

Izjeme, prekinitve ter zunanje prekinitve v STM32F4 (Cortex-M)

Izjeme (angl. exceptions)

Izjeme (angl. exceptions) so dogodki, ki lahko prekinejo centralno procesno enoto med izvajanjem nekega programa ali niza ukazov z namenom izvedbe niza ukazov, ki izjemo servisira. Odločitev o tem, ali bo določena izjema dejansko prekinila izvajanje je odvisna od tega kakšno prioriteto imata izjema ter program, ki ga izjema prekinja. V primeru, da ima izjema višjo prioriteto, se zgodi servisiranje izjeme, v nasprotnem primeru pa izjema čaka dalje. Obstaja več tipov izjem, ki jih bomo spoznali v nadaljevanju.

Prekinitve (angl. interrupts)

Najpogosteje se srečujemo s tipom izjem, ki jim rečemo prekinitve. Prekinitve so izjeme, ki jih dvignejo periferne naprave, da sporočijo centralni procesni enoti (CPE), da se je zgodilo nekaj, kar bi CPE znalo "zanimati". Primeri so:

- pritisk gumba,
- senzor je zaznal prosti pad,
- temperaturni senzor je zaznal visoko temperaturo,
- vezje za branje stanja baterije je zaznalo, da je baterija prazna,
- zaznan je bil klik na zaslonu na dotik,
- senzor gibanja je zaznal gibanje.

Poleg omenjenih izjem med prekinitve štejemo tudi dve prekinitvi, ki se prožita znotraj procesorja. Prva je `SysTick`, ki jo periodično generira sistemski časovnik. To prekinitvev pogosto uporabljajo operacijski sistemi za merjenje časovnih rezin in preklapljanje opravil. Tudi funkcija `HAL_Delay()`, ki ste jo že uporabili na preteklih vajah, se zanaša na omenjeno prekinitvev.

Druga prekinitvev, ki se generira znotraj procesorja je `PendSV`, to prekinitvev prožimo programsko in se jo uporablja za preklap konteksta (angl. context switch) v večopravilnih operacijskih sistemih.

Poznamo še posebno vrsto prekinitev, ki se imenuje ne-maskirajoče prekinitev (angl. non-maskable interrupt). Te so posebne v tem, da imajo izjemno visoko prioriteto, ki jo presega zgolj reset, kar pomeni da bodo praktično vedno prekinile trenutno izvajajoči proces.

Napake (angl. fault)

Napake so posledica neobičajnih dogodkov, ki jih zazna procesor. Ti dogodki so lahko notranji na primer napake med komunikacijo s pomnilnikom ali drugimi napravami. Napake običajno signalizirajo resne težave v strojni ali programski opremi, ki onemogočajo normalno izvajanje programa. Poznamo sledeče napake:

- **UsageFault** napaka, ki se sproži, ko ukaza ne moremo izvesti. Bodisi ker ukaz ni definiran ali ga ni možno izvesti. Primer bi bilo deljenje z ničlo.
- **BusFault** se proži ob napaki na ukaznem ali podatkovnem vodilu ob dostopu do pomnilnika.
- **MemManage** se proži ob dostopu do zaščitenega dela pomnilnika.
- **HardFault** se proži, če do napake pride med servisiranjem drugih izjem. Pod posebnimi pogoji lahko tudi ostale izjeme eskalirajo v napako tipa HardFault.

Do eskalacije napak v HardFault pride, ko se napaka pripeti v funkcijah, ki servisirajo izjeme z enako ali višjo prioriteto (v tem primeru se napaka ne bi servisirala) ali pa se zgodi napaka za katero ni definirane funkcije, ki se v tem primeru izvede.

Nadzorni klic (angl. supervisor call)

Ta izjema se pripeti ob izvedbi posebnega supervisor call (SVC) ukaza. Ti se uporabljajo kot priročen način, da običajni programi zahtevajo, da jedro operacijskega sistema (angl. operating system kernel) izvede določene funkcije v njihovem imenu. S pomočjo tega ukaza lahko naši mikrokrmilniški programi, ki tečejo v nepriviligiranih načinih, dobijo dostop do posebnih registrov za katere sicer nimajo zadosti pravic.

