UART komunikace s DMA, EEPROM na I2C

1 Zadání

- Vytvořte projekt s aktivním rozhraním USART2, realizujte loopback a ověřte funkci.
- Rozšiřte obsluhu USARTu o využití DMA pro příjem v kruhovém módu. Otestujte zpětným výpisem přijatých příkazů **(1,5b)**.
- Vytvořte parser pro jednoduchý textový protokol. Pomocí tohoto protokolu realizujte ovládání LED1 a LED2 na vývojovém kitu **(1b)**.
- Doplňte podporu externí I2C EEPROM paměti. Rozšiřte protokol o funkci čtení a zápisu 8bitového čísla na definovanou adresu v EEPROM paměti **(1b)**. Implementujte příkaz pro vypsání (dump) prvních 16 bajtů externí paměti v hexadecimálním vyjádření včetně odřádkování po osmi bajtech **(0,5b)**.

Příkazy textového protokolu:

- HELLO
- LED1 ON|OFF
- LED2 ON|OFF
- STATUS
- READ adresa
- WRITE adresa hodnota
- DUMP

2 Návod

2.1 Základní seznámení

- Vytvořte si pracovní kopii svého repozitáře z Githubu (Git Clone), příp. aktualizujte repozitář ze serveru (Git Pull).
- Založte nový projekt přes File / New / STM32 Project / Board Selector / NUCLEO-F030R8. Budeme využívat HAL knihovny, proto ponechte Targeted Project Type na STM32Cube. Potvrďte inicializaci všech periferií do výchozího nastavení.

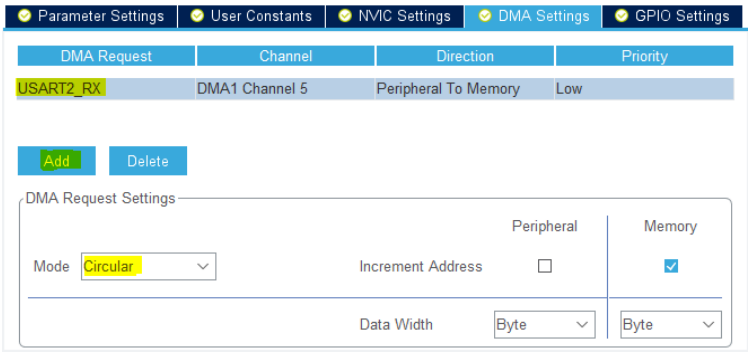
2.2 UART loopback

- Zařízením loopback se rozumí okamžité odeslání každého přijatého bajtu zpět odesílateli. Pro testování sériové komunikace je optimální program Termite, výchozí rychlost portu je 38400 Bd.
- Funkci realizujeme pomocí HAL knihoven v nekonečné smyčce main()u:

```
uint8_t c;
HAL_UART_Receive(&huart2, &c, 1, HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, &c, 1, HAL_MAX_DELAY);
```

2.3 Použití kruhového bufferu s DMA

- Pro rozhraní USART2 nastavte DMA Request pro příjem, využijte kruhový režim (circular).



- Příjem je nejvýhodnější realizovat s využitím kruhového bufferu. Jednotlivé přijímané znaky jsou postupně ukládány do bufferu, po dosažení jeho konce se automaticky pokračuje znovu od začátku. Zápisový index lze tedy vypočítat na základě délky bufferu a registrů DMA přenosu, čtecí index se obsluhuje softwarově.
- Deklarujeme buffer a indexy čtení a zápisu:

```
#define RX_BUFFER_LEN 64
static uint8_t uart_rx_buf[RX_BUFFER_LEN];
static volatile uint16_t uart_rx_read_ptr = 0;
#define uart_rx_write_ptr (RX_BUFFER_LEN - hdma_usart2_rx.Instance->CNDTR)
```

- Nejdříve je třeba aktivovat DMA čtení z UARTu:

```
HAL_UART_Receive_DMA(&huart2, uart_rx_buf, RX_BUFFER_LEN);
```

