VJEŽBA 3: STM32F407 UART SUČELJE. RUKOVANJE IZNIMKAMA.

I. Cilj vježbe

Koristiti UART periferiju mikroupravljača za komunikaciju s osobnim računalom. Upoznati se s iznimkama na mikroupravljaču STM32F407VG. Upotreba različite periferije mikroupravljača kao izvora prekida: GPIO pin, *SysTick* brojač, A/D pretvornik i USART.

II. Opis vježbe

Na laboratorijskim vježbama koristi se STM32F4 Discovery razvojna ploča kao hardverska komponenta dok se Atollic TrueStudio koristi kao pripadno integrirano razvojno okruženje.. U vježbi se studeni upoznavanju s USART periferijom koja omogućava serijsku komunikaciju mikroupravljača s drugim uređajima kao npr. s osobnim računalom. Nadalje, studenti se upoznaju s iznimkama te njihovim rukovanjem. Za demonstraciju rukovanja s iznimkama kao izvor prekida koristi se GPIO pin na koji je spojeno tipkalo, *SysTick* brojač vremena, A/D pretvornik te UART sučelje.

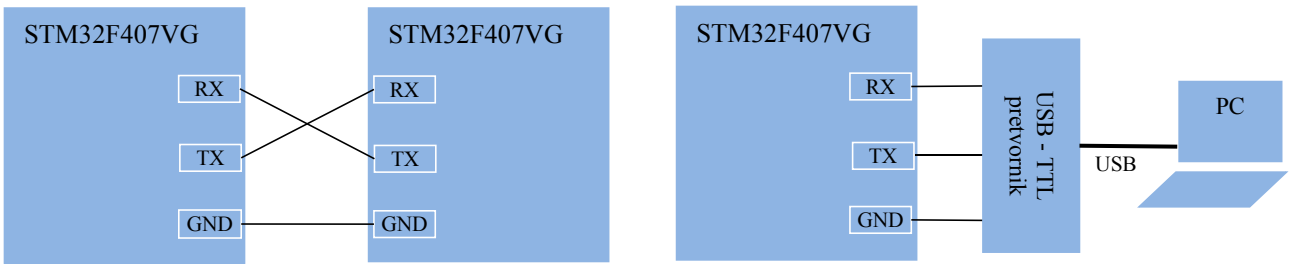
Važno. STM32F407 Discovery razvojna ploča ima eksterni oscilator iznosa 8 MHz. Provjerite u datoteci `stm32f4xx.h` vašeg C projekta kolika je vrijednost konstante HSE_VALUE te ju po potrebi izmijenite u 8000000. Također u datoteci `system_stm32f4xx.c` postavite vrijednost PLL_M na 8. Ako ove vrijednosti nisu dobro postavljene, sva periferija koja izravno ovisi o točnoj vrijednosti korištenog oscilatora neće ispravno raditi.

II.1. USART

U(S)ART (engl. Universal synchronous asynchronous receiver transmitter) je periferija mikroupravljača koja omogućava serijsku full-duplex komunikaciju mikroupravljača s različitim uređajima, poput osobnog računala, GPS modula, različitih senzora i sl. U(S)ART se sastoji od prijemnika i predajnika koji imaju odgovarajuće pinove na mikroupravljaču koji se označavaju s RX (pin za prijem podataka) i TX (pin za slanje podataka) i preko kojih se odvija komunikacija serijski. Generalno, slanje podataka odvija se zapisivanjem vrijednosti u podatkovni registar predajnika, dok se primanje podataka odvija čitanjem podatkovnog registra prijemnika. Konfiguracija U(S)ART-a se, obavlja na sličan način kao i kod ostale periferije mikroupravljača - podešavanjem vrijednosti u odgovarajućim konfiguracijskim registarima U(S)ART-a. Glavne odlike U(S)ART-a mikroupravljača STM32F407VG su sljedeće:

- full duplex asinkrona komunikacija,
- programirljiva veličina riječi (8 ili 9 bitova), broj stop bitova (1 ili 2 bita), paritet i brzina prijenosa (bitova po sekundi ili baud),
- hardverska kontrola toka (linije CTS i RTS),
- 10 izvora prekida (prijemni registar pun, završen prijenos, greška pariteta i sl.),
- mogućnost sinkrone komunikacije (takt CK),
- mogućnost zasebne aktivacije prijemnika i predajnika,
- podrška za DMA.

Na slici 3.1 a) prikazan je primjer spajanja dva mikroupravljača STM32F407VG preko UART periferije u asinkronom načinu rada bez hardverske kontrole toka. Kako bi se komunikacija pravilno odvijala, osim unakrsnog spajanja linija RX i TX potrebno je još povezati i GND na oba mikroupravljača. Na slici 3.1 b) prikazan je način spajanja mikroupravljača STM32F407VG na osobno računalo koje nema serijski port. U tom slučaju je potrebno koristiti pretvornik USB na TTL.



Sl. 3.1 Komunikacija pomoću UART-a a) između dva mikroupravljača, b) između mikroupravljača i osobnog računala.