Reset

Posebna izjema je **reset**, ki se pripeti ob vsakem vklopu procesorja ter v primeru tako imenovanega toplega reseta (angl. warm reset). Tega na naših razvojnih ploščah dosežemo s pritiskom črne (reset) tipke. Ta izjema je posebna v tem, da lahko prekine delovanja v vsakem primeru. Razlikuje se tudi v načinu delovanja, kar bomo spoznali v nadaljevanju.

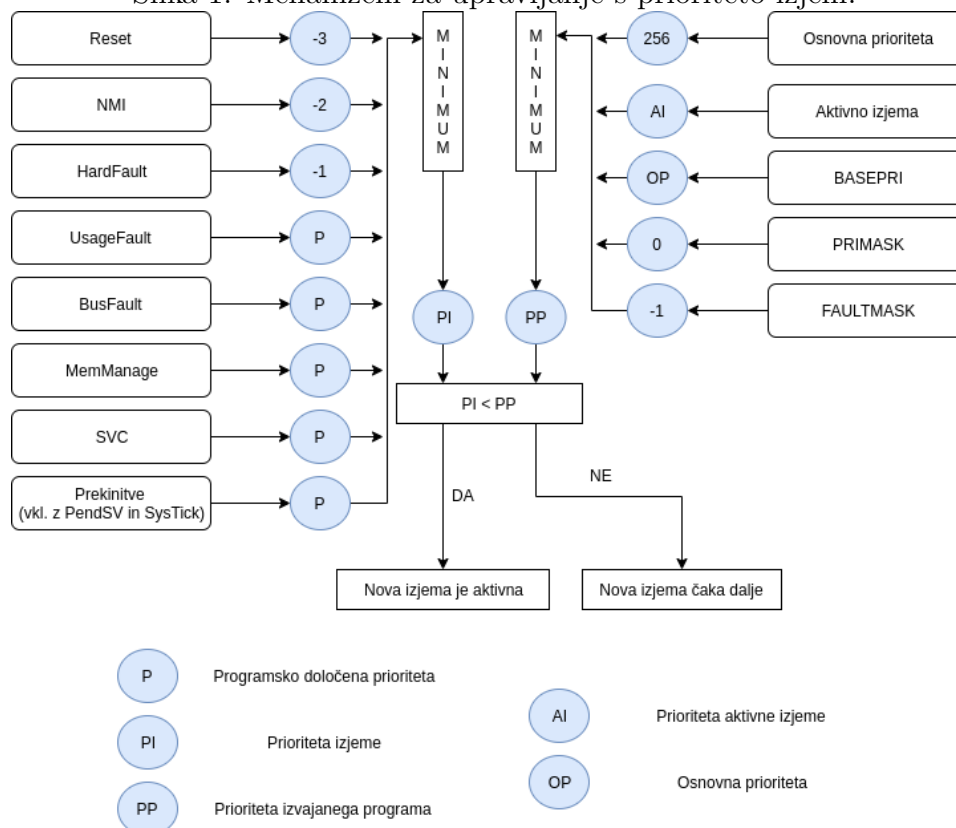
Prioritete izvajanj

V mikrokrmilnikih Cortex-M, med katere spada tudi STM32F4, so prioritete označene s celimi števili, nižje število pa pomeni višjo prioriteto. Najvišjo prioriteto z vrednostjo -3 ima izjema **Reset**. Sledijo ne-maskirajoče prekinitve (NMI) s prioriteto -2 in **HardFault** s prioriteto -1. Ostale izjeme imajo programsko nastavljivo prioriteto med 0 in 255.

