KATEDRA ZA ELEKTRONIKU

SISTEMI U REALNOM VREMENU

HARIS TURKMANOVIĆ - VEŽBE 2023/24 - ČAS 1

ORGANIZACIJA

OPŠTE NAPOMENE VEZANE ZA TOK IZVOĐENJA VEŽBI

HARDWARE I SOFTWARE POTREBAN ZA VEŽBE

Opšte napomene

Organizacija

- Vežbe se održavaju Online
  - *U okviru kanala „Vežbe“ grupe „Sistemi u realnom vremenu 2024/25“ objavljivaće se informacije vezane za izvođenje vežbi kao i materijali koji prate vežbe*
- Vežbe se održavaju u blokovima od po 3 sata
  - *Predviđena su 4 bloka računskih vežbi:*

    - 01.08.2025 (Petak) od 11 do 14h (Online MS Teams)

    - 09.08.2025 (Petak) od 11 do 14h (Online MS Teams)

    - 16.08.2025 (Petak) od 11 do 14h (Online MS Teams)

    - 12.09.2025 (Petak) od 11 do 14h (Online MS Teams)
- Laboratorijske vežbi se održavaju **uživo** u paviljonu – **Lab 18**
  - 2 laboratorijske vežbe koje se izvode u grupama od po 2 studenta i traju 3h

    - Septembar 2025 I LAB

    - Septembar 2025 II LAB

  - Link za prijavu grupe biće objavljen nakon prvog termina računskih vežbi

    - Dovoljno je da jedan student iz grupe prijavi grupu
    - Ukoliko se student samostalno ne prijavi u okviru neke grupe, biće mu dodeljena generička grupa.

Opšte napomene

Organizacija

- Vežbe se održavaju uživo
  - *U okviru kanala „Vežbe“ grupe „Sistemi u realnom vremenu 2022/23“ objavljivaće se informacije vezane za izvođenje vežbi kao i materijali koji prate vežbe*
- Vežbe se održavaju u blokovima od po 3 sata
  - *Predviđena su 4 bloka računskih vežbi:*
    - *01.08.2025 (Petak) od 11 do 14h (Online MS Teams)*
    - *09.08.2025 (Petak) od 11 do 14h (Online MS Teams)*
    - *16.08.2025 (Petak) od 11 do 14h (Online MS Teams)*
    - *12.09.2025 (Petak) od 11 do 14h (Online MS Teams)*
- Laboratorijske vežbi se održavaju **uživo** u paviljonu – **Lab 18**
  - Detaljan raspored grupa po terminima, kao i studenata po grupama, biće objavljen najkasnije 2 dana pre početka lab vežbe. **Dozvoljena je zamena termina između grupa**
  - Lab vežbe moguće je nadoknaditi isključivo u okviru termina regulatnih lab vežbi !?

    … ukoliko postoje slobodni računari

Opšte napomene

Organizacija

- U okviru dela vežbi moguće je osvojiti maks **50 poena**
  - Lab vežbe – 20 poena
    - *I Lab vežba – **7 poena***
    - *II Lab vežba – **13 poena***
    - *III Lab vežba – **Ne***
  - Projekat – 30 poena
    - Dizajn softvera i funkcionalnost realizacije – **10 poena**
    - Teorijski apekti realizacije – **20 poena**
- Lab vežbe pokrivaju određeni deo gradiva
  - I Lab vežba
    - Koncepti RTOS

      Dekompozicija na taskove, koncepti sinhronizacije taskova, signaliziranja događaja, prekidne rutine
    - Podrazumeva se korišćenje sledećih objekata FreeRTOSa
      - *Taskovi*
      - *Semafori*
      - *Softverski tajmeri*

Opšte napomene

Organizacija

- U okviru dela vežbi moguće je osvojiti maks **50 poena**
  - Lab vežbe – 30 poena
    - *I Lab vežba – **5 poena***
    - *II Lab vežba – **10 poena***
    - *III Lab vežba – **15 poena***
  - Projekat – 20 poena
    - Dizajn softvera i funkcionalnost realizacije – **10 poena**
    - Teorijski apekti realizacije – **10 poena**
- Lab vežbe pokrivaju određeni deo gradiva
  - II Lab vežba
    - Koncepti RTOS

      Dekompozicija na taskove, koncepti sinhronizacije taskova, signaliziranja događaja, deljena memorija, prekidne rutine
    - Podrazumeva se korišćenje sledećih objekata FreeRTOSa
      - *Taskovi*
      - *Semafori*
      - *Softverski tajmeri*
      - *Mutexi*
      - *Queue-ovi*

Opšte napomene

Organizacija

- U okviru dela vežbi moguće je osvojiti maks **50 poena**
  - Lab vežbe – 30 poena
    - *I Lab vežba – **5 poena***
    - *II Lab vežba – **10 poena***
    - *III Lab vežba – **15 poena***
  - Projekat – 20 poena
    - Dizajn softvera i funkcionalnost realizacije – **10 poena**
    - Teorijski apekti realizacije – **10 poena**
- Lab vežbe pokrivaju određeni deo gradiva
  - III Lab vežba
    - Koncepti RTOS

      Dekompozicija na taskove, koncepti sinhronizacije taskova, signaliziranja događaja, deljena memorija, prekidne rutine, zadovoljavanje strogih RT performansi
    - Podrazumeva se korišćenje sledećih objekata FreeRTOSa
      - *Taskovi*
      - *Semafori*
      - *Softverski tajmeri*
      - *Mutexi*
      - *Queue-ovi*
      - *Grupe događaja*
      - *Notifikacija taska*
      - *…*

Opšte napomene

Organizacija

- U okviru dela vežbi moguće je osvojiti maks **50 poena**
  - Lab vežbe – 30 poena
    - *I Lab vežba – **5 poena***
    - *II Lab vežba – **10 poena***
    - *III Lab vežba – **15 poena***
  - Projekat – 20 poena
    - Dizajn softvera i funkcionalnost realizacije – **10 poena**
    - Teorijski apekti realizacije – **10 poena**
- Projekat
  - Tekst postavke projektnog zadatka se objavljuje nakon završenih vežbi iz predmeta.
    - *Jasno definisani zahtevi koje embedded aplikacija bazirana na FreeRTOSu mora da zadovolji*
  - *Kompletna realizacija projektnog zadatka treba da sadrži asastoji iz dva glavna dela*
    1. *Opis arhitekture aplikacije bazirane na FreeRTOSu*
       - *U formi izveštaja opisati arhitekturu aplikacije*
       - *Jasno naznačiti taskove, njihovu međusobnu sinhronizaciju, način obrade događaja, ...*
       - *Cilj ove faze jeste da se prikaže način na koji je urađena dekompozicija zahteva na objekte RTOSa i da se približi način na koji će funkcionalnosti biti realizovane*

Opšte napomene

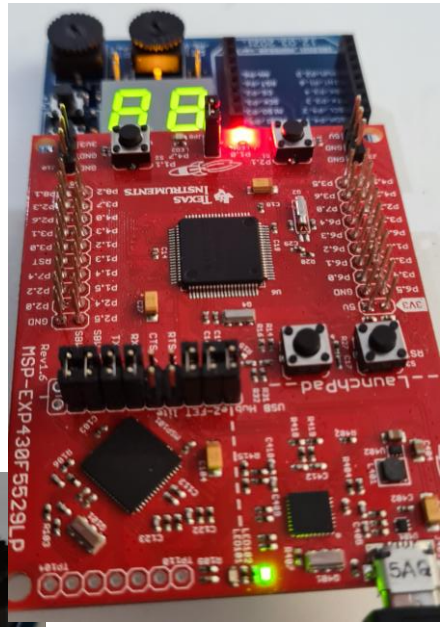
Organizacija

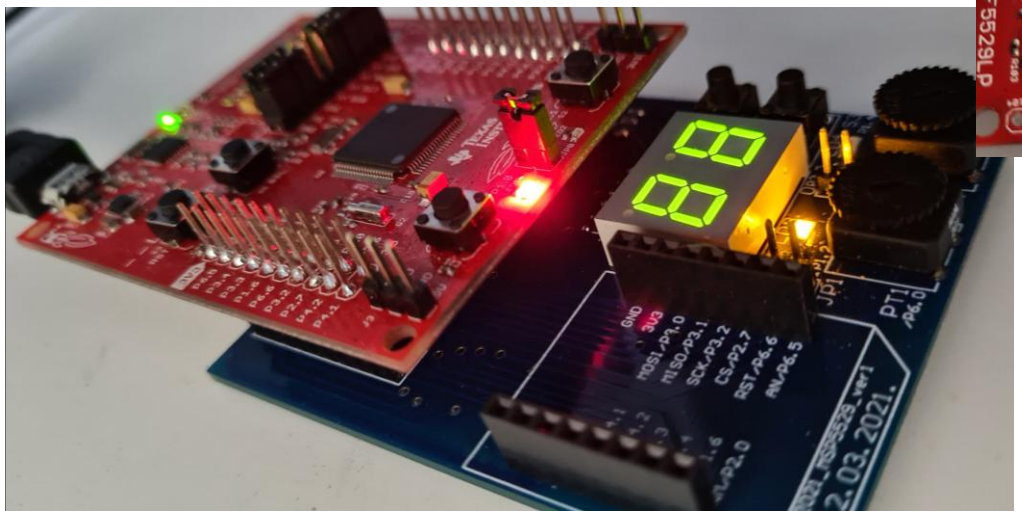
- U okviru dela vežbi moguće je osvojiti maks **50 poena**
  - Lab vežbe – 30 poena
    - *I Lab vežba – **5 poena***
    - *II Lab vežba – **10 poena***
    - *III Lab vežba – **15 poena***
  - Projekat – 20 poena
    - Dizajn softvera i funkcionalnost realizacije – **10 poena**
    - Teorijski apekti realizacije – **10 poena**
- Projekat
  - Tekst postavke projektnog zadatka se objavljuje nakon završenih vežbi iz predmeta.
    - *Jasno definisani zahtevi koje embedded aplikacija bazirana na FreeRTOSu mora da zadovolji*
  - *Kompletna realizacija projektnog zadatka treba da sadrži asastoji iz dva glavna dela*
    2. *Realizacija funkcionalnosti na hardverskoj platformi*
       - *Implementacija mehanizama opisanih u fazi 1 koristeći FreeRTOS i MSP hardversku platformu*

