buffer block. The module design process is presented in the next section below.

### Signal description

Table 2.1 Signal description APB UART

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

After designing the block diagram and naming the signals used, we proceed with programming based on the parts just outlined. Details of the code will be placed in the appendix.

## Design the UART FIFO

### Block diagram

A white rectangular object with black text

Description automatically generated

Figure 2.4 UART FIFO block diagram

### Signal description

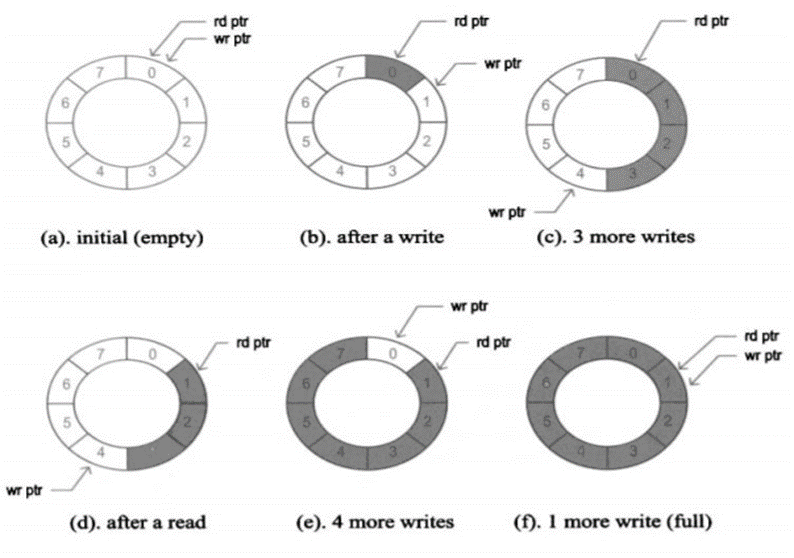
Table 2.2 UART FIFO signal description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| clk | 1 | Input | Clock |
| reset\_n | 1 | Input | Reset asynchronous active LOW |
| clr\_i | 1 | Input | Clear FIFO |
| ready\_i | 1 | Input | Data input ready |
| valid\_i | 1 | Input | Data input valid |
| data\_i | 8 | Input | Data input |
| ready\_o | 1 | Input | Data output ready |
| valid\_o | 1 | Input | Data output valid |
| elements\_o | 1 | Input | Number of elements in FIFO |
| data\_o | 8 | Output | Data output |
| empty | 1 | Output | FIFO empty |
| full | 1 | Output | FIFO full |
| error | 1 | Output | Error when write into full FIFO |

### Functional description

The buffer block has the function of temporarily storing received and sent data (data\_w) when the RX and TX units have not yet had time to read or send previously arrived data. To store data, the buffer block must have a register file. In addition, to perform read and write, we need a read signal (rd), a write signal (wr) and a pointer signal pointing to the address of the file you want to read or write. The block outputs a full signal (full) = 1 when the queue is full and an empty signal (empty) = 1 when the queue is empty and data is read (data\_r). Because the block contains memory elements, we need to use clk pulses and reset signals.

I will go to design the FIFO function of reading and writing in the buffer block. Because UART transmit and receive data is 8 bits, each register row will have 8 bits. Suppose that the queue length is 8 corresponding to 3 bit addresses numbered from 0 to 7 as shown.



A diagram of a diagram

Description automatically generated with medium confidence

Figure 2.5 Operating mechanism of FIFO

We call the read and write pointers rd ptr and wr ptr respectively. Initially the queue is empty, signal empty = 1. wr ptr and rd ptr will carry the address value at row 0 of the register file – Figure a. To perform writing, signal wr = 1, we will write the value at each positive edge of the clk cycle. After each write, the address of pointer wr ptr will be added by 1. Figures b and c depict the process of writing four values to the register file. This takes place over 4 clock cycles. To perform a read, let the signal rd = 1. The read will not depend on the clock cycle and the rd pointer will be added by 1 at each positive edge of the clock cycle. Figure d depicts this process. For the full output signal, it signals full when we set wr = 1, rd = 0 and wr ptr increases until it catches up with rd ptr as shown in figure f. For the empty output signal, it will be reported as empty when we set wr = 0, rd = 1 and rd ptr will increase until it catches up with wr ptr. In case we both read and write, the buffer will never be empty or full. The above process will be described by the following summary table. We define signal\_next as the value of that signal in the next positive edge of the clock cycle.