Ko pride do izjeme se bo ta servisirala v primeru, da je njena prioriteta nižja od trenutne prioritete izvajanja. Osnovna prioriteta izvajanega programa je 256. V primeru, da se izvaja servisiranje izjeme je prioriteta izvajanega programa enaka prioriteti izjeme. Osnovno prioriteto lahko spremenimo z vpisom vrednosti v poseben register **BASEPRI**. Če v ta register vpišemo vrednost 5, bo to pomenilo, da se bodo lahko servisirale zgolj izjeme, ki imajo višjo prioriteto od 5 (-3, -2, -1, 0, 1, 2, 3 ali 4). Poleg prioritete imamo za nadziranje izjem in prekinitev še dva dodatna enobitna registra, ki določene izjeme maskirata (onemogočita). V primeru, da je postavljen bit 0 v registru **PRIMASK** je najnižja prioriteta izvajanega programa 0. To pomeni, da se bodo servisirale zgolj izjeme **HardFault**, **NMI** in **Reset**.

V primeru, da je postavljen bit 0 v registru **FAULTMASK** je najnižja prioriteta izvajanega programa -1. To pa pomeni, da se bodo servisirale zgolj izjeme **NMI** in **Reset**. Grafični prikaz mehanizma za prioriteta je prikazan na sliki 1.

Slika 1: Mehanizem za upravljanje s prioriteto izjem.



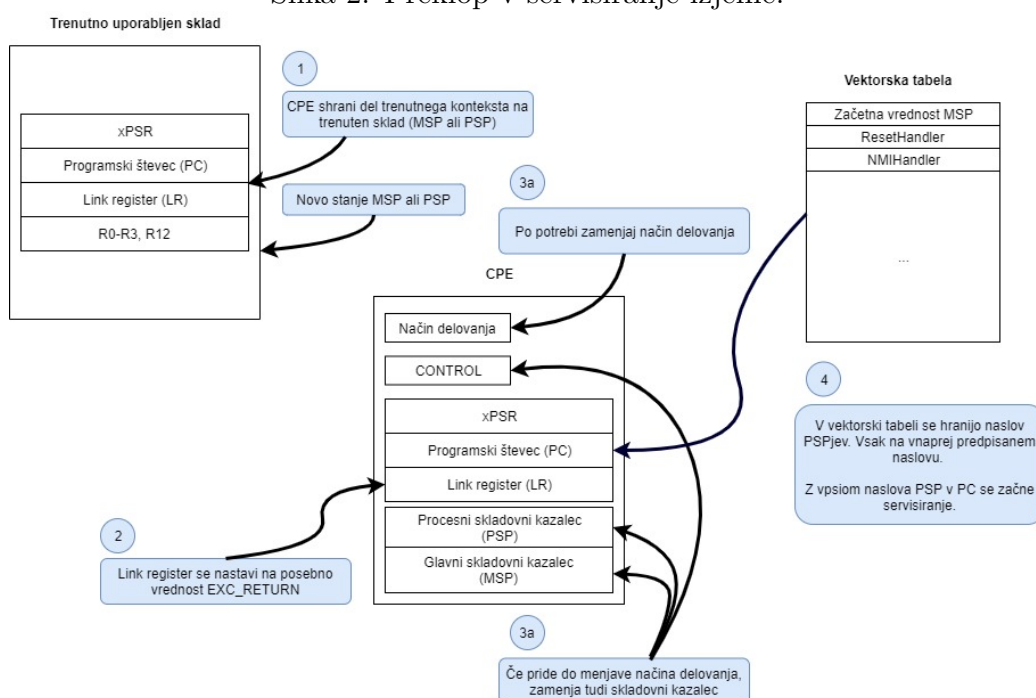
Načini delovanja

Mikrokontrolniki iz družine Cortex-M imajo 2 načina delovanja: Thread in Handler. Razlika med njima je v pravicah dostopa do posebnih registrov mikrokontrolnika (npr. CONTROL). V Handler načinu delovanja centralna procesna enota preklopi kadar servisira izjeme. Edina izjema, za katero to ne drži, je **Reset**, servisiranje te izjeme se namreč izvaja v Thread načinu. Prav tako je pogosto razlika med načini delovanja tudi v skladovnem kazalcu, ki se med servisiranjem uporablja. Tega, tako kot običajno hranimo v registru **r13**. V Handler načinu se uporabi glavni skladovni kazalec (angl. Main Stack pointer - MSP), v Thread načinu pa procesni skladovni kazalec (angl. Process Stack Pointer - PSP).