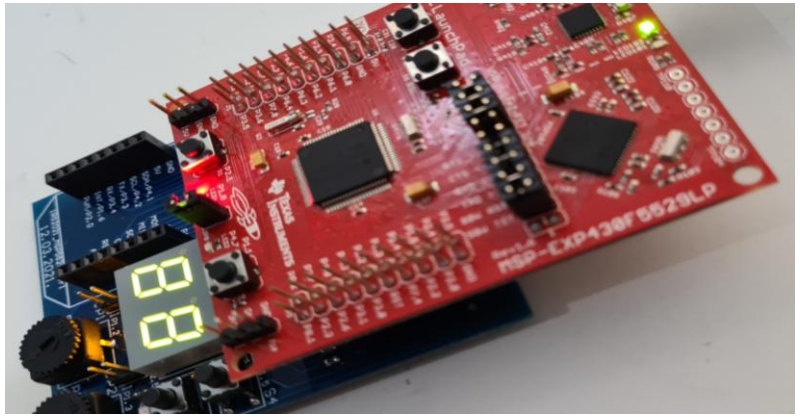
Opšte napomene

Resursi - Hardver

- Koristi se razvojna ploča bazirana na **MSP430F5529** mikrokontroleru
- Kako bi olakšali integraciju razvojne ploče (crvena ploča) u okviru mnogobrojnih embedded aplikacija, za potrebe predmeta SRV i IRS na katedri za elektroniku, kreiran je *schield* (plava pločica) koji na sebi sadrži:
  - *Dva Tastera*
  - *Dva LE displeja*
  - *Dva potenciometra*
  - *Dve LE diode*
  - *1 uBus slot*

