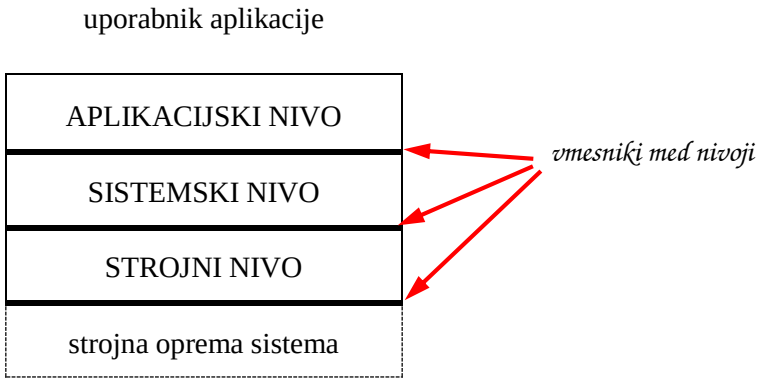
Osnove mikroprocesorske elektronike

Vaja 5: Tri-nivojsko programiranje in LED skupina

Pred vami je vaja, ki je pomembna iz več vidikov. Pri tej vaji se boste namreč srečali z idejo *tri-nivojskega programiranja*. Gre za pomembno *strategijo pri organizaciji programske opreme* vgrajenega sistema, ki programsko kodo *razdeli na tri nivoje glede na funkcionalnost*, ki jo koda implementira:

- na najnižji *strojni nivo*,
- na srednji *sistemski nivo* ter
- na najvišji *aplikacijski nivo*.

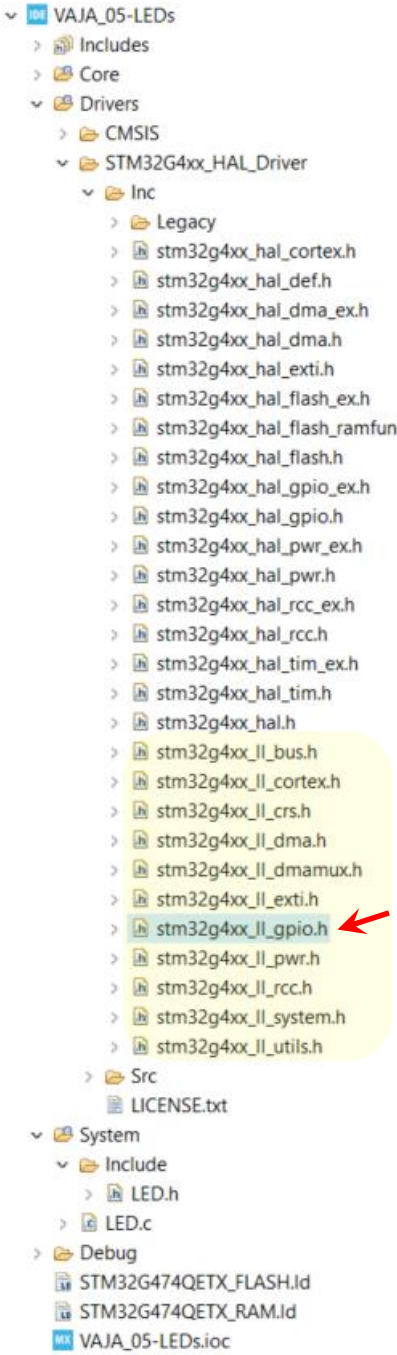
To preprosto idejo prikazuje slika spodaj.



Slika 1 – ideja razdelitve programske opreme vgrajenega sistema na tri osnovne nivoje. Nivoji so med seboj povezani preko vmesnikov (angl. interfaces).

