

ADSD Special Assignment

Design of UART receiver Based on Verilog HDL and implement on DE2 board.

Student name: Samir Khanusiya

Roll number: 24MRE007

Abstract:- As a two-way transmission channel, Universal Asynchronous Receiver/Transmitter (UART) not only greatly improves the efficiency information transmission between computers and external devices, but also ensures the accuracy and consistency of information by eliminating metastable state, setting baud rate and other means. In this paper, on the basis of fully understanding the definition and function of UART, based on Verilog HDL language to build UART receiver, and through Quartus simulation, image and data. The experimental results show that the receiving and sending module of this module works well and meets the requirements of full-duplex serial communication equipment. There is no doubt that the design of this paper has made a more detailed explanation of the basic operating principle of UART receiver, which will contribute to its further development.

Keywords: UART receiver, Verilog HDL, full-duplex serial communication, Altera Quartus, DE2 Board, Digilent Waveform.

1. Introduction

Computer is an indispensable part of the development of modern science and technology, and any development of it is of great significance to the progress of the age of science and technology. Universal Asynchronous Receiver/Transmitter is a universal serial data bus, which can meet the needs of information transmission between peripherals and computers, greatly increasing the efficiency of information transmission, and achieving full-duplex serial communication. It was a major milestone in the history of computing. This paper will fully understand the structure and principle of UART, function and implementation on the basis of the use of Verilog HDL language, by describing its function, to achieve the construction of UART. By writing the code and running the simulation, the results this paper get agree with the expectation, which proves the feasibility of the construction method. UART can be divided into sending module and receiving module according to functions. It transmits binary codes bit by bit and receives binary codes bit by bit. It is worth noting that in order to take into account the accuracy and efficiency of information transmission, the sending module and the receiving module have different methods of confirming information.

2. Implementation of UART function

2.1. UART operational principle

2.1.1. UART brief introduction

UART is a universal asynchronous receiver/transmitter that converts data to be transmitted between serial communication and parallel communication, enabling full-duplex transmission and receiving. It has a serial data stream that converts the parallel data inside the computer into the output, and the input serial data into the byte transmission bit by bit; Parity check of input and output data streams; Add functions such as deleting start and stop tags to the output data stream. The frame format of asynchronous communication is usually 1 bit at the start, 5 to 8 bits at the data bit, parity bit (optional), parity or no parity, and stop bit (1, 1.5, or 2 bits). UART data format [2]. As can be seen Figure 1.

To speed up the operation process, the parity bit is not set in this operation. The data frame format is 1-bit start bit, 8-bit data bit, no parity bit, and 1-bit stop bit. The communication interface standard is Rs-232 [3] and full-duplex communication mode [4].

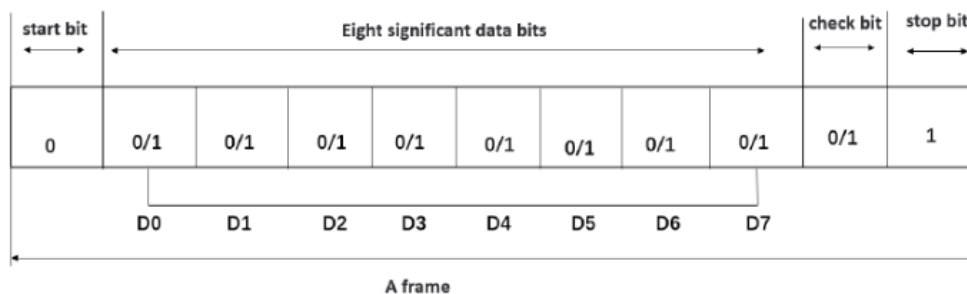


Fig. 1 Data frame format of UART

2.1.2 Division of modules

UART can be divided into two sub-modules: serial port sending module and serial port receiving module. The working diagram is shown in Figure 2 and Figure 3.

