Domača naloga 2

Aplikacija za STM32H7

Jakob Jelovčan

January 14, 2023

1. Uvod

Za drugo domačo nalogo sem za sistem STM32H7 razvil aplikacijo za oddajanje Morse kode. Program iz serijskega vmesnika prebere niz znakov, ga pretvori v Morseovo kodo ter jo odda z utripajočo led diodo. Če uporabnik pritisne na moder gumb, se velike in male črke v nizu zamenjajo, nov niz pa se preko serijske povezave pošlje nazaj na računalnik. Aplikacijo sem napisal v zbirniku in programskem jeziku C.

2. Struktura projekta

Kodo za delo z vhodno/izhodnimi napravami sem napisal v zbirniku, kodo za pretvarjanje in oddajanje Morseovega sporočila pa sem napisal v programskem jeziku C, predvsem zato, ker sem se želel naučiti kako v istem projektu uporabljati oba jezika. Zaradi preglednosti sem kodo glede na njeno funkcionalnost razdelil v več datotek. Funkcije, ki so potrebne za uporabo vhodno/izhodnih naprav so globalne, ostale pomožne funkcije pa so dostopne le znotraj datoteke.

- Main.s
  - Vsebuje glaven program
- GPIO.s
  - Vsebuje kodo za delo z led diodami in gumbom
- SysTick.s
  - Vsebuje kodo za delo s časovnikom
- USART3.s
  - Vsebuje kodo za delo s serijsko povezavo
- Morse.c
  - Vsebuje kodo za pretvarjanje in oddajanje Morseovega sporočila

3. Branje nizov

Za komunikacijo z računalnik sem uporabil serijsko povezavo. Niz se bere po eno črko na enkrat, dokler ni zaznan znak za novo vrstico (0x0D) oziroma zmanjka prostora (maksimalna dolžina niza je 127 znakov). Niz se zaključi s praznim znakom (0x0). Naslov za shranjevanje niza je podan v registru **r0**, maksimalna dolžina niza pa je podana v registru **r1**.

Koda za branje

```
receive_string_usart3:
    push { r4, lr }

    mov r4, r0
receive_string_loop:
    bl receive_char_usart3
    str r0, [r4], #1

    subs r1, #1
    IT ne
    cmpne r0, #0x0D
    bne receive_string_loop

    mov r0, #0
    str r0, [r4, #-1]

    pop { r4, pc }
```

4. Pretvarjanje sporočil

Sporočila, ki jih program prejme prek serijske povezave so zakodirana v ASCII kodiranju. Znaki, ki jih program pretvori v Morse kodo so črke (velike in male), številke in presledek. Ostali znaki so ignorirani. Za pretvarjanje ASCII znakov v Morse kodo sem v jeziku C ustvaril tabelo 36 bajtov, kjer vsak bajt vsebuje Morse kodo za posamezen znak. Prvi trije biti (MSB) vsebujejo dolžino kode, ostalih pet bitov pa predstavlja dejansko kodo, kjer je 0 kratek signal, 1 pa dolg.

Pretvarjanje nizov sem izvedel v jeziku C. Funkcija **string_to_morse** je poklicana iz zbirnika in kot argument dobi naslov niza znakov. Nato vsak niz pretvori v Morse kodo in ga odda z utripajočo led diodo. Led dioda je za kratek signal prižgana 500 ms, za dolg signal pa 1500 ms. Za merjenje časa sem uporabil časovnik SysTick.

5. Zelena led dioda

Za označevanje konca oddajanja Morseovega sporočila sem uporabil zeleno led diodo. Vklop zelene diode je enak kot vklop rdeče diode, razen tega, da se zelena dioda nahaja na nožici 3 **GPIOJ**.

Koda za vklop zelene led diode:

```
.equ    RCC_AHB4ENR,        0x580244E0
.equ    GPIOJ_BASE,         0x58022400
.equ    GPIOx_MODER,        0x00

.type   init_green_led %function
.global init_green_led

init_green_led:
    push { r0, r1, lr }
    ldr r0, =RCC_AHB4ENR
    ldr r1, [r0]
    orr r1, #(1 << 9)
    str r1, [r0]

    ldr r0, =GPIOJ_BASE
    ldr r1, [r0, #GPIOx_MODER]
    and r1, #0xFFFFFFCF
    orr r1, #0x00000010
    str r1, [r0, #GPIOx_MODER]

    pop { r0, r1, pc }

    .type   init_red_led %function
    .global init_red_led
```

6. Modri gumb

Ko program konča z oddajanjem Morseovega sporočila, lahko uporabnik pretvorjen niz pošlje nazaj na računalnik, preko serijske povezave, s pritiskom na modri gumb. Program v neskončni zanki čaka na pritisk na gumb.

