Naloga 3 – UART in PREKINITVE

Pri tej vaji boste spoznali asinhrono serijsko komunikacijo (angl. *UART*) in uporabo prekinitev (angl. *interrupts*). UART je pogosto uporabljen komunikacijski protokol med (običajno) dvema napravama. Iz samega poimenovanja lahko razberemo dve lastnosti: je asinhrona (ne vsebuje ločenega urnega signala za sinhronizacijo) in serijska (poteka bit po bit) komunikacija. Povezava je preprosta in vsebuje zgolj dve liniji: *transmiter* (Tx) in *receiver* (Rx). Več v dokumentu priloženem pri nalogi in v specifikacijah procesorja.

1 Naloge

V tem sklopu imate na voljo več nalog, ki so različno ocenjene (pri vsaki nalogi je zapisano, koliko je največjo število točk, ki jih lahko dobite). Odločitev, katero nalogo boste reševali, je prepuščena vam. Točke se ne seštevajo.

1.1 Pošiljanje niza [največ 1T]

Implementirajte funkciji InitUART(uint32_t hitrost) in SendString(char* niz, uint8_t len), ki bosta izvedli:

- InitUART(uint32_t hitrost): konfiguracijo UART1 na **hitrost/8-N-1** [hitrost je lahko: 9600, 19200 ali 115200]
- SendString(char * niz, uint8_t len): pošiljanje niza znak po znak preko UART1 na računalnik (pošiljanje je lahko narejeno *bločno – program v zanki čaka, da se znak pošlje*).

Na računalniku se bodo podatki sprejemali v programu *RealTerm* (ali kakšnem drugem *terminalu*). Za prikaz delovanja v glavnem programu ustvarite poljuben niz, ki ga potem pošiljate z nekaj sekundnim zamikom (časovni zamik lahko naredite s preprosto zanko).

1.2 Ukazi preko konzole [največ 3T]

Aplikacija bo sprejemala znake in jih shranjevala v polje znakov (skupaj tvorijo ukaz). Ukaz/vnos se zaključi z znakom '\n' (*new line)* ali ko uporabnik vnese 10 znakov (kaj se zgodi prej). Po koncu pošiljanja aplikacija izvede zahtevano akcijo in uporabniku vrne povratno sporočilo ter čaka na nov ukaz. Prejemanje je potrebno implementirati z uporabo **prekinitev**, kjer se sproži prekinitvena rutina vsakič, ko aplikacija prejme en znak.

Seznam ukazov, ki jih lahko uporabnik pošlje je prikazan v spodnji tabeli v prvem stolpcu. Drugi stolpec vsebuje opis akcije, ki jo mora aplikacija izvesti, tretji stolpec pa povratno sporočilo, ki ga bo aplikacija poslala nazaj na računalnik, ko izvede ukaz.

Ukaz	Opis (Akcija)	Povratno sporočilo
»L13o\n«	Vključi se LED na PG13	»Green LED ON\r\n«
»L13f\n«	Izključi se LED na PG13	»Green LED OFF\r\n«
»L14o\n«	Vključi se LED na PG14	»Red LED ON\r\n«
»L14f\n«	Izključi se LED na PG14	»Red LED OFF\r\n«
Karkoli drugega	/	»Neznan ukaz\r\n«

Za *razpoznavo* ukazov lahko uporabite funkcijo strcmp (String.h).

1.3 Shranjevanje podatkov v krožna vrsto [največ 5T]

Pri tej nalogi je potrebno implementirati prejemanje znakov z uporabo prekinitev in njihovo shranjevanje v t.i. *krožno vrsto* (program si shranjuje znak v *buffer* dolžine *N* znakov). Ko program prejme več kot *N* znakov, začne prepisovati *najstarejše podatke*).

Dodajte tudi konfiguracijo tipke (**PA0**), ki bo ob pritisku sprožila prekinitev (več o tem najdete v dokumentaciji – podpoglavji 8.3.8 in 12.2). V prekinitveni rutini program preko UART-a pošlje najstarejši znak (samo enega) v *vrsti*.

Vrednost *N* definirajte z uporabo stavka *define.*

Nalogo je mogoče rešiti na obeh razvojnih ploščicah:

- *STM32F429*: PC (preko terminala) pošilja podatke, ki jih potem sprejemate z USART1 (PA9 in PA10). Ob kliku na tipko se pošlje najstarejši znak nazaj na PC.
- *STM32F411*: Pri tej rešitvi bo potrebno malo domišljije 😊. Potrebno je narediti konfiguracijo dveh USART-ov, in sicer: USART1 in USART6, ki ju med seboj povežete:

  USART6_TX (PC6) -> USART1_RX (PA10)
  USART6_RX (PC7) <- USART1_TX (PB6)

Ideja za STM32F411: USART1 konfigurirajte kot tistega, ki bo sprejemal podatke in jih shranjeval v vrsto in USART6 kot tistega, ki bo pošiljal podatke in prejemal podatke ob kliku na tipko. Moj predlog bi bil sledeč: v neskončni znaki se vsakih *n* sekund (delay z zankami) pošlje en znak [recimo, da

pošljete celotno abecedo] ali pa niz preko USART6 na USART1, kjer se znak/niz shrani v vrsto.

```
int count = 0;
for(;;){
    for(int i = 0; i < 1000; i++)
        for(int j = 0; j < 10000; j++)
            asm("nop");


    //POŠLJI ('A' + (++count%25))

}
```

2 Dodatni pogoji

Pri implementaciji upoštevajte naslednje:

- Dovoljena je uporaba knjižnice *stm32f4xx.h*, ki je naložena na sistemu eŠtudij in *string.h* za primerjavo nizov.
- **Uporaba Cube generatorja kode ni dovoljena!**
- Kodo razdelite v ločene datoteke

3 Konfiguracija priključkov

Glede na izbiro implementacijo boste morali v dokumentaciji plošče poiskati, katere priključke (pine) uporabljajo izbrani UART-i (npr. USART1 na STM32F429 Discovery, ki je povezan na USB, uporablja **PA9** in **PA10**). To so GPIO priključki, ki imajo strojno podporo še dodatno (*alternate function*) funkcionalnost. (Seznam alternativnih funkcij na posameznih priključkih vidite v dokumentu Dodaten opis plošče STM32F429I v tabeli 10). Pri nastavitvah GPIO priključkov jih je potrebno konfigurirati kot *alternate function mode* v registru MODER ter v registra AFRL oziroma AFRH zapisati vrednosti AFn, ki jih najdete na sliki na strani 275. V registra morate zapisati n, ki pripada izbranemu USART-u.

4 Konfiguracija UART-a

Glede konfiguracije in opisa registrov imate pri nalogi priložen dokument s kratkim opisom UART protokola. V poglavjih 30.3.1 do 30.3.5 specifikacij procesorja je zapisano, katere registre je potrebno nastaviti za pošiljanje ter za prejemanje enega znaka. UART lahko »pošlje« sočasno samo en znak, tako da je potrebno za pošiljanje niza postopek pošiljanja postaviti v zanko.

V zadnji nalogi je potrebno prejemanje podatkov rešiti z uporabo prekinitev. Nekaj o tem je zapisano v specifikacijah, nekaj osnovnih idej in nasvetov:

- V registru CR1 je potrebno izbrati ustrezen »interrupt request« [v nalogi je potrebno prekinitve narediti za prejemanje podatkov]
- V datoteki »stm32f4xx.h« je funkcija NVIC_EnableIRQ, ki jo morate uporabiti [poiščite ustrezen parameter v enumu definiranem zgoraj]
- V datoteki »startup_stm32f411vetx.s« poiščite v vektorski tabeli ustrezno (ustrezni) IRQHandler (za vsak UART enega). Ta zapis/labela je potem ime funkcije [void, (void)], ki jo definirate v main.c.
- Obstaja »več« pravil, kako pisati prekinitvene rutine. Moj predlog je ta: prekinitvena rutina (ISR) naj bo čim krajša. Preberite znak in ga shranite v buffer. Če je potrebno v ISR storiti še kaj drugega, potem postavite *flag*. Ta flag potem preverjate v neskončni zanki v mainu. Ko program *obdela* kar bi moral, postavite ta *flag* nazaj na 0.

5 Navodila za oddajo

Pred oddajo naloge je potrebno v projekt razvojnem okolju **najprej očistiti** (v meniju izberite možnost Clean Project), nato pa **ga združiti v datoteki formata ZIP**.