

Certainly! Let's go through the code step by step:

Initialization and Setup

Includes and Initialization

c

Copy code

```
#include "uart.h"

// Init UART
void UART_init(void) {
    // Set GPIO pins
```

1. **Include the Header File:** This includes the `uart.h` file, which likely contains the definitions and prototypes for UART-related functions and settings.
2. **UART Initialization Function:** This function sets up the GPIO pins and configures the UART peripheral for communication.

GPIO Configuration for UART

The code provides multiple configurations based on the value of `UART_MAP`. The following explains the configuration for each map:

Map 0: PA2 (TX) and PA3 (RX)

c

Copy code

```
#if UART_MAP == 0
    // Setup pin PA2 (TX) and pin PA3 (RX)
    RCC->IOPENR |= RCC_IOPENR_GPIOAEN;
    GPIOA->MODER = (GPIOA->MODER & ~( ((uint32_t)0b11<<(2<<1)) |
((uint32_t)0b11<<(3<<1)) ))
                                | ( ((uint32_t)0b10<<(2<<1)) |
((uint32_t)0b10<<(3<<1)) );
    GPIOA->OTYPER &= ~ ((uint32_t)0b1 <<(2<<0));
    GPIOA->PUPDR = (GPIOA->PUPDR & ~(
((uint32_t)0b11<<(3<<1)) ))
                                | (
((uint32_t)0b01<<(3<<1)) );
```

```

GPIOA->AFR[0] = (GPIOA->AFR[0] & ~( ((uint32_t)0xf <<(2<<2)) |
((uint32_t)0xf <<(3<<2)) ))
| ( ((uint32_t)0x1 <<(2<<2)) |
((uint32_t)0x1 <<(3<<2)) );

```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA2 and PA3 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA2 and PA3 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA2 and PA3 to push-pull mode (clearing the bit).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA3.
 - `GPIOA->AFR[0]:` Sets the alternate function for PA2 and PA3 to AF1 (USART1).

Other Maps

The configurations for other maps (PA7/PA8, PA9/PA10, PA14/PA13, PA14/PA15, PB6/PB7, PF1/PF0) follow a similar pattern, adjusting the specific pins and alternate functions according to the mapping.

UART Peripheral Configuration

c

Copy code

```

// Setup and start UART (8N1, RX/TX, default BAUD rate)
RCC->APBENR2 |= RCC_APBENR2_USART1EN;
USART1->BRR = ((2 * F_CPU / UART_BAUD) + 1) / 2;
USART1->CR1 = USART_CR1_RE | USART_CR1_TE | USART_CR1_UE;
}

```

1. **Enable USART1 Clock:** `RCC->APBENR2 |= RCC_APBENR2_USART1EN;` enables the clock for USART1.
2. **Set Baud Rate:** `USART1->BRR` sets the baud rate. The formula $((2 * F_CPU / UART_BAUD) + 1) / 2$ calculates the value based on the system clock (`F_CPU`) and the desired baud rate (`UART_BAUD`).
3. **Configure USART1:**
 - `USART_CR1_RE:` Enable receiver.
 - `USART_CR1_TE:` Enable transmitter.
 - `USART_CR1_UE:` Enable USART.

UART Read and Write Functions

Reading a Byte

c

Copy code

```
// Read byte via UART
char UART_read(void) {
    while(!UART_available());
    return USART1->DR;
}
```

1. **Wait for Data:** `while(!UART_available());` waits until data is available in the receive data register (`USART1->DR`).
2. **Return Data:** `return USART1->DR;` reads and returns the data from the receive data register.

Sending a Byte

c

Copy code

```
// Send byte via UART
void UART_write(const char c) {
    while(!UART_ready());
    USART1->DR = c;
}
```

1. **Wait for Transmit Ready:** `while(!UART_ready());` waits until the transmit data register is empty and ready to send new data.
2. **Send Data:** `USART1->DR = c;` writes the data to the transmit data register.

Summary

This code initializes and configures the UART peripheral on the PY32F0xx microcontroller. It sets up the GPIO pins for alternate function (UART), configures the UART baud rate, enables the UART peripheral, and provides functions to read and write data via UART. The configuration supports multiple pin mappings through conditional compilation based on the `UART_MAP` value.

4o

Please explain the full code step by step

ChatGPT

Sure! Let's go through the entire code in detail, step by step.

Header Inclusion

c

Copy code

```
#include "uart.h"
```

This line includes the `uart.h` header file. This file likely contains function prototypes, macro definitions, and possibly some configuration settings for the UART functionality.