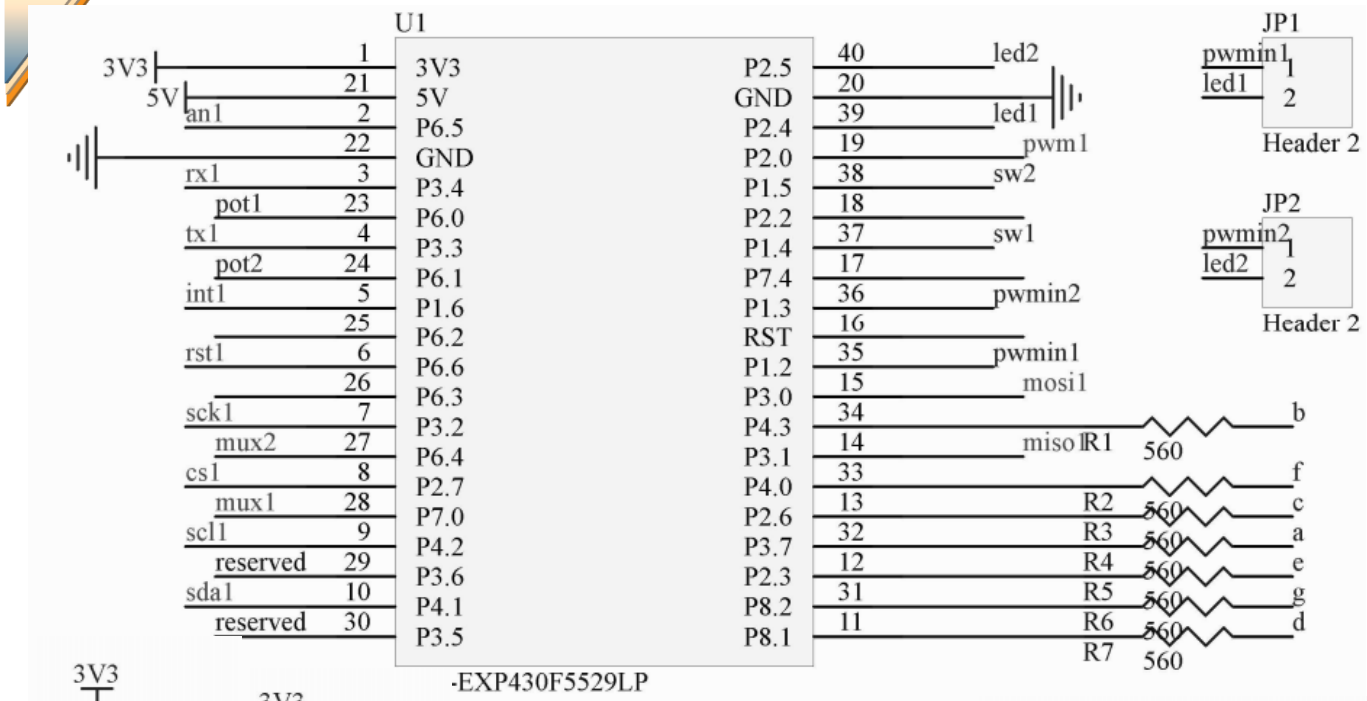


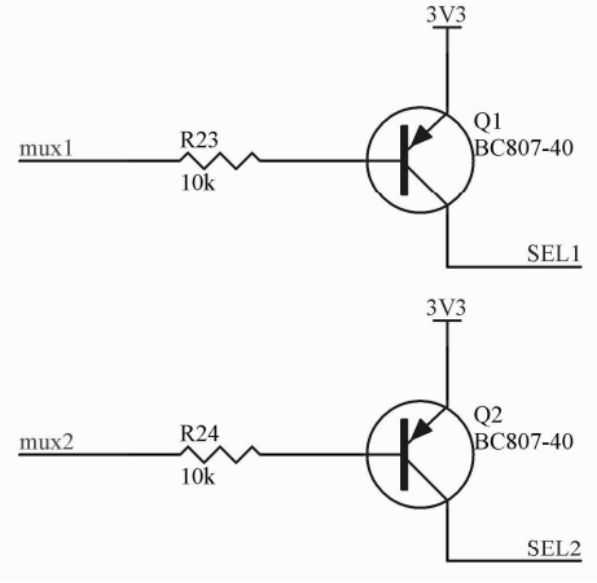


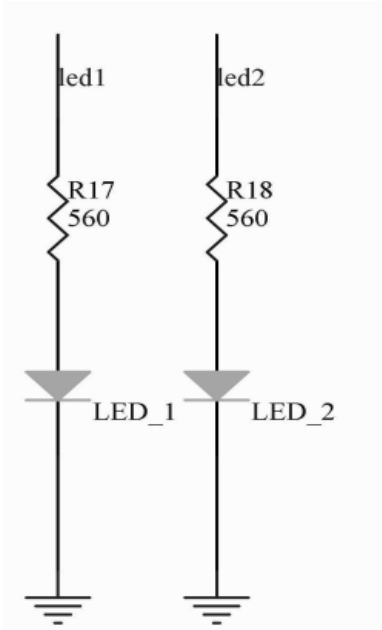


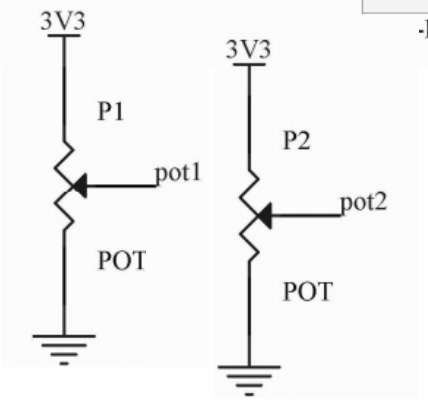
Opšte napomene

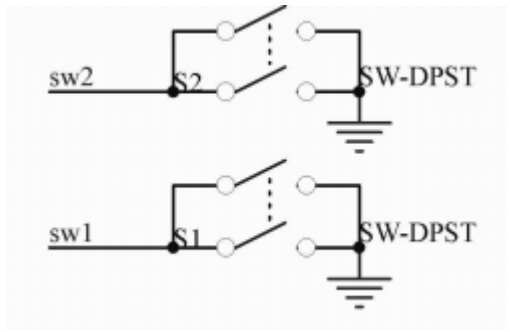
Resursi – Hardver *(Schematic)*

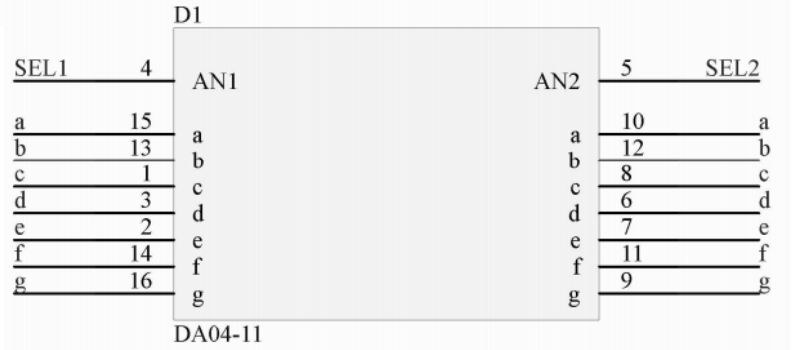


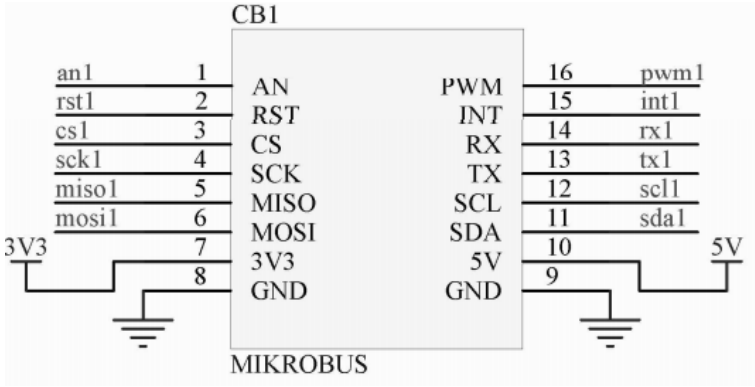










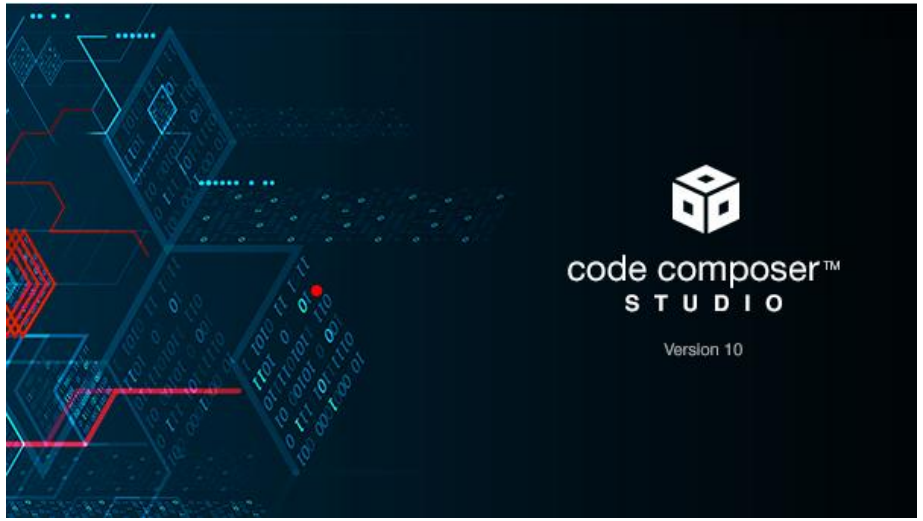


Opšte napomene

Resursi - Softver

- U cilju ilustracije bazičnih kocepata koji se koriste pri projektovanju RT embedded softvera, u okviru vežbi iz ovog predmeta koristićemo RTOS pod nazivom ***FreeRTOS***

- Softver baziran na *FreeRTOS*-u **razvijaćemo** koristeći razvojno okruženje ***Code Composer Studio***
  - *Razvojno okruženje razvijeno od strane Texas Instruments-a kako bi se omogućilo programiranje njihovih mikrokontrolera*
  - *Razvojno okruženje bazirano na **eclipse** platformi*
  - *Moguće preuzeti sa https://www.ti.com/tool/CCSTUDIO*

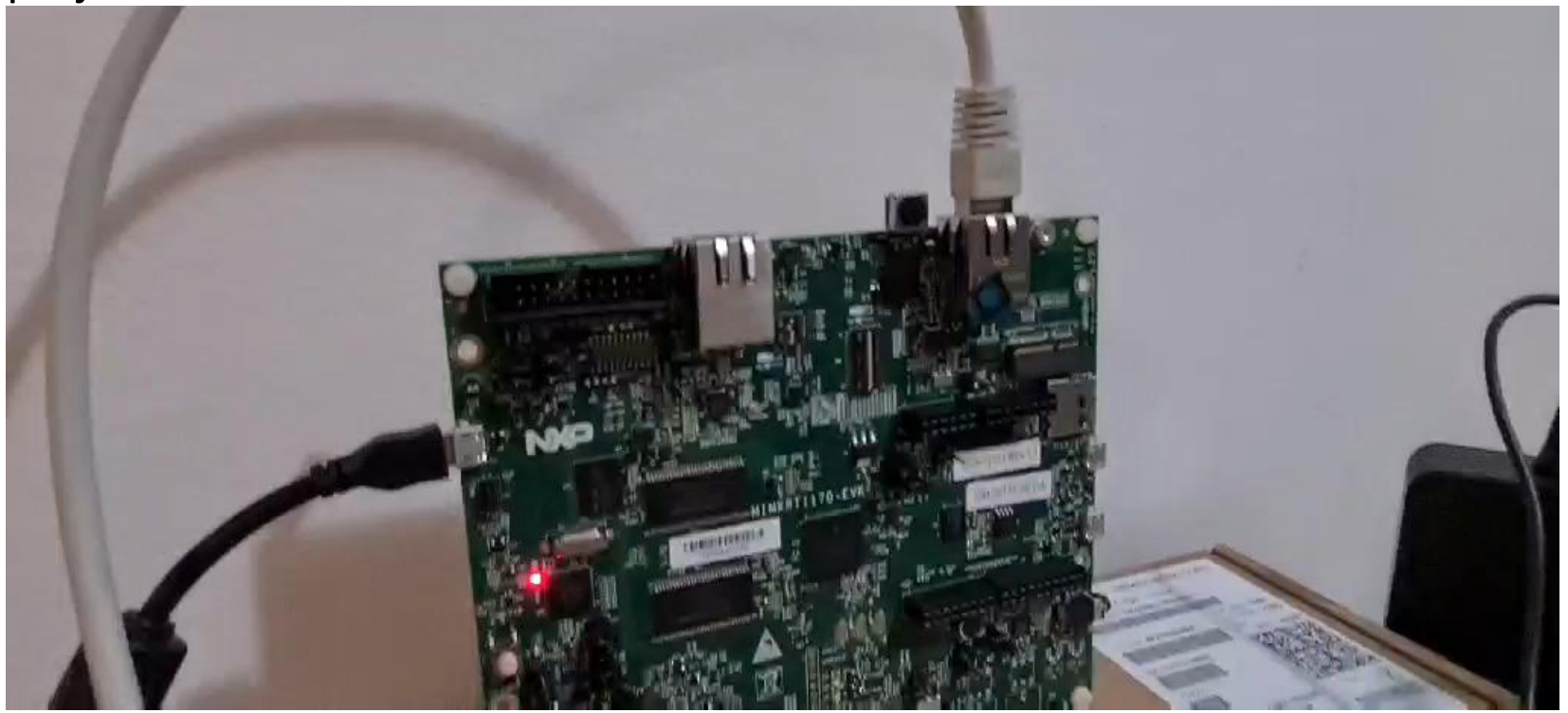


- Kako bi olakšali razvoj softvera i veću pažnju posvetili samom projektovanju softvera, u okviru materijala za prvi čas nalazi se ***Template*** projekat koji će biti polazna osnova pri pisanju softvera u ***Code Composer Studio***

FreeRTOS projekti

Resursi - Softver

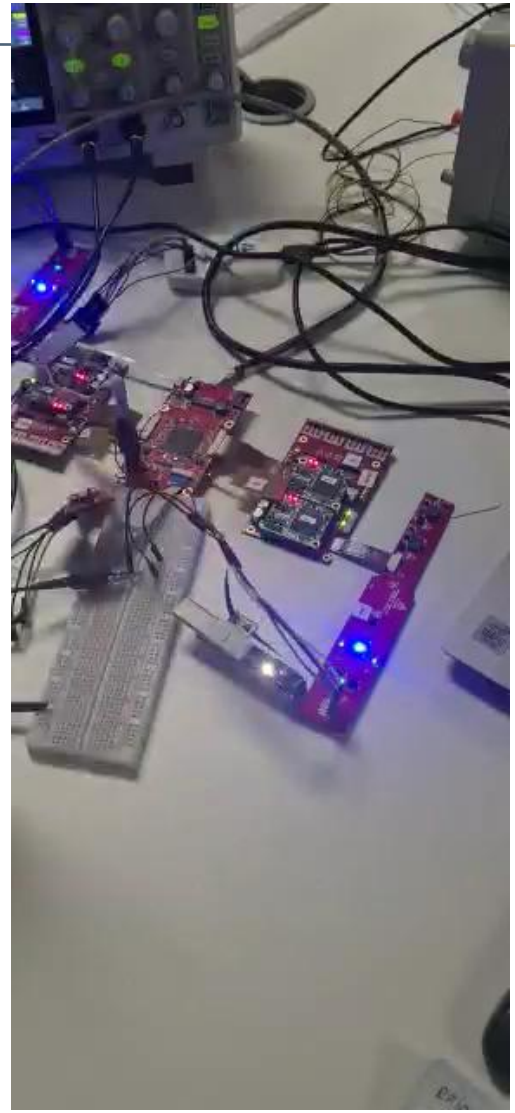
- Različite oblasti industrije
  - *Automotive, Medical, IoT*
- Neki od projekata



