Projektna naloga 1 – FreeRTOS osnove

V okviru projekta boste implementaciji dodali realno-časovni operacijski sistem FreeRTOS. Cilj prve naloge je priprava integracije vašega projekta z FreeRTOS. Realno-časovni operacijski sistem [RTOS] (primeri: FreeRTOS, VxWorks, QNX), je operacijski sistem namenjen zagonu realno-časovno aplikacij (več o tem ste slišali na predavanjih), ki ga običajno uporabljamo pri aplikacijah z več medsebojno povezanimi funkcionalnostmi (več na tem linku[1]). Osnovna komponenta programa so opravila (angl. *Tasks*), ki so v osnovi C funkcije, ki (običajno) implementirajo eno funkcionalnost in jih operacijski sistem (oz. razvrščevalnik) razvršča glede na prioritete. Razvrščevalnik (angl. Scheduler) je del jedra operacijskega sistema, ki določa, katero opravilo se bo izvajalo ob katerem časovnem trenutku. RTOS običajno vsebuje še MMU, mehanizme za obravnavanje kritičnih sekcij, mehanizme za *varno* komunikacijo med opravili, implementacijo preklopa konteksta, …

Pri projektu boste uporabljali prosto-dostopen operacijski sistem FreeRTOS (link), ki je najbolj pogosto uporabljen RTOS in je tudi integriran v STM32CubeIDE. Izvožen je na veliko različnih platform, dobro dokumentiran, dosti pomoči se najde tudi na spletu. V večini je napisan v programskem jeziku C, samo najbolj nujni deli so implementirani v zbirniku (npr. preklop konteksta).

1 Naloge

Naloga je razdeljena na več pod-nalog, ki so različno ocenjene. Točke se seštevajo.

1.1 Vprašanja o FreeRTOS [2T]

Cilj te naloge je, da se čim bolj spoznate z FreeRTOS, zato je potrebno odgovoriti na naslednja vprašanja, katerih odgovore boste našli v dokumentaciji FreeRTOS (link):

1) Kaj je opravilo? S katero funkcijo se ustvari? Kaj se zgodi z opravilom, ko se enkrat ustvari? Kako izgleda TCB?
2) Opišite diagram prehajanja stanj za opravila v FreeRTOS.
3) Na kakšen način FreeRTOS razvrsti opravila? Katere se izvedejo najprej?
4) Kaj je *pre-empitve* in kaj *co-operative* scheduling? Kako se nastavi v FreeRTOS?
5) Kaj je time-slicing in kje se definira/uporablja?
6) Poiščite/Skopirajte kodo za preklop konteksta.
7) Kakšen je namen datoteke FreeRTOSConfig.h?
8) Katere sheme alokacij pomnilnik podpira FreeRTOS?
9) Kaj je CMSIS? Kakšne je povezava z FreeRTOS?

[1] Why Use a Real-Time Operating System in MCU Applications

1.2 Priprava načrta za projekt [2T]

Pri tej nalogi si boste morali že malo zamisliti, kako bo izgledal vaš projekt na STM32 ploščici in sicer, katere funkcionalnosti (na primer IMU, mikrofon, komunikacija z ESP-01, TOF, ...) boste podprli in kako jih boste implementirali (npr. z uporabo prekinitev ali s pollingom). Ideja v FreeRTOS je, da poskusite (vsako) funkcionalnost pretvoriti v opravilo. Zamislite si tudi, kako bo potekala morebitna medsebojna komunikacija (pošiljanje podatkov med opravili) med opravili.