UART Initialization Function

c

Copy code

```
void UART_init(void) {  
    // Set GPIO pins
```

The `UART_init` function initializes the UART peripheral and sets up the GPIO pins for UART communication. The setup depends on the value of `UART_MAP`, which determines which GPIO pins are used for UART.

Pin Configuration Based on UART_MAP

The code defines multiple configurations based on the value of `UART_MAP`. Each configuration sets up different pins for UART communication.

UART_MAP == 0: PA2 (TX) and PA3 (RX)

c

Copy code

```
#if UART_MAP == 0  
    // Setup pin PA2 (TX) and pin PA3 (RX)  
    RCC->IOPENR |= RCC_IOPENR_GPIOAEN;  
    GPIOA->MODER = (GPIOA->MODER & ~( ((uint32_t)0b11<<(2<<1)) |  
((uint32_t)0b11<<(3<<1)) ))  
| ( ((uint32_t)0b10<<(2<<1)) |  
((uint32_t)0b10<<(3<<1)) );  
    GPIOA->OTYPER &= ~ ((uint32_t)0b1 <<(2<<0));  
    GPIOA->PUPDR = (GPIOA->PUPDR & ~(  
((uint32_t)0b11<<(3<<1)) ))  
| (  
((uint32_t)0b01<<(3<<1)) );
```

```

GPIOA->AFR[0] = (GPIOA->AFR[0] & ~( ((uint32_t)0xf <<(2<<2)) |
((uint32_t)0xf <<(3<<2)) ))
| ( ((uint32_t)0x1 <<(2<<2)) |
((uint32_t)0x1 <<(3<<2)) );

```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA2 and PA3 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA2 and PA3 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA2 to push-pull mode (clearing the bit for PA2).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA3.
 - `GPIOA->AFR[0]:` Sets the alternate function for PA2 and PA3 to AF1 (USART1).

UART_MAP == 1: PA7 (TX) and PA8 (RX)

c

Copy code

```

#elif UART_MAP == 1
    // Setup pin PA7 (TX) and pin PA8 (RX)
    RCC->IOPENR |= RCC_IOPENR_GPIOAEN;
    GPIOA->MODER = (GPIOA->MODER & ~( ((uint32_t)0b11<<(7<<1)) |
((uint32_t)0b11<<(8<<1)) ))
| ( ((uint32_t)0b10<<(7<<1)) |
((uint32_t)0b10<<(8<<1)) );
    GPIOA->OTYPER &= ~ ((uint32_t)0b1 <<(7<<0));
    GPIOA->PUPDR = (GPIOA->PUPDR & ~(
((uint32_t)0b11<<(8<<1)) ))
| (
((uint32_t)0b01<<(8<<1)) );
    GPIOA->AFR[0] = (GPIOA->AFR[0] & ~( ((uint32_t)0xf <<(7<<2))
))
| ( ((uint32_t)0x8 <<(7<<2))
);
    GPIOA->AFR[1] = (GPIOA->AFR[1] & ~(
((uint32_t)0xf <<(0<<2)) ))
| (
((uint32_t)0x8 <<(0<<2)) );

```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA7 and PA8 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA7 and PA8 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA7 to push-pull mode (clearing the bit for PA7).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA8.
 - `GPIOA->AFR[0]:` Sets the alternate function for PA7 to AF8.
 - `GPIOA->AFR[1]:` Sets the alternate function for PA8 to AF8.

UART_MAP == 2: PA9 (TX) and PA10 (RX)

C

Copy code

```
#elif UART_MAP == 2
    // Setup pin PA9 (TX) and pin PA10 (RX)
    RCC->IOPENR    |= RCC_IOPENR_GPIOAEN;
    GPIOA->MODER    = (GPIOA->MODER & ~( ((uint32_t)0b11<<(9<<1)) |
    ((uint32_t)0b11<<(10<<1)) ))
                                | ( ((uint32_t)0b10<<(9<<1)) |
    ((uint32_t)0b10<<(10<<1)) );
    GPIOA->OTYPER   &= ~ ((uint32_t)0b1 <<(9<<0));
    GPIOA->PUPDR    = (GPIOA->PUPDR & ~(
    ((uint32_t)0b11<<(10<<1)) ))
                                | (
    ((uint32_t)0b01<<(10<<1)) );
    GPIOA->AFR[1]   = (GPIOA->AFR[1] & ~( ((uint32_t)0xf <<(1<<2)) |
    ((uint32_t)0xf <<( 2<<2)) ))
                                | ( ((uint32_t)0x1 <<(1<<2)) |
    ((uint32_t)0x1 <<( 2<<2)) );
```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA9 and PA10 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA9 and PA10 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA9 to push-pull mode (clearing the bit for PA9).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA10.
 - `GPIOA->AFR[1]:` Sets the alternate function for PA9 and PA10 to AF1 (USART1).

UART_MAP == 3: PA14 (TX) and PA13 (RX)

c

Copy code

```
#elif UART_MAP == 3
    // Setup pin PA14 (TX) and pin PA13 (RX)
    RCC->IOPENR    |= RCC_IOPENR_GPIOAEN;
    GPIOA->MODER    = (GPIOA->MODER & ~( ((uint32_t)0b11<<(14<<1)) |
((uint32_t)0b11<<(13<<1)) ))
                                                    | ( ((uint32_t)0b10<<(14<<1)) |
((uint32_t)0b10<<(13<<1)) );
    GPIOA->OTYPER   &= ~ ((uint32_t)0b1 <<(14<<0));
    GPIOA->PUPDR    = (GPIOA->PUPDR & ~(
((uint32_t)0b11<<(13<<1)) ))
                                                    | (
((uint32_t)0b01<<(13<<1)) );
    GPIOA->AFR[1]   = (GPIOA->AFR[1] & ~( ((uint32_t)0xf <<( 6<<2)) |
((uint32_t)0xf <<( 5<<2)) ))
                                                    | ( ((uint32_t)0x1 <<( 6<<2)) |
((uint32_t)0x8 <<( 5<<2)) );
```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA14 and PA13 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA14 and PA13 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA14 to push-pull mode (clearing the bit for PA14).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA13.
 - `GPIOA->AFR[1]:` Sets the alternate function for PA14 and PA13 to AF1 and AF8, respectively.

UART_MAP == 4: PA14 (TX) and PA15 (RX)

c

Copy code

```
#elif UART_MAP == 4
    // Setup pin PA14 (TX) and pin PA15 (RX)
    RCC->IOPENR    |= RCC_IOPENR_GPIOAEN;
    GPIOA->MODER    = (GPIOA->MODER & ~( ((uint32_t)0b11<<(14<<1)) |
((uint32_t)0b11<<(15<<1)) ))
                                                    | ( ((uint32_t)0b10<<(14<<1)) |
((uint32_t)0b10<<(15<<1)) );
    GPIOA->OTYPER   &= ~ ((uint32_t)0b1 <<(14<<0));
```

```

    GPIOA->PUPDR    = (GPIOA->PUPDR & ~(
((uint32_t)0b11<<(15<<1)) ))
                                | (
((uint32_t)0b01<<(15<<1)) );
    GPIOA->AFR[1]    = (GPIOA->AFR[1] & ~( ((uint32_t)0xf <<( 6<<2)) |
((uint32_t)0xf <<( 7<<2)) ))
                                | ( ((uint32_t)0x1 <<( 6<<2)) |
((uint32_t)0x1 <<( 7<<2)) );

```

1. **Enable GPIOA Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOAEN;` enables the clock for GPIO port A.
2. **Configure PA14 and PA15 as Alternate Function:**
 - `GPIOA->MODER:` Sets PA14 and PA15 to alternate function mode (`0b10`).
 - `GPIOA->OTYPER:` Sets PA14 to push-pull mode (clearing the bit for PA14).
 - `GPIOA->PUPDR:` Enables a pull-up resistor on PA15.
 - `GPIOA->AFR[1]:` Sets the alternate function for PA14 and PA15 to AF1.

UART_MAP == 5: PB6 (TX) and PB7 (RX)

c

Copy code

```

#elif UART_MAP == 5
    // Setup pin PB6 (TX) and pin PB7 (RX)
    RCC->IOPENR    |= RCC_IOPENR_GPIOBEN;
    GPIOB->MODER    = (GPIOB->MODER & ~( ((uint32_t)0b11<<(6<<1)) |
((uint32_t)0b11<<(7<<1)) ))
                                | ( ((uint32_t)0b10<<(6<<1)) |
((uint32_t)0b10<<(7<<1)) );
    GPIOB->OTYPER   &=                                ~ ((uint32_t)0b1 <<(6<<0));
    GPIOB->PUPDR    = (GPIOB->PUPDR & ~(
((uint32_t)0b11<<(7<<1)) ))
                                | (
((uint32_t)0b01<<(7<<1)) );
    GPIOB->AFR[0]   &=                                ~( ((uint32_t)0xf <<(6<<2)) |
((uint32_t)0xf <<(7<<2)) );

```

1. **Enable GPIOB Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOBEN;` enables the clock for GPIO port B.
2. **Configure PB6 and PB7 as Alternate Function:**

- **GPIOB->MODER**: Sets PB6 and PB7 to alternate function mode (0b10).
- **GPIOB->OTYPER**: Sets PB6 to push-pull mode (clearing the bit for PB6).
- **GPIOB->PUPDR**: Enables a pull-up resistor on PB7.
- **GPIOB->AFR[0]**: Clears the alternate function for PB6 and PB7 (likely sets them to AF0).

UART_MAP == 6: PF1 (TX) and PF0 (RX)

c

Copy code

```
#elif UART_MAP == 6
    // Setup pin PF1 (TX) and pin PF0 (RX)
    RCC->IOPENR    |= RCC_IOPENR_GPIOFEN;
    GPIOF->MODER    = (GPIOF->MODER & ~( ((uint32_t)0b11<<(1<<1)) |
((uint32_t)0b11<<(0<<1)) ))
                                                    | ( ((uint32_t)0b10<<(1<<1)) |
((uint32_t)0b10<<(0<<1)) );
    GPIOF->OTYPER   &= ~ ((uint32_t)0b1 <<(1<<0));
    GPIOF->PUPDR    = (GPIOF->PUPDR & ~(
((uint32_t)0b11<<(0<<1)) ))
                                                    | (
((uint32_t)0b01<<(0<<1)) );
    GPIOF->AFR[0]   = (GPIOF->AFR[0] & ~( ((uint32_t)0xf <<(1<<2)) |
((uint32_t)0xf <<(0<<2)) ))
                                                    | ( ((uint32_t)0x8 <<(1<<2)) |
((uint32_t)0x8 <<(0<<2)) );
```

1. **Enable GPIOF Clock:** `RCC->IOPENR |= RCC_IOPENR_GPIOFEN`; enables the clock for GPIO port F.
2. **Configure PF1 and PF0 as Alternate Function:**
 - **GPIOF->MODER**: Sets PF1 and PF0 to alternate function mode (0b10).
 - **GPIOF->OTYPER**: Sets PF1 to push-pull mode (clearing the bit for PF1).
 - **GPIOF->PUPDR**: Enables a pull-up resistor on PF0.
 - **GPIOF->AFR[0]**: Sets the alternate function for PF1 and PF0 to AF8.

Default Warning

c

Copy code

```
#else
    #warning No automatic pin mapping for USART1
```

```
#endif
```

If `UART_MAP` is not defined or doesn't match any of the predefined values, a warning is issued indicating no automatic pin mapping is available for USART1.

UART Setup and Start

c

Copy code

```
// Setup and start UART (8N1, RX/TX, default BAUD rate)
RCC->APBENR2 |= RCC_APBENR2_USART1EN;
USART1->BRR = ((2 * F_CPU / UART_BAUD) + 1) / 2;
USART1->CR1 = USART_CR1_RE | USART_CR1_TE | USART_CR1_UE;
}
```

1. **Enable USART1 Clock:** `RCC->APBENR2 |= RCC_APBENR2_USART1EN;` enables the clock for USART1.
2. **Set Baud Rate:** `USART1->BRR` sets the baud rate. The formula $((2 * F_CPU / UART_BAUD) + 1) / 2$ calculates the value based on the system clock (`F_CPU`) and the desired baud rate (`UART_BAUD`).
3. **Configure USART1:**
 - `USART_CR1_RE`: Enable receiver.
 - `USART_CR1_TE`: Enable transmitter.
 - `USART_CR1_UE`: Enable USART.

UART Read Function

c

Copy code

```
// Read byte via UART
char UART_read(void) {
    while(!UART_available());
    return USART1->DR;
}
```

1. **Wait for Data:** `while(!UART_available());` waits until data is available in the receive data register (`USART1->DR`).
2. **Return Data:** `return USART1->DR;` reads and returns the data from the receive data register.

UART Write Function

c

Copy code

```
// Send byte via UART
void UART_write(const char c) {
    while(!UART_ready());
    USART1->DR = c;
}
```

1. **Wait for Transmit Ready:** `while(!UART_ready());` waits until the transmit data register is empty and ready to send new data.
2. **Send Data:** `USART1->DR = c;` writes the data to the transmit data register.

Summary

This code initializes and configures the UART peripheral on the PY32F0xx microcontroller. It sets up the GPIO pins for alternate function (UART), configures the UART baud rate, enables the UART peripheral, and provides functions to read and write data via UART. The configuration supports multiple pin mappings through conditional compilation based on the `UART_MAP` value.