Najčešća konfiguracija UART-a je tzv. "bps, 8, N, 1" pri čemu je:
- bps - brzina prijenosa u bitovima po sekundi (npr. 9600 bps)
- 8 - broj bitova koji se koristi po znaku
- N - komunikacija bez pariteta
- 1 - broj stop bitova

Mikroupravljač STM32F407VG ima ukupno 6 U(S)ART-a (USART1, USART2, USART3, UART4, UART5) koji su dostupni na odgovarajućim GPIO pinovima kao što je dano u tablici 3.1 (izdvojeno iz tablice 7 STM32F407 datasheet). Primijetite da 4. i 5. podržavaju samo asinkronu komunikaciju. USART1 i USART6 imaju maksimalnu brzinu prijenosa od 10.5 Mbit/s, a preostali maksimalnu brzinu od 5.25 Mbit/s. Prilikom korištenja USART-a, potrebno je pin konfigurirati tako da ima alternativnu funkciju.

Tablica 3.1 GPIO pinovi koji podržavaju U(S)ART periferiju

U(S)ART	TX	RX	CTS	RTS	CK
USART1	PA9 PB6	PA10 PB7	PA11	PA12	PA8
USART2	PA2 PD5	PA3 PD6	PA0 PD3	PA1 PD4	PA4 PD7
USART3	PB10 PD8 PC10	PB11 PD9 PC11	PB13 PD11	PB14 PD12	PB12 PD10 PC12
UART4	PA0 PC10	PA1 PC11	-	-	-
UART5	PC12	PD2	-	-	-
USART6	PC6	PC7	-	-	PC8

Rad s UART-om na mikroupravljaču STM32F407VG podrazumijeva nekoliko koraka. U tablici 3.2. dan je primjer podešavanja i korištenja UART1 pomoću Standard Peripheral Library pri čemu se koriste GPIO pinovi PB6 i PB7 za primanje i slanje podataka. Najprije je potrebno uključiti takt za USART1 i port B, te se nakon toga konfiguriraju pinovi PB6 i PB7 odnosno koristi se njihova alternativna funkcija. Nadalje potrebno je "povezati" pinove PB6 i PB7 s njihovom ulogom na USART1 (TX odnosno RX) pomoću funkcije `GPIO_PinAFConfig`. Nakon toga se podešava UART1 pri čemu se koristi konfiguracija "9600, 8, N, 1". Na kraju se omogućava rad UART1 periferije pomoću funkcije `USART_Cmd`.

Tablica 3.2 Podešavanje UART1, slanje i primanje podataka pomoću SPL.

```
GPIO_InitTypeDef GPIO_InitStruct;              //inicijalizacijska struktura za GPIO
USART_InitTypeDef USART_InitStruct;            //inicijalizacijska struktura za USART

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);   //uključi takt za port B
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);  //uključi takt za USART1

// koristi se alternativna funkcija GPIO pinova PB6 i PB7
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_Init(GPIOB, &GPIO_InitStruct);

GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);       //PB6 se koristi kao USART1_Tx
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);       //PB7 se koristi kao USART1_Rx

USART_InitStruct.USART_BaudRate = 9600;                         //brzina prijenosa 9600 bps
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None; //ne koristi se CTS i RTS
USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;    //koristi se i prijemnik i predajnik
USART_InitStruct.USART_Parity = USART_Parity_No;                //komunikacija bez pariteta
USART_InitStruct.USART_StopBits = USART_StopBits_1;             //1 stop bit
USART_InitStruct.USART_WordLength = USART_WordLength_8b;        //digitalna riječ je veličine 8 bita
USART_Init(USART1, &USART_InitStruct);                          //zapisivanje UART1 postavki

USART_Cmd(USART1, ENABLE);                                      //omogući rad UART1 periferije
```

Slanje i primanje podataka moguće je obaviti korištenjem SPL funkcija `USART_ReceiveData` i `USART_SendData` pri čemu se zapisuje odnosno čita vrijednost u USART podatkovnom registru (`USART_DR`). Prilikom slanja i primanja podataka preporuča se provjeriti stanje podatkovnog registra za slanje (`TDR`) odnosno stanje podatkovnog registra za primanje podataka (`RDR`). Ove informacije moguće je dohvatiti iz USART statusnog registra (`USART_SR`). Ako je vrijednost zastavice `TXE` jednaka 1, tada je `TDR` prazan i moguće je u njega upisivati nove

podatke za slanje. Ako je vrijednost zastavice `RXNE` jednaka 1, tada je `RDR` registar prebačen u podatkovni `USART_DR` registar i moguće je pročitati primljeni podatak. Zapisivanjem vrijednosti u `USART_DR` registar automatski se postavlja `TXE` na 0 odnosno čitanjem vrijednosti iz `USART_DR` automatski se postavlja `RXNE` na 0. Primjer korištenja ovih zastavica prilikom slanja odnosno primanja znaka dan je u tablici 3.3 gdje se koristi princip prozivke UART-a (engl. *polling*) za primanje odnosno slanje podataka. Više detalja o USART periferiji moguće je pronaći u RM0090 Reference manual od stranice 968.

Tablica 3.3 Slanje i primanje podataka pomoću UART-a korištenjem SPL.

