Uporaba STM HAL knjižnice za delo s splošno namenskim vhodom/izhodom

Na tokratni vaji bomo spoznali še tretji in zadnji način razvoja programske opreme za mikrokrmilnike. Prvi je bil neposredno pisanje na naslove, drugi pa priprava lastnih funkcij oziroma knjižnice za delo z GPIO napravami. Tokrat bomo spoznali še knjižnico, ki jo ponuja proizvajalec mikrokrmilnika. Knjižnica je narejena na podoben način kot tista, ki ste jo začeli graditi na zadnji vaji. Razlika je le v tem, da proizvajalčeva knjižnica podpira vse funkcionalnosti vseh naprav na mikrokrmilniku in da to knjižnico uporablja milijone razvijalcev po svetu in je tako možnost za hrošče bistveno manjša. Knjižnica, ki jo je razvil proizvajalec mikrokrmilnika, se imenuje HAL Driver Library oziroma krajše HAL. Glavnina datotek knjižnice se nahaja v mapi `Drivers/STM32F4xx_HAL_Driver`. Tu najdete dve podmapi, `src`, kjer se nahajajo .c datoteke s funkcijami ter `inc` s .h datotekami, kjer najdemo konstante in prototipe funkcij.

Prvi del knjižnice, ki ga bomo spoznali na tokratni vaji, omogoča delo z digitalnim vhodom in izhodom, torej z GPIO napravami.

Vklop ure GPIO naprave

Podobno kot na zadnji vaji, bomo tudi tokrat začeli z vklopom ure GPIO naprav. Za ta namen ste na zadnji vaji razvili funkcijo `clock_on()`. HAL knjižnica ponuja za ta namen več funkcij, vsako za eno izmed GPIO naprav. Funkcija `__HAL_RCC_GPIOA_CLK_ENABLE()` tako vklopi uro naprave GPIOA, `__HAL_RCC_GPIOB_CLK_ENABLE()` vklopi uro naprave GPIOB in tako naprej do `__HAL_RCC_GPIOI_CLK_ENABLE()`, ki vklopi uro naprave GPIOI. Takšne skupine funkcij običajno krajše zapišemo z `__HAL_RCC_GPIOx_CLK_ENABLE()`, kjer `x` nadomestimo s črkami A do I. Definicije teh funkcij se nahajajo v datotekah `stm32f4xx_hal_rcc.h` ter `stm32f4xx_hal_rcc_ex.h`.

HAL ponuja še funkcije za izklop ure, `__HAL_RCC_GPIOx_CLK_DISABLE()` ter funkcije, ki določajo kaj se dogaja z urami GPIO naprav, ko mikrokrmilnik preklopimo v način delovanja, ki omogoča nižjo porabo energije (angl. Low Power Mode). To sta funkciji `__HAL_RCC_GPIOx_CLK_SLEEP_ENABLE()` in `__HAL_RCC_GPIOx_CLK_SLEEP_DISABLE()`.

Inicializacija GPIO naprave

Pri knjižnic HAL za inicializacijo uporabljamo funkcijo `HAL_GPIO_Init`. Ta sprejme dva parametra, kazalec na strukturo `GPIO_TypeDef` ter kazalec na strukturo `GPIO_InitTypeDef`. V datoteki `stm32f407.h` je definirana struktura `GPIO_TypeDef`, ki je prikazana spodaj.

```
typedef struct
{
  __IO uint32_t MODER;
  __IO uint32_t OTYPER;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

Ta struktura bi vam morala biti bolj ali manj že poznana, saj smo podobno strukturo zgradili že pri zadnji vaji. Opazimo pa lahko tri razlike. Prva je, da imajo vsi elementi predpono `__IO`, gre zgolj za preimenovano rezervirano besedo `volatile`. V naši knjižnici smo to izpustili, je pa za pomnilniško preslikane lokacije (registre) dobra praksa, da se spremenljivkam dodeli status `volatile`. Druga razlika je, da sta tu registra `Bit Set` in `Bit Reset` združena v register `Bit Set Reset (BSRR)`. Knjižnica tako namesto dveh registrov uporablja zgornjo in spodnjo polovico večjega registra. Tretja in zadnja razlika so dodatni registri: `LCKR` omogoča zaklep nastavitev, `AFR[0]` in `AFR[1]` pa omogočata nastavitve alternativnih funkcij, kadar pinu določimo ta način delovanja. Pri naši prejšnji vaji smo jih izpustili, ker jih nismo potrebovali.

V datoteki `stm32f407.h` najdemo tudi naslednje konstante:

```
#define GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD                ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE                ((GPIO_TypeDef *) GPIOE_BASE)
#define GPIOF                ((GPIO_TypeDef *) GPIOF_BASE)
```

```
#define GPIOG                   ((GPIO_TypeDef *) GPIOG_BASE)
#define GPIOH                   ((GPIO_TypeDef *) GPIOH_BASE)
#define GPIOI                   ((GPIO_TypeDef *) GPIOI_BASE)
```

Tudi takšen zapis konstant bi vam moral biti že znan. V vaših knjižnicah ste uporabili konstante `GPIOAd, GPIOBd, ..., GPIOId`. Črko d smo dodali razno zato, da ni bilo konfliktov z že definiranimi konstantami. Pri vaših knjižnicah ste namesto imen `GPIOD_BASE` zapisali kar neposredne naslove. Tu je vse skupaj zgradeno modularno, če pogledamo na primeru `GPIOD`:

```
#define PERIPH_BASE             0x40000000UL

#define AHB1PERIPH_BASE         (PERIPH_BASE + 0x00020000UL)

#define GPIOD_BASE              (AHB1PERIPH_BASE + 0x0C00UL)

#define GPIOD                   ((GPIO_TypeDef *) GPIOD_BASE)
```

Če ročno izračunamo naslove dobimo `GPIOD = 0x40020C00`, kar je enaka vrednost, kot ste jo uporabili v vaših knjižnicah.

Prvi argument funkcije `HAL_GPIO_Init` je tako konstanta `GPIOx`, podobno kot pri funkciji `init_GPIO()` iz prejšnje vaje. Namesto preostalih petih argumentov naše funkcije imamo pri HAL zgolj enega. Kot že rečeno, gre za kazalec na strukturo `GPIO_InitTypeDef`. V to strukturo zapišemo želene nastavitve pinov, ki jih funkcija `HAL_GPIO_Init` nato preslika v registre, podobno kot so to počele vaše funkcije v zadnji nalogi. Struktura `GPIO_InitTypeDef` je definirana v datoteki `stm32f4xx_hal_gpio.h`:

```
typedef struct
{
  uint32_t Pin;  /*!< Specifies the GPIO pins to be configured.
   This parameter can be any value of @ref GPIO_pins_define */
  uint32_t Mode;      /*!< Specifies the operating mode for the
   selected pins. This parameter can be a value of @ref
   GPIO_mode_define */
  uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down
   activation for the selected pins. This parameter can be a
   value of @ref GPIO_pull_define */
  uint32_t Speed;     /*!< Specifies the speed for the selected
   pins. This parameter can be a value of @ref GPIO_speed_define
    */
```

```
7   uint32_t Alternate;  /*!< Peripheral to be connected to the
      selected pins. This parameter can be a value of @ref
      GPIO_Alternate_function_selection */
8 }GPIO_InitTypeDef;
```

Struktura ima torej 5 elementov, s katerimi določimo nastavitve za inicializacijo naprave. Vsak izmed elementov ima omejen nabor vrednosti, ki mu jih lahko določimo. V komentarju je za vsakega zapisano ime nabora vrednosti, ki jih lahko uporabimo (`GPIO_pins_define`, `GPIO_mode_define`, ...).

Mode

Nabor možnih vrednosti za element `Mode` je:

```
1 #define  GPIO_MODE_INPUT                0x00000000U
2 #define  GPIO_MODE_OUTPUT_PP            0x00000001U
3 #define  GPIO_MODE_OUTPUT_OD            0x00000011U
4 #define  GPIO_MODE_AF_PP                0x00000002U
5 #define  GPIO_MODE_AF_OD                0x00000012U
6 #define  GPIO_MODE_ANALOG               0x00000003U
```

V primerjavi z vašimi funkcijami, tu z elementom `Mode` ne določimo zgolj načina delovanja ampak, v primeru, da uporabimo izhodni način delovanja, določimo tudi tip izhoda (push-pull/open-drain). Isto velja za alternativne funkcije.

Speed

Nabor možnih vrednost za element `Speed` je:

```
1 #define  GPIO_SPEED_FREQ_LOW            0x00000000U
2 #define  GPIO_SPEED_FREQ_MEDIUM         0x00000001U
3 #define  GPIO_SPEED_FREQ_HIGH           0x00000002U
4 #define  GPIO_SPEED_FREQ_VERY_HIGH      0x00000003U
```

Tukaj knjižnica namesto številčnih (2MHz, 25MHz, 50MHz, ...) uporablja opisna imena. Opisna imena uporablja zato, ker knjižnica deluje z različnimi mikrokrmilniki. Pri ostalih mikrokrmilnikih hitrosti ne bodo nujno enake tem, ki jih uporablja STM32F407.

Pull

Nabor možnih vrednost za element `Pull` je zapisan spodaj. Kot vidite tu knjižnica uporablja enake vrednosti, kot ste jih uporabili pri vaši rešitvi prejšnje vaje.

```
#define  GPIO_NOPULL          0x00000000U
#define  GPIO_PULLUP          0x00000001U
#define  GPIO_PULLDOWN        0x00000002U
```

Pin

Največje razlike v primerjavi z vašimi knjižnicami se pojavijo pri naboru vrednost za `Pin`:

```
#define GPIO_PIN_0                 ((uint16_t)0x0001)
#define GPIO_PIN_1                 ((uint16_t)0x0002)
#define GPIO_PIN_2                 ((uint16_t)0x0004)
#define GPIO_PIN_3                 ((uint16_t)0x0008)
#define GPIO_PIN_4                 ((uint16_t)0x0010)
#define GPIO_PIN_5                 ((uint16_t)0x0020)
#define GPIO_PIN_6                 ((uint16_t)0x0040)
#define GPIO_PIN_7                 ((uint16_t)0x0080)
#define GPIO_PIN_8                 ((uint16_t)0x0100)
#define GPIO_PIN_9                 ((uint16_t)0x0200)
#define GPIO_PIN_10                ((uint16_t)0x0400)
#define GPIO_PIN_11                ((uint16_t)0x0800)
#define GPIO_PIN_12                ((uint16_t)0x1000)
#define GPIO_PIN_13                ((uint16_t)0x2000)
#define GPIO_PIN_14                ((uint16_t)0x4000)
#define GPIO_PIN_15                ((uint16_t)0x8000)
#define GPIO_PIN_All               ((uint16_t)0xFFFF)
```

Namesto številčnih oznak za pine (0, 1, 2, ...) uradna knjižnica uporablja imenske konstante, za katerimi se skrivajo zastavice. Prednost tega pristopa je, da lahko z enim klicem funkcije za inicializacijo nastavimo več pinov, medtem ko ste pri vaši knjižnici za inicializacijo LED diod potrebovali štiri klice. Primer inicializacije izhodnih pinov `PE3` in `PE6` je prikazan spodaj:

```
__HAL_RCC_GPIOE_CLK_ENABLE();

GPIO_InitTypeDef init_structure;
init_structure.Pin = GPIO_PIN_3 | GPIO_PIN_6;
init_structure.Mode = GPIO_MODE_OUTPUT_PP;
init_structure.Pull = GPIO_NOPULL;
init_structure.Speed = GPIO_SPEED_FREQ_LOW;

HAL_GPIO_Init(GPIOE, &init_structure);
```

Branje vhoda GPIO naprave

HAL ima za branje vhoda na voljo funkcijo `HAL_GPIO_ReadPin()`. Funkciji podamo kazalec na strukturo `GPIO_TypeDef` ter oznako pina. Primer branja vrednosti na pinu `PE3`: `HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_3)`. V kodi funkcije, ki je prikazana spodaj, lahko vidite, da je implementacija branja vhoda v HAL podobna vašim implementacijam. Dodano je le nekaj preverjanj vhodnih vrednostih.