## Design the UART transmiter

### Block diagram

A white rectangular object with black text

Description automatically generated

Figure 2.6 UART transmiter block diagram

### Signal description

Table 2.3 UART transmiter signal description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| clk | 1 | Input | Clock |
| reset\_n | 1 | Input | Reset asynchronous active LOW |
| cfg\_en\_i | 1 | Input | Enable UART transmiter |
| cfg\_div\_i | 1 | Input | Division for baudrate generator |
| cfg\_parity\_en\_i | 1 | Input | Enable parity bit |
| cfg\_bits\_i | 1 | Input | Configuration number of data bits |
| cfg\_stop\_bits\_i | 1 | Input | Configuration number of stop bits |
| tx\_data\_i | 8 | Input | Data input |
| tx\_valid\_i | 1 | Input | Data input valid |
| tx\_ready\_o | 1 | Output | Data output ready |
| tx\_o | 1 | Output | TX |
| busy\_o | 1 | Output | Status busy |

### Functional description

The Uart transmitter acts as a parallel-to-serial shift register. Initially, after the transmission signal is turned on, the transmitter will receive a block of data from the transmitter's fifo block. The Uart transmitter checks the number of bits 0 and 1 in the received data block and then stores this value in a parity bit. The transmitter transfers the entire data block and parity bits to a register. Then the transmission begins. The output of the Uart transmitter block is always in standby state. The start bit is detected when the data line transitions from logic “1” state to logic “0” state. Next, each bit of the data block and the parity bit in the stored register are transmitted to the output of the transmitter. The stop bit is transmitted last. After transmitting the stop bit, a signal will be turned on to indicate that a transmission has been completed.

1. During transmission, each bit is sampled 16 times. Because this Uart protocol transmits 8 data bits, 1 parity bit, 1 stop bit, we have the following operation process:

2. The signal received from the queue is checked for the number of parity bits in the data frame, the check results are stored in the parity bit. Move the parity bit and data frame into a register

3. Wait until the transmit notification signal is turned on, perform the transmission starting from the start bit when the incoming signal is low. The active tick counter counts the required number of ticks for the start bit. The tick counter is reset and the data transfer proceeds

4. When the tick counter reaches the value 15, one bit of data is completed. The register storing the data bit is shifted to the right by 1 bit and the tick counter is reset.