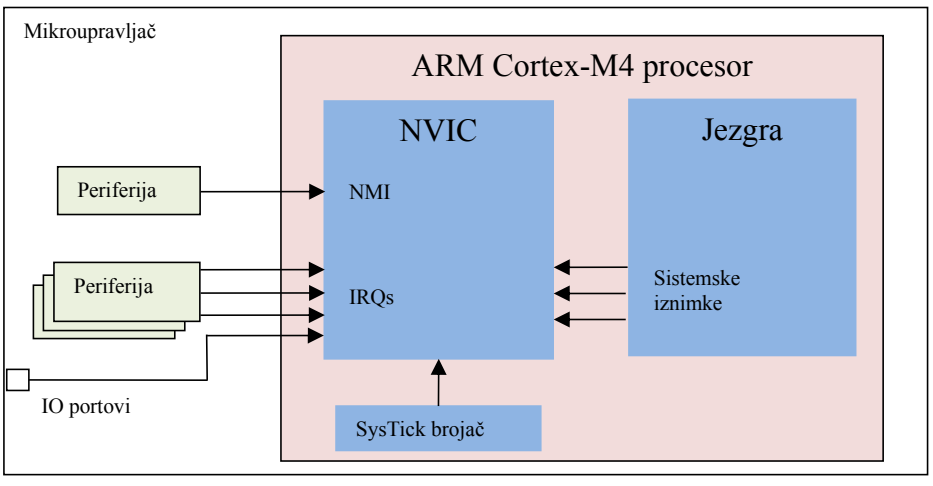
```
uint16_t c;

while (!USART_GetFlagStatus(USART1, USART_FLAG_TXE));    //čekaj dok TDR nije prazan
USART_SendData(USART1, c);                               //pošalji podatak c

while (!USART_GetFlagStatus(USART1, USART_FLAG_RXNE));   //čekaj dok se primi podatak
c = USART_ReceiveData(USART1);                           //pročitaj primljeni podatak na USART1
```

II.2. Iznimke

STM32F407VG mikroupravljač zasniva se na ARM Cortex-M4 jezgri koja podržava niz različitih iznimki. Generalno, iznimke su događaji koji prekidaju normalan tijek programa i usmjeravaju procesor na izvođenje odgovarajućeg dijela koda koji se naziva rukovatelj iznimkom (engl. *exception handler*). Dio mikroupravljača zadužen za upravljanje iznimkama kod ARM Cortex-M4 mikroupravljača naziva se NVIC (engl. *Nested Vectored Interrupt Controller*) i blokovski je prikazan na slici 3.2. Iznimke imaju prioritet što znači da iznimka većeg prioriteta može prekinuti izvođenje iznimke nižeg prioriteta. Neke iznimke imaju fiksni prioritet, a nekima ga je moguće softverski dodijeliti. Iznimke se mogu podijeliti u dvije glavne kategorije:

1) Sistemske iznimke u koje pripadaju:
   a) *NMI* - nemaskirajući zahtjev (engl. *Non-maskable Interrupt*) za prekidom koji može generirati periferija ili vanjski izvori,
   b) Pogreške (engl. *faults*) - abnormalne pojave koje detektira procesor bilo interno ili tijekom komunikacije s memorijom ili ostalim uređajima (*Usage fault*, *Bus fault*, *Memory management fault*, *Hard fault*),
   c) *SVCall* - iznimka koju generira SVC instrukcija (engl. *supervisor call*),
   d) *Debug Monitor* - iznimke koje se generiraju kod softverskog otklanjanja pogreški, npr. zbog korištenja *breakpoints-a,*
   e) *PendSV* - iznimka koja se generira samo softverski i često se koristi od strane OS-a, te
   f) *SysTick* - iznimka koju periodički generira 24-bitni brojač vremena.

2) Zahtjevi za prekidom (engl. *Interrupt ReQuests* - IRQs) su iznimke koje generira periferija kako bi signalizirala procesoru da se pojavio događaj vezan za periferiju (npr. podatak je pristigao na serijsko sučelje, A/D pretvornik je završio s pretvorbom i sl.). IRQ može generirati i software i u tom slučaju se naziva softverski IRQ. Svi ovi zahtjevi su maskirajući zahtjevi za prekidom što znači da se moraju softverski omogućiti prije upotrebe.



Sl. 3.2 NVIC i izvori iznimki.

Iznimke koje postoje na mikroupravljaču STM32F407VG su dane u tablici 3.4 gdje je za svaku iznimku dan njen prioritet, akronim, adresa u vektorskoj tablici (sadrži početnu adresu rukovatelja iznimke) te kratki opis iznimke. NVIC upravlja sa 16 sistemskih iznimki i 82 maskirajuća IRQ koji potječu od periferije (UART, CAN, SPI, brojači vremena i sl.). Više detalja o iznimkama moguće je pronaći u RM0090 Reference manual od stranice 371.

Tablica 3.4 Iznimke na mikroupravljaču STM32F407VG.