Preklop v servisiranje izjeme

V primeru, da ima izjema, ki se pojavi, dovolj visoko prioriteto, da postane aktivna, se začne servisiranje izjeme. Preklop v servisiranje izjeme je prikazan na sliki 2. Najprej se shrani del stanja (konteksta) procesorja na sklad. Shranijo se programski števec (angl. program counter - PC), register za povratni naslov (angl. link register - LR), statusni register (angl. program status register - xPSR) ter splošno-namenski registri R0, R1, R2, R3 in R12. Procesor ima na voljo dva sklada in posledično dva skladovna kazalca, glavni skladovni kazalec (MSP) ter procesni skladovni kazalec (PSP). Običajno se uporabi glavni skladovni kazalec, razen če programsko določimo drugače.

Slika 2: Preklop v servisiranje izjeme.



Po shranjevanju na sklad se v LR zapiše posebna vrednost, ki jo ARM označuje kot `EXC_RETURN`. V primeru, da je potrebno, sledi preklop načina delovanja in menjava skladovnega kazalca. Hkrati se v register `IPSR` shrani oznaka izjeme. Zadnji korak preklopa v servisiranje izjeme je nastavljanje programskega števca iz vektorske tabele. Kateri vnos iz vektorske tabele

procesor vzame je odvisno od oznake izjeme. Prvi vnos v vektorski tabeli pripada začetnemu naslovu funkcije, ki servisira **Reset** izjemo. Sledi začetni naslov funkcije za servisiranje ne-maskirajočih prekinitev, in tako dalje. Vektorska tabela se nahaja na začetku pomnilniškega prostora, pred njo se nahaja zgolj začetna vrednost glavnega skladovnega kazalca. Vektorska tabela se v projektu za STM32CubeIDE nahaja v `Startup/startup_stm32f407vgtx.s`. Izsek vektorske tabele je prikazan spodaj.

```
1 g_pfnVectors :
2   .word   _estack
3   .word   Reset_Handler
4   .word   NMI_Handler
5   .word   HardFault_Handler
6   .word   MemManage_Handler
7   .word   BusFault_Handler
8   .word   UsageFault_Handler
9   .word   0
10  .word   0
11  .word   0
12  .word   0
13  .word   SVC_Handler
14  .word   DebugMon_Handler
15  .word   0
16  .word   PendSV_Handler
17  .word   SysTick_Handler
18
19  /* External Interrupts */
20  .word   WWDG_IRQHandler
21  .word   PVD_IRQHandler
22  .word   TAMP_STAMP_IRQHandler
23  .word   RTC_WKUP_IRQHandler
24  .word   FLASH_IRQHandler
25  .word   RCC_IRQHandler
26  .word   EXTI0_IRQHandler
27  .word   EXTI1_IRQHandler
28  .word   EXTI2_IRQHandler
29  .word   EXTI3_IRQHandler
30  .word   EXTI4_IRQHandler
```

Pseudoukaz `.word` označuje, da na tem mestu shranjujemo 32-bitno vrednost. Kot vrednosti so zapisana imena podprogramov v zbirniku ali funkcij v C-ju. Ime funkcije pa je dejansko naslov začetka funkcije, podobno kot ime polja predstavlja naslov začetka polja.