- V hlavní smyčce kódu pak postupně zpracováváme jednotlivé bajty z bufferu. Dojde-li k zablokování kódu po určitou dobu (např. vlivem zpracování ISR), data se „nahromadí“ v bufferu a zpracují se, jakmile je k dispozici procesorový čas:

```
while (uart_rx_read_ptr != uart_rx_write_ptr) {
    uint8_t b = uart_rx_buf[uart_rx_read_ptr];
    if (++uart_rx_read_ptr >= RX_BUFFER_LEN) uart_rx_read_ptr = 0; // increase read pointer

    uart_byte_available(b); // process every received byte with the RX state machine
}
```

- Funkce **`uart_byte_available()`** ukládá tisknutelné znaky (ASCII kódy 32 až 126) do dalšího bufferu, který již obsahuje poskládaný textový příkaz. Po nalezení konce řádku zavolá obsluhu pro vyhodnocení příkazu:

```
static void uart_byte_available(uint8_t c)
{
    static uint16_t cnt;
    static char data[CMD_BUFFER_LEN];

    if (cnt < CMD_BUFFER_LEN && c >= 32 && c <= 126) data[cnt++] = c;
    if ((c == '\n' || c == '\r') && cnt > 0) {
        data[cnt] = '\0';
        uart_process_command(data);
        cnt = 0;
    }
}
```

- Vhodnou velikost bufferu pro textový příkaz **CMD_BUFFER_LEN** je třeba definovat (např. 256B).
- Do funkce **uart_process_command()** je pro otestování vhodné doplnit zpětný výpis přijatého příkazu:

```
printf("prijato: '%s'\n", cmd);
```

- Aby fungoval výpis pomocí **printf()**, je třeba inkludovat **<stdio.h>** a vytvořit systémovou funkci **_write()**:

```
int _write(int file, char const *buf, int n)
{
    /* stdout redirection to UART2 */
    HAL_UART_Transmit(&huart2, (uint8_t*)(buf), n, HAL_MAX_DELAY);
    return n;
}
```

- Proveďte commit pracovní kopie.

2.4 Obsluha textového protokolu

- Implementuje příkazy HELLO, LED1, LED2 a STATUS.
  - Příkaz HELLO vypíše uvítací text.
  - Příkazy LED1 a LED2 rozsvítí nebo zhasnou jednotlivé LED kontrolky podle údaje ON nebo OFF.
  - Příkaz STATUS vrátí stav obou LED.
- Použijte parsování příkazů pomocí funkce **strtok()** http://www.cplusplus.com/reference/cstring/strtok/, oddělovačem bude znak mezery, inkludujte hlavičkový soubor **<string.h>**. Jednotlivé příkazy rozlišujte se zanedbáním velikosti znaků pomocí funkce **strcasecmp()** http://www.cplusplus.com/reference/cstring/strcmp/.
- Opakovaná volání **strtok()** s prvním parametrem **NULL** vrací další tokeny ve formě C řetězců. Tímto postupným způsobem lze parsovat i komplexní příkazy s mnoha parametry.

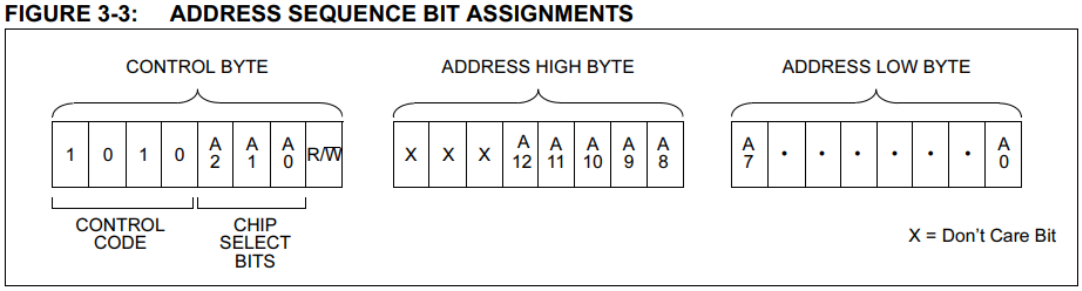
```
char *token;
token = strtok(cmd, " ");

if (strcasecmp(token, "HELLO") == 0) {
    printf("Komunikace OK\n");
}
else if (strcasecmp(token, "LED1") == 0) {
    token = strtok(NULL, " ");
    if (strcasecmp(token, "ON") == 0) ...
    else if (strcasecmp(token, "OFF") ...
    printf("OK\n");
}
else if (strcasecmp...
```