Broj iznimke	Prioritet	Akronim	Adresa
1	-3	Reset	0x0000 0004
2	-2	NMI	0x0000 0008
3	-1	HardFault	0x0000 000C
4	podesiv	MemManage	0x0000 0010
5	podesiv	BusFault	0x0000 0014
6	podesiv	UsageFault	0x0000 0018
7-10	NA	Rezervirano	
11	podesiv	SVCall	0x0000 002C
12	podesiv	Debug Monitor	0x0000 0030
13	NA	Rezervirano	
14	podesiv	PendSV	0x0000 0038
15	podesiv	SysTick	0x0000 003C
16	podesiv	WWDG	0x0000 0040
17	podesiv	PVD	0x0000 0044
	...		
22	podesiv	EXTI0	0x0000 0058
	...		
34	podesiv	ADC	0x0000 0088
	...		
96	podesiv	HASH_RNG	0x0000 0180
97	podesiv	FPU	0x0000 0184

Svaka iznimka ima vrijednost prioritet pri čemu manja vrijednost označava veći prioritet prilikom izvođenja rukovatelja iznimke. Kod mikroupravljača STM32F407VG *RESET*, *NMI* i *HardFault* imaju fiksan prioritet dok se ostalim iznimkama može podesiti vrijednost prioriteta. Za implementaciju razina priorieta se koriste 4 najznačajnija bita 8-bitnog Interrupt Priority Registra (IPR). Na taj način omogućeno je do 16 različitih razina prioriteta. Dodatno, ovih 4 bita je moguće podijeliti u:

1) bitovi za definiciju razina grupnog prioriteta (engl. *group priority, pre-emption priority*),
2) bitovi za definiciju prioriteta na razini grupe (engl. *subpriority*).

Grupna razina prioriteta definira može li zahtjev za prekidom prekinuti izvođenje nekog drugog rukovatelja iznimke. Prioriteta na razini grupe definira koji rukovatelj iznimke će se prvi izvoditi ako se pojave dva zahtjeva za prekidom s istim grupnim prioritetom. Ovo se tipično događa kada neki rukovatelj iznimke traje relativno dugo. Količinu bitova koji se koriste za definiranje razina grupnog prioriteta i za definiranje prioriteta na razini grupe moguće je podesiti.

Kako bi NVIC mogao primati zahtjeve za prekidima, on se najprije mora konfigurirati te se mora omogućiti njegov rad. Kako bi se olakšao rad s NVIC i iznimkama, CMSIS biblioteka pruža definicije registara i različite pomoćne funkcije koje se mogu pronaći unutar datoteke `core_cm4.h`. Najprije je potrebno podesiti željenu raspodjelu između grupnog prioritera i subprioriteta. U tablici 3.5. dana je oznaka načina grupiranja koja se predaje funkciji `NVIC_SetPriorityGrouping` kako bi se postigla željena raspodjela.

Tablica 3.5 Moguće raspodjele prioriteta.

Oznaka načina grupiranja	Broj grupnih prioriteta	Broj prioriteta na razini grupe
0,1,2,3	16	1
4	8	2
5	4	4
6	2	8
7	N/A	16

U tablici 3.6 dan je konkretan primjer korištenja dostupnih funkcija. U danom primjeru koriste se 2 bita za definiciju razine grupnog prioriteta i 2 bita za definiciju prioriteta na razini grupe. Nadalje, iznimke `ADC_IRQn` i `USART1_IRQn` imaju grupni prioritet jednak 0 dok iznimka `EXTI0_IRQn` ima grupni prioritet 1 što znači da ne može prekinuti izvođenje rukovatelja za prekide `ADC_IRQn` i `USART1_IRQn` dok će ove dvije iznimke prekinuti izvođenje

rukovatelja prekida `EXTI0_IRQn`. U slučaju da se pojave u isto vrijeme iznimke `ADC_IRQn` i `USART1_IRQn`, najprije će se izvoditi rukovatelj iznimke `ADC_IRQn` zbog većeg prioriteta na razini grupe.

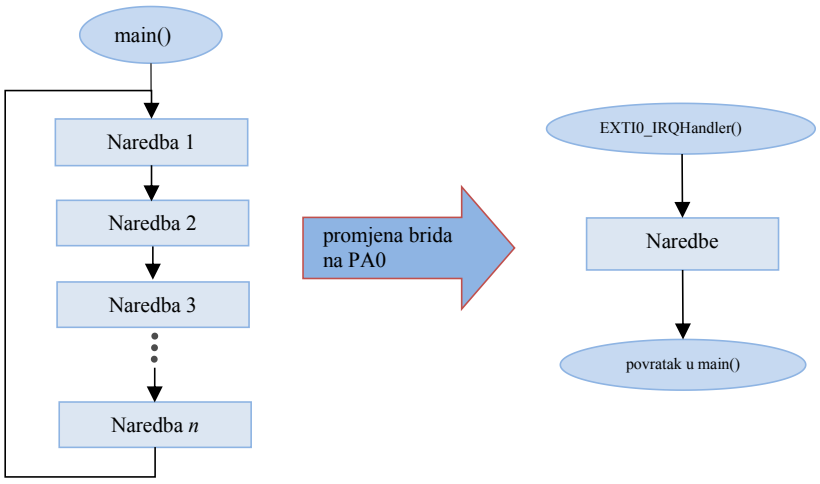
Tablica 3.6 Podešavanje razina prioriteta.

```
//podesi raspodjelu bitova prioriteta: 2 bita za pre-emption priority, 2 bita za subpriority
NVIC_SetPriorityGrouping(5u);

//postavi prioritet pojedinog prekida
NVIC_SetPriority(ADC_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,0));
NVIC_SetPriority(USART1_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,1));
NVIC_SetPriority(EXTI0_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),1,0));

//omogući prekide
NVIC_EnableIRQ(ADC_IRQn);
NVIC_EnableIRQ(USART1_IRQn);
NVIC_EnableIRQ(EXTI0_IRQn);
```