FreeRTOS projekti

Resursi - Softver

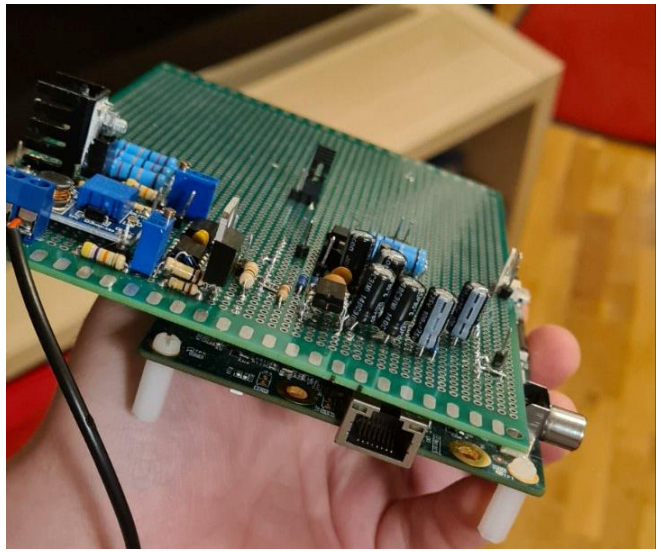
- Različite oblasti industrije
  - *Automotive, Medical, IoT*
- Neki od projekata

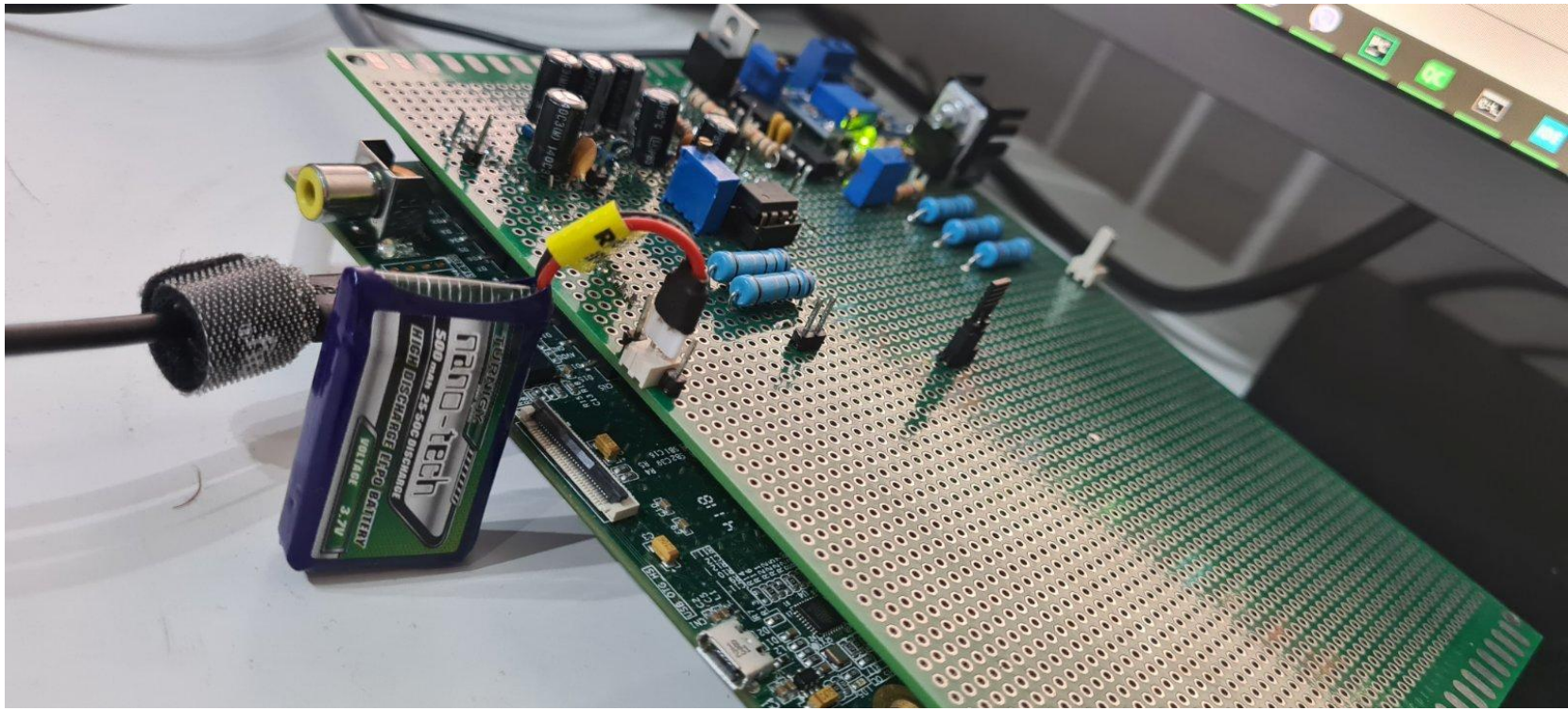


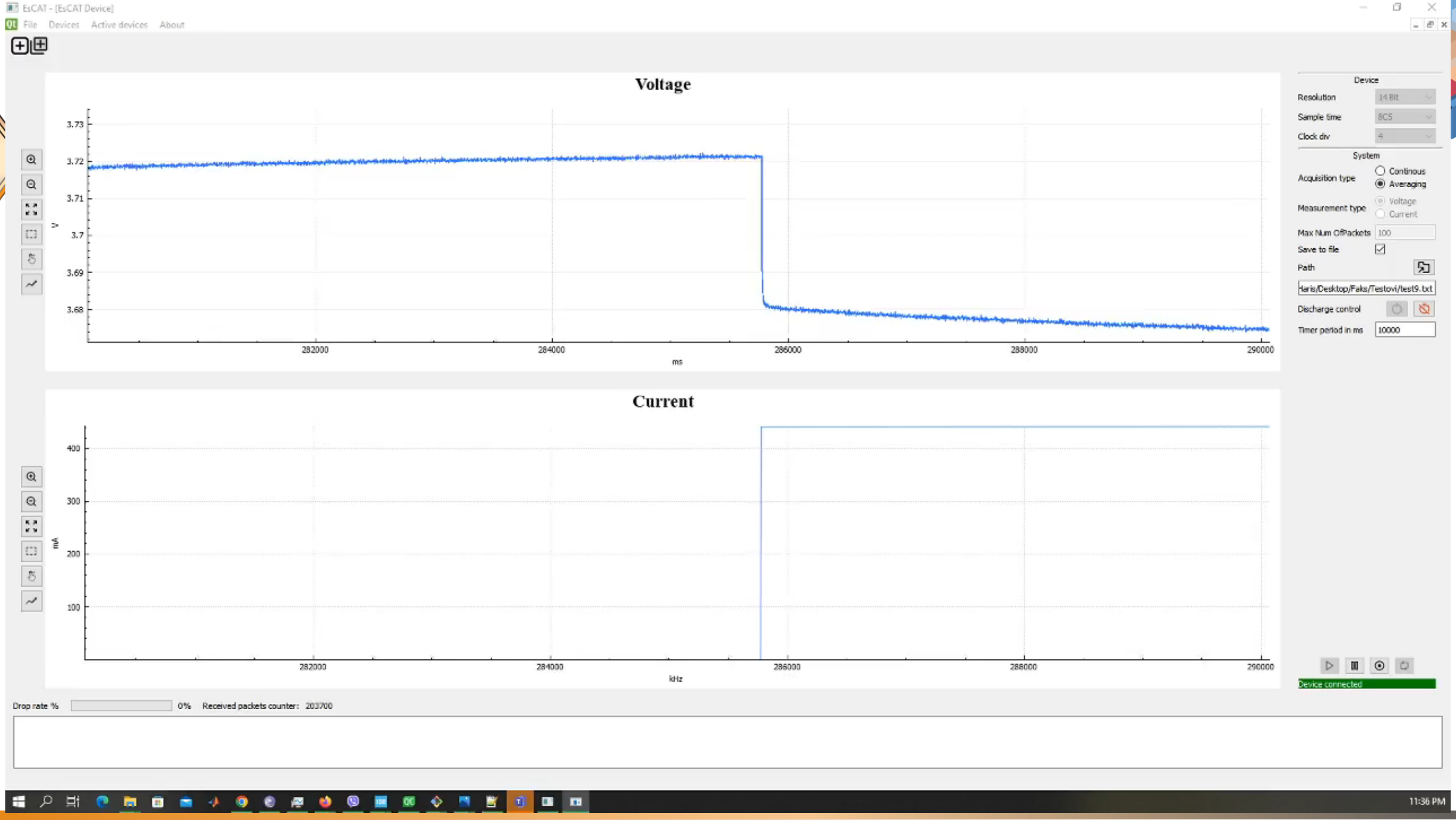
Gde mogu da upotrebim stečeno znanje

Resursi - Softver

- Različite oblasti industrije
  - *Automotive, Medical, IoT*
- Neki od projekata