Prva vrstica vektorske tabele označuje začetno vrednost glavnega skladovnega kazalca MSP. Sledi oznaka `Reset_Handler`. Ta pomeni, da se bo v primeru, da pride do izjeme `Reset`, izvedla funkcija z imenom `Reset_Handler`. Ta se nahaja v isti datoteki in je napisana v ARM zbirniku. Spodaj so navedena tudi imena funkcij, ki se bodo izvedle v primeru prekinitiv. Tako funkcijo imenujemo prekinitveno servisni program (angl. interrupt request handler), zato se končajo s končnico `IRQHandler`. Za delo s prekinitvami je potrebno poznati tako oznake izjem kot imena prekinitveno servisnih programov. V `Drivers/CMSIS/Device/ST/STM32F4xx/Include/stm32f407xx.h` lahko najdete oznake izjem. Izsek oznak je prikazan spodaj.

```

1 typedef enum
2 {
3  /***** Cortex-M4 Processor Exceptions Numbers
4     *****/
5     NonMaskableInt_IRQn    = -14, /*!< 2 Non Maskable */
6     MemoryManagement_IRQn = -12, /*!< 4 Cortex-M4 Memory
7     Management */
8     BusFault_IRQn          = -11, /*!< 5 Cortex-M4 Bus Fault */
9     UsageFault_IRQn        = -10, /*!< 6 Cortex-M4 Usage Fault */
10    SVCall_IRQn             = -5,  /*!< 11 Cortex-M4 SV Call */
11    DebugMonitor_IRQn       = -4,  /*!< 12 Cortex-M4 Debug Monitor
12    */
13    PendSV_IRQn             = -2,  /*!< 14 Cortex-M4 Pend SV */
14    SysTick_IRQn           = -1,  /*!< 15 Cortex-M4 System Tick */
15  /***** STM32 specific Interrupt Numbers
16     *****/
17    WWDG_IRQn              = 0,    /*!< Window WatchDog Interrupt */
18    PVD_IRQn               = 1,    /*!< PVD through EXTI Line detection */
19    TAMP_STAMP_IRQn        = 2,    /*!< Tamper and TimeStamp interrupts */
20    RTC_WKUP_IRQn         = 3,    /*!< RTC Wakeup interrupt */
21    FLASH_IRQn            = 4,    /*!< FLASH global Interrupt */
22    RCC_IRQn              = 5,    /*!< RCC global Interrupt */
23    EXTI0_IRQn            = 6,    /*!< EXTI Line0 Interrupt */
24    EXTI1_IRQn            = 7,    /*!< EXTI Line1 Interrupt */
25    EXTI2_IRQn            = 8,    /*!< EXTI Line2 Interrupt */
26    EXTI3_IRQn            = 9,    /*!< EXTI Line3 Interrupt */
27    EXTI4_IRQn            = 10,   /*!< EXTI Line4 Interrupt */
28    ...

```

V datoteki `Src/stm32f4xx_it.c` najdemo ogrodja servisnih funkcij za ostale napake. Za preostale izjeme je v datoteki definirano, da se v primeru,

da ne obstaja funkcija z enakim imenom kot je definiran v vektorski tabeli, kliče funkcijo `Default_Handler`. Ta je definirana v isti datoteki kot vektorska tabela, v njej je pa implementirana neskončna zanka.

V primeru, da želimo napisati prekinitveno servisni program za katero od preostalih izjem, moramo v eno izmed datotek projekta zapisati funkcijo z imenom, kot je zapisan v vektorski tabeli. Dobra praska je, da servisne programe izjem pišemo v datoteko `Src/stm32f4xx_it.c`.

Prekinitveni krmilnik NVIC

Vse prekinitve z izjemo prekinitvev `SysTick` in `PendSV` do mikrokrmilnika pridejo iz zunanjih naprav. Vendar pa ne neposredno, temveč preko prekinitvenega krmilnika. Ta omogoča, da lahko vsak posamičen prekinitveni vir programsko vklopimo ali izklopimo ter mu določamo prioriteto. S knjižico STM32 HAL to storimo preprosto z dvema ukazoma, kot je prikazano spodaj.

```
1 // za prekinitveni vir določimo prioriteto
2 HAL_NVIC_SetPriority(RCC_IRQn, 1, 2);
3
4 // vklopimo prekinitveni vir
5 HAL_NVIC_EnableIRQ(RCC_IRQn);
```

Prioriteto v prekinitvenem krmilniku določamo na dveh nivojih z vrednostmi od 0 do 15. Kot je bilo omenjeno na začetku tega dokumenta, to pomeni, da imamo skupaj 256 prioritetenih nivojev. Najprej se upošteva prioriteto na prvem nivoju in v primeru enakosti na prvem nivoju se naredi primerjava še na drugem nivoju.

