Osnove mikroprocesorske elektronike

Vaja 6: Tipkovnica

V sklopu te vaje boste na sistemskem nivoju *implementirali funkcionalnost* tipkovnice. Videli boste, da je *metodologija*, ki jo uporabimo za implementacijo tipkovnice, popolnoma enaka metodologiji, ki smo jo uporabili za implementacijo funkcionalnosti LED modula, in vsebuje sledeče ključne korake:

1. *definicija podatkovne strukture* za delo z modulom (tj. "handle structure"),
2. *inicializacija* "handle" strukture,
3. *implementacija funkcionalnosti* modula na sistemskem nivoju s pomočjo nizko-nivojskih strojnih funkcij,
4. *testiranje* sistemskih funkcij.

Ta *pristop* se bo ponavljal tekom vseh naslednjih vaj in vam lahko *predstavlja osnovno ogrodje* vsakokrat, ko se lotite implementacije funkcionalnosti modula *na sistemskem nivoju*.

Slika 1 - tokrat bomo "oživili" tipkovnico Miškota

Naloge vaje

Implementirajte *funkcionalnost tipkovnice* tako, da izpolnite sledeče naloge:

1. ***Definirajte podatkovno strukturo, ki bo hranila vse potrebne parametre za delo s tipkovnico.***

   Definirali jo boste s pomočjo naslednjih korakov:

   - definirajte naštevni *tip* `buttons_enum_t`, kjer boste *definirali imena vseh tipk*, ki so na voljo v sistemu (`kbd.h` datoteka); poleg `NUM_OF_` elementa dodajte tudi fiktivno tipko `BTN_NONE`, s katero bo sistem sporočal, da ni bila pritisnjena nobena tipka
   - definirajte *tip* "handle" strukture `button_handle_t`, ki hrani vse potrebne parametre za delo *z eno samo* tipko
   - definirajte *tip* "handle" strukture `keyboard_handle_t`, ki pa bo hranila vse potrebne parametre za delo *s celotno* tipkovnico
   - na podlagi zgornjega tipa definirajte *globalno* spremenljivko `keyboard`, ki pa bo naša "handle" struktura za tipkovnico

2. ***Inicializirajte "handle" strukturo*** `keyboard` ***znotraj*** `KBD_init()` ***funkcije.***

   Pri inicializaciji "handle" strukture boste poskrbeli:

   - da *za vsako tipko v tipkovnici specificirate, na kateri GPIO pin in port je priključena* (ta informacija je ključna, če želite stanje tipk brati s pomočjo strojne funkcije iz LL knjižnice)
   - da *inicializirate začetno vrednost sistemskih spremenljivk*, ki se tičejo posameznih tipk (za vsako tipko se namreč hrani vrednost prejšnjega in trenutnega stanja)
   - da *inicializirate ciklični medpomnilnik*, ki bo pomnil, katere tipke so bile pritisnjene, da se bomo lahko kasneje odzvali na njihov pritisk

3. ***Implementirajte sistemske funkcije za delo s tipkovnico.***

   Do konca boste implementirali boste sledeče sistemske funkcije:

   - `KBD_scan()` – funkcija za branje trenutnega stanja tipk ter detekcijo pritiska tipk (glejte namig spodaj)
   - `KBD_get_pressed_key()` – funkcija, ki iz medpomnilnika vrne informacijo o tem, katera je naslednja pritisnjena tipka, na katero se še nismo odzvali
   - `KBD_get_button_state()` – funkcija, ki vrne trenutno stanje posamezne tipke

4. ***Znotraj funkcije*** `KBD_demo()` ***spišite testno kodo***, ki bo *preizkusila* in hkrati *demonstrirala* sistemske funkcije tipkovnice.

   Pri tem si lahko pomagate z že zastavljeno rešitvijo, ki preizkusi tipkovnico s pomočjo LED modula: pritisk tipke obrne stanje določene LEDice (angl. toggle).