```
typedef enum
{
  GPIO_PIN_RESET = 0,
  GPIO_PIN_SET
} GPIO_PinState;

GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx,
                               uint16_t GPIO_Pin)
{
  GPIO_PinState bitstatus;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
  {
    bitstatus = GPIO_PIN_SET;
  }
  else
  {
    bitstatus = GPIO_PIN_RESET;
  }
  return bitstatus;
```

```
24 }
```

Nastavljanje izhoda GPIO naprave

Za nastavljanje izhoda imamo na voljo dve funkciji: `HAL_GPIO_WritePin()`, ki nastavi pin na poljubno vrednost ter `HAL_GPIO_TogglePin()`, ki negira trenutno vrednost pina. Podobno kot pri branju, pri obeh podamo kazalec na strukturo naprave, podobno kot pri branju. Pri `HAL_GPIO_WritePin()` dodatno podamo `GPIO_PIN_RESET` (logična ničla) ter `GPIO_PIN_SET` (logična enica). Primeri uporabe omenjenih funkcij so prikazni spodaj:

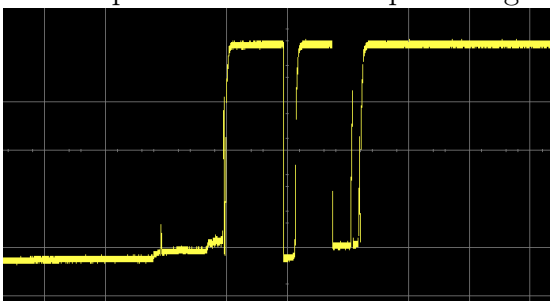
```
1 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
2 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12 | GPIO_PIN_13, GPIO_PIN_SET
      );
3 HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
```

Kot vidite v zgornjem primeru tudi tukaj lahko nastavljamo vrednost večih pinov hkrati. Konstante `GPIO_PIN_X` so namreč vrednosti, ki imajo enico zgolj na pinu `X`, povsod drugje pa vrednost 0, tako lahko z logičnim ali izberemo več pinov hkrati. Pine lahko izberemo tudi tako, da vrednost parametra za izbiro pinov določimo samo, vrednost 0x7002 (binarno 0111 0000 0000 0010), bi tako izbrala pine 14, 13, 12 in 1.

Branje stanja gumba

Zadnja tema, ki se jo bomo lotili na tej vaji, je branje gumba. Pri branju gumbov sicer zaenkrat nismo imeli bistvenih težav. Namerno se namreč nismo lotili nalog, kjer bi do težav lahko prišlo. Če pa bi s kodo iz zadnje vaje želeli implementirati števec pritiskov gumba, bi zelo hitro opazili, da bi vaša koda velikokrat zaznala več klikov gumba. Razlog tiči v fizikalnih lastnostih gumba. Namesto čistega in lepega prehoda iz ničle v enico, se namreč na vhodni liniji pojavi signal podoben temu na sliki 1.

Slika 1: Napetost na vhodu ob pritisku gumba.



Najenostavnejša rešitev te težave, ki se ji angleško reče *button debounce* je, da po tem, ko zaznamo spremembo signala iz nič v ena, počakamo preden ponovno beremo stanje gumba. Po domače povedano v naš program vstavimo zakasnitev (angl. delay). Ta je lahko zelo kratka (nekaj 10-100ms), saj napetost dokaj hitro izniha.

Psevdokoda, ki šteje število pritiskov v globalno spremenljivko vendar ne rešuje omenjene težave z debounce-om je prikazana spodaj.

```
1 uint8_t counter = 0;
2 uint8_t old = 0;
```

```
uint8_t new = 0;

while(1) {
    // preberemo novo stanje gumba
    // funkcija read_button
    new = read_button();

    // ce je bilo prejsnje stanje gumba 0
    // sedaj pa je 1, se je zgodil klik (prehod iz 0 v 1)
    if (old == 0 && new == 1) {
        counter++;
    }

    // shranimo trenutno stanje gumba kot prejsnje
    old = new;
}
```

Naloga

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 2.

Slika 2: Prikaz vrednosti števca na LED diodah (od 0 do 4).

Ob kratkem pritisku gumba povečajte vrednost števca za 1, ob dolgem pritisku (2 sekundi ali več) pa nastavite vrednost števca na 0. Za implementacije morebitnih zakasnitev lahko uporabite funkcijo `HAL_Delay(ms)`, ki ji podate število milisekund za zakasnitev. Zaznavanje dolgega klika ne sme blokirati zaznavanja kratkih klikov, torej rešitev kjer za ločevanje kratkih in dolgih klikov uporabite 2 sekundno zakasnitev ni primerna. Za izhodišče vzemite prazen projekt v razvojnem okolju.