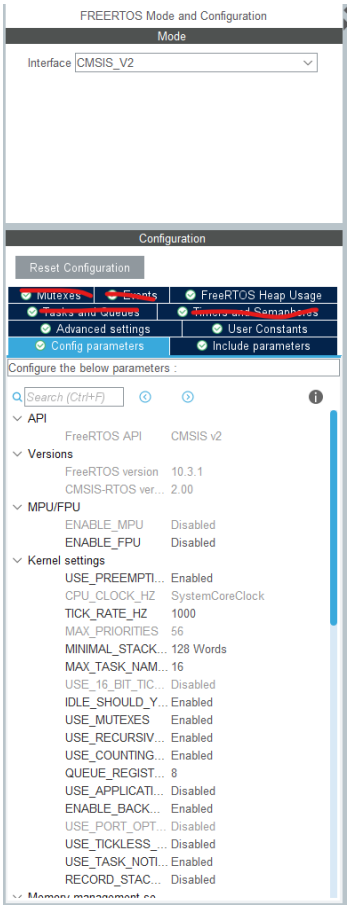
V nalogi je torej potrebno pripraviti sliko/shemo opravil/funkcionalnosti (recimo s kvadratki) in njihovo medsebojno komunikacijo (opravila z medsebojno komunikacijo povežite med seboj).

1.3 Priprava ogrodja za projekt [2T]

V tej naloge boste *ustvarili* opravila skladno s sliko/shemo, sestavljeno v prejšnji nalogi. Za zdaj je dovolj, da jih samo ustvarite (torej xTaskCreate in njihova implementacija – opravilo je C funkcija) s pravilnimi prioritetami (opravilo razvrstite po prioriteti).

Z STM32CubeIDE generatorjem kode ustvarite projekt, kjer je potrebno dodati *Middleware* FreeRTOS (uporabite *Interface CMSIS_V2*) in opravite osnovno konfiguracijo z nastavljanjem *Config parameters.* Ker je cilj projektnih nalog uporabljati *čisti* FreeRTOS, opravil, semaforjev, vrst ne boste definirali preko GUI vmesnika, ki nam ustvari *CMSIS interface* funkcije, ampak jih definirate v kodi v funkciji *main*. Opravila ustvarite z uporabo funkcije *xTaskCreate* in sicer jo je potrebno klicati PRED *osKernelStart()* (zažene se razvrščevalnik in program se nikoli več ne vrne v main).

- Opomba 1: *Timebase Source* nastavite na poljuben timer (recimo: Tim4)
- Opomba 2: USE_NEWLIB_REENTRANT nastavite na *Enabled*
- Opomba 3: v *main.c* je potrebno dodati: `#include "FreeRTOS.h"`



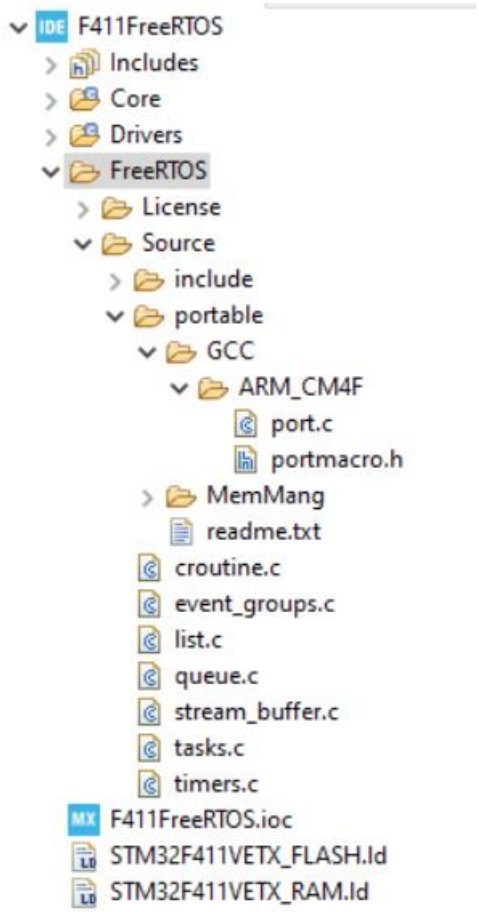
1.4 Ročna vključitev FreeRTOS [1T]

V tej nalogi je potrebno ustvariti nov STM32 projekt, kjer bomo *ročno* dodali FreeRTOS, s prenosom kode iz uradne spletne strani FreeRTOS (ta verzija je novejša, je pa to tudi vaja, kako bi to naredili, če bi uporabljali kakšno drugo ploščico, ki nima na voljo razvojnega okolja z integriranim FreeRTOS). Prikažite delovanje programa z ustvarjanjem dveh opravil, kjer vsaka vklaplja in izklaplja svojo diodo (eno opravilo vsako sekundo, drugo pa vsake 3 sekunde).

Postopek je naslednji:

1. Ustvarite projekt z uporabo STM32CubeIDE Cube generatorja kode (določite, katero periferijo boste potrebovali, porte, SPI, UART, …). *Tukaj bi lahko določili dodali tudi FreeRTOS, ampak tega ne bomo naredili, saj nam razvojno okolje doda še en dodani nivo kode (CMSIS) nad FreeRTOS.*
2. Prenesite si FreeRTOS s te spletne strani: link. Kodo razpakirajte.
3. V projektu ustvarite mapo FreeRTOS (lahko na isti nivo kot sta mapi *Core* in *Drivers*). V raziskovalcu odprite to mapo.

4. V novo ustvarjeno mapo iz razpakirane kode FreeRTOS:
    a. Skopirajte mapo *License*
    b. Skopirajte mapo *Source*
    c. Premaknite se v mapo Source / portable. Iz te mape odstranite vse, razen:
        - mapo GCC
        - mapo MemMang
        - datoteko readme.txt
    d. Premaknite se v mapo GCC (vsebuje različne kode potrebne za t.i. *porte* – gre se za kodo, ki je specifična za posamezen HW in je zapisana v assemblerju) in odstranite vse, razen mape ARM_CM4F.
5. Greste v STM32CubeIDE in z desnim tipko na miško pritisnite na mapo FreeRTOS in izberete Refresh. To vam prikaže novo dodane datoteke.



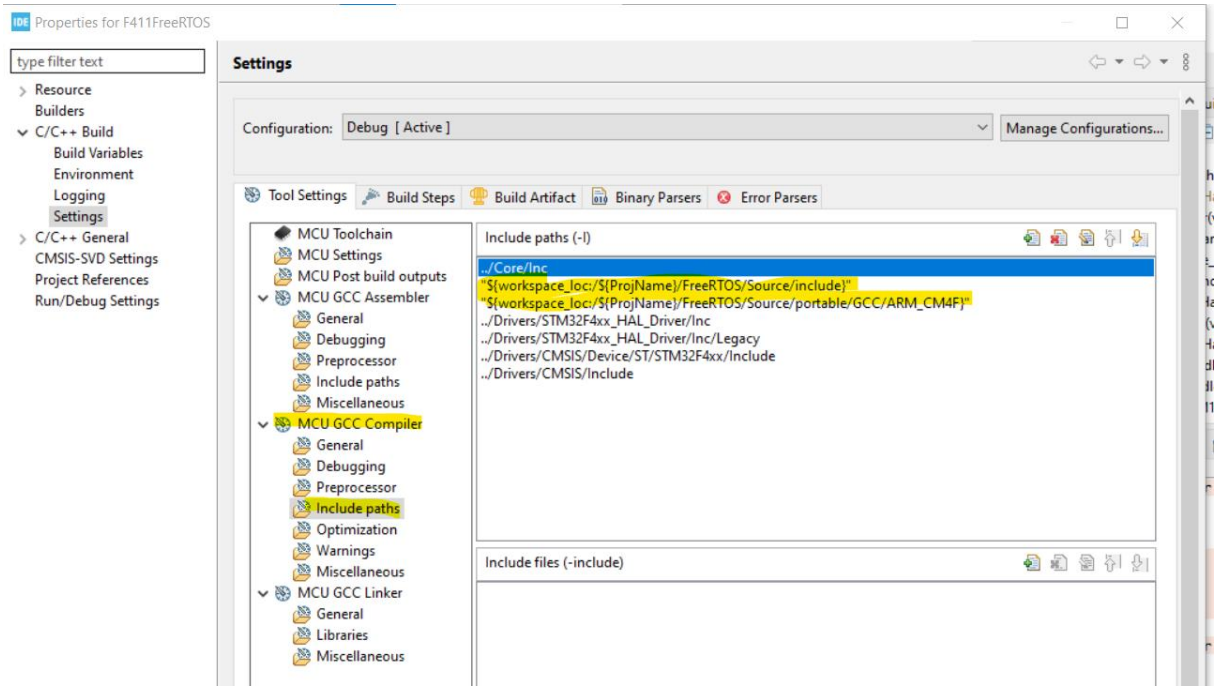
6. Sedaj je potrebno urediti še *include* stavke znotraj projekta:
    a. Najprej izberite mapo FreeRTOS in stisnite z desno tipko ter odprite lastnosti (P*roperties*). Premaknite se v *C/C++ Build* in odstranite (če še ni) kljukico pri zapisu *Exclude resource from build.*

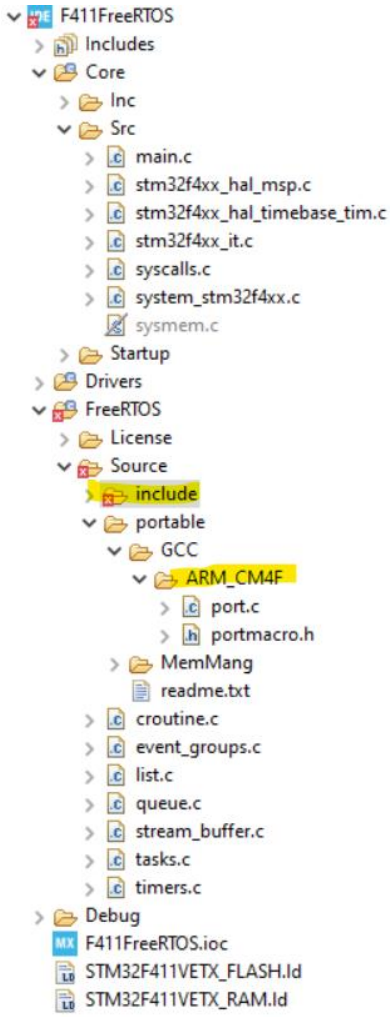
b. Izberite datoteko *Core/Src/sysmem.c* in stisnite z desno tipko ter odprite lastnosti (P*roperties*). Premaknite se v *C/C++ Build* in dodajte kljukico pri zapisu *Exclude resource from build.* FreeRTOS ima svoje načine/sheme alokacij pomnilnika.

c. V mapi FreeRTOS/Source/portable/MemMang imate različne sheme. Odstranite vse razen heap_4.c.

d. Dodajmo pot do spodaj označenih map.

+Nadaljevanje po naslednji strani.



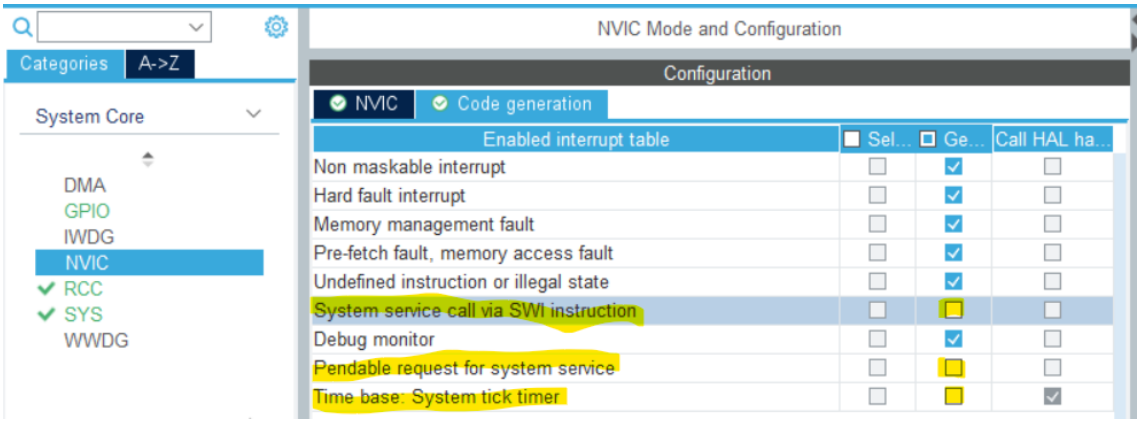


7. Dodati je potrebno datoteko FreeRTOSConfig.h. Najbolj preprost način je, da poiščemo demo za našo (ali podobno) razvojno ploščico in skopiramo FreeRTOSConfig.h od tam. Primer: FreeRTOSv202111.00\FreeRTOS\Demo\CORTEX_M4F_STM32F407ZG-SK. Skopirajte FreeRTOSConfig.h datoteko in jo prilepite v mapo *Source*. To mapo je potem potrebno dodati v *pot* (kot v prejšnjem koraku).
8. Prevajalnik bo javil napako, da ni definirana ura. Razlog se nahaja v FreeRTOSConfig.h datoteki. V datoteki je na vrhu napisan ifdef stavek, ki preverja, kateri prevajalnik se uporablja. Mi ne uporabljamo ICCARM prevajalnik, temveč GCC. Najpreprostejši način za reševanje tega problema je, da za komentiramo »ifdef« in »endif« vrstici.

```
//#ifdef __ICCARM__
#include <stdint.h>
extern uint32_t SystemCoreClock;
//#endif
```

9. Sedaj pa se nam pojavi nov problem. Pri prevajanju pride do treh *multiple-definition* napak in sicer: SVC_Handler, PendSV_Handler in SysTick_Handler. To so vse funkcije, ki nam jih je

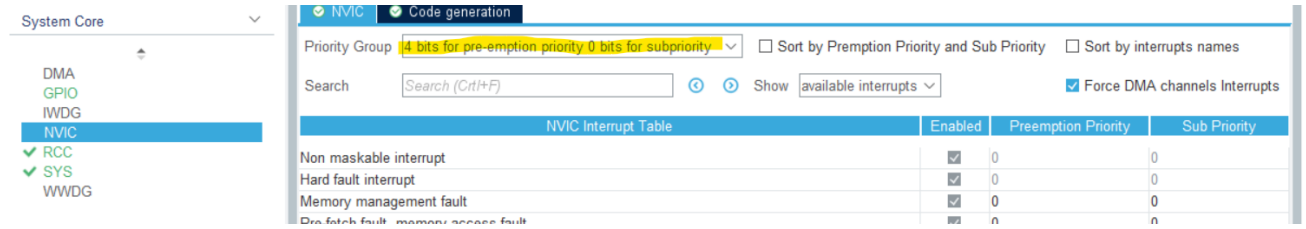
privzeto ustvaril Cube, ampak FreeRTOS vsebuje v datoteki *port.c* svoje implementacije teh funkcij, kar pomeni, da moramo privzete odstraniti. Ker uporabljamo Cube, jih ne moremo preprosto izbrisati, saj bi jih Cube ustvaril nazaj ob naslednjem generiranju kode. Zato moramo to narediti v konfiguraciji in sicer:



10. Naslednji *napaki* sta tudi vezani na FreeRTOSConfig.h datoteko. Spremeniti je potrebno dva define stavka:

    - configUSE_TICK_HOOK na 0
    - configCHECK_FOR_STACK_OVERFLOW na 0

11. Privzeto FreeRTOS in HAL za svoje delovanje uporablja SysTick, kar seveda ne gre. Zato je potrebno v SYS izbrati drug Timebase Source. Recimo izberite TIM5.
12. Spremenite nastavitev v NVIC:



To bi moralo biti vse. Če vam javi še kakšen HOOK error, ga odstranite v FreeRTOSConfig.h. V main.c vključite FreeRTOS.h in tasks.h. Ne pozabite PRED while(1) zanko klicati funkcije `vTaskStartScheduler();`