Tak način *organizacije programske opreme* je pri vgrajenih sistemih zelo pogost, saj v programsko kodo vnaša *modularnost*, enostavno *povezljivost* in veliko *prenosljivost rešitev* (angl. reusability). Namreč, ker so *posamezni nivoji pravzaprav neodvisni drug od drugega* in komunicirajo le preko *vmesnikov* (debele črne črte na zgornji sliki), je modul zelo enostavno prenesti v drug sistem (s tujko je tako zasnovana programska koda poimenovana "*loosely coupled code*"). Vse kar je potrebno v novem sistemu zagotoviti je, da se modul "ujame" z drugimi moduli *na nivoju vmesnika*. Preostala programska koda, ki pa pravzaprav implementira *funkcionalnost* modula, pa praktično lahko ostane nespremenjena! In poglejte: če pa zamenjamo strojno opremo sistema, zato ni potrebno spreminjati funkcij aplikacijskega ali sistemskega nivoja, pač pa le funkcije strojnega nivoja!

Strojni nivo



V primeru naših laboratorijskih vaj bodo ***funkcije strojnega nivoja,* ki *upravljajo s strojno opremo* mikrokrmilnika *na najnižjem nivoju***, že na voljo v obliki nizko-nivojskih funkcij iz tako imenovane *"low-layer drivers" knjižnice* (s kratico: LL knjižnice).

LL knjižnico nam pomaga v naš projekt pravilno vključiti kar samo razvojno orodje STM Cube IDE oziroma grafični vmesnik CubeMX z avtomatskim generiranjem programske kode s pomočjo te knjižnice. Poglejte primer projekta na sliki levo. Datoteke, ki v imenu vsebujejo "_ll_", so datoteke LL knjižnice. In mimogrede vidite, da je na voljo tudi knjižnica za delo z GPIO periferijo.

Več o LL knjižnici si lahko preberete v uvodnem poglavju dokumentacije knjižnice (kratek izsek je na voljo spodaj).

Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features.

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

Sistemski nivo in aplikacijski nivo

Vaša naloga tekom te vaje ter večine naslednjih vaj pa bo, da implementirate funkcije sistemskega nivoja.

Sistemske funkcije so funkcije srednjega nivoja in njihova naloga je, da uporabijo funkcionalnost, ki jo nudi strojni nivo, ter z njegovo pomočjo "oživijo" module sistema (kot npr. LED skupina, tipkovnica, serijski vmesnik ipd.).

Razložimo še drugače: na *sistemskem programskem nivoju* je potrebno poskrbeti, da se funkcije *strojnega* nivoja uporabijo tako, da *moduli sistema postanejo na voljo za uporabo*. Vendar pa sistemski nivo sam po sebi (načeloma) ne uporablja teh modulov. Te sistemske module nato uporablja *aplikacijski* nivo! ***Aplikacija je tista, ki preko uporabe modulov sistemskega nivoja uporablja strojno opremo tako, kot to želi uporabnik!***

Če ponazorimo s primerom iz sveta računalništva: operacijski sistem Windows (tj. sistemski nivo) uporablja zvočno kartico računalnika (tj. modul sistema, "device") tako, kot to zahteva aplikacija za predvajanje videa (tj. aplikacijski nivo). Da pa sistem lahko pravilno uporablja zvočno kartico, pa potrebuje gonilnike, ki z zvočno kartico upravljajo na najnižjem nivoju (tj. strojni nivo).

Ta razloček je pomemben, saj nam pomaga ločiti sistemski nivo od aplikacijskega. Če tega ne razumemo dobro, se lahko hitro zgodi, da se programje sistemskega nivoja prične mešati s programjem aplikacijskega nivoja. Saj stvar bo še vedno delovala, zapletlo pa se bo, ko bo potrebno bodisi spremeniti funkcionalnost sistemskega nivoja ali pa spremeniti funkcionalnost aplikacije! Zapletlo se bo pa zato, ker bosta oba nivoja združena in zato soodvisna (t. i. "tightly coupled code").