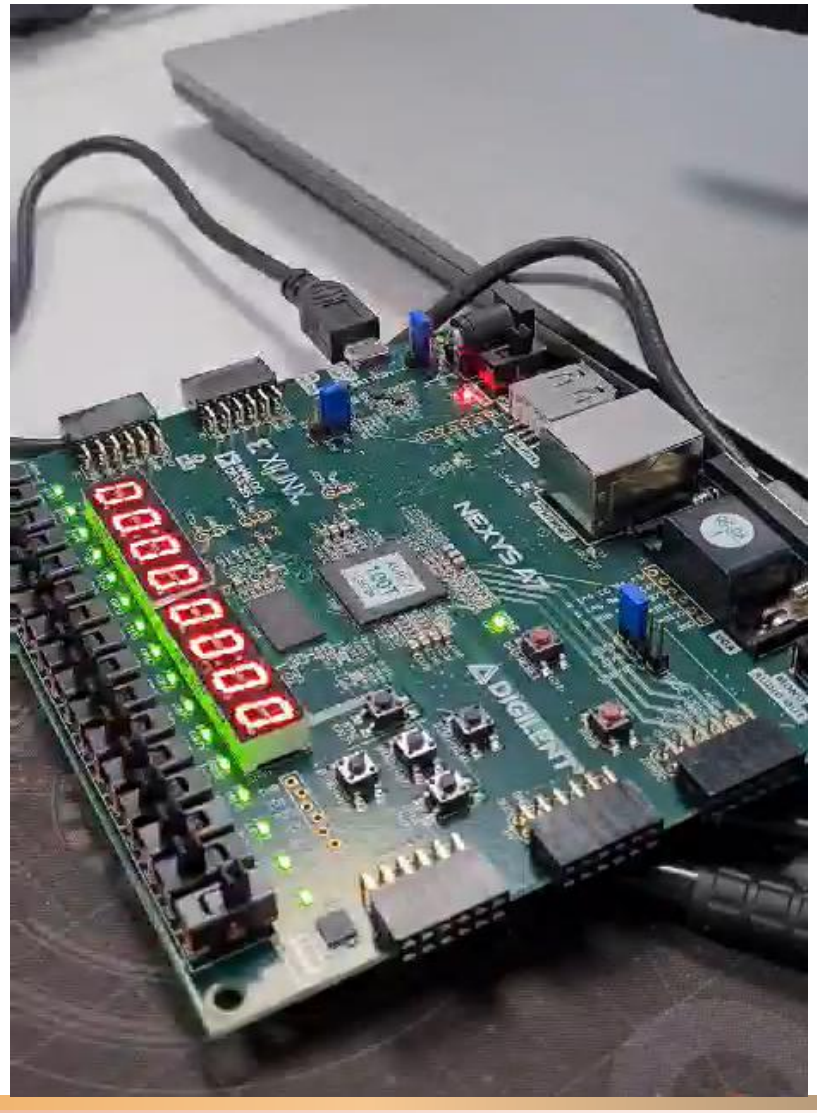



EsCAT - [EsCAT Device]
File
Devices
Active devices
About
Voltage
3.73
3.72
3.71
3.7
3.69
3.68
V
282000
284000
286000
288000
290000
ms
Current
400
300
200
100
mA
282000
284000
286000
288000
290000
kHz
Device
Resolution
14 Bit
Sample time
8CS
Clock div
4
System
Acquisition type
Continous
Averaging
Measurement type
Voltage
Current
Max Num OfPackets
100
Save to file
Path
Haris/Desktop/Faks/Testovi/test9.txt
Discharge control
Timer period in ms
10000
Device connected
Drop rate %
0%
Received packets counter: 203700
11:36 PM

Gde mogu da upotrebim stečeno znanje

Resursi - Softver

- Različite oblasti industrije
  - *Automotive, Medical, IoT*
- Neki od projekata



SUPER-LOOP

OSNOVNI KONCEPTI

SUPER-LOOP VS REAL-TIME

POLIRANJE I PREKIDI

MOTIV ZA UVOĐENJE RTOS-A

Super-loop

Najednostavniji metod realizacije programskog toka

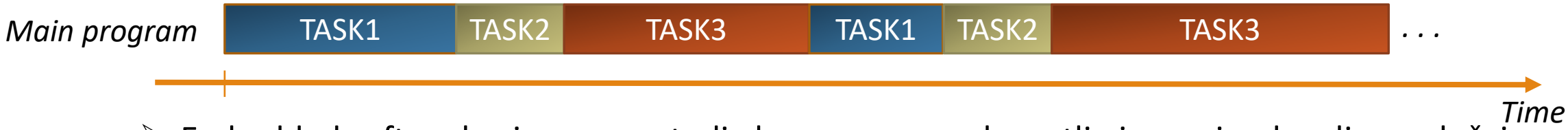
- Jedan od glavnih zahteva koji softver pisan za embedded uređaje treba da realizuje jeste kontinuirano izvršavanje na nekoj embedded platformi
  - *Program se startuje odmah nakon dovođenja napajanja i izvršava se sve dok postoji napajanje u sistemu*
  - *Jedna od ključnih razlika u odnosu na programe pisane, na primer za PC računare gde se posrdestvom korisnika može prekinuti izvršavanje programa*
- Najednostavnija, i hronološki najstarija, **metoda** koja omogućava realizaciju ovog zahteva jeste implemenetacija funkcionalnosti softvera u jednoj beskonačnoj programskoj petlji (***super-loop***)
  - Beskonačna programska petlja koja se poziva odmah nakon ulaska u program (funkcije `main`), naziva se glavna programska petlja
  - *Pseudo kod koji ilustruje koncept glavne programske petlje:*

```
void main(){
    while(1){
        Task1();//Logic group of functionalities (algorithms) related to the
                //Task 1
        Task2();//Logic group of functionalities (algorithms) related to the
                //Task 2
        Task3();//Logic group of functionalities (algorithms) related to the
                //Task 3
        //Possible to have more other functionalities
    }
}
```

Super-loop

Najjednostavniji metod realizacije programskog toka

- Funkcionalnosti definisane u okviru glavne programske petlje izvršavaju se sekvencijalno
  - *Naredna funkcionalnost u sekvenci **ne započinje** sve dok se **prethodna ne završi***

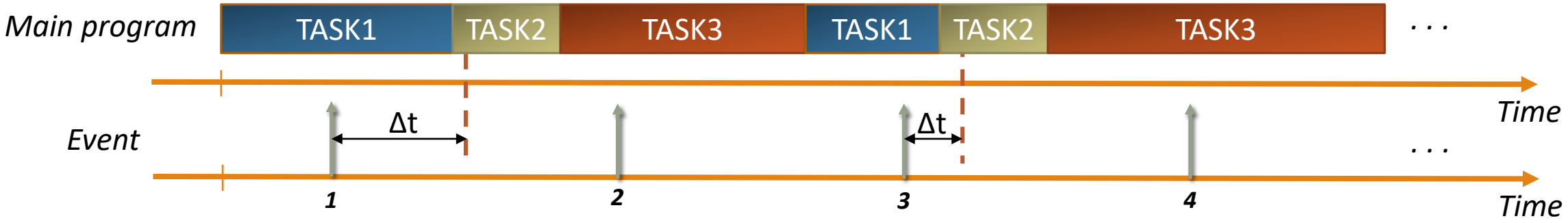


- Embedded softver baziran na metodi glavne programske petlje je sasvim dovoljan u slučaju jednostavnijih embedded aplikacija.
  - *U slučaju da imamo jednu ili dve jednostavne fukcionalnosti (na primer sistem koji samo čita vrednost sa AD konvertora i šalje informaciju o tome da li je vrednost prešla neki prag, jednostavno upravljanje nekim motorom, ...)*
- U slučaju realizacije embedded softvera, u okviru koje koristimo glavnu programsku petlju, možemo reći da imamo **jednostavan** i **razumljiv** programski tok.
- Kako se softver baziran na glavnoj programskoj petlji ponaša u slučaju odziva na spoljašnje događaje?
- Kako pristupiti projektovanju embedded softvera u slučaju kompleksnijih zahteva (obrada više događaja, komunikacija sa više periferija, kompleksni komunikacioni stekovi, ...)?

Super-loop

Analiza odziva na događaje u sistemu

- Posmatrajmo jednostavan primer gde do generisanja događaja dolazi periodično
  - *U okviru programske celine **Task2** potrebno je obraditi periodični događaj*



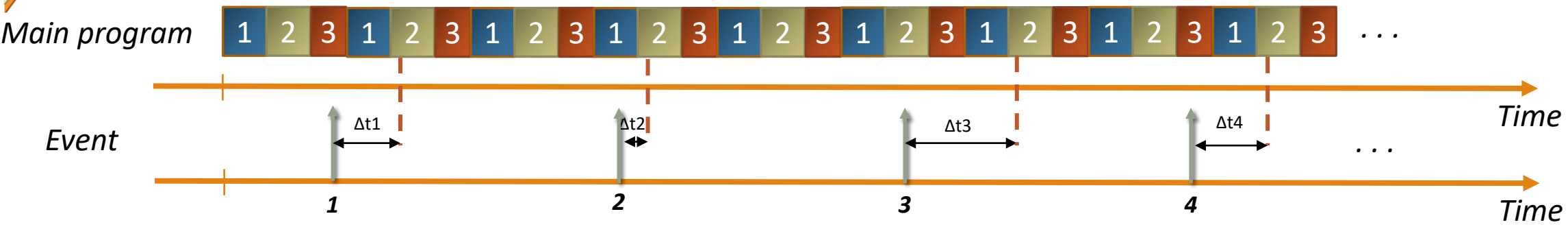
- Na osnovu vremenskog dijagrama možemo uočiti sledeće:
  - Nije moguće garantovati (najgore) vreme potrebno za obradu nekog generisanog događaja Δt
    - Vreme potrebno za obradu događaja se menja
    - Sistem **nema determinističko ponašanje** što je veoma bitno u slučaju *Real-Time* sistema
  - U slučaju **promenljivog** vremena trajanja određenih **logičkih celina** može doći i do propuštanja obrade nekog događaja
    - Šta bi se desilo u slučaju da generisani događaji predstavljaju prijem karaktera putem UART-a?

Kako rešiti uočene probleme?