Dodatna pojasnila

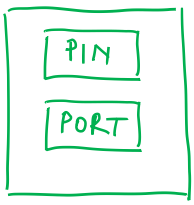
Ideja "handle" podatkovne strukture za tipkovnico

Ponazorimo idejo te podatkovne strukture s skicami, podobno kot pri prejšnji vaji.

Če bi želeli s pomočjo funkcije LL knjižnice le brati *trenutno stanje* enega digitalnega vhoda, potem bi v "handle" strukturi morali hraniti le dva parametra:

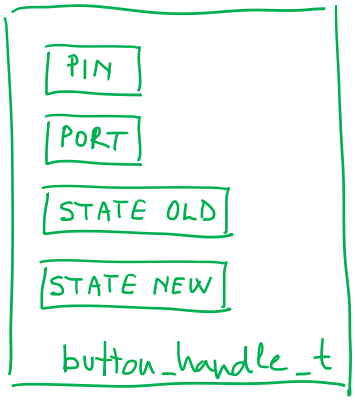
- na katerem *pinu* se nahaja naš digitalni vhod ter
- h kateremu *portu* pripada ta pin.

Zelo podobna situacija kot pri "handle" strukturi za upravljanje ene same LEDice. Glejte spodaj.



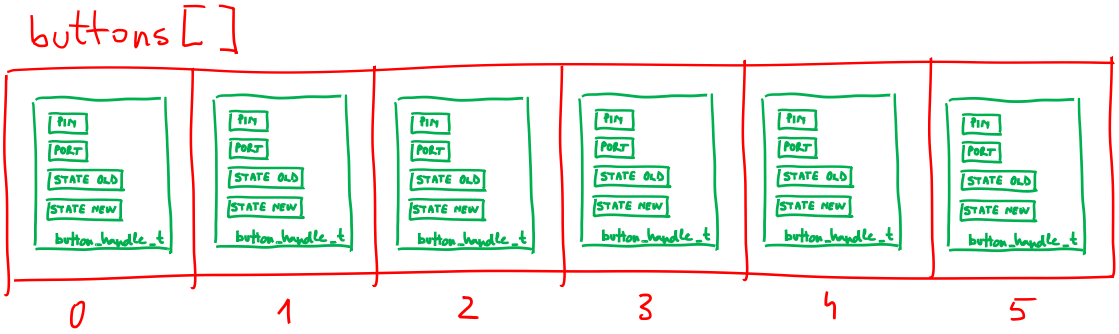
Vendar mi pa želimo, da naš sistem zaznava *pritisk* tipke in ne le njeno *trenutno stanje*!

Če pa želimo zaznati pritisk tipke, moramo biti sposobni zaznati *spremembe stanja* tipke. To pa pomeni, da moramo poznati *prejšnje staro stanje* tipke ter *trenutno novo stanje* tipke. Zato je smiselno, da "handle" strukturi dodamo še dve spremenljivki, ki bosta pomnili prejšnje in trenutno stanje tipke. Tako se dokopljemo do `button_handle_t` strukture, ki hrani vse potrebne parametre za delo z eno tipko na tipkovnici.