Zadnji korak je, da dodamo prekinitveno servisni program za vklopljen prekinitveni vir. Za zgornji primer je to funkcija `RCC_IRQHandler`.

Prekinitveno servisni programi

Pri predmetu ORS bomo pisali predvsem prekinitveno servisne programe (PSP). Torej funkcije, ki servisirajo prekinitve. Bistveno pri PSP je, da v čim krajšem času odpravi razlog za prekinitvev. Običajno je vsak PSP razdeljen v tri dele:

- Preverjanje prekinitvenih virov: najprej moramo preveriti, da smo v PSP ravno zaradi izbranega prekinitvenega vira. To preverjamo tudi zato, ker je lahko isti PSP odgovoren za več prekinitvenih virov.

- Odpravljanje razloga za prekinitvev.
- Brisanje prekinitvenih zahtev: pred izhodom iz PSP moramo obvezno poskrbeti, da je prekinitvena zahteva, ki smo jo pravkar servisirali, pobrisana.

Glede na to, da je zaželjen čim hitrejši izhod iz PSP-jev, je v njih odsvetovano uporabljati zakasnitve (npr. `HAL_Delay()`).

Vračanje iz servisiranja izjem

Po zaključku prekinitveno servisnih programov oziroma funkcij, procesor prebere vrednost shranjeno v registru `lr`. Če prebere prej omenjeno posebno vrednost `EXC_RETURN`, ve da je funkcija, iz katere se vračamo, servisirala izjemo. Ob povratku v izvorni program preveri ali so izpolnjeni vsi pogoji za vračanje iz servisne funkcije, nato pa povrne stanje procesorja iz sklada. Povrnjeno stanje programskega števca tudi pomeni, da program nadaljuje na mestu kjer je bil prekinjen zaradi izjeme.

Zunanje prekinitve

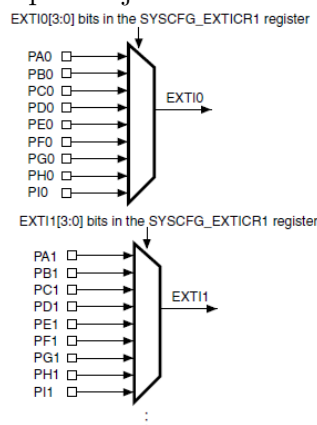
Prvi vir prekinitvev, ki ga bomo podrobneje spoznali, so zunanje prekinitve (angl. external interrupts). To so prekinitveni viri, ki se uporabljajo predvsem za to, da lahko digitalni vhodni pini prožijo prekinitvev. Tako bomo na primer znali doseči, da bo pritisk na gumb sprožil prekinitvev.

Vhodni pini namreč sami ne morejo prožiti prekinitvev preko GPIO naprave, ampak za to skrbi ločena naprava za zunanje prekinitvev. Ta ima oznako `EXTI`. `EXTI` naprava omogoča zaznavo spremembe signala iz 0 v 1 (prva fronta, angl. rising edge) ali iz 1 v 0 (zadnja fronta, angl. falling edge) in proženje prekinitvev ob zaznavanju ene ali obeh front.

Naprava `EXTI` ima 23 vhodnih linij, ki lahko prožijo prekinitvev. Nas bodo zanimale predvsem linije 0 do 15. Na linijo 0 so namreč priklopljeni vsi pini z oznako `Px0` (`PA0`, `PB0`, `PC0`, `PD0`, ...), na linijo ena vsi pini z oznako `Px1`, in tako naprej do linije 15. Pini so na `EXTI` linijo priklopljeni preko multiplekserjev prikazanih na sliki 3.

Izhodi iz multiplekserjev so vhod v vezje na sliki 4. Na levi strani vidimo, da je izhod iz vezja povezan na `NVIC`. Izhod vezja predstavlja stanje registra `Pending request register`. Ta register se nastavlja iz izhoda `AND` vrat.