Modri gumb se nahaja na nožici 13 **GPIOC**. Za aktivacijo gumba je potrebno vključiti **GPIOC**, nožico 13 nastaviti v **input** način v registru **GPIOx_MODER** ter v **pull down** način v registru **GPIOx_PUPDR**. Nastavitev registra **GPIOx_PUPDR** zagodavlja, da se vrednost v **GPIOx_IDR** registru ponastavi, ko uporabnik spusti gumb. Stanje gumba je na voljo na bitu 13 registra **GPIOx_IDR**.

```
.equ    RCC_AHB4ENR,        0x580244E0
.equ    GPIOC_BASE,         0x58020800
.equ    GPIOx_MODER,        0x00
.equ    GPIOx_PUPDR,        0x0C
.equ    GPIOx_PUPDR_CLEAR,  0xF3FFFFFF
.equ    GPIOx_PUPDR_SET,    0x08000000

.type   init_button %function
.global init_button
init_button:
    push { r0, r1, r2, lr }
    ldr r0, =RCC_AHB4ENR
    ldr r1, [r0]
    orr r1, #0x00000004
    str r1, [r0]

    ldr r0, =GPIOC_BASE
    ldr r1, [r0, #GPIOx_MODER]
    and r1, #0xF3FFFFFF
    str r1, [r0, #GPIOx_MODER]

    ldr r1, [r0, #GPIOx_PUPDR]
    ldr r2, =GPIOx_PUPDR_CLEAR
    and r1, r2
    ldr r2, =GPIOx_PUPDR_SET
    orr r1, r2
    str r1, [r0, #GPIOx_PUPDR]

    pop { r0, r1, r2, pc }


    .type   wait_button %function
    .global wait_button
wait_button:
    push { r0, r1, lr }

    ldr r0, =GPIOC_BASE
loop_button:
    ldr r1, [r0, #GPIOx_IDR]
    tst r1, #(1 << 13)
    beq loop_button

    pop { r0, r1, pc }
```

7. Interakcija zbirnik - C

Glavni razlog za to, da sem v nalogi uporabil programski jezik C je bil ta, da me je zanimalo kako lahko iz jezika C kličem funkcije napisane v zbirniku in obratno.

Klicanje zbirniške funkcije iz C-ja:

1. Zbirniško funkcijo deklariramo kot `.global`
2. V C dodamo `extern` deklaracijo zbirniške funkcije

Klicanje C funkcije iz zbirnika:

1. C funkcijo definiramo kot `extern`
2. V zburniško datoteko dodamo `.global` deklaracijo C funkcije

Argumenti funkcije:

- Če funkcija vrača strukturo, `r0` vsebuje naslov na katerega naj bo struktura shranjena, argumenti 1-3 so v registrih `r1-r3`, ostali pa na skladu
- Sicer so argumenti 1-4 v registrih `r0-r3`, ostali pa na skladu

Rezultat funkcije:

- Če funkcija vrača primitiven tip, je vrednost v `r0`
- Če funkcija vrača strukturo, se njen naslov nahaja v `r0`