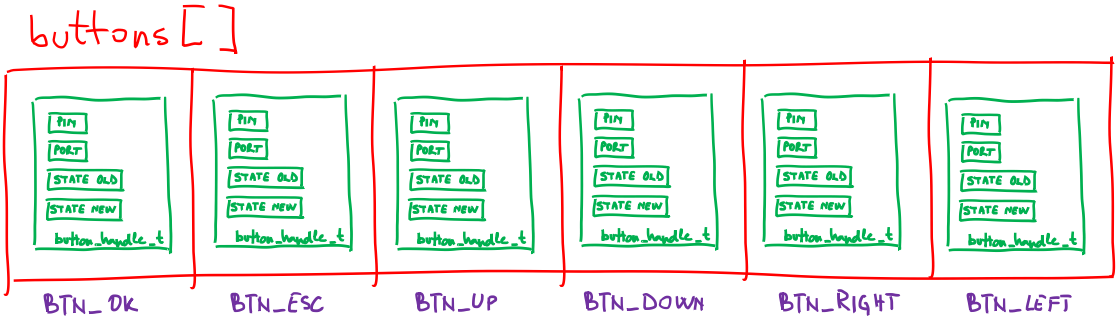


Ker pa imamo v sistemu Miško več kot le eno tipko, bomo potrebovali več kot le eno `button_handle_t` strukturo. Za vsako tipko bomo potrebovali svojo `button_handle_t` strukturo. In to množico `button_handle_t` struktur je seveda smiselno urediti v tabelo `buttons[]`, podobno kot smo to storili pri prejšnji vaji s "handle" strukturami za posamezne LEDice.

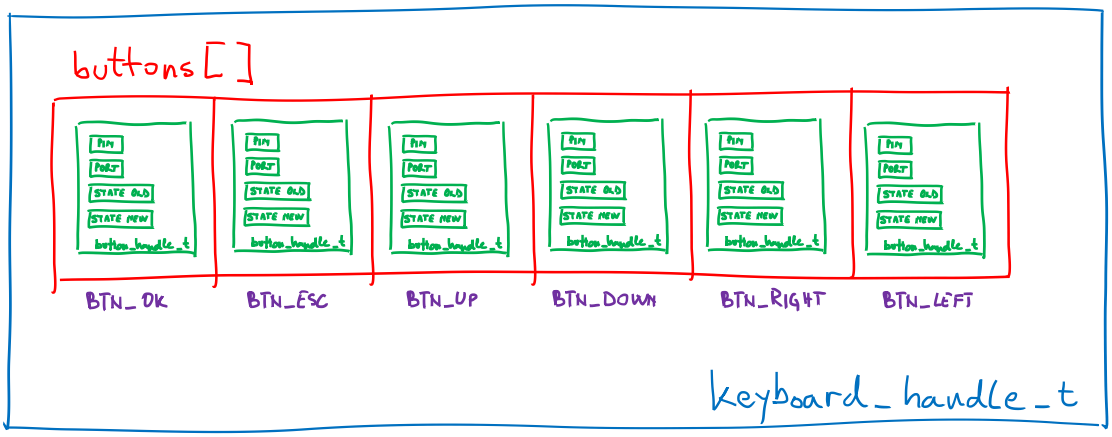
Tako dobimo sledečo tabelo "handle" struktur za vse tipke sistema.



In da ne bomo pozabili ali pomešali, v katerem elementu tabele `buttons[]` se hrani "handle" struktura za točno določeno tipko, si pomagamo tako, da si pametno in pomenljivo pripravimo naštevni tip `buttons_enum_t`, ki vsebuje *seznam vseh tipk v sistemu*. In nato za naslavljanje elementov tabele uporabljamo kar vrednosti naštevnega tipa. Tako dobimo sledečo situacijo:



Zgornjo tabelo je potrebno le še "zaviti" v "handle" strukturo `keyboard_handle_t` za upravljanje celotne tipkovnice.