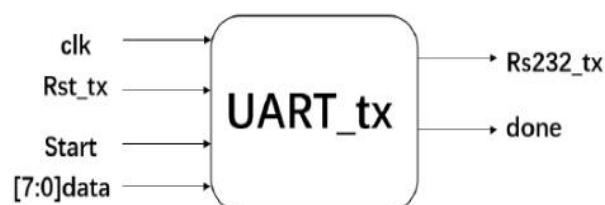


Fig. 2 UART sending module



Fig. 3 UART receiving module

Clk stands for master clock, rst_n stands for reset signal, start stands for start signal, [7:0]data stands for data bits (8 bits); Rs232_tx indicates data transmission and done indicates termination signal.

2.3. Receiving module

2.3.1. The elimination of metastable state

When the input of the first-stage trigger does not meet its establishment holding time, the output metastable data will stabilize before the next pulse edge arrives, and the stable data will meet the establishment time of the second-stage trigger. If the requirements are met, then the second stage trigger will not be metastable when the next pulse edge arrives [7]

2.3.2 Receiver timing design

Every time the level changes from "0" to "1", it can be regarded as the arrival of the starting bit of a frame data. However, the noise on the communication line is also likely to make logic "1" jump to logic "0", in order to ensure the accuracy of the information exchanged between the communication parties. In this paper, falling edge detection is used to filter the interference of communication line noise. baud_cnt is the baud rate setting, while bit_cnt and bit_flag jointly control whether rx_data receives each bit of data transmitted by Rs232. Finally, the receiving stops, the state signal is pulled down, and the done signal is pulled up. When the sampling counter counts, all data bits have been entered [8]. As can be seen Figure 5

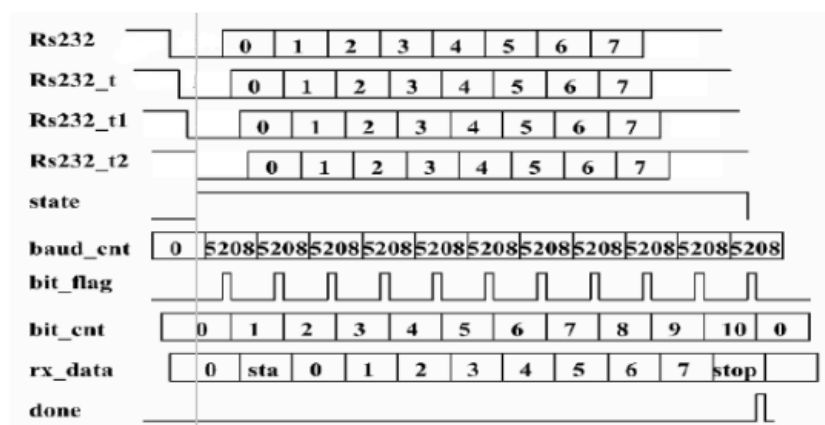


Fig. 5 Timing diagram of receiving module

3. Experiment Content

3.1.2 UART receiving module

```
module uart_rx(  
    input wire clk,        // 50 MHz clock input  
    input wire reset_n,    // Active low reset  
    input wire rx,         // UART receive input  
    output reg [7:0] rx_data, // 8-bit received data  
    output reg rx_ready    // Indicates data received and ready to be  
read  
);  
  
    // Parameters  
    localparam BAUD_RATE = 115200;  
    localparam CLOCK_FREQ = 50000000; // 50 MHz clock  
    localparam BAUD_DIV = CLOCK_FREQ / BAUD_RATE; // Baud rate  
divider  
  
    localparam IDLE = 2'b00, START = 2'b01, DATA = 2'b10, STOP =  
2'b11;  
  
    reg [1:0] state = IDLE;    // FSM state  
    reg [9:0] shift_reg;       // Shift register to store incoming bits  
    reg [15:0] baud_counter = 0; // Baud rate counter  
    reg [3:0] bit_index = 0;    // Bit counter to track the current bit
```

```
always @(posedge clk or negedge reset_n) begin
```

```
    if (!reset_n) begin
```

```
        state <= IDLE;
```

```
        rx_ready <= 0;
```

```
        baud_counter <= 0;
```

```
        bit_index <= 0;
```

```
    end else begin
```

```
        case (state)
```

```
            IDLE: begin
```

```
                rx_ready <= 0;
```

```
                if (!rx) begin // Detect start bit (low)
```

```
                    state <= START;
```

```
                    baud_counter <= 0;
```

```
                end
```

```
            end
```

```
            START: begin
```

```
                if (baud_counter < (BAUD_DIV / 2)) begin
```

```
                    baud_counter <= baud_counter + 1;
```

```
                end else begin
```

```
                    baud_counter <= 0;
```

```
                    state <= DATA;
```

```

        bit_index <= 0;
    end
end

DATA: begin
    if (baud_counter < BAUD_DIV) begin
        baud_counter <= baud_counter + 1;
    end else begin
        baud_counter <= 0;
        shift_reg[bit_index] <= rx; // Sample the data bit
        bit_index <= bit_index + 1;
        if (bit_index == 7) begin
            state <= STOP;
        end
    end
end
end