II.2.1. GPIO sučelje kao izvor prekida

Promjena naponskog signala (rastući ili padajući brid) na nekom GPIO pinu može biti izvor tzv. vanjskog prekida (engl. *external interrupt* - EXTI). Dio mikroupravljača koji je zadužen za interpretaciju signala na GPIO sučelju te proslijeđivanje NVIC-u je EXTI kontroler (engl. *External interrupt/event controller*). Sastoji se od 23 detektora bridova koji generiraju zahtjeve za prekidom. Svaki se detektor može zasebno konfigurirati (detekcija rastućeg brida, padajućeg brida ili oboje). Kod mikroupravljača STM32F407VG ukupno 140 GPIO pinova je spojeno na 16 detektora (npr. na EXTI0 spojeni su PA0, PB0, ..., PI0).

Za potrebe ilustracije rada s vanjskim prekidima koristi se ugradbeno tipkalo na Discovery razvojnoj ploči koje je spojeno na pin PA0. Pritisak na ugradbeno tipkalo (tj. promjena brida na određenom GPIO pinu) treba uzrokovati vanjski prekid koji će prekinuti normalno izvođenje glavnog programa te će se započeti izvoditi odgovarajuća prekidna rutina (engl. *Interrupt Service Routine* - ISR) kao što je prikazano na slici 3.3.



Prekidna rutina nema povratnu vrijednost niti prima argumente.

Ako se žele razmijenjivati vrijednosti, onda je potrebno koristiti varijable datotečnog dosega. Koristite modifikator `volatile`!

Programski kod u prekidnim rutinama treba biti što kraći.

Sl. 3.3 GPIO pin PA0 kao izvor prekida.

Osnovni softverski princip prilikom rada s vanjskim prekidima sastoji se od nekoliko koraka. U slučaju generiranja prekida preko pina PA0 to su:

1) Konfigurirati PA0 pin kao ulazni,
2) Omogućiti SYSCFG takt (takt sistemskog konfiguratora) koji je potreban za podešavanje EXTI linija,
3) Za EXTI0 definirati pin PA0 kao izvorni pomoću sistemskog konfiguratora,
4) Omogućiti EXTI0 prekid,
5) Definirati generira li se prekid u slučaju rastućeg brida, padajućeg brida ili u oba slučaja,
6) Postaviti prioritet za EXTI0 u NVIC,
7) Omogućiti IRQ za EXTI0 u NVIC,
8) Napisati odgovarajuću ISR, tj. `void EXTI_0(void){}`. Unutar ISR obrisati zastavicu prekida koja je postavljena od strane hardwarea.

Odgovarajući programski kod temeljen na CMSIS biblioteci dan je u tablici 3.7. Na sličan način moguće je konfigurirati i ostale GPIO pinove koji su spojeni na druge EXTI linije, kao npr. `PB2` i sl.

Tablica 3.7 Podešavanje vanjskog prekida za pin PA0 i odgovarajuća prekidna rutina.

```
#include "stm32f4xx.h"

GPIO_InitTypeDef GPIO_InitButn;         //inicijalizacijska struktura za tipkalo
EXTI_InitTypeDef EXTI_InitStruct;       //inicijalizacijska struktura za EXTI kontroler
NVIC_InitTypeDef NVIC_InitStruct;       //inicijalizacijska struktura za NVIC

//PA0 se koristi kao ulazni pin
GPIO_InitButn.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitButn.GPIO_Pin = GPIO_Pin_0;
GPIO_InitButn.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitButn.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &GPIO_InitButn);

RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);  //uključivanje takta za sistemski konfigurator
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);  //PA0 se koristi kao izvor za EXTI0

EXTI_InitStruct.EXTI_Line = EXTI_Line0;                 //EXTI linija 0 se konfigurira
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;        //prekidni način
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;    //padajući brid uzrokuje prekid
EXTI_InitStruct.EXTI_LineCmd = ENABLE;                  //omogućavanje EXTI prekida
EXTI_Init(&EXTI_InitStruct);                            //zapisivanje postavki EXTI konfiguracije

//podesi raspodjelu bitova prioriteta: 4 bita za pre-emptive priority, 0 bita za subpriority
NVIC_SetPriorityGrouping(0u);

//postavi prioritet prekida EXTI0_IRQn na 1
NVIC_SetPriority(EXTI0_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),1,0));

//omogući EXTI0_IRQn
NVIC_EnableIRQ(EXTI0_IRQn);


void EXTI0_IRQHandler(void)
{
     /* kod prekidne rutine
     ...
     ...
     */

     //brisanje zastavice prekida (postavljanje odgovarajućeg bita na 0 u EXTI_PR registru)
     EXTI_ClearITPendingBit(EXTI_Line0);
}
```

II.2.2. SysTick brojač vremena

ARM Cortex-M4 mikroupravljači sadrži 24-bitni sistemski brojač vremena naziva *SysTick* koji broji prema "dolje" od određene vrijednosti (`RELOAD`) s odgovarajućim taktom. `RELOAD` vrijednost se automatski učita u brojač kada brojač dostigne vrijednost 0. Na taj način *SysTick* brojač vremena generira prekide u regularnim intervalima. Ovaj brojač se obično koristi kod OS za *context switching* kako bi se omogućila višezadaćnost. Međutim, u aplikacijama koje ne zahtijevaju OS ovaj se brojač može koristiti za izvršavanje određene zadaće u definiranom vremenskom intervalu ili za mjerenje (proteklog) vremena.