Da ponazorimo delitev na tri programske nivoje še na primeru naše vaje z LEDicami:

- funkcije strojnega nivoja bomo uporabili *za krmiljenje digitalnih GPIO izhodov,*
- sistemski nivo bo take digitalne izhode uporabil tako, da bo z njimi oživil *LED modul,*
- aplikacija na koncu semestra pa bo na primer LED modul uporabila kot *indikator točnosti*, ki pokaže, kako dobro ste z merkom na puški ujeli tarčo.

Opaziti je potrebno še, kako na poti od strojnega nivoja proti aplikacijskemu nivoju dobiva strojna oprema oziroma sistemski modul *čedalje bolj specifičen namen*. Poglejte spodaj.

strojni nivo	**sistemski nivo**	**aplikacijski nivo**
digitalni GPIO izhodi	LED modul	LED indikator točnosti

zelo nizek nivo, zelo blizu strojni opremi mikrokrmilnika; ***splošen namen uporabe*** (npr. digitalni izhod lahko krmili rele, luč, piskač ipd.)

s pomočjo strojnega nivoja smo implementirali funkcionalnost sistemskemu modulu

modul je sedaj ***na voljo*** za uporabo, vendar pa ***še nima specifičnega namena uporabe***

aplikacija uporabi sistemski LED modul *za točno določen namen*, ki je aplikaciji pač potreben

Naloge vaje

Kot že nakazano zgoraj, vaša naloga pri tej vaji bo, da *implementirate funkcije sistemskega nivoja*, ki bodo oživile LED modul.

LED modul na našem Miško3 sistemu sestavljati dve skupini LEDic, vsaka s štirimi LEDicami. Čeprav so *LEDice fizično ločene na tiskanini* (glejte spodaj), bi jih *želeli uporabljati na tak način, kot da je vseh osem LEDic urejenih v vrsto*: od manj pomembne LSB LEDice `LED0` na desni pa do najbolj pomembne MSB LEDice `LED7` na levi. Vidite torej, da je potrebno na sistemskem nivoju implementirati nekakšno v vrsto *urejeno LED skupino*. **To boste storili tako, da boste ustrezno dopolnili izhodiščni `LED.c` in `LED.h` datoteki, ki ste jih tekom priprave dodali vašemu projektu.**

Slika 2 – Dve skupini po štiri LEDice bi želeli uporabljati na tak način, kot da je vseh osem LEDic urejenih v vrsto eno za drugo.

Implementirajte torej funkcionalnost *urejene LED skupine* tako, da izpolnite sledeče naloge:

1. ***Definirajte podatkovno strukturo, ki bo hranila vse potrebne parametre za upravljanje LED skupine.***

   Taki podatkovni strukturi bomo s tujko v žargonu rekli ***"handle structure"*** oziroma "handle" struktura. Definirali jo boste s pomočjo naslednjih korakov:

   - definirajte naštevni tip `LEDs_enum_t`, kjer boste definirali imena vseh LEDic, ki so na voljo v sistemu (`LED.h` datoteka)
   - definirajte tip "handle" strukture `LED_handle_t`, ki hrani vse potrebne parametre za upravljanje ene same LEDice
   - definirajte tip "handle" strukture `LED_group_handle_t`, s pomočjo katere pa bomo lahko upravljali vse LEDice v LED skupini
   - na podlagi zgornjega tipa definirajte *globalno* spremenljivko `LED_group`, ki pa bo naša "handle" struktura za LED skupino

2. Inicializirajte "handle" strukturo `LED_group` ***znotraj*** `LED_init()` ***funkcije.***

Pri inicializaciji boste poskrbeli, *da za vsako LEDico v skupini specificirate, na kateri GPIO pin in port je priključena*. Ta informacija je ključna, če želite LEDice krmiliti s pomočjo strojnih funkcij iz LL knjižnice.

3. Implementirajte sistemske funkcije za upravljanje LED skupine.

Ko imate enkrat na voljo "handle" strukturo, ki vam pomaga upravljati modul na strojnem nivoju, se pa lahko lotite implementacije funkcij na sistemskem nivoju.