- Využijte funkcí **HAL_GPIO_WritePin()**, **HAL_GPIO_ReadPin()** a formátovaný výpis. Výstupní piny připojené k LED (PA4 a PB0) nastavte v CubeMX.
- Proveďte commit pracovní kopie.

2.5 Externí EEPROM

- EEPROM paměť je připojená na I2C sběrnici, piny PB9 (SDA) a PB8 (SCL) – piny je nutné přemapovat (v pinoutu CubeMX stisknout Ctrl + chytit pin levým tlačítkem myši, nástroj ukáže možná mapování, přetáhnout na cíl). Pro obsluhu povolíme režim I2C periferie I2C1 v konfiguraci CubeMX a následně využijeme funkce z HAL knihoven **`HAL_I2C_Mem_Read()`**, **`HAL_I2C_Mem_Write()`** a **`HAL_I2C_IsDeviceReady()`**.
- Externí EEPROM je typu Microchip 24LC64-I/SN. V datasheetu lze nalézt informace o adrese (1-0-1-0-A2-A1-A0-RW), piny A0 až A2 jsou připojené na GND, adresa **`EEPROM_ADDR`** je tedy 0b10100000 = 0xA0. Dále je třeba dohledat informaci o způsobu adresování – adresa uvnitř EEPROM se skládá z vyššího a nižšího byte, jedná se tedy o 16bitové adresování (konstanta **`I2C_MEMADD_SIZE_16BIT`**).

FIGURE 3-3: ADDRESS SEQUENCE BIT ASSIGNMENTS



- Čtení jednoho bajtu **`value`** z adresy **`addr`**, resp. zápis jednoho bajtu **`value`** na adresu **`addr`**:

```
HAL_I2C_Mem_Read(&hi2c1, EEPROM_ADDR, addr, I2C_MEMADD_SIZE_16BIT, &value, 1, 1000);
HAL_I2C_Mem_Write(&hi2c1, EEPROM_ADDR, addr, I2C_MEMADD_SIZE_16BIT, &value, 1, 1000);
```

- Po každém zápisu do EEPROM je nutné počkat na dokončení operace, např. takto:

```
/* Check if the EEPROM is ready for a new operation */
while (HAL_I2C_IsDeviceReady(&hi2c1, EEPROM_ADDR, 300, 1000) == HAL_TIMEOUT) {}
```

- Implementujte do textového komunikačního protokolu příkazy pro práci s EEPROM:
  - Příkaz READ přečte z paměti zadanou adresu a vypíše její obsah.
  - Příkaz WRITE zapíše na zadanou adresu paměti zadanou hodnotu. Počká na dokončení operace a potvrdí vykonání.
  - Příkaz DUMP hexadecimálně vypíše hodnoty na adresách 0x0000 až 0x000F.
- Pro převod řetězců vzniklých tokenizací na čísla využijte funkci **`atoi()`** z hlavičkového souboru **`<stdlib.h>`**.
- Příklad ověření funkce v terminálu:

```
read 2
Adresa 0x0002 = 0xFF
write 2 50
OK
read 2
Adresa 0x0002 = 0x32
dump
FF FF 32 FF FF FF FF FF
FF FF FF FF FF FF FF FF
```

- Proveďte commit pracovní kopie do Gitu, uložte repozitář pomocí Git Push.