Slika 3: Multiplekserji na vhodnih linijah EXTI.



Ta je torej aktiven, ko sta oba vhoda aktivna. Prvi je **Interrupt mask register**, drugi pa izhod iz OR vrat. Če je bit v Interrupt mask register postavljen na 0, s tem onemogočimo, da bi linija prožila prekinitev. Ne glede na to kaj je na drugem vhod IN vrat, če je eden izmed vhodov nič, bo namreč izhod IN vrat vedno logična ničla. V primeru, da je bit v **Interrupt mask register** postavljen, pa za proženje prekinitve potrebujemo še logično enico iz OR vrat. To dobimo na dva načina. Prvi je, da postavimo bit v **Software interrupt event register**, ki omogoča programsko proženje EXTI prekinitev. Drugi način, ki je tudi tisti, ki bo nam prišel prav, je da vezje za zaznavanje fronte (angl. Edge detect circuit) zazna fronto na vhodnem signalu. To je tudi primarna funkcija EXTI naprave.

V tabeli 1 so zapisane prekinitvene oznake in imena prekinitveno servisnih programov vseh šestnajstih prekinitvenih linij, ki so vezane na GPIO pine.

Vklop prekinitev za vhodni pin

Preden določimo prioriteto in vklopimo prekinitveni vir, moramo vklopiti proženje prekinitev v EXTI ter določiti na katero fronto želimo, da vhodni pin proži prekinitev. To storimo tako, da pri inicializaciji pina za način delovanja izberemo enega izmed spodnjih možnosti:

- `GPIO_MODE_IT_RISING`
- `GPIO_MODE_IT_FALLING`

Slika 4: EXTI naprava.

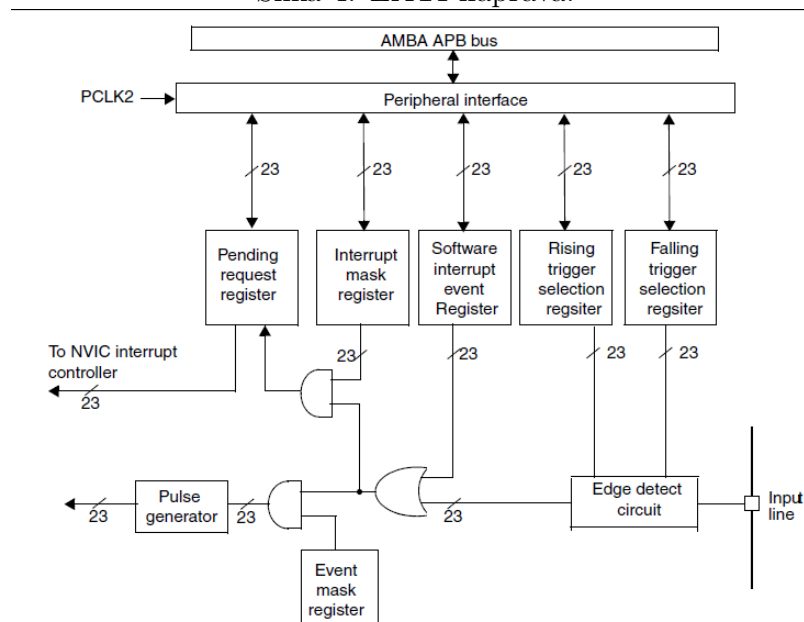


Tabela 1: Oznake prekinitvenih virov in imena PSP-jev.

EXTI linija	Oznaka prek. vira	Ime PSP-ja
EXTI0	EXTI0_IRQn	EXTI0_IRQHandler
EXTI1	EXTI1_IRQn	EXTI1_IRQHandler
EXTI2	EXTI2_IRQn	EXTI2_IRQHandler
EXTI3	EXTI3_IRQn	EXTI3_IRQHandler
EXTI4	EXTI4_IRQn	EXTI4_IRQHandler
EXTI5 - EXTI9	EXTI9_5_IRQn	EXTI9_5_IRQHandler
EXTI10-EXTI15	EXTI15_10_IRQn	EXTI15_10_IRQHandler

- GPIO_MODE_IT_RISING_FALLING

Celoten primer vklopa prekinitve za pin PE3 je prikazan spodaj.

```

1 // clock on
2 _HAL_RCC_GPIOE_CLK_ENABLE();
3

```

```
4 // init
5 GPIO_InitTypeDef init_structure;
6 init_structure.Pin = GPIO_PIN_3;
7 init_structure.Mode = GPIO_MODE_IT_RISING;
8 init_structure.Pull = GPIO_NOPULL;
9 init_structure.Speed = GPIO_SPEED_FREQ_LOW;
10 HAL_GPIO_Init(GPIOE, &init_structure);
11
12 // za prekinitveni vir določimo prioriteto
13 HAL_NVIC_SetPriority(EXTI3_IRQn, 1, 2);
14
15 // vklopimo prekinitveni vir
16 HAL_NVIC_EnableIRQ(EXTI3_IRQn);
```

Prekinitveno servisni program

V zgornjem primeru smo vklopili prekinitve za vir `EXTI3_IRQn`, kar pomeni, da moramo implementirati funkcijo `EXTI3_IRQHandler`. To naredimo tako, da v datoteko `Inc/stm32f4xx_it.h` dodamo prototip funkcije, ki je prikazan spodaj:

```
1 void EXTI3_IRQHandler(void);
```

V datoteko `Src/stm32f4xx_it.c` moramo nato dodati prekinitveno servisno funkcijo. V njej s funkcijo `__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_3)` najprej preverimo, če je prekinitvev sprožil izbrani prekinitveni vir. Na koncu PSP-ja pa kličemo funkcijo `__HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_3)`, da pobrišemo prekinitveno zahtevo. Celoten primer PSP-ja, ki ob prekinitvi poveča vrednost števca za 1 je prikazan spodaj.

```
1 volatile int counter = 0;
2
3 void EXTI3_IRQHandler() {
4     if(__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_3) != 0)
5     {
6         counter++;
7         __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_3);
8     }
9 }
```

SysTick prekinitve

V projektih, ki smo jih do sedaj uporabljali na vajah, so že bile aktivne prekinitve, a se tega nismo niti zavedali. Funkcija `HAL_Init()`, katere klic je avtomatsko dodan v telo `main` funkcije, namreč že inicializira in vklopi proženje prekinitve naprave `SysTick`. Ta vsako milisekundo proži prekinitve, kar pomeni, da se vsako milisekundo izvede prekinitveno servisni program oziroma funkcija `SysTick_Handler()`. To funkcijo lahko najdete v datoteki `Src/stm32f4xx_it.c`.

`SysTick_Handler()` bomo lahko uporabljali za enostavno merjenje časa (na milisekundnem nivoju) in ostala opravila, ki zahtevajo delo s časom. Kasneje v semestru bomo podrobneje spoznali časovnike, ki nam bodo omogočali tudi bolj zahtevno delo s časom.

Naloga za vajo

Za izhodišče vzemite prazen projekt v razvojnem okolju. Realizirajte števec pritiskov na gumb. Števec naj šteje do vključno 15. Stanje števca v binarni obliki prikažite s pomočjo LED diod. Pri številu 14 (binarno 1110) naj bo prva LED dioda ugasnjena, ostale 3 pa prižgane. Pri številu 8 (binarno 1000) naj gori zgolj zadnja LED dioda. Primer štetja od 0 do 4 je prikazan na sliki 5.

Slika 5: Prikaz vrednosti števca na LED diodah (od 0 do 4).



Ob dolgem pritisku (2 sekundi ali več) nastavite vrednost števca na 0. Celotno logiko štetja ter prižiganja LED diod izvedite v prekinitveno servisnih programih. V `main` funkciji naj bo zgolj logika, ki inicializira potrebne naprave in pine. Temu naj sledi zgolj prazna neskončna zanka.

Za vsa opravila, ki se tičejo zakasnitev in merjenja časa, uporabljajte `SysTick_Handler()`.