Super-loop

Poliranje kao mehanizam obrade događaja

- Problem propuštanja događaja možemo rešiti češćom proverom

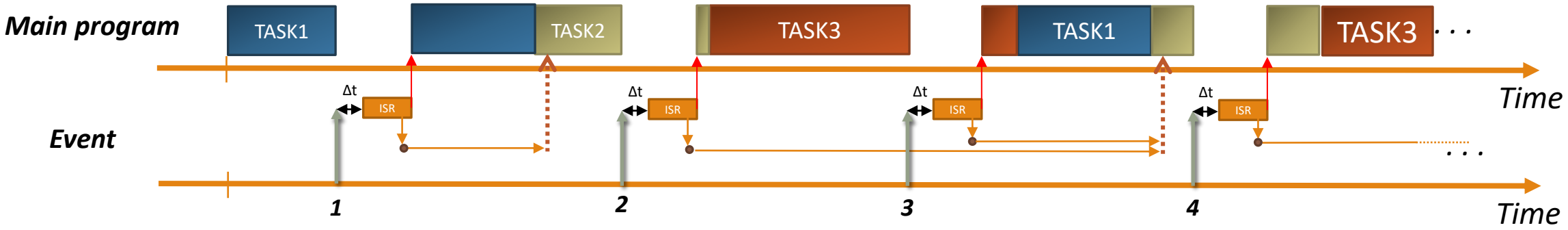


- Postiže se optimizacijom ostalih logičkih celina ili odabirom platforme sa većim procesorskim mogućnostima
- Ovakav mehanizam u kome se **konstantno vrši provera,** kako bi se utvrdilo da li je došlo do generisanja događaja u sistemu, naziva se mehanizam **Poliranja**
  - Na primer, u slučaju jednostavne aplikacije (detekcija pritiska tastera, procesiranje UART poruka, ...) ovakav pristup može dati zadovoljavajuće rezultate
  - Da li ovakav sistem može biti Real-Time?
- Može se desiti da je perioda provere događaja MNOGO MANJA od periode generisanja događaja. Kako to utiče na potrošnju energije?
- Ostaje **problem promenljivog vremena potrebnog za detekciju događaja**
- Šta se dešava u slučaju potrebe za obradom više događaja (Event 1, 2, 3, ... N)?

Super-loop

Prekid kao mehanizam obrade događaja

- Izjednačavanje vremena potrebnog za detekciju događaja
  - Većina uP koja se koristi u okviru embedded sistema ima podršku za **mehanizam Prekida**
  - Ovaj mehanizam podrazumeva poziv odgovarajuće funkcije, **prekidne rutine** (***I**nterrupt **S**ervice **R**outine*), nakon generisanja događaja kome je ta funkcija dodeljena

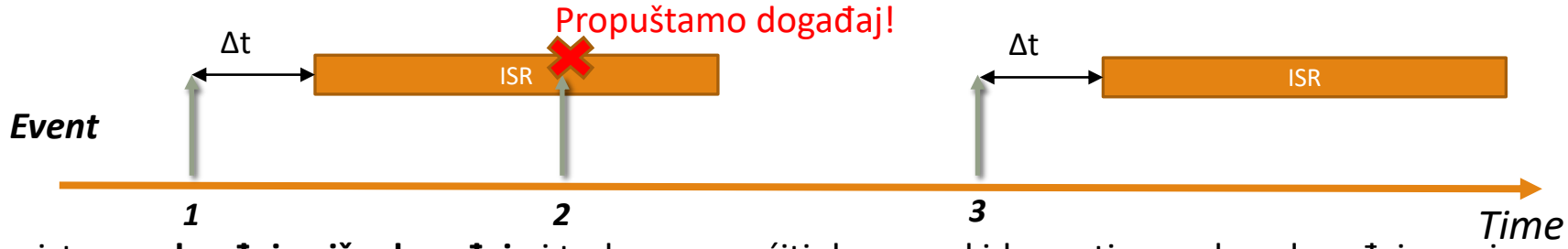


  - Vreme odziva na generisani događaj je sada determinističko
    - Odgovara **vremenu potrebnom za poziv prekidne rutine**
    - **Zavisi od platforme** za koju pišemo softver i vrednost ovog vremenskog intervala može se naći u okviru *Datasheet*-a proizvođača platforme
  - Uvedena najjednostavnija forma „paralelizacije“
    - Prekidna rutina se izvršava **nezavisno** od programske logičke celine (*Task1,2,3, ... N*) koju je ta prekidna rutina prekinula

Super-loop

Prekid kao mehanizam obrade događaja – potencijalni problemi

- Prekidnu rutinu je potrebno dizajnirati tako da bude što kraća po pitanju vremena izvršavanja
  - U prekidnoj rutini treba provesti što manje vremena kako bi sprečili gubitak događaja u sistemu ali isto tako i garantovali vreme odziva što je bitno kada dizajniramo *Real-Time embedded software*
    - Ako se događaj u sistemu javlja često, ukoliko prekidna rutina nije napisana tako da se brzo izvršava, može doći do pozivanja prekidne rutine za trenutni događaj, a da prekidna rutina koja odgovara prethodnom događaju nije završena



    - Često se u sistemu **obrađuje više događaja** i treba omogućiti da se prekidna rutina svakog događaja poziva pre isteka nekog kritičnog vremena definisanog zahtevima *Real-Time* sistema
  - Kako bi smanjili vreme provedeno u prekidnoj rutini možemo koristiti hardversku podršku za paralelizaciju u vidu posebnih periferija koje većina današnjih uC sadrži (DMA, razne periferije koje ubrzavaju obradu podataka, itd).

Super-loop

Prekid kao mehanizam obrade događaja – potencijalni problemi

- Dakle, na osnovu prethodno iznetih karakteristika i navedenih primera jasno je da je moguće projektovati embedded software koji **zadovoljava *Real-Time* zahteve** isključivo **korišćenjem prekida** uz potencijalnu **eksploataciju dodatnih periferija kao što je DMA** kako bi se smanjilo vreme provedeno u prekidnoj rutini.
  - Moguće je imati i određen nivo paralelizacije korišćenjem prekidnih rutina
- Zbog čega onda uopšte postoji potreba za *Real-Time* Operativnim Sistemom na nekoj embedded platformi?
  - U slučaju jednostavnih embedded aplikacija (kao što su jednostavna očitavanja, jednostavna komunikacija UART ili SPI, ...) vrlo verovatno da **nema potrebe** da projektujemo softver baziran na nekom **RTOS-u.** Softver **baziran na beskonačnoj programskoj petlji će biti sasvim dovoljan**
    - ✓ Jednostavne funkcionalnosti
    - ✓ Efikasni i kratki algoritmi
    - ✓ Nema obrade velike količine podataka
    - ✓ Resursi platforme za koju pišemo Real-Time softver su oskudni (mala brzina CPUa, mala količina dostupne memorije, ...)
  - **Potreba** za postojanjem nekog RTOS koji bi omogućio **paralelizaciju** i **Real-Time** performanse se javlja u slučaju projektovanja **složenijih** embedded aplikacija.

RTOS

TASKOVI

KONCEPT TASKA

KONKURENTNOST IZVRŠAVANJA TASKOVA

ALGORITMI RASPOREĐIVANJA

RTOS

- Prilikom projektovanja softvera za složenije *embedded* aplikacije javlja se potreba za procesiranjem različitih događaja u sistemu
  - Broj događaja koje treba obraditi je relativno veliki
  - Algoritmi koji se koriste za obradu takvih događaja su kompleksni (složena komunikacija, komunikacioni stekovi, rad sa grafičkim elementima, ...)
- Često takav softver treba da zadovoljava i *real-time* zahteve
  - Vreme od trenutka generisanja događaja do obrade događaja ne sme biti veće od jasno definisanog vremena u okviru zahteva projekta
- Razvoj, a kasnije i održavanje softvera, treba što je moguće više pojednostaviti
  - Treba težiti razvoju **modularnog** *embedded* softvera
  - Funkcionalnosti softvera treba **podeliti** na određene **logičke celine**

    - Često se softver za kompleksne emebedded aplikacije razvija u timu sastavljenom od više inženjera

    - Ukoliko je softver dobro organizovan, pojedincima u timu se može dodeliti razvoj dogovarajuće **nezavisne** logičke celine softvera i na taj način **ubrzati** razvoj a kasnije **testiranje** i **održavanje** softvera.
  - Prilikom deljenja softvera u odgovarajuće logičke celine treba razmišljati o tome da se te logičke celine što lakše mapiraju u odgovarajuće programske celine zvane „Taskovi“.

Koncept Taska

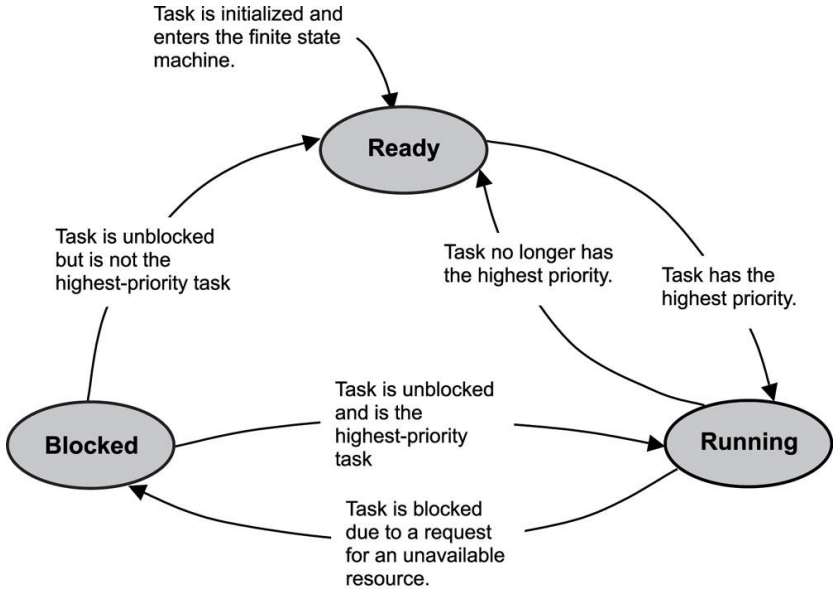
- **Task** predstavlja **nezavisnu logičku celinu** u kodu enkapsuliranu u vidu programske funkcije
  - Sa stanovišta programskog jezika **Task** je specifična funkcija koja:
    - u okviru svoje definicije ima jednu beskonačnu programsku petlju
    - ne poziva se kao druge funcije već se samo jednom kreira od strane ***Schedulera*** operativnog sistema
  - Pseudo-kod koji opisuje **definiciju** jedne tipične **Task funkcije** je:

```
void Task(arguments){
    //Task data initialization
    //go to endless loop
    while(1){
        //Task algorithm
    }
}
```