STM32F407VG mikroupravljač koji se nalazi na Discovery razvojnoj ploči može imati različite izvore takta koji se koriste za sistemski takt. Uz pretpostavku da je sistemski takt iznosa 168 MHz (može se vidjeti u datoteci `system_stm32f4xx.c`), vrijednost 24-bitnog registra SysTick brojača smanjuje se svakih 1/(168 MHz) ≈ 5.95 ns (često se naziva i otkucaj (engl. tick)). Postavljanjem odgovarajuće RELOAD vrijednosti SysTick brojača, moguće je generirati prekid u željenim vremenskim intervalima.

Podešavanje *SysTick* brojača vremena moguće je pomoću `SysTick_Config` funkcije koja se nalazi u `core_cm4.h` datoteci. Ova funkcija prima broj otkucaja između dva prekida. Definicija odgovarajuće prekidne rutine nalazi se u datoteci `stm32f4xx_it.c` pod nazivom `SysTick_Handler`. Funkcija `SysTick_Config` također automatski podešava prioritet *SysTick* prekida te ga omogućuje.

U tablici 3.8 dan je primjer podešavanja Systick brojača vremena. Prekid se želi generirati svake milisekunde. Stoga, funkciji `SysTick_Config` se predaje cjelobrojna vrijednost tj. broj otkucaja koji odgovara umnošku željenog perioda i sistemske frekvencije (1 ms * 168 MHz = 168000).

Tablica 3.8 Podešavanje *SysTick* brojača i odgovarajući rukovatelj.

```
#include "stm32f4xx.h"

//SysTick prekid svake milisekunde
SysTick_Config(168000);

void SysTick_Handler(void)
{
      /* kod prekidne rutine
      ...
      ...
      */
}
```

II.2.3. A/D pretvornik kao izvor prekida

A/D pretvornici koji su integrirani u STM32F407VG mikroupravljač također mogu biti izvor prekida. Četiri su moguća uzroka generiranja prekida prilikom korištenja A/D pretvornika:

1) završetak pretvorbe (engl. *end of conversion*),
2) završetak injektirane pretvorbe (engl. *end of injected conversion*),
3) analog watchdog, i
4) prepisivanje rezultata pretvorbe (engl. *overrun*).

Primjer generiranja prekida pomoću prvog A/D pretvornika dan je u tablici 3.9. U danom primjeru A/D pretvornik koristi se u kontinuiranom načinu rada pri čemu se radi pretvorba kanala 9 tj. napona na pinu PB1 u 12-bitnu digitalnu riječ. Nakon podešavanja A/D pretvornika potrebno je omogućiti A/D zahtjeve za prekidom pomoću funkcije `ADC_ITConfig`. Nakon toga prekid se dodaje u NVIC. U danom primjeru se generira prekid u slučaju završetka konverzije na prvom A/D pretvorniku (u slučaju postavljanja zastavice `EOC` u `ADC_SR`) te program započinje izvođenje rukovatelja prekidom `ADC_IRQHandler`. Dana funkcija `ADC_ITConfig` praktički postavlja bit `EOCIE` (*End Of Conversion Interrupt Enable*) u kontrolnom registru 1 A/D pretvornika (`ADC_CR1`).

Tablica 3.9 Podešavanje A/D pretvornika 1 i odgovarajuća prekidna rutina.

```
// uključi takt za port B
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

// pin PB1 se koristi u analognom načinu rada
GPIO_InitTypeDef GPIO_InitStructADC;
GPIO_InitStructADC.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructADC.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructADC.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOB, &GPIO_InitStructADC);


// omogućavanje takta za prvi A/D pretvornik
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

// inicijalizacijska struktura za prvi A/D pretvornik
ADC_InitTypeDef ADC1_InitStruct;
// rezolucija A/D pretvorbe
ADC1_InitStruct.ADC_Resolution = ADC_Resolution_12b;
// pretvorba jednog kanala
ADC1_InitStruct.ADC_ScanConvMode = DISABLE;
// koristi se kontinuirani način rada
ADC1_InitStruct.ADC_ContinuousConvMode = ENABLE;
// ne koristi se vanjski okidač za pokretanje pretvorbe
ADC1_InitStruct.ADC_ExternalTrigConv = DISABLE;
ADC1_InitStruct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
// desno poravnanje rezultata u ADC_DR registru
ADC1_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
// broj pretvorbi
ADC1_InitStruct.ADC_NbrOfConversion = 1;

// zapisivanje postavki u konfiguracijske registre A/D pretvornika
ADC_Init(ADC1, &ADC1_InitStruct);
// pin PB1 je spojen na 9 kanal prvog A/D pretvornika
ADC_RegularChannelConfig(ADC1, ADC_Channel_9, 1, ADC_SampleTime_480Cycles);
// omogući rad prvog A/D pretvornika
ADC_Cmd(ADC1, ENABLE);

//omogući ADC prekid uslijed završetka pretvorbe (EOC)
```

```
ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);

//podesi priority grouping: 4 bita za preemptive priority, 0 bita za subpriority
NVIC_SetPriorityGrouping(0u);

//postavi prioritet prekida ADC_IRQn na 0
NVIC_SetPriority(ADC_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,0));

//omogući ADC_IRQn prekid
NVIC_EnableIRQ(ADC_IRQn);

//započni kontinuiranu A/D pretvorbu
ADC_SoftwareStartConv(ADC1);

void ADC_IRQHandler(void)
{
     /* kod prekidne rutine
     ...
     ...
     */

     //obriši zastavicu EOC (automatski se briše i čitanjem rezultata pretvorbe, tj. registra ADC_DR)
     ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
}
```