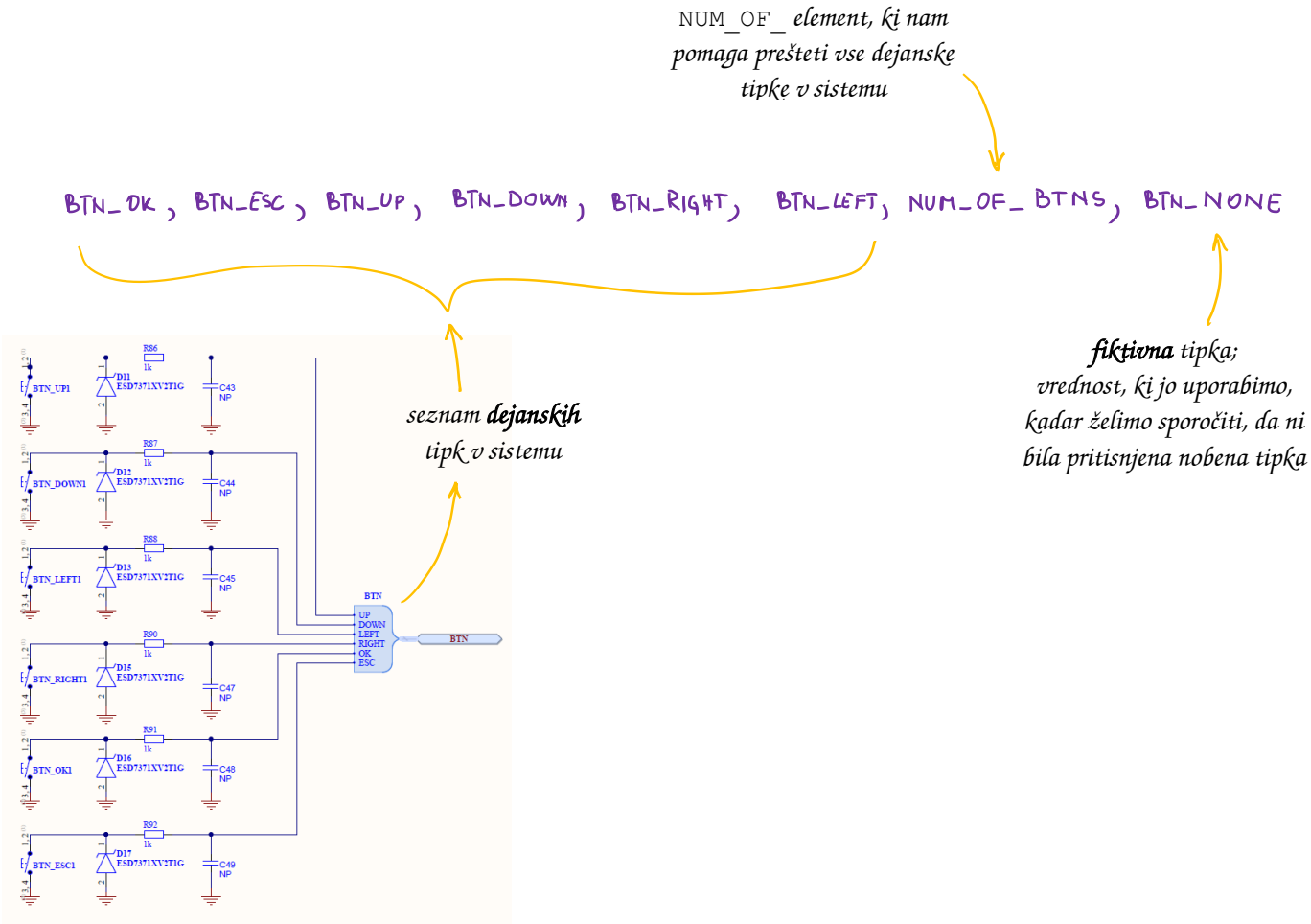
In s pomočjo tako definiranega *tipa* `keyboard_handle_t` bomo nato definirali *strukturno spremenljivko* `keyboard`. In še primer za boljšo predstavo: če bi nato želeli dostopati do "handle" strukture za tipko `UP`, bi to storili takole:

```
keyboard.buttons[BTN_UP]
```

Kot zanimivost: kasneje pri eni od naslednjih vaj bomo branje tipkovnice implementirali s pomočjo časovnika (angl. timer). In takrat bomo to isto *podatkovno strukturo razširili* tako, da bo vsebovala še parametre časovnika, ki bo uporabljen pri implementaciji. Takrat boste lahko spoznali, kako uporaba "handle" strukture elegantno omogoča razširljivost funkcionalnosti (angl. flexibility).