- Kada se task izvršava na nekoj platformi on **koristi resurse te platfome** (registre, stek, ...) kao da je jedinstveni task u sistemu
- Kako bi se zapamtili resursi dodeljeni tasku ali i sačuvalo stanje određenih resursa (na primer stanje registara CPUa) bitnih za izvršavanje taska, svakom kreiranom tasku se dodeljuje struktura pod nazivom **„Kontekst taska“**
- Svakom novokreiranom tasku **dodeljuje se prioritet**
  - U slučaju korišćenja algoritma raspoređivanja koji koriste prioritet taskova, prioritet taska određuje koji od dva taska će dobiti procesorsko vreme ukoliko su oba taska spremna za izvršavanje.

Koncept Taska

- U većini realizacija RTOS-a, Task može imati neko od sledećih stanja
  - ***Running*** – Task se izvršava na procesoru
  - ***Ready*** – Task je spreman za izvršavanje, ali se ne izvršava (verovatno jer se trenutno izvršava task većeg ili istog prioriteta)
  - ***Blocking*** – Task je blokiran i čeka na neki događaj u sistemu
  - U zavisnosti od realizacija RTOSa mogu postojati i dodatna stanja(na primer ***Suspended*** u slučaju FreeRTOS-a). U većini slučajeva uvek postoje stanja kao što su ***Running***, ***Ready*** and ***Blocking***



Konkurentnost taskova

- Jedan od zadataka operativnog sistema je da omogući konkurentnost u izvršavanju taskova

SUPER-LOOP Realizacija

```
void main(){
    while(1){
        Task1();//Logic group of functionalities (algorithms) related to the
                //Task 1
        Task2();//Logic group of functionalities (algorithms) related to the
                //Task 2
        Task3();//Logic group of functionalities (algorithms) related to the
                //Task 3
        //Possible to have more other functionalities
    }
}
```

Realizacija korišćenjem taskova

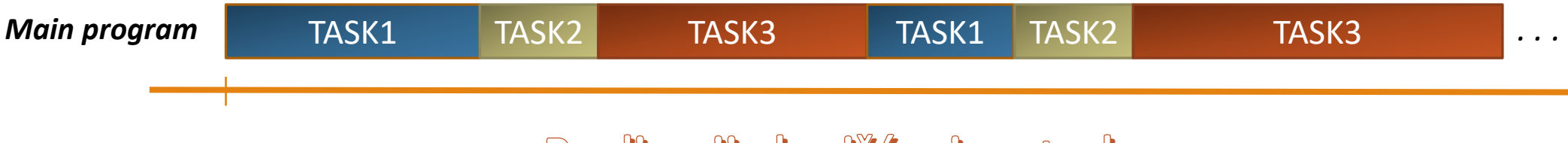
```
void Task1(arguments){
    //Task1 data initialization
    //go to endless loop
    while(1){
            //Task1 algorithm
    }
}
void Task2(arguments){
    //Task2 data initialization
    //go to endless loop
    while(1){
            //Task2 algorithm
    }
}
void Task3(arguments){
    //Task3 data initialization
    //go to endless loop
    while(1){
            //Task3 algorithm
    }
}
void main(){
    CreateTask(Task1);//Create task with Task1 as task function
    CreateTask(Task2);//Create task with Task2 as task function
    CreateTask(Task3);//Create task with Task3 as task function
    StartSchedulrer();//Start RTOS Scheduler - never return from this function
                      //except if there was some errors
}
```

- Potrebno je uočiti da za razliku od realizacije u vidu beskonačne programske petlje, gde imamo jednu beskonačnu programsku petlju, u slučaju softvera baziranog na RTOS svaki od taskova ima svoju beskonačnu programsku petlju
  - „Svaki task je program za sebe“?

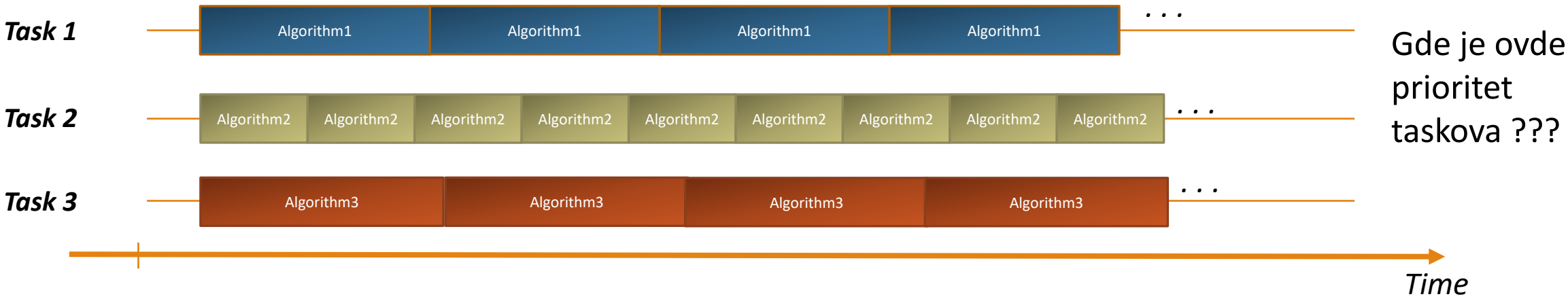
Konkurentnost taskova

- Jedan od zadataka operativnog sistema je da omogući konkurentnost u izvršavanju taskova

SUPER-LOOP Realizacija



Realizacija korišćenjem taskova



Gde je ovde prioritet taskova ???

Kako je moguće da se na jednom uP izvršava više taskova istovremeno???

Konkurentnost taskova

- Kako je uopšte moguće postići da se na **jednom** uP istovremeno izvršava više taskova?
  - Objektivno, na jednom uP zaista nije moguće paralelizovati izvršavanje taskova
  - Međutim, ako uzmemo u obzir činjenicu da većina modernih procesora može da izvrši milione instrukcija u jednoj sekundi, onda možemo dosta dobro realizovati konkurentnost u izvršavanju taskova (*fokusirati se na kasniji primer o Round-Robin algoritmu raspoređivanja*)
- U okviru kernela operativnog sistema, komponenta pod nazivom ***Scheduler*** (jezgro kernela) ima zadatak da vrši raspoređivanje taskova
  - Raspoređivanje taskova zapravo podrazumeva odlučivanje o tome kom će tasku **u posmatranom trenutku** biti dodeljeno procesorsko vreme
- Funkcionalnost ***Schedulera*** bazirana je na algoritmima za raspoređivanje taskova
- Postoji mnoštvo različitih algoritama za raspoređivanje taskova koji prilikom raspoređivanja razmatraju različite parametre taska ili sistema. Dva najčešća algoritma raspoređivanja su:
  - Round-Robin algoritam

    - Svaki task dobija isto vreme za izvršavanje
  - Priority Round-Robin algoritam

    - Task najvećeg prioriteta dobija procesorsko vreme

Zamena konteksta

- Kada dolazi do pozivanja *Scheduler-a*, tj. algoritama za raspoređivanje taskova, zavisi od konkretne implementacije RTOS. Međutim, tri najčešća mesta u toku izvršavanja programa gde se poziva Scheduler su:
  1. Kada trenutni **task** koji se izvršava **odlazi u blokirano stanje,** a nije došlo do **isteka sistemskog tajmera**
  2. Kada **usled generisanja prekida** task većeg prioriteta prelazi iz stanja „Blokiran“ u stanje „Spreman za izvršavanje“
  3. Usled isteka sistemskog tajmera
- Prilikom pozivanja ***Schedulera***, ukoliko dolazi do dodele procesorskog vremena nekom tasku koji nije trenutni task koji se izvršava na procesoru, dolazi do **procesa zamene konteksta**

  - Dok neki posmatrani task **ne koristi procesorsko** vreme (na primer nalazi se u stanju „Blokiran“) **neki drugi task u sistemu može koristi** procesorske registre, stek, itd ...

  - Kada task dobije procesorsko vreme kontekst procesora na kome se task izvršava (sadržaj steka stek, vrednosti registara CPUa) **mora biti isti kao kontekst procesora neposredno pre odlaska taska u stanje „Blokiran“ ili stanje „Spreman za izvršavanje“**

  - Kako omogućiti da kontekst procesora(vrednosti registara, sadržaj steka) bude isti kao kontekst procesora neposredno pre odlaska Taska u stanje „Blokiran“ ili stanje „Spreman za izvršavanje“?