II.2.4. USART kao izvor prekida

USART periferija STM32F407VG mikroupravljača može se podesiti tako da generira zahtjev za prekidom u određenim uvjetima. Osam je mogućih uzroka generiranja zahtjeva za prekidom prilikom korištenja U(S)ART-a. Često korišten izvor prekida je primanje novog podatka. U tom se slučaju umjesto periodičkog prozivanja U(S)ART-a (provjere statusa zastavice `RXNE`), podatak može primiti pomoću odgovarajuće prekidne rutine. Primjer generiranja prekida pomoću USART1 dan je u tablici 3.10. Slično kao kod ostale periferije, nakon incijalizacije USART-a potrebno je registrirati prekid kod NVIC pri čemu se definira prioritet prekida te se nakon toga omogućuju USART1 zahtjevi za prekidom u slučaju primanja novog podatka pomoću funkcije `USART_ITConfig` (postavlja se bit `RXNEIE` u prvom kontrolnom registru USART-a, USART_CR1).

Tablica 3.10 UART1 kao izvor prekida prilikom primanja podataka.

```
GPIO_InitTypeDef GPIO_InitStruct;              //inicijalizacijska struktura za GPIO
USART_InitTypeDef USART_InitStruct;            //inicijalizacijska struktura za USART

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);   //uključi takt za port B
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);  //uključi takt za USART1

// koristi se alternativna funkcija GPIO pinova PB6 i PB7
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_Init(GPIOB, &GPIO_InitStruct);

GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);       //PB6 se koristi kao USART1_Tx
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);       //PB7 se koristi kao USART1_Rx

USART_InitStruct.USART_BaudRate = 9600;                         //brzina prijenosa 9600 bps
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None; //ne koristi se CTS i RTS
USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;    //koristi se i prijemnik i predajnik
USART_InitStruct.USART_Parity = USART_Parity_No;                //komunikacija bez pariteta
USART_InitStruct.USART_StopBits = USART_StopBits_1;             //1 stop bit
USART_InitStruct.USART_WordLength = USART_WordLength_8b;        //digitalna riječ je veličine 8 bita
USART_Init(USART1, &USART_InitStruct);                          //zapisivanje UART1 postavki

USART_Cmd(USART1, ENABLE);                                      //omogući rad UART1 periferije

USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);     //omogući USART1 prekid uslijed primanja podataka

//podesi priority grouping: 4 bita za preemptive priority, 0 bita za subpriority
NVIC_SetPriorityGrouping(0u);

//postavi prioritet USART1_IRQn prekida
NVIC_SetPriority(USART1_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,0));

//omogući USART1_IRQn prekid
NVIC_EnableIRQ(USART1_IRQn);

void USART1_IRQHandler(void)
```

```
{
      /* kod prekidne rutine
      ...
      ...
      */

     USART_ClearITPendingBit(USART1, USART_IT_RXNE);
}
```

III. Priprema za vježbu

1. UART periferija se često koristi za komunikaciju s različitim uređajima. Znate li konkretan primjer upotrebe ove komunikacije? Koje su najčešće brzine prijenosa?

2. Koji je minimalni, a koji maksimalni mogući vremenski period između dva uzastopna zahtjeva za prekidom koji se generira *SysTick* brojačem uz određeni takt?

3. Pronađite u RM0090 Reference manual dio koji opisuje U(S)ART periferiju. Možete li reći koje su sve mogući izvori prekida kod korištenja U(S)ART periferije? Možete li objasniti što predstavlja *overrun error*? Koji točno bit mora biti aktiviran kako bi ova pogreška generirala zahtjev za prekidom (vidi tablicu 147.)? Koji problem ovo uzrokuje prilikom korištenja prekida za dohvaćanje primljenih podataka?

4. Pročitajte datasheet temperaturnog senzora TMP36. Koju preciznost možete očekivati prilikom rada s ovim senzorom? Što je izlaz ovog senzora i kako (programski) doći do vrijednosti temperature?