5. Repeat step 3 seven times to transmit all 8 bits of data.

6. Repeat step 3 one more time to transmit the parity bit.

7. Repeat step 3 again to complete stop bit transmission.

### FSMD

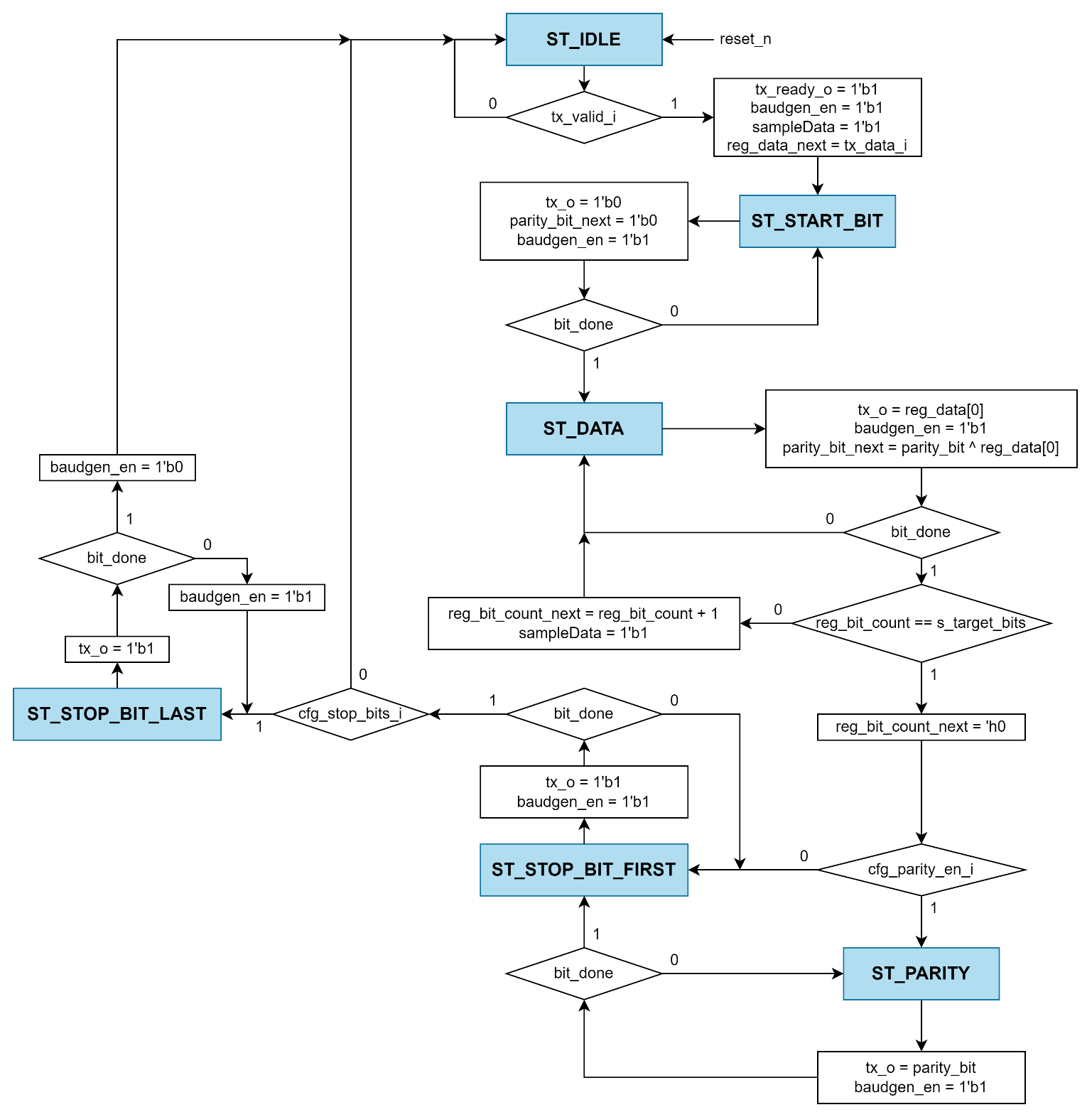


Figure 2.7 UART transmiter FSMD

## Design the UART receiver

### Block diagram

A white rectangular object with black text

Description automatically generated

Figure 2.8 UART receiver block diagram

### Signal description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Width** | **Port** | **Description** |
| clk | 1 | Input | Clock |
| reset\_n | 1 | Input | Reset asynchronous active LOW |
| rx\_i | 1 | Input | RX |
| cfg\_div\_i | 1 | Input | Division for baudrate generator |
| cfg\_en\_i | 1 | Input | Enable UART RX |
| cfg\_parity\_en\_i | 1 | Input | Enable parity bit |
| cfg\_bits\_i | 1 | Input | Configuration number of data bits |
| err\_clr\_i | 1 | Input | Status error when clear FIFO |
| rx\_ready\_i | 1 | Input | Data input ready |
| busy\_o | 1 | Output | Status busy |
| err\_o | 1 | Output | Status error |
| rx\_data\_o | 8 | Output | Data output |
| rx\_valid\_o | 1 | Output | Data output valid |

### Functional description

During reception, the UART will be in standby mode and always expect to receive the start bit. The start bit is detected when the data line transitions from logic “1” state to logic “0” state. The UART will record the data and check the parity bit after the receiving block has received the data. Parity bit to check the accuracy of received data compared to the transmitter. If the parity bit returns true, then data from the receiving block is pushed out to the receiver's buffer block, otherwise, if the parity bit returns false, data from the receiving block is not pushed out to the receiver's buffer block.

During reception, a sampling rate of 16 times the baud rate is used, which means each serial bit is sampled 16 times. Because the Uart protocol is chosen to receive 8 data bits, 1 parity bit, 1 stop bit, we have the following operating process:

1. The initial signal is always high, wait until the incoming signal drops to level 0, which is the starting point of the start bit, the tick counter begins to operate, performing signal sampling counting.

2. When the tick counter reaches 7, now in the middle of the start signal, receive the start bit and restart the counter.

3. When the counter reaches the value 15, now in the middle of the first data bit, receive the data bit, then transfer it to the register and reset the counter.

4. Repeat step 3 7 more times to receive all 8 bits of data.

5. Repeat step 3 one more time to get the parity bit.

6. Repeat step 3 again to receive the stop bit

### FSMD

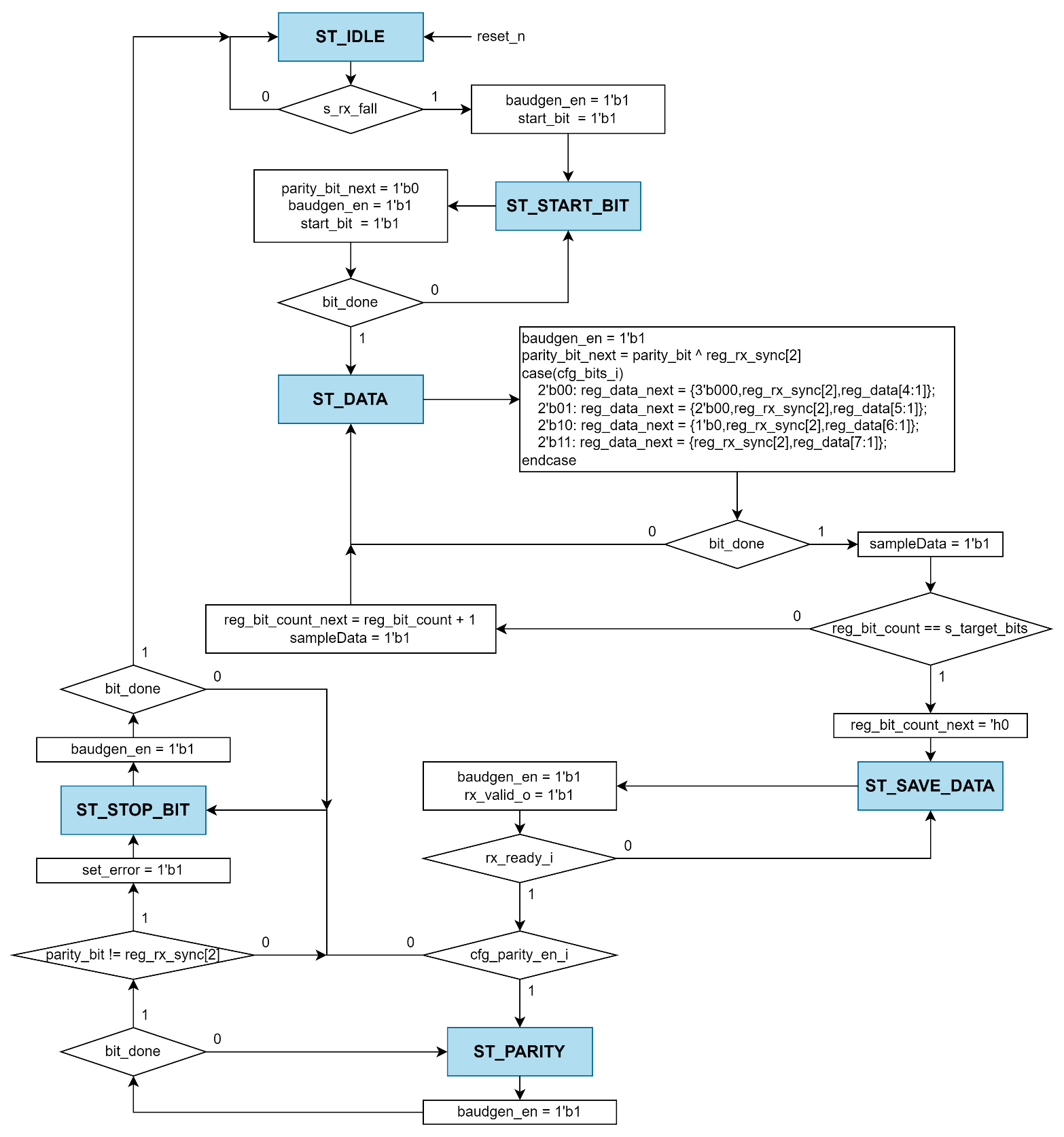


Figure 2.9 UART receiver FSMD

## Design the Register block

### Block diagram

A black and white rectangular object with text

Description automatically generated

Figure 2.10 Register block block diagram

### Register summary

Table 2.4 Register summary

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Address**  **(Hex)** | **SW Access** | **Description** |
| rbr | 000 | RO | Receiver Buffer Register |
| thr | 000 | WO | Transmitter Holding Register |
| ier | 004 | RW | Interrupt Enable Register |
| iir | 008 | RO | Interrupt Identification Register |
| fcr | 008 | WO | FIFO Control Register |
| lcr | 00C | RW | Line Control Register |
| mcr | 010 | RW | Modem Control Register |
| lsr | 014 | RW | Line Status Register |
| msr | 018 | RW | Modem Status Register |
| scr | 01C | RW | Scratch Pad Register |
| dll | 020 | RW | Divisor LSB Latch |
| dlh | 024 | RW | Divisor MSB Latch |
| mdr | 028 | RW | Mode Definition Register |
| stt | 02Cs | RO | Status Register |

### rbr

Address: 0x000

SW Access: RO

Description: Receiver Buffer Register

Table 2.5 rbr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| data | [7:0] | 0 | Received data. |
| rfu | [31:8] | 0 | Reserved |

### thr

Address: 0x000

SW Access: WO

Description: Transmitter Holding Register

Table 2.6 thr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| data | [7:0] | 0 | Data to transmit. |
| rfu | [31:8] | 0 | Reserved |

### ier

Address: 0x004

SW Access: RW

Description: Interrupt Enable Register

Table 2.7 ier field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| erbi | 0 | 0 | Receiver data available interrupt and character timeout indication interrupt enable.  0 = Receiver data available interrupt and character timeout indication interrupt is disabled.  1 = Receiver data available interrupt and character timeout indication interrupt is enabled. |
| etbei | 1 | 0 | Transmitter holding register empty interrupt enable.  0 = Transmitter holding register empty interrupt is disabled.  1 = Transmitter holding register empty interrupt is enabled. |
| elsi | 2 | 0 | Receiver line status interrupt enable.  0 = Receiver line status interrupt is disabled.  1 = Receiver line status interrupt is enabled. |
| edssi | 3 |  | Enable modem status interrupt. |
| rfu | [31:4] | 0 | Reserved |

### iir

Address: 0x008

SW Access: RO

Description: Interrupt Identification Register

Table 2.8 iir field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| ipend | 0 | 0 | Interrupt pending.  When any UART interrupt is generated and is enabled in IER, IPEND is forced to 0. IPEND remains 0 until all pending interrupts are cleared or until a hardware reset occurs. If no interrupts are enabled, IPEND is never forced to 0.  0 = Interrupts pending.  1 = No interrupts pending. |
| intid | [3-1] | 0 | Interrupt type.  0 = Reserved.  1 = Transmitter holding register empty (priority 3).  2 = Receiver data available (priority 2).  3 = Receiver line status (priority 1, highest).  4 = Reserved.  5 = Reserved.  6 = Character timeout indication (priority 2).  7 = Reserved. |
| rfu1 | [5-4] | 0 | Reserved |
| fifoen | [7-6] | 0 | FIFOs enabled.  0 = Non-FIFO mode.  1 = Reserved.  2 = Reserved.  3 = FIFOs are enabled. FIFOEN bit in the FIFO control register (FCR) is set to 1. |
| rfu2 | [31:8] | 0 | Reserved |

### fcr

Address: 0x008

SW Access: WO

Description: FIFO Control Register

Table 2.9 fcr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| fifoen | 0 | 0 | Transmitter and receiver FIFOs mode enable. FIFOEN must be set before other FCR bits are written to or the FCR bits are not programmed. Clearing this bit clears the FIFO counters.  0 = Non-FIFO mode. The transmitter and receiver FIFOs are disabled, and the FIFO pointers are cleared.  1 = FIFO mode. The transmitter and receiver FIFOs are enabled. |
| rxclr | 1 | 0 | Receiver FIFO clear. Write a 1 to RXCLR to clear the bit.  0 = No effect.  1 = Clears receiver FIFO and resets the receiver FIFO counter. The shift register is not cleared. |
| txclr | 2 | 0 | Transmitter FIFO clear. Write a 1 to TXCLR to clear the bit.  0 = No effect.  1 = Clears transmitter FIFO and resets the transmitter FIFO counter. The shift register is not cleared. |
| dmamode1 | 3 | 0 | DMA MODE1 enable if FIFOs are enabled. Always write 1 to DMAMODE1. After a hardware reset, change DMAMODE1 from 0 to 1. DMAMOD1 = 1 is a requirement for proper communication between the UART and the EDMA controller.  0 = DMA MODE1 is disabled.  1 = DMA MODE1 is enabled. |
| rfu1 | [5-4] | 0 | Reserved |
| rxfiftl | [7-6] | 0 | Receiver FIFO trigger level. RXFIFTL sets the trigger level for the receiver FIFO. When the trigger level is reached, a receiver data-ready interrupt is generated (if the interrupt request is enabled). When the FIFO drops below the trigger level, the interrupt is cleared.  0 = 1 byte.  1 = 4 bytes.  2 = 8 bytes.  3 = 14 bytes. |
| rfu2 | [31:8] | 0 | Reserved |

### lcr

Address: 0x00C

SW Access: RW

Description: Line Control Register

Table 2.10 lcr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| wls | [1-0] | 0 | Word length select. Number of bits in each transmitted or received serial character. When STB = 1, the WLS bit  determines the number of STOP bits.  0 = 5 bits  1 = 6 bits  2 = 7 bits  3 = 8 bits |
| stb | 2 | 0 | Number of STOP bits generated. STB specifies 1, 1.5, or 2 STOP bits in each transmitted character. When STB = 1, the WLS bit determines the number of STOP bits. The receiver clocks only the first STOP bit, regardless of the number of STOP bits selected.  0 = One STOP bit is generated.  1 = WLS bit determines the number of STOP bits:  » When WLS = 0, 1.5 STOP bits are generated.  » When WLS = 1h, 2h, or 3h, 2 STOP bits are generated. |
| pen | 3 | 0 | Parity enable. The PEN bit works in conjunction with the SP and EPS bits.  0 = No PARITY bit is transmitted or checked.  1 = Parity bit is generated in transmitted data and is checked in received data between the last data word bit and the first STOP bit. |
| eps | 4 | 0 | Even parity select. Selects the parity when parity is enabled (PEN = 1). The EPS bit works in conjunction with  the SP and PEN bits.  0 = Odd parity is selected (an odd number of logic 1s is transmitted or checked in the data and PARITY bits).  1 = Even parity is selected (an even number of logic 1s is transmitted or checked in the data and PARITY bits). |
| sp | 5 | 0 | Stick parity. The SP bit works in conjunction with the EPS and PEN bits.  0 = Stick parity is disabled.  1 = Stick parity is enabled.  When odd parity is selected (EPS = 0), the PARITY bit is transmitted and checked as set.  When even parity is selected (EPS = 1), the PARITY bit is transmitted and checked as cleared. |
| bc | 6 | 0 | Break control.  0 = Break condition is disabled.  1 = Break condition is transmitted to the receiving UART. A break condition is a condition in which the UARTn\_TXD signal is forced to the spacing (cleared) state. |
| dlab | 7 | 0 | Divisor latch access bit. The divisor latch registers (DLL and DLH) can be accessed at dedicated addresses or at addresses shared by RBR, THR, and IER. Using the shared addresses requires toggling DLAB to change which registers are selected. If the dedicated addresses are used, DLAB can = 0.  0 = Allows access to the receiver buffer register (RBR), the transmitter holding register (THR), and the interrupt enable register (IER) selected. At the address shared by RBR, THR, and DLL, the CPU can read from RBR and write to THR. At the address shared by IER and DLH, the CPU can read from and write to IER.  1 = Allows access to the divisor latches of the baud generator during a read or write operation (DLL and DLH). At the address shared by RBR, THR, and DLL, the CPU can read from and write to DLL. At the address shared by IER and DLH, the CPU can read from and write to DLH. |
| rfu | [31:8] | 0 | Reserved |

### mcr

Address: 0x010

SW Access: RW

Description: Modem Control Register

Table 2.11 mcr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| rfu1 | 0 | 0 | Reserved |
| rts | 1 | 0 | RTS control. When AFE = 1, the RTS bit determines the autoflow control enabled. Note that all UARTs do not  support this feature. See the device-specific data manual for supported features. If this feature is not available,  this bit is reserved and should be cleared to 0.  0 = UARTn\_RTS is disabled, only UARTn\_CTS is enabled.  1 = UARTn\_RTS and UARTn\_CTS are enabled. |
| out1 | 2 | 0 | OUT1 Control Bit. |
| out2 | 3 | 0 | OUT2 Control Bit. |
| loop | 4 | 0 | Loopback mode enable. LOOP is used for the diagnostic testing using the loopback feature.  0 = Loopback mode is disabled.  1 = Loopback mode is enabled. When LOOP is set, the following occur:  » The UARTn\_TXD signal is set high.  » The UARTn\_RXD pin is disconnected  » The output of the transmitter shift register (TSR) is looped back in to the receiver shift register (RSR) input. |
| afe | 5 | 0 | Autoflow control enable. Autoflow control allows the UARTn\_RTS and UARTn\_CTS signals to provide  handshaking between UARTs during data transfer. When AFE = 1, the RTS bit determines the autoflow control  enabled. Note that all UARTs do not support this feature. See the device-specific data manual for supported  features. If this feature is not available, this bit is reserved and should be cleared to 0.  0 = Autoflow control is disabled.  1 = Autoflow control is enabled:  » When RTS = 0, only UARTn\_CTS is enabled.  » When RTS = 1, UARTn\_RTS and UARTn\_CTS are enabled. |
| rfu2 | [31:6] | 0 | Reserved |

### lsr

Address: 0x014

SW Access: RW

Description: Line Status Register

Table 2.12 lsr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| dr | 0 | 0 | Data-ready (DR) indicator for the receiver. If the DR bit is set and the corresponding interrupt enable bit is set (ERBI = 1 in IER), an interrupt request is generated. |
| oe | 1 | 0 | Overrun error (OE) indicator. An overrun error in the non-FIFO mode is different from an overrun error in the FIFO mode. If the OE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated. |
| pe | 2 | 0 | Parity error (PE) indicator. A parity error occurs when the parity of the received character does not match the parity selected with the EPS bit in the line control register (LCR). If the PE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated. |
| fe | 3 | 0 | Framing error (FE) indicator. A framing error occurs when the received character does not have a valid STOP bit. In response to a framing error, the UART sets the FE bit and waits until the signal on the RX pin goes high. When the RX signal goes high, the receiver is ready to detect a new START bit and receive new data. If the FE bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated. |
| bi | 4 | 0 | Break indicator. The BI bit is set whenever the receive data input (UARTn\_RXD) was held low for longer than a full-word transmission time. A full-word transmission time is defined as the total time to transmit the START, data, PARITY, and STOP bits. If the BI bit is set and the corresponding interrupt enable bit is set (ELSI = 1 in IER), an interrupt request is generated. |
| thre | 5 | 0 | Transmitter holding register empty (THRE) indicator. If the THRE bit is set and the corresponding interrupt enable bit is set (ETBEI = 1 in IER), an interrupt request is generated. |
| temt | 6 | 0 | Transmitter empty (TEMT) indicator. |
| rxfifoe | 7 | 0 | Receiver FIFO error. |
| rfu2 | [31:8] | 0 | Reserved |

### msr

Address: 0x018

SW Access: RW

Description: Modem Status Register

Table 2.13 msr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| dcts | 0 | 0 | Change in CTS indicator bit. DCTS indicates that the CTS input has changed state since the last time it was read by the CPU. When DCTS is set (autoflow control is not enabled and the modem status interrupt is enabled), a modem status interrupt is generated. When autoflow control is enabled, no interrupt is generated. |
| ddsr | 1 | 0 | Change in DSR indicator bit. DDSR indicates that the DSR input has changed state since the last time it was read by the CPU. When DDSR is set and the modem status interrupt is enabled, a modem status interrupt is generated. |
| teri | 2 | 0 | Trailing edge of RI (TERI) indicator bit. TERI indicates that the RI input has changed from a low to a high. When TERI is set and the modem status interrupt is enabled, a modem status interrupt is generated. |
| dcd | 3 | 0 | Change in DCD indicator bit. DCD indicates that the DCD input has changed state since the last time it wasread by the CPU. When DCD is set and the modem status interrupt is enabled, a modem status interrupt is generated. |
| cts | 4 | 0 | Complement of the Clear To Send input. When the UART is in the diagnostic test mode (loopback mode MCR[4] = 1), this bit is equal to the MCR bit 1 (RTS). |
| dsr | 5 | 0 | Complement of the Data Set Ready input. When the UART is in the diagnostic test mode (loopback mode MCR[4] = 1), this bit is equal to the MCR bit 0 (DTR). |
| ri | 6 | 0 | Complement of the Ring Indicator input. When the UART is in the diagnostic test mode (loopback mode MCR[4] = 1), this bit is equal to the MCR bit 2 (OUT1). |
| cd | 7 | 0 | Complement of the Carrier Detect input. When the UART is in the diagnostic test mode (loopback mode MCR[4] = 1), this bit is equal to the MCR bit 3 (OUT2). |
| rfu | [31:8] | 0 | Reserved |

### scr

Address: 0x01C

SW Access: RW

Description: Scratch Pad Register

Table 2.14 scr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| scr | [7-0] | 0 | These bits are intended for the programmer's use as a scratch pad in the sense that it temporarily holds the programmer's data without affecting any other UART operation. |
| rfu | [31:8] | 0 | Reserved |

### dll

Address: 0x020

SW Access: RW

Description: Divisor LSB Latch

Table 2.15 dll field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| dll | [7-0] | 0 | The 8 least-significant bits (LSBs) of the 16-bit divisor for generation of the baud clock in the baud rate generator. |
| rfu | [31:8] | 0 | Reserved |

### dlh

Address: 0x024

SW Access: RW

Description: Divisor MSB Latch

Table 2.16 dlh field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| dlh | [7-0] | 0 | The 8 most-significant bits (MSBs) of the 16-bit divisor for generation of the baud clock in the baud rate generator. Maximum baud rate is 128 kbps. |
| rfu | [31:8] | 0 | Reserved |

### mdr

Address: 0x028

SW Access: RW

Description: Mode Definition Register

Table 2.17 mdr field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| osmsel | 0 | 0 | Over-Sampling Mode Select.  0 = 16× oversampling.  1 = 13× oversampling. |
| rfu | [31:1] | 0 | Reserved |

### stt

Address: 0x02C

SW Access: RW

Description: Status Register

Table 2.18 stt field descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Bit** | **Reset value** | **Description** |
| txff\_full | 0 | 0 | FIFO transmiter full |
| txff\_empty | 1 | 1 | FIFO transmiter empty |
| rxff\_full | 2 | 0 | FIFO receiver full |
| rxff\_empty | 3 | 1 | FIFO receiver empty |
| rfu | [31:4] | 0 | Reserved |

# SIMULATION AND VERIFICATION

In this chapter, I present the results of simulating the UART module. I conducted tests to check the simulation results of four modules: the transmitter, receiver, communication controller, and decoder. Then, I tested the simulation when connecting the transmitter and receiver to ensure the accuracy and performance of the UART system.

During the simulation process, I used appropriate simulation tools and integrated development environment (IDE) to create realistic simulation conditions. I used different input values to test boundary cases and special situations while ensuring that the system operates as expected in all scenarios.

# CONCLUSION

In this Project, we successfully designed and implemented a UART communication protocol tailored for integration within Very Large Scale Integration (VLSI) systems. Through a systematic approach, we addressed key challenges inherent in VLSI design, including area, power, and timing constraints, while ensuring the functionality and reliability of the UART protocol.

Our design process began with a thorough understanding of UART specifications and protocols, laying the groundwork for the development of a robust architecture. Leveraging VHDL or Verilog, we crafted a scalable and efficient design that encompassed essential features such as baud rate generation, data framing, error detection, and flow control mechanisms.

In conclusion, this project not only deepened our understanding of VLSI design principles but also equipped us with practical experience in implementing complex communication protocols within the constraints of modern semiconductor technology. Moving forward, the insights gained from this project will serve as a valuable foundation for further exploration and refinement of VLSI-based communication systems.

# TÀI LIỆU THAM KHẢO

1. ARM, “AMBA AHB Protocol Specification” [Online]. Available: <https://developer.arm.com/documentation/ihi0033/latest/>
2. ARM, “AMBA AXI Protocol Specification” [Online]. Available: <https://developer.arm.com/documentation/ihi0022/c/?lang=en>
3. <https://verificationguide.com/systemverilog/systemverilog-testbench/>, truy cập lần cuối ngày 02/03/2023
4. <https://verificationguide.com/uvm/uvm-tutorial/>, truy cập lần cuối cùng ngày 03/03/2023
5. <https://www.chipverify.com/uvm/uvm-tutorial>, truy cập lần cuối cùng ngày 03/03/2023
6. Dolphin Technology Vietnam Center, “AHB to AXI Bridge Specification”, 2022

PHỤ LỤC

## Phụ lục 1. Testplan lõi IP AHB to AXI

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu đề** | **Mô tả** | **Trọng số** | **Mục tiêu** |
| AP\_AHB2AXI\_HBURST\_SINGLE\_AW | AHB single transfer burst are converted to AXI4 INCR with length 0 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR\_AW | AHB incrementing burst of undefined length transfers are converted to AXI4 INCR with length 0 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP4\_AW | AHB 4-beat wrapping burst are converted to AXI4 WRAP with length 3 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR4\_AW | AHB 4-beat incrementing burst are converted to AXI4 INCR with length 3 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP8\_AW | AHB 8-beat wrapping burst are converted to AXI4 WRAP with length 7 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR8\_AW | AHB 8-beat incrementing burst are converted to AXI4 INCR with length 7 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP16\_AW | AHB 16-beat wrapping burst are converted to AXI4 WRAP with length 15 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR16\_AW | AHB 16-beat incrementing burst are converted to AXI4 INCR with length 15 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_SINGLE\_AR | AHB single transfer burst are converted to AXI4 INCR with length 0 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR\_AR | AHB incrementing burst of undefined length transfers are converted to AXI4 INCR with length 0 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP4\_AR | AHB 4-beat wrapping burst are converted to AXI4 WRAP with length 3 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR4\_AR | AHB 4-beat incrementing burst are converted to AXI4 INCR with length 3 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP8\_AR | AHB 8-beat wrapping burst are converted to AXI4 WRAP with length 7 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR8\_AR | AHB 8-beat incrementing burst are converted to AXI4 INCR with length 7 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_WRAP16\_AR | AHB 16-beat wrapping burst are converted to AXI4 WRAP with length 15 | 1 | 100% |
| AP\_AHB2AXI\_HBURST\_INCR16\_AR | AHB 16-beat incrementing burst are converted to AXI4 INCR with length 15 | 1 | 100% |
| AP\_AHB2AXI\_HSIZE\_AW | AXI4 transfer size must be the same as AHB transfer size | 1 | 100% |
| AP\_AHB2AXI\_HSIZE\_AR | AXI4 transfer size must be the same as AHB transfer size | 1 | 100% |
| AP\_AHB2AXI\_HWRITE\_AW | HWRITE is high, AXI4 write channel sends transfer and HWRITE is low, AXI4 read channel sends transfer | 1 | 100% |
| AP\_AHB2AXI\_HWRITE\_AR | HWRITE is high, AXI4 write channel sends transfer and HWRITE is low, AXI4 read channel sends transfer | 1 | 100% |
| AP\_AHB2AXI\_HADDR\_AW | AXI4 address must be the same as AHB address | 1 | 100% |
| AP\_AHB2AXI\_HADDR\_AR | AXI4 address must be the same as AHB address | 1 | 100% |