LED skupino bomo upravljali na dva načina:

1) upravljanje posamezne LEDice
2) upravljanje vseh LEDic kot urejene LED skupine

V sklopu upravljanja posamezne LEDice implementirajte sledeče sistemske funkcije:

- `void LED_on( LEDs_enum_t LEDn )`
- `void LED_off( LEDs_enum_t LEDn )`
- `void LED_toggle( LEDs_enum_t LEDn )`
- `void LED_set( LEDs_enum_t LEDn, uint8_t state )`

V sklopu upravljanja LEDic kor urejene LED skupine implementirajte sledeče sistemske funkcije:

- `void LEDs_on(uint8_t LED_bitmask)`
- `void LEDs_off(uint8_t LED_bitmask)`
- `void LEDs_write(uint8_t value)`

Implementirajte tudi funkcijo, ki vrača informacijo o trenutnem stanju LEDice:

- `uint8_t LED_is_on(LEDs_enum_t LEDn)`

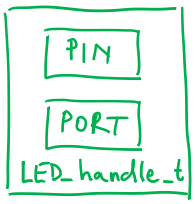
Zahtevano delovanje vseh funkcij je obrazloženo v komentarjih znotraj `LED.c` datoteke.

4. Znotraj funkcije `LED_demo()` ***po lastni presoji spišite kodo, ki bo stestirala zgorjnje sistemske funkcije.***

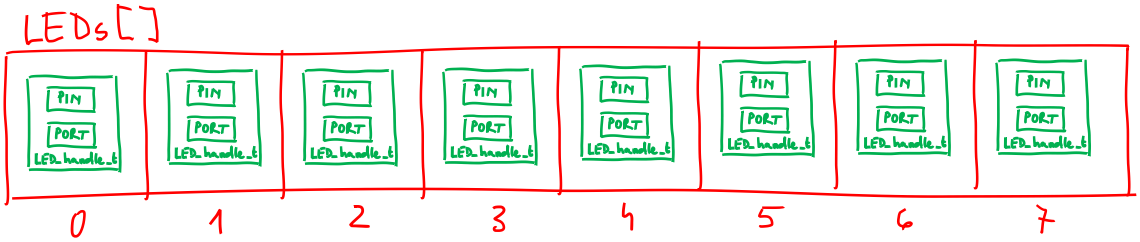
Dodatna pojasnila

Ideja "handle" podatkovne strukture za LED skupino

Ponazorimo idejo te podatkovne strukture na kratko kar s skicami. Če bi bilo potrebno upravljati *eno samo* LEDico, bi potrebovali eno samo `LED_handle_t` strukturo, ki bi hranila informacijo o tem, na kateri GPIO pin in na kateri GPIO port je LEDica priključena. Poglejte spodaj.



Ker pa imamo opravka z več kot eno LEDico v LED skupini, si moramo za vsako od LEDic zapomniti isto informacijo. Torej za vsako od teh LEDic potrebujemo eno `LED_handle_t` strukturo. In to množico struktur je smiselno urediti v tabelo, takole:

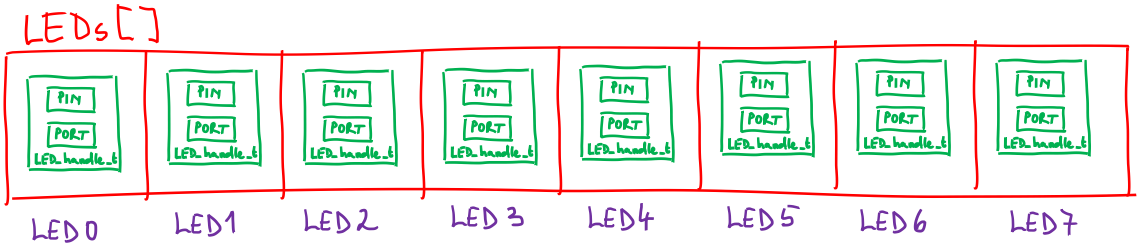


Vsak element tabele je ena `LED_handle_t` struktura za upravljanje ene LEDice v našem sistemu.