```

```

STOP: begin
    if (baud_counter < BAUD_DIV) begin
        baud_counter <= baud_counter + 1;
    end else begin
        baud_counter <= 0;
        rx_data <= shift_reg[7:0]; // Store received data
    end
end

```

```

        rx_ready <= 1'b1;        // Data is ready to be read

        state <= IDLE;

    end

    end

    endcase

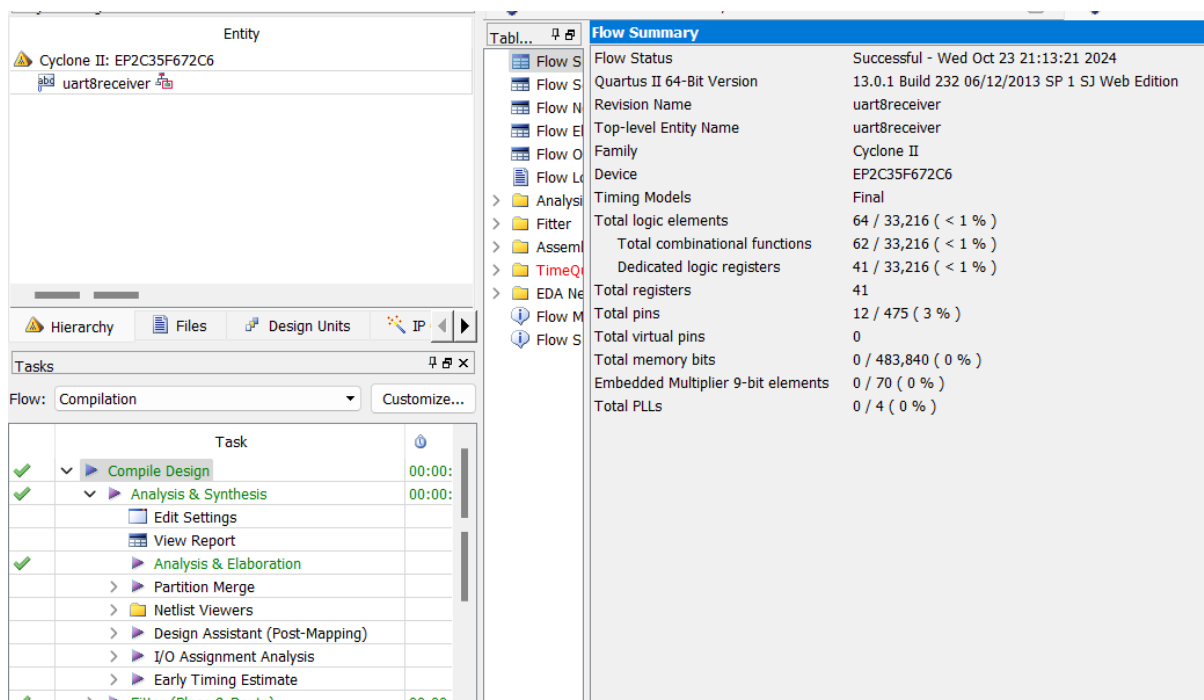
    end

    end

endmodule