Dva dodatna elementa naštevnega tipa

Naštevni tip `buttons_enum_t` nam pomaga pri naslavljanju elementov tabele s "handle" strukturami za posamezne tipke. Lahko pa ga še malenkost dopolnimo in nam pomaga še pri dveh stvareh. Poglejte idejo spodaj.



Element `NUM_OF_` že poznate iz prejšnjih vaj. Ta element nosi informacijo o številu elementov v seznamu naštevnega tipa. Podobno vlogo ima tudi v seznamu zgoraj. Na koncu seznama pa smo dodali še en element: *fiktivno tipko* `BTN_NONE`. Ta element dodamo kot nekakšno posebno kodo, s katero bo sistem sporočal, da ni bila pritisnjena nobena (nova) dejanska tipka.

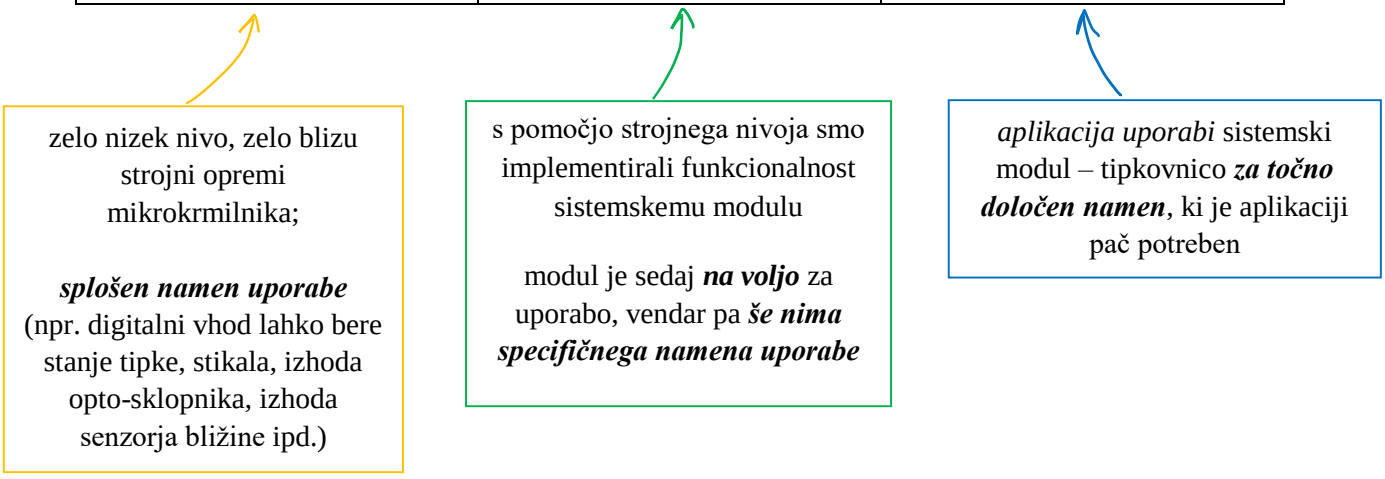
Razdelitev na tri programske nivoje na primeru vaje s tipkovnico

Da ponazorimo delitev na tri programske nivoje še na primeru tokratne vaje s tipkovnico:

- funkcije strojnega nivoja bomo uporabili *za branje digitalnih GPIO vhodov*,
- sistemski nivo bo take digitalne vhode uporabil tako, da bo z njimi oživil *tipkovnico*,
- aplikacija na koncu semestra pa bo na primer tipkovnico uporabila za *navigacijo med meniji* igrice

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 vhodi	tipkovnica	tipke za navigacijo med meniji



Namigi

Algoritem za detekcijo pritiska tipke

Znotraj funkcije `KBD_scan()` boste morali implementirati algoritem za *detekcijo pritiska* tipke. Naj vam spodnja izseka iz predavanj služita za osvežitev problematike detekcije pritiska.