Device Configuration Parser and Validator

Problem Statement

Build a program that parses and validates device configuration files commonly used in embedded Linux systems. The program should read hardware peripheral configurations (GPIO, I2C, SPI, UART), validate parameters, detect conflicts (like pin collisions), and generate a C header file with initialization macros.

Real-world context: In embedded Linux systems, device tree files or configuration scripts define how hardware peripherals are connected. Before the system boots, these configurations must be validated to prevent hardware conflicts (two devices using the same pin) or invalid settings (addresses out of range).

Problem Requirements

Requirement 1: Define Peripheral Structures

Create structures and enumerations to represent different peripheral types.

What you need to do:

```
// Enumeration for peripheral types
enum PeriphType { GPIO_DEV, I2C_DEV, SPI_DEV, UART_DEV };

// Enumeration for GPIO modes
enum PinMode { INPUT, OUTPUT, ALTERNATE };

// Structure for GPIO configuration
typedef struct {
    int pin;
    enum PinMode mode;
    int pull;  // 0=none, 1=up, 2=down
} GPIOConfig;

// Union to hold different peripheral configs
typedef union {
    GPIOConfig gpio;
    I2CConfig i2c;
    SPIConfig spi;
    UARTConfig uart;
} PeriphConfig;

// Main peripheral structure
typedef struct {
    enum PeriphType type;
    char name[32];
    PeriphConfig config;
    int enabled;
} Peripheral;
```

Requirements:

- Define enums for: PeriphType, PinMode, ErrorCode
- Create typedef structs for: GPIOConfig, I2CConfig, SPIConfig, UARTConfig
- Use a union (PeriphConfig) to efficiently store different config types
- Create main Peripheral struct containing type, name, config, and enabled flag

---

Requirement 2: Parse Configuration File

Read and parse a device configuration file with the following format:

Configuration File Format:

```
# Comments start with #
GPIO_LED: pin=17, mode=OUTPUT, pull=NONE
I2C_SENSOR: addr=0x48, speed=100000
SPI_FLASH: cs=8, clock=1000000, mode=0
UART_DEBUG: tx=14, rx=15, baud=115200
```

Requirements:

- Read configuration from `device_config.txt`
- Parse 4 device types: GPIO, I2C, SPI, UART
- Extract device name and parameters from each line
- Support comments (# prefix) and blank lines
- Store parsed configurations in an array
- Handle both decimal and hexadecimal values (0x prefix)

Functions to implement:

- `parse_config_file(filename)` - Main parser
- `parse_gpio_line(line, peripheral)` - Parse GPIO config
- `parse_i2c_line(line, peripheral)` - Parse I2C config
- `parse_spi_line(line, peripheral)` - Parse SPI config
- `parse_uart_line(line, peripheral)` - Parse UART config
- `trim_whitespace(str)` - Remove leading/trailing spaces

Requirement 3: Validate Configurations and Detect Conflicts

Validate all peripheral configurations and detect hardware conflicts.

Validation Rules:

GPIO:

- Pin number must be 0-31
- Detect if pin already allocated to another device

I2C:

- Address must be 0x08-0x7F
- Detect if address already used by another I2C device

SPI:

- Chip select pin must be 0-31
- Detect if CS pin already allocated

UART:

- TX and RX pins must be 0-31
- Detect if either pin already allocated

Bit Manipulation for Pin Tracking:

```
static unsigned int gpio_pin_map = 0;  // 32-bit bitmask

// Check if pin N is allocated
if (gpio_pin_map & (1 << pin)) {
    // Pin already in use!
}

// Mark pin N as allocated
gpio_pin_map |= (1 << pin);
```

Requirements:

- Use a bitmask (unsigned int) to track allocated GPIO pins
- Use bitwise operations (|, &, <<) for pin allocation tracking
- Implement validator functions for each peripheral type
- Use function pointers to call appropriate validator
- Report clear error messages for conflicts

Functions to implement:

- `validate_peripheral(peripheral)` - Main validator
- `validate_gpio(peripheral)` - GPIO-specific validation
- `validate_i2c(peripheral)` - I2C-specific validation
- `validate_spi(peripheral)` - SPI-specific validation
- `validate_uart(peripheral)` - UART-specific validation
- `check_conflicts()` - Check all peripherals for conflicts

Requirement 4: Generate Output Header File

Generate a C header file (`device_init.h`) with initialization macros.

Generated Header Format:

```
/* Auto-generated device initialization header */

#ifndef DEVICE_INIT_H
#define DEVICE_INIT_H

// GPIO: GPIO_LED
#define GPIO_LED_PIN 17
#define GPIO_LED_MODE 1
#define GPIO_LED_PULL 0

// I2C: I2C_SENSOR
#define I2C_SENSOR_ADDR 0x48
#define I2C_SENSOR_SPEED 100000

// SPI: SPI_FLASH
#define SPI_FLASH_CS_PIN 8
#define SPI_FLASH_CLOCK 1000000
#define SPI_FLASH_MODE 0

// Configuration checksum
#define CONFIG_CHECKSUM 0xAB

#endif
```

Requirements:

- Generate `device_init.h` from parsed configurations
- Create #define macros for each peripheral's parameters
- Include comments identifying each device
- Calculate and include a checksum of the configuration
- Use proper header guards (#ifndef, #define, #endif)

Functions to implement:

- `generate_output_file(filename)` - Generate header file
- `calculate_checksum()` - Calculate config checksum using XOR

Output Specification

1. Console Output

During Execution:

```
=== Device Configuration Parser ===

Parsing configuration file...
Parsed 12 peripherals

Validating peripherals...
```

```
All peripherals validated successfully!

=== Configuration Statistics ===
Total Peripherals: 12
  GPIO devices: 6
  I2C devices: 3
  SPI devices: 2
  UART devices: 1
GPIO pins allocated: 14/32
Configuration checksum: 0xAB
================================

Generated output file: device_init.h

Configuration processing completed successfully!
```

Error Example (if conflict detected):

```
Parsing configuration file...
Parsed 10 peripherals

Validating peripherals...
ERROR: GPIO pin 17 already allocated
Validation failed for GPIO_LED_2

Configuration validation failed
```

2. Generated Header File (`device_init.h`)

```
/* Auto-generated device initialization header */
/* Generated from device configuration */

#ifndef DEVICE_INIT_H
#define DEVICE_INIT_H

// GPIO: GPIO_LED_RED
#define GPIO_LED_RED_PIN 17
#define GPIO_LED_RED_MODE 1
#define GPIO_LED_RED_PULL 0

// GPIO: GPIO_LED_GREEN
#define GPIO_LED_GREEN_PIN 18
#define GPIO_LED_GREEN_MODE 1
#define GPIO_LED_GREEN_PULL 0

// GPIO: GPIO_BUTTON_1
#define GPIO_BUTTON_1_PIN 5
#define GPIO_BUTTON_1_MODE 0
#define GPIO_BUTTON_1_PULL 1
```

```
// I2C: I2C_TEMP_SENSOR
#define I2C_TEMP_SENSOR_ADDR 0x48
#define I2C_TEMP_SENSOR_SPEED 100000

// I2C: I2C_EEPROM
#define I2C_EEPROM_ADDR 0x50
#define I2C_EEPROM_SPEED 400000

// SPI: SPI_FLASH
#define SPI_FLASH_CS_PIN 8
#define SPI_FLASH_CLOCK 1000000
#define SPI_FLASH_MODE 0

// UART: UART_DEBUG
#define UART_DEBUG_TX_PIN 14
#define UART_DEBUG_RX_PIN 15
#define UART_DEBUG_BAUD 115200

// Configuration checksum
#define CONFIG_CHECKSUM 0xA7

#endif // DEVICE_INIT_H
```

3. Sample Input File (`device_config.txt`)

```
# Device Configuration File
# Format: DEVICE_TYPE_NAME: parameter=value, parameter=value

# GPIO Configurations
GPIO_LED_RED: pin=17, mode=OUTPUT, pull=NONE
GPIO_LED_GREEN: pin=18, mode=OUTPUT, pull=NONE
GPIO_BUTTON_1: pin=5, mode=INPUT, pull=UP
GPIO_BUTTON_2: pin=6, mode=INPUT, pull=UP
GPIO_RESET: pin=22, mode=OUTPUT, pull=NONE

# I2C Configurations
I2C_TEMP_SENSOR: addr=0x48, speed=100000
I2C_EEPROM: addr=0x50, speed=400000
I2C_RTC: addr=0x68, speed=100000

# SPI Configurations
SPI_FLASH: cs=8, clock=1000000, mode=0
SPI_ADC: cs=9, clock=500000, mode=1

# UART Configurations
UART_DEBUG: tx=14, rx=15, baud=115200
UART_GPS: tx=12, rx=13, baud=9600
```

Compilation and Execution

Compile:

```
gcc -Wall -Wextra -o device_parser device_parser.c
```

Run:

```
./device_parser
```

Required Files:

- **Input:** `device_config.txt` (device configuration)
- **Output:** `device_init.h` (generated header file)

This assignment provides practical experience with real embedded Linux device configuration workflows!