```

3.2. Experimental Results



The screenshot displays the Quartus II software interface. On the left, the 'Entity' pane shows the project 'Cyclone II: EP2C35F672C6' with a component 'uart8receiver'. Below it, the 'Tasks' pane shows a list of tasks for the 'Compilation' flow, including 'Compile Design', 'Analysis & Synthesis', 'Edit Settings', 'View Report', 'Analysis & Elaboration', 'Partition Merge', 'Netlist Viewers', 'Design Assistant (Post-Mapping)', 'I/O Assignment Analysis', and 'Early Timing Estimate'. On the right, the 'Flow Summary' pane provides a detailed overview of the compilation process, including the flow status, version, revision name, top-level entity name, family, device, timing models, and resource usage statistics.

Flow Summary		
Flow Status	Successful - Wed Oct 23 21:13:21 2024	
Flow S	Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Flow N	Revision Name	uart8receiver
Flow E	Top-level Entity Name	uart8receiver
Flow O	Family	Cyclone II
Flow L	Device	EP2C35F672C6
Analysis	Timing Models	Final
Fitter	Total logic elements	64 / 33,216 (< 1 %)
Assembl	Total combinational functions	62 / 33,216 (< 1 %)
TimeQ	Dedicated logic registers	41 / 33,216 (< 1 %)
EDA Ne	Total registers	41
Flow M	Total pins	12 / 475 (3 %)
Flow S	Total virtual pins	0
	Total memory bits	0 / 483,840 (0 %)
	Embedded Multiplier 9-bit elements	0 / 70 (0 %)
	Total PLLs	0 / 4 (0 %)

- The design uses simulation software modelsim, clearly simulated UART send and receive data waveform

Groups

Named: *

Node Name

rx_data[7..0]