Primer:

```
//code_table
struct test_struct
{
    int a, b, c, d, e, f;
};

extern void asm_function1(int, int, int, int, int, int);
extern struct test_struct asm_function3(void);

extern void c_function1(void) {
    asm_function1(1,2,3,4,5,6);
}

extern void c_function2(int a, int b, int c, int d, int e, int f) {

}

extern void c_function3(void) {
    struct test_struct t = asm_function3();
}

extern struct test_struct c_function4(void) {
    struct test_struct t = { 1, 2, 3, 4, 5, 6 };
    return t;
}
```

```
//zbirnik
.type   main %function
.global main

main:
    bl c_function1
    bl asm_function2
    bl c_function3
    bl asm_function4
__end: b __end

.type   asm_function1 %function
.global asm_function1

asm_function1:
    push { r4, r5, lr }

    ldr r4, [sp, #12]
    ldr r5, [sp, #16]

    pop { r4, r5, pc }


.type   asm_function2 %function
.global asm_function2

asm_function2:
    push { r4, lr }
    ldr r0, =1
    ldr r1, =2
    ldr r2, =3
    ldr r3, =4

    ldr r4, =6
    str r4, [sp, #-4]!
    ldr r4, =5
    str r4, [sp, #-4]!

    bl c_function2

    add sp, #8 //Iz sklada je potrebno odstraniti argumente funkcije

    pop { r4, pc }
```

```
.type   asm_function3 %function
.global asm_function3


asm_function3:
    push { r4, lr }

    ldr r4, =1
    str r4, [r0]
    ldr r4, =2
    str r4, [r0, #4]
    ldr r4, =3
    str r4, [r0, #8]
    ldr r4, =4
    str r4, [r0, #12]
    ldr r4, =5
    str r4, [r0, #16]
    ldr r4, =6
    str r4, [r0, #20]

    pop { r4, pc }

.type   asm_function4 %function
.global asm_function4

asm_function4:
    push { r4, r5, r6, lr }

    sub sp, #24
    mov r0, sp

    bl c_function4

    ldr r1, [r0]
    ldr r2, [r0, #4]
    ldr r3, [r0, #8]
    ldr r4, [r0, #12]
    ldr r5, [r0, #16]
    ldr r6, [r0, #20]

    add sp, #24 //Iz sklada je potrebno odstraniti strukturo

    pop { r4, r5, r6, pc }
```

8. Ostala koda

```
#include <ctype.h>
extern void red_led_on(void);
extern void red_led_off(void);
extern void delay_tc(int);
void display_signal(char);
char code_table[] = {0x48, 0x90, 0x94, 0x70, 0x20, 0x84, 0x78, 0x80, 0x40, 0x8E, 0x74, 0x88, 0x58,
     0x50, 0x7C, 0x46, 0x4D, 0x68, 0x60, 0x30, 0x64, 0x82, 0x6C, 0x92, 0x4B, 0x4C, 0xBF, 0xAF,
        0xA7, 0xA3, 0xA1, 0xA0, 0xB0, 0xB8, 0xBC, 0xBE};

extern char char_to_morse(char c) {
    char c_lower = tolower(c);
    if(isdigit(c_lower)) {
        return code_table[c_lower - 22];
    } else if(isalpha(c_lower)) {
        return code_table[c_lower - 97];
    }
    else {
        return 0;
    }
}

extern void short_signal() {
    red_led_on();
    delay_tc(500);
    red_led_off();
    delay_tc(500); //Pause after letter
}

extern void long_signal() {
    red_led_on();
    delay_tc(1500);
    red_led_off();
    delay_tc(500); //Pause after letter
}

extern void space() {
    delay_tc(2500);
}

extern void display_morse(char* str) {
    while(*str) {
        if(*str == ' ') {
            space();
        }
        else {
            char morse = char_to_morse(*str);
            display_signal(morse);
        }
        ++str;
    }
}

void display_signal(char morse) {
    char len = (morse & 0xE0) >> 5;
    morse <<= 3;
```

```
    while(len--) {
        char signal = morse & 0x80;
        morse <<= 1;
        if(signal) short_signal();
        else long_signal();
    }
}

main:
    bl init_tc0_ms
    bl init_usart3
    bl init_red_led
    bl red_led_off
    bl init_green_led
    bl green_led_off
    bl init_button

    ldr r0, =input
    ldr r1, =0x80
    bl receive_string_usart3

    ldr r0, =input
    bl display_morse

    bl green_led_on

    bl wait_button

    ldr r0, =input
    bl change
    ldr r0, =input
    bl send_string_usart3
    bl green_led_off

.type   change %function
change:
    push { r4, lr }

loop:
    ldrb r4, [r0]
    cmp r4, #0x41
    blo skip
    cmp r4, #0x7A
    bhi skip
    cmp r4, #0x5A
    bls switch
    cmp r4, #0x61
    bhs switch
    b skip

    switch:
    eor r4, #(1 << 5)

    skip:
    strb r4, [r0], #1
    cmp r4, #0
    bne loop

    pop { r4, pc }
```