IV. Rad na vježbi

1. Pokrenite Atollic True Studio. Potrebno je kreirati novi Projekt naziva *LV3* u kojem će se izraditi jednostavni program za STM32F4 Discovery razvojnu ploču. Odaberite *File-->New-->C project* te odaberite opciju *Embedded C Project*. Kako biste uspješno kreirali projekt potrebno je:
   a. dati odgovarajući naziv projektu (LV_3)
   b. odabrati raspoloživi hardware pod opcijom *Target* (točan naziv mikroupravljača pročitajte s raspoložive Discovery pločice)
   c. odabrati odgovarajući debug hardware (pročitajte s Discovery pločice)

   Ostale postavke nije potrebno mijenjati. Pritiskom na tipku *Finish* nastat će novi projekt naziva "LV_3".

2. Napišite program koji će periodički uključivati i isključivati ugrađenu LED koja je spojena na pin PD15. Za potrebe definiranja vremenskog intervala između uključivanja i isključivanja LED koristite `for` ili `while` petlju u kojoj se inkrementira neka varijabla do određene vrijednosti. Pritiskom na ugradbeno tipkalo na razvojnoj ploči treba se smanjiti interval periodičkog uključivanja odnosno isključivanja LED za određenu vrijednost. Pri tome je potrebno koristiti opciju vanjskog prekida prilikom pritiskanja ugradbenog tipkala.

3. Napišite program koji će svake sekunde uključiti odnosno isključiti LED koja je koja je spojena na pin PD15. Koristite *Systick* brojač i odgovarajući rukovatelj ove iznimke. Pri tome je potrebno u rukovatelju *SysTick* iznimke učiniti uključivanje ili isključivanje LED.

4. Dodajte u rješenje prethodnog zadatka vanjski prekid na padajući brid signala na pinu PA0. U odgovarajućoj prekidnog rutini za vanjski prekid potrebno je:
   a. Uključiti LED koja je spojena na pin PD14.
   b. Napraviti *delay* petlju pomoću for ili while naredbe, npr.
      ```
      i=0; while(i < 100000000) {i++;}
      ```
   c. Isključiti LED koja je spojena na pin PD14.

   Što primjećujete u radu? Koja iznimka ima veći prioritet, *SysTick* ili vanjski prekid? Što trebate učiniti kako bi LED spojena na PD15 i dalje periodički uključivala i isključivala bez obzira na pojavu vanjskog prekida?

5. Povežite STM32F4 Discovery razvojnu ploča s osobnim računalom koristeći pretvornik PL2303TA. Nemojte spajati žicu +5V! Napišite program koji će primljeni podatak (znak) odmah poslati natrag (funkcija *echo*). Koristite SPL. Za ispitivanje rješenja koristite PC aplikaciju *Putty*.

Boja	Funkcija
	+5V
	GND
	PC TX
	PC RX

Sl. 3.4 Pretvornik PL2303TA

6. Napišite program koji će putem UART-a s osobnog računala primiti trenutno vrijeme (npr. u obliku sati:minute:sekunde). Nakon primljenog vremena mikroupravljač započinje mjeriti vrijeme te ga ispisuje korisniku na UART kada korisnik pritisne tipkalo na razvojnoj ploči. Primanje trenutnog vremena učiniti pomoću odgovarajućeg UART prekida, mjerenje vremena učiniti pomoću *SysTick* brojača, a pritisak tipkala detektirati u programu pomoću vanjskog prekida. Što se događa ako vrlo često pritišćete tipkalo? Pokazuje li sat i dalje točno vrijeme ili počinje kasniti?

7. Napišite program koji će mjeriti temperaturu svake sekunde pomoću temperaturnog senzora TMP36. Svakih 30 sekundi potrebno je ispisati srednju vrijednost temperature za prošlih 30 sekundi na zaslonu PC dok se pritiskom na ugrađeno tipkalo ispisuje zadnja izmjerena temperatura. Ako temperatura prijeđe 30°C potrebno je ispisati ALARM na osobno računalo te uključiti crvenu LED na ploči. Kada temperatura padne ispod 30°C program opet periodički ispisuje srednju temperaturu. Prilikom izračunavanja temperature uzmite u obzir da ovaj senzor daje 10mV/°C te da mu je izlazu dodan naponski *offset* iznosa 0.5V. Pri temperaturi od 25°C tipično daje na izlazu napon od 750mV. Nadalje, uzmite da je referentni napon A/D pretvornika jednak 2.9V prilikom izračunavanje temperature. Prilikom spajanja senzora na razvojnu ploču prvi pin senzora spojite na 3V, drugi pin senzora spojite na GPIO pin koji podržava A/D pretvorbu, a treći pin senzora GND. Na slici 3.5 a) prikazan je senzor u TO-92 kućištu, a na slici b) dan je raspored pinova ako se senzor promatra "od ispod").

a) b)

Sl. 3.5 TMP36 temperaturni senzor a) izgled, b) raspored pinova.

V. Za one koji žele znati više

A/D pretvornik na mikroupravljaču STM32F407VG ima ugrađen analogni *watchdog*. Riješite zadatak broj 7. na način da alarmiranje (ispis ALARM na osobnom računalu i uključivanje crvene LED) izvršite korištenjem analognog *watchdog*-a i odgovarajućeg rukovatelja iznimke.