<<new group>>

Top View - Wire Bond

Options: EPIC3000000

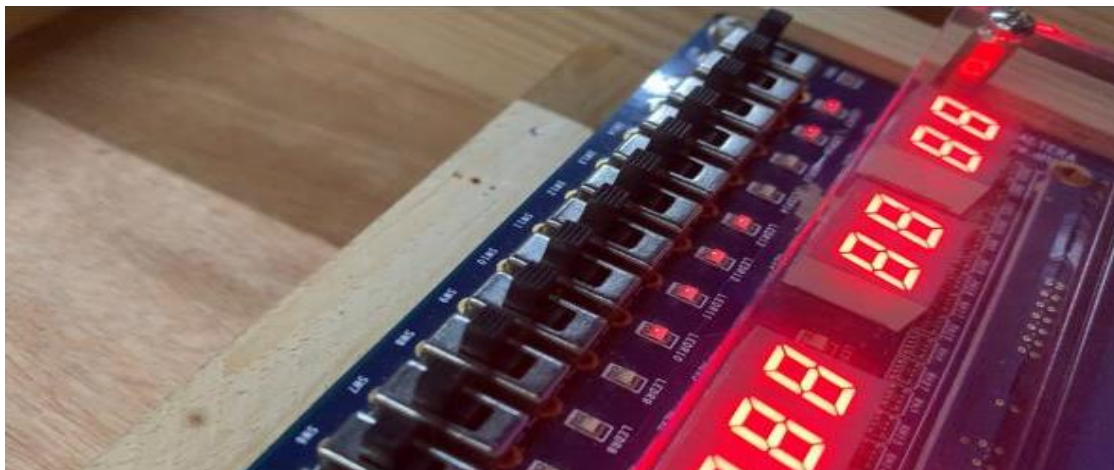
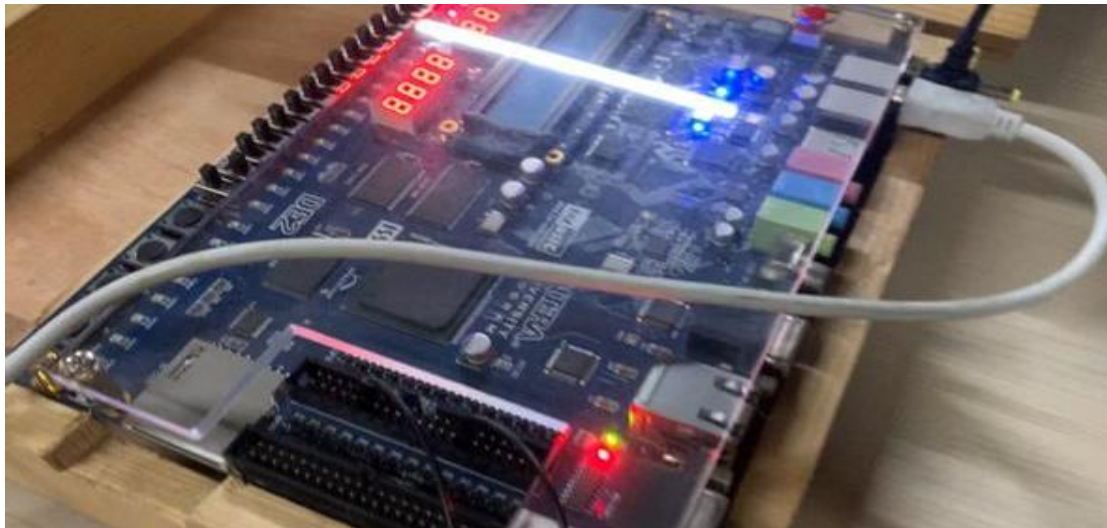
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_N2	2	B2_N1	PIN_N2	3.3-V L...efault		24mA (default)	
reset_n	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V L...efault		24mA (default)	
rx	Input	PIN_C25	5	B5_N0	PIN_C25	3.3-V L...efault		24mA (default)	
rx_data[7]	Output	PIN_AD12	8	B8_N0	PIN_AD12	3.3-V L...efault		24mA (default)	
rx_data[6]	Output	PIN_AE12	8	B8_N0	PIN_AE12	3.3-V L...efault		24mA (default)	
rx_data[5]	Output	PIN_AE13	8	B8_N0	PIN_AE13	3.3-V L...efault		24mA (default)	
rx_data[4]	Output	PIN_AE13	8	B8_N0	PIN_AE13	3.3-V L...efault		24mA (default)	
rx_data[3]	Output	PIN_AE15	7	B7_N1	PIN_AE15	3.3-V L...efault		24mA (default)	
rx_data[2]	Output	PIN_AD15	7	B7_N1	PIN_AD15	3.3-V L...efault		24mA (default)	
rx_data[1]	Output	PIN_AC14	7	B7_N1	PIN_AC14	3.3-V L...efault		24mA (default)	
rx_data[0]	Output	PIN_AA13	7	B7_N1	PIN_AA13	3.3-V L...efault		24mA (default)	
rx_ready	Output	PIN_Y18	7	B7_N0	PIN_Y18	3.3-V L...efault		24mA (default)	

Named: *

Edit: X

Filter

<<new node>>



As can be seen from Figure , rs232_t2 is the serial data to be received (directly calling the data of r_data in the sending module). rx_data is the received parallel data. rx_data on the receiving end does not receive the start and end bits, but receives eight data bits "1", "0", "1", "0", "1", "0", "1", and "0" in sequence from low to high. The data equivalent of rs232_t2 is 55H (01010101B), which proves that the module design is correct.

Summary

This paper introduces the definition, function and module division of UART in detail, and verifies the feasibility of the program by running simulation. From the above experiments, this paper successfully built UART through Verilog HDL language, respectively simulated the sending and receiving modules, took into account the speed and accuracy, realized the high-speed and accurate information exchange between the computer and peripheral, and achieved the ideal result. This design achieves full-duplex transmission and reception, complete the design objectives, so that the data transmission process is efficient, clear, accurate and stable. This paper has certain reference value for the development of UART.