Kako si pa bomo zapomnili, katera "handle" struktura je uporabljena za katero od LEDic v našem sistemu? Uporabimo idejo, ki ste jo srečali na prvi laboratorijski vaji:

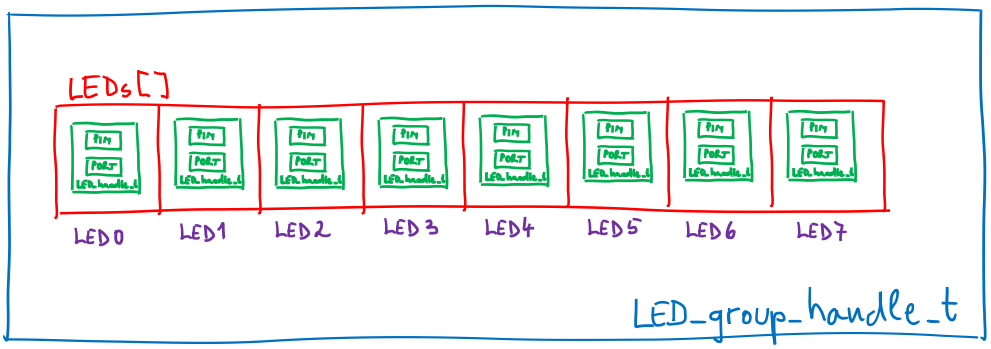
> za naslavljanje elementov tabele uporabimo vrednosti naštevnega tipa!

Če si pametno pripravimo naštevni tip `LEDs_enum_t`, potem lahko namesto številskih indeksov uporabljamo pomenljive elemente naštevnega tipa za naslavljanje elementov tabele, takole:



Sedaj pa vse kar je še potrebno narediti, je to, da zgornjo tabelo "zavijemo" v "handle" *strukturo* za upravljanje LED skupine.

Tako dobimo sledečo podatkovno strukturo *tipa* `LED_group_handle_t`:

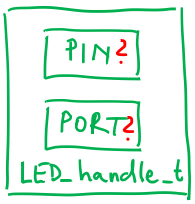


Vidite, da *ko ustvarimo spremenljivko* `LED_group` tipa `LED_group_handle_t`, dobimo strukturo, ki vsebuje tabelo, kjer vsak element tabele pomaga upravljati z eno od LEDic v skupini. Da pa ne pomešamo "handle" struktur posameznih LEDic med seboj, si pa pomagamo z naslavljanjem tabele s pomočjo elementov naštevnega tipa (vijolični del).

Vprašanje: zakaj je potrebno tabelo `LEDs[]` še "tlačiti" v strukturo? Zakaj ne uporabljamo zgolj tabele?

Programerska praksa je pokazala, da je ugodno parametre, ki jih potrebujete za upravljanje določenega modula, imeti združene znotraj strukture. Na ta način so parametri smiselno urejeni v skupino, ki pa *omogoča zelo enostavne nadgradnje in modifikacije brez velikih sprememb v programski kodi*. Prednost je torej predvsem v *fleksibilnosti* tako urejenih parametrov.

Kakšnega tipa naj bosta parametra "pin" in "port"?



Takole razmišljamo: parametra "pin" in "port" naj bosta takega tipa, da bomo z njima lahko kar *neposredno poklicali nizko-nivojske LL funkcije* za upravljanje digitalnih GPIO izhodov.

Kako pa izvemo, kakšen tip natančno je potreben? Pogledati je potrebno definicijo katere od teh LL GPIO funkcij, ki jih bomo uporabljali. Poglejte primer spodaj iz dokumentacije. Tip je naveden pri definiciji funkcije.

Function name

`__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)`

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values: