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 prekinitev 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 prekinitev.

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

Poznamo še posebno vrsto prekinitev, ki se imenuje ne-maskirajoče prekinitve (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.

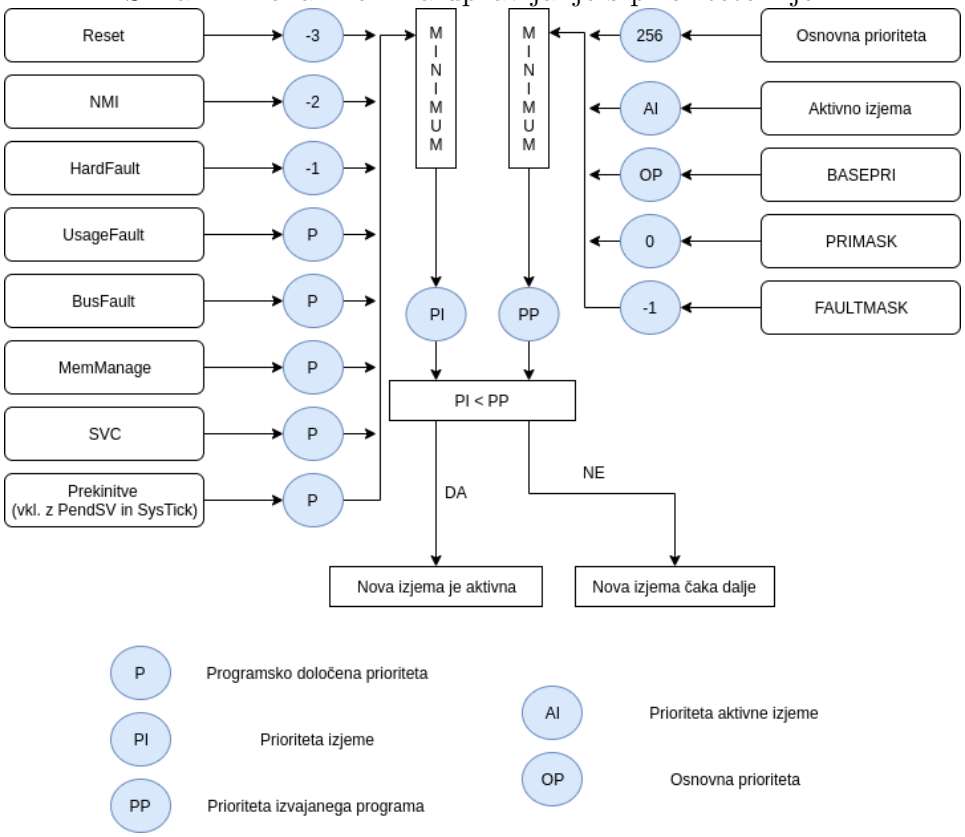
Prioritete izvajanj

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 priororitete 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.



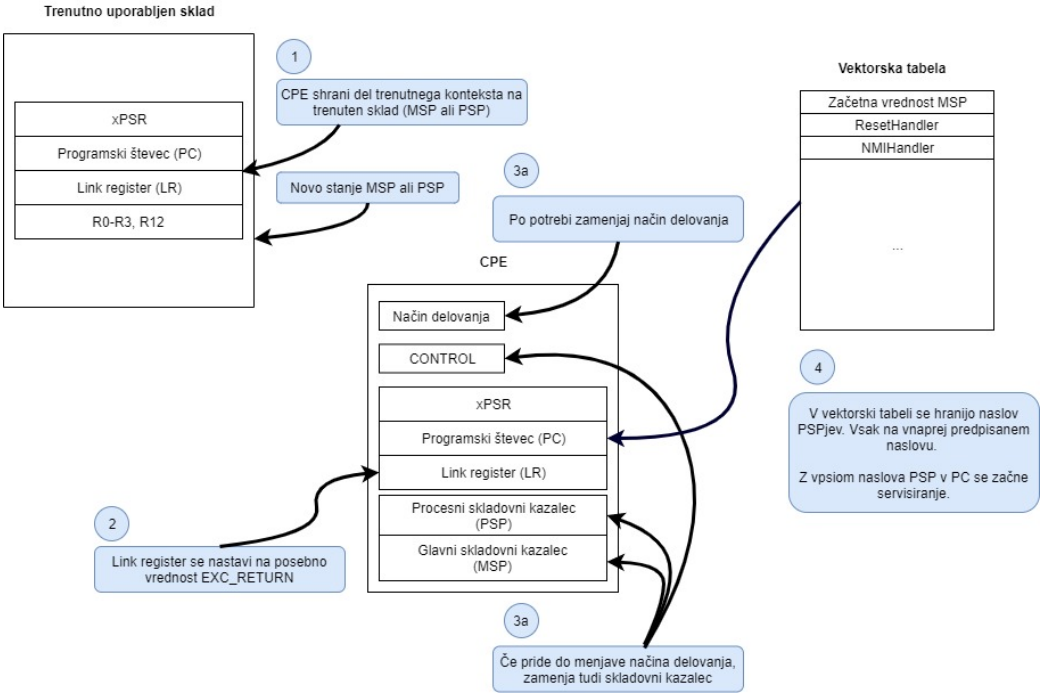
Načini delovanja

Mikrokrmilniki iz družine Cortex-M imajo 2 načina delovanja: Thread in Handler. Razlika med njima je v pravicah dostopa do posebnih registrov mikrokrmilnika (npr. CONTROL). V Handler način 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 običjano 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.

```
g_pfnVectors:
  .word  _estack
  .word  Reset_Handler
  .word  NMI_Handler
  .word  HardFault_Handler
  .word  MemManage_Handler
  .word  BusFault_Handler
  .word  UsageFault_Handler
  .word  0
  .word  0
  .word  0
  .word  0
  .word  SVC_Handler
  .word  DebugMon_Handler
  .word  0
  .word  PendSV_Handler
  .word  SysTick_Handler

  /* External Interrupts */
  .word      WWDG_IRQHandler
  .word      PVD_IRQHandler
  .word      TAMP_STAMP_IRQHandler
  .word      RTC_WKUP_IRQHandler
  .word      FLASH_IRQHandler
  .word      RCC_IRQHandler
  .word      EXTI0_IRQHandler
  .word      EXTI1_IRQHandler
  .word      EXTI2_IRQHandler
  .word      EXTI3_IRQHandler
  .word      EXTI4_IRQHandler
```

Psevdoukaz `.word` označuje, da na tem mestu shranjujejmo 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 prekinitev. 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.

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

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 prekinitev `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 dolocimo 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 prioritetnih 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 prekinitev. 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 prekinitev.
- Brisanje prekinitvenih zahtev: pred izhodom iz PSP moramo obvezno poskrbeti, da je prekinitvena zahteva, ki smo jo pravkar servisirali, pobrisana.

Vračanje iz servisiranja izjem

Po zaključku prekinitveno servisnih programov oziroma funkcij, procesor prebere vrednost shranjeno v regirstru `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

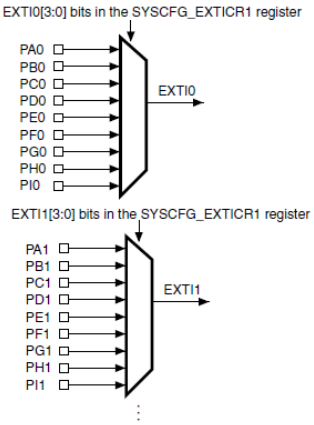
Prvi vir prekinitev, ki ga bomo podrobneje spoznali, so zunanje prekinitve (angl. external interrupts). To so prekinitveni viri, ki se uporabljajo predvsem za to, da lahko digitalni vhodni pini prožijo prekinitev. Tako bomo na primer znali doseči, da bo pritisk na gumb sprožil prekinitev.

Vhodni pini namreč sami ne morejo prožiti prekinitev preko GPIO naprave, ampak za to skrbi ločena naprava za zunanje prekinitve. 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 prekinitev ob zaznavanju ene ali obeh front.

Naprava EXTI ima 23 vhodnih linij, ki lahko prožijo prekinitve. 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. 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

Slika 3: Multiplekserji na vhodnih linijah EXTI.



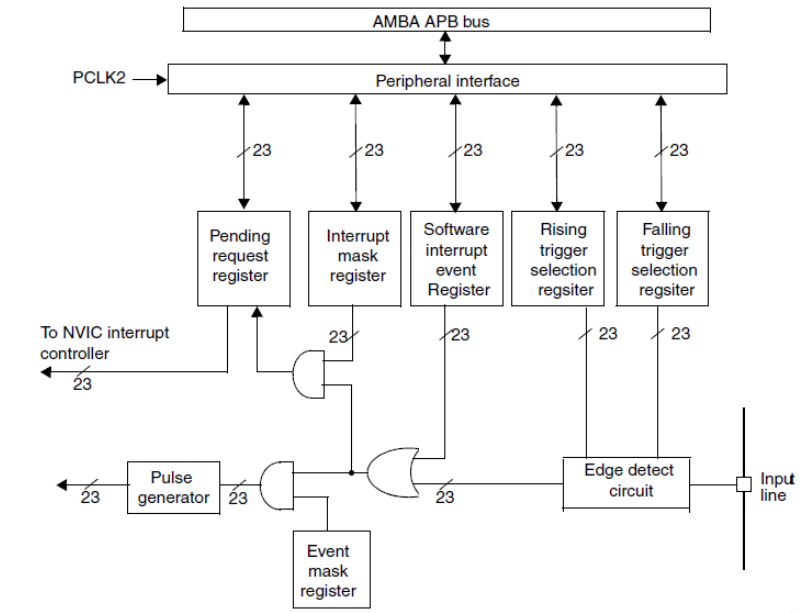
postavljen na 0, s tem onemogočimo, da bi linija prožila prekinitev. Ne glede na to kaj je na drugem vhod IN vrat, če je eden izmed vhodov nič, bo namreč izhod IN vrat vedno logična ničla. V primeru, da je bit v `Interrupt mask register` postavljen, pa za proženje prekinitve potrebujemo še logično enico 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.

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

Slika 4: EXTI naprava.



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`
- `GPIO_MODE_IT_RISING_FALLING`

Celoten primer vklopa prekinitve za pin PE3 je prikazan spodaj.

```
// clock on
__HAL_RCC_GPIOE_CLK_ENABLE();

// init
GPIO_InitTypeDef init_structure;
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 dolocimo 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 funckije, 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 prekinitev sprožil izbrani prekinitven 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 }
```

Naloga

Za izhodišče vzemite prazen projekt v razvojnem okolju. Realizirajte števec pritiskov na gumb na podoben način kot v prejšnji vaji. 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 pritisku gumba povečajte vrednost števca za 1. Celotno logiko štetja ter prižiganja LED diod izvedite v prekinitveno servisnih programih. V main funkciji naj bo zgolj logika, ki inicializira potrebne naprave. Temu naj sledi zgolj prazna neskončna zanka.

Za reševanje odboja gumba (angl. button debouncing) namesto funkcije `HAL_Delay` uporabite PSP `SysTick_Handler`. Ta je že deklariran v datoteki, kjer so zbrani vsi PSPji (`Src/stm32f4xx_it.c`). Za prekinitev `SysTick` vemo, da se proži vsako milisekundo.