CH32V307 LED Control via UART Commands

Project Report – Task 2

Abstract

This report details the design, implementation, and testing of an embedded system for controlling an LED via Universal Asynchronous Receiver-Transmitter (UART) communication. The system is based on the WCH CH32V307, a RISC-V architecture microcontroller. The project's objective is to receive specific hexadecimal commands (0xAA for ON, 0xBB for OFF) from a host PC to control the state of an onboard LED. This document covers the hardware configuration, software architecture, development process, and validation results, demonstrating a functional and reliable command-response system.

1. Introduction

1.1 Project Objectives

The primary objective was to implement a reliable communication interface that allows precise control of an onboard LED through specific UART commands. The implementation had to meet the following requirements:

- Establish a serial communication link between the CH32V307 development board and a PC.
- Configure the microcontroller to recognize and differentiate between two distinct byte commands (0xAA and 0xBB).
- Control the state of an onboard LED (ON/OFF) based exclusively on the received commands.
- Provide serial feedback to the user confirming the action taken.

1.2 Scope

The scope of this project encompasses the hardware setup, software development, and functional testing of the UART-based LED control system. This includes MCU peripheral initialization, command parsing logic, and validation of the system's response to valid and invalid inputs.

2. System Architecture

The system consists of three main components: a host PC running a serial terminal, a USB-to-UART interface, and the CH32V307 microcontroller which directly controls the LED.

3. Hardware Design

3.1 Components Used

- **CH32V307 Development Board:** The core of the system, featuring the RISC-V MCU.
- **Onboard LED:** Connected to pin PB1 in an active-low configuration.
- **USB-to-UART Converter:** Bridges communication between the PC's USB port and the MCU's UART pins (or an onboard equivalent like CH340).
- **Host PC:** Used for sending commands via terminal emulation software.

3.2 Hardware Configuration

The hardware setup is minimal, relying on the integrated components of the development board.

Function	MCU Pin	Description
LED	PB1	Onboard LED (Active-low: LOW signal turns it ON)
UART TX	PA9	Transmits data from MCU to PC
UART RX	PA10	Receives command data from PC to MCU

4. Software Design

4.1 Software Architecture and Flow

The software is designed as a continuous loop that waits for incoming UART data, processes it, and acts accordingly. The architecture is straightforward and event-driven, with the arrival of a byte on the UART RX line being the primary event trigger.

4.2 Key Software Components

The C code for this project is organized into functions for hardware initialization and a main processing loop. The complete source code is available in **Appendix C**.

GPIO Configuration

The LED is connected to pin PB1. The GPIO_LED_Init function configures this pin as a push-pull output and sets its initial state to HIGH, which keeps the active-low LED off at startup.

```
/**
  * @brief  Initialize GPIO for LED control
  */
void GPIO_LED_Init(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOB, &GPIO_InitStructure);

   // Ensure LED starts in OFF state (active-low LED)
   GPIO_SetBits(GPIOB, GPIO_Pin_1);
}
```

UART Configuration

The UART_Init function configures the USART1 peripheral for communication at 115200 baud, 8 data bits, no parity, and 1 stop bit (115200-8-N-1). It also configures the corresponding GPIO pins (PA9 for TX, PA10 for RX) for their alternate function roles.

```
/**
  * @brief  Initialize UART for serial communication
  */
void UART_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE);
    // Configure TX pin
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // Configure RX pin
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // Configure UART parameters
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE);
}
```

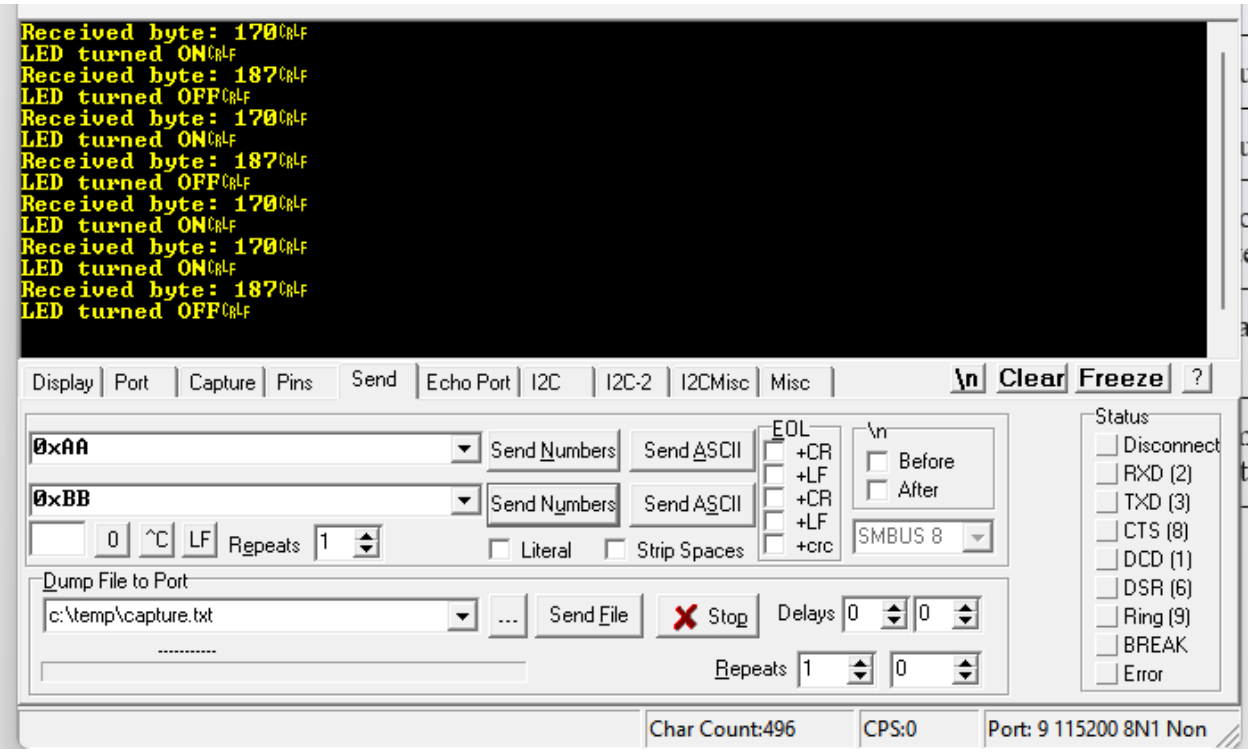
5. Implementation and Testing

5.1 Development Environment

- **IDE:** MounRiver Studio
- **Compiler:** RISC-V Embedded GCC
- **Programming Language:** C
- **Target MCU:** CH32V307

5.2 Testing and Validation

The system was tested by sending hexadecimal commands from a serial terminal program on a host PC. The commands 0xAA (170) and 0xBB (187) were used to control the LED. The test confirmed that the system responds correctly and provides appropriate feedback.



The following table summarizes the test results:

Test Case	Description	Expected Result	Actual Result	Status
TC01	Send 0xAA command	LED turns ON	LED turned ON	PASS
TC02	Send 0xBB command	LED turns OFF	LED turned OFF	PASS
TC03	Alternate commands rapidly	LED toggles correctly	LED toggled as expected	PASS
TC04	Send invalid command (e.g., 0xCC)	LED state unchanged	No change in LED state	PASS
TC05	System restart	Initializes with LED OFF	System initialized correctly	PASS