Zamena konteksta

- **Proces zamene konteksta se** sastoji se od dve faze
  1. Čuvanje konteksta procesora trenutnog taska

     - Čuva se kontekst procesora u okviru strukture „Kontekst taska“ trenutnog taska koji se izvršava na sistemu i taj task odlazi u stanje **„Blokiran“** ili **„Spreman za izvršavanje“**
  2. Restauracija konteksta procesora

     - Kontekst procesora se inicijalizuje na vrednosti koje se nalaze u okviru strukture „Kontekst taska“ taska koji je dobio procesorsko vreme
- **Proces zamene konteksta** je jako bitan kada pričamo o operativnim sistemima

  - Omogućava da se svaki task ponaša kao nezavisna celina
- Prilikom projektovanja softvera koji je baziran na RTOS i koji treba da zadovolji Real-Time zahteve treba uzeti u obzir da proces zamene konteksta **traje određeno vreme**

Algoritmi raspoređivanja – Round Robin – Zadatak 1

- Round-Robin
  - Svakom tasku se dodeljuje jednako procesorsko vreme za izvršavanje
  - Task koji je iskoristio svoje procesorsko vreme, a nije se do kraja izvršio, ide na kraj liste čekanja
- Zadatak 1

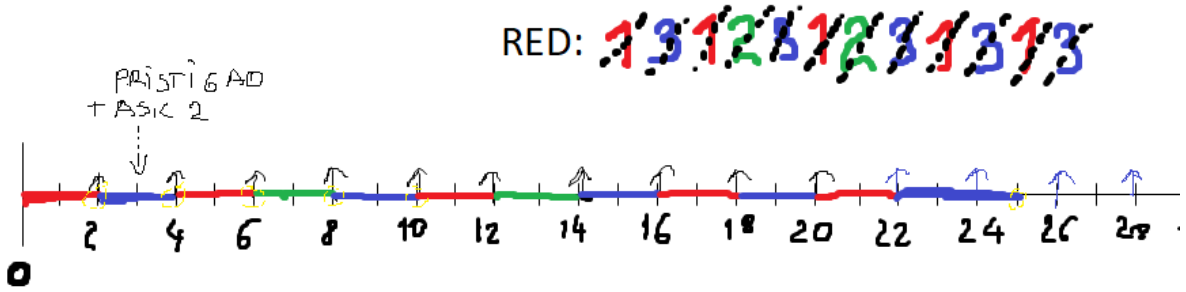
  *Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **2**. Ako Scheduler ovog RTOS-a koristi **Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

Naziv taska	Vreme pristizanja	Vreme izvršavanja
T_1	0	10
T_2	3	4
T_3	0	11

- Zadatak 1 – diskusija rešenja
  - Jednaka raspodela procesorskog vremena za sve taskove.
    - Postignut efekat konkurentnosti u izvršavanju taskova
    - Šta se dešava u slučaju periodičnih taskova?
  - Kakve su Real-Time karakteristike sistema koji koristi isključivo ovakav algoritam raspoređivanja?

Algoritmi raspoređivanja – Round Robin – Zadatak 1

- Round-Robin
  - Svakom tasku se dodeljuje jednako procesorsko vreme za izvršavanje
  - Task koji je iskoristio svoje procesorsko vreme, a nije se do kraja izvršio, ide na kraj liste čekanja
- Zadatak 1 - Rešenje



NAZIV	Preostalo vreme	Vreme Pristizanja	Početak	Kraj
1	10 8 6 4 2 0	0	0	22
2	4 2 0	3	6	14
3	17 9 7 5 3 0	0	2	25

Algoritmi raspoređivanja – Priority R-R – Zadatak 2

- Priority Round-Robin algoritam raspoređivanja
  - Task najvećeg prioriteta dobija procesorsko vreme
  - Ukoliko na sistemu postoje 2 ili više taskova istog prioriteta koji su spremni za izvršavanje, raspoređuju se primenom Round-Robin algoritma
- Zadatak 2

  *Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **2**. Ako Scheduler ovog RTOS-a koristi **Priority Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

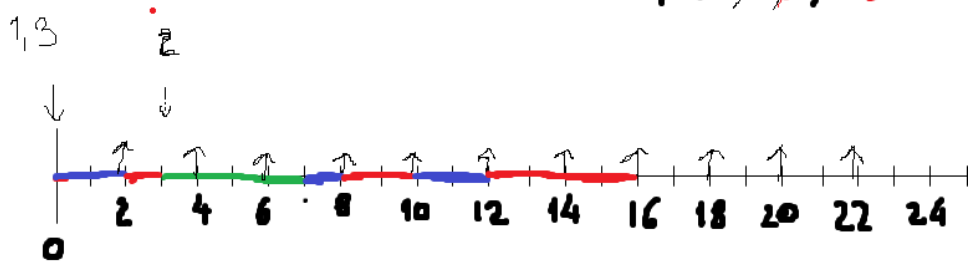
Naziv taska	Vreme pristizanja	Vreme izvršavanja	Prioritet
1	0	7	1(Low)
2	3	4	2 (High)
3	0	5	1(Medium)

- Zadatak 2 – diskusija rešenja
  - Task najvećeg prioriteta dobija procesorsko vreme
    - Šta se dešava u slučaju periodičnih taskova velike učestanosti?
  - Kakve su Real-Time karakteristike sistema koji koristi ovakav algoritam raspoređivanja?

Algoritmi raspoređivanja – Priority R-R – Zadatak 2

- Priority Round-Robin algoritam raspoređivanja
  - Task najvećeg prioriteta dobija procesorsko vreme
  - Ukoliko na sistemu postoje 2 ili više taskova istog prioriteta koji su spremni za izvršavanje, raspoređuju se primenom Round-Robin algoritma
- Zadatak 2 - Rešenje

RED: 3 1 3 1 3 1



NAZIV	Preostalo vreme	Vreme Pristizanja	Početak	Kraj	Prioritet
1	7 6 4	0	2	16	1
2	4 0	3	3	7	2
3	5 3 1 0	0	0	12	1

Algoritmi raspoređivanja – Zadaci za samostalni rad

- Zadatak 2.1

*Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **2**. Ako Scheduler ovog RTOS-a koristi **Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

Naziv taska	Vreme pristizanja	Vreme izvršavanja	Periodičan task
1	0	10	Da
2	4	4	Da
3	2	11	Da

- Zadatak 2.2.

*Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **3**. Ako Scheduler ovog RTOS-a koristi **Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

Naziv taska	Vreme pristizanja	Vreme izvršavanja	Periodičan task
1	0	10	Da
2	4	4	Da
3	2	11	Da
4	1	7	Ne

Algoritmi raspoređivanja – Zadaci za samostalni rad

- Zadatak 2.3

*Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **2**. Ako Scheduler ovog RTOS-a koristi **Priority Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

Naziv taska	Vreme pristizanja	Vreme izvršavanja	Periodičan task	Prioritet
1	0	10	Da	1(Low)
2	4	4	Da	2(High)
3	2	11	Da	1(Low)

- Zadatak 2.4.

*Za embedded sistem poznato je da poseduje RTOS čija **rezolucija sistemskog tajmera** iznosi **1**. Ako Scheduler ovog RTOS-a koristi **Priority Round-Robin** algoritam za raspoređivanje taskova, vremenskim dijagramom ilustrovati izvršavanje taskova na ovom sistemu. Broj taskova, kao i informacije za svaki task pojedinačno, definisani su tabelom.*

Naziv taska	Vreme pristizanja	Vreme izvršavanja	Periodičan task	Prioritet
1	0	10	Da	1 (Low)
2	4	4	Da	2 (Medium)
3	2	11	Da	1 (Low)
4	1	7	Ne	3 (High)

FreeRTOS

Uvod

- OSOBINE
- KONFIGURACIJA
- SISTEMSKI TAJMER
- TOK IZVRŠAVANJA *MAIN* FUNKCIJE

FreeRTOS

Glavne osobine

- *Open Source* RTOS kernel pogodan za implementaciju ***Real-Time*** embedded aplikacija
  - Zvanični web-site http://www.freertos.org
  - Sva dokumentacija http://www.freertos.org/Documentation/RTOS_book.html

    - Mastering the FreeRTOS Real Time Kernel – a Hands On Tutorial Guide

    Uputstvo u okviru koga su objašnjeni osnovni gradivni elementi ***FreeRTOSa***

    - FreeRTOS Reference Manual

    Uputstvo u okviru koga su objašnjene ***FreeRTOS* API** funkcije
- Glavne osobine
  - Nudi više vrsta raspoređivanja
  - Podržava tickless mod (u cilju uštede energije)
  - Dostupni različiti mehanizmi sinhronizacije i komunikacije
  - „Ne postoji ograničenje broja kreiranih taskova“
  - Ne postoji ograničenje broja prioriteta taskova
  - Širok spektar dostupnih biblioteka

    - Neke su komercijalne a neke su besplatne

    - CLI, IO, SSL, IoT, FAT FileSystem, TCP/IP, UDP ...

FreeRTOS

Konfigurisanje

- **Konfigurisanje** FreeRTOS-a se vrši u okviru ***FreeRTOSConfig.h*** header fajla
  - U okviru ovog fajla moguće je izvršiti podešavanje konfigurabilnih parametara FreeRTOS-a
  - Fajl uglavnom čine makroi. Promenom vrednosti maroa u header fajlu moguće je uključiti/isključiti određene funkcionalnosti FreeRTOS-a

```
#define configUSE_PREEMPTION            1
#define configUSE_IDLE_HOOK             1
#define configUSE_TICK_HOOK             1
#define configCPU_CLOCK_HZ              ( 10000000UL )
#define configLFXT_CLOCK_HZ             ( 32768L )
#define configTICK_RATE_HZ              ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES            ( 8 )
#define configTOTAL_HEAP_SIZE           ( ( size_t ) ( 5 * 1024 ) )
#define configMAX_TASK_NAME_LEN         ( 10 )
#define configUSE_TRACE_FACILITY        0
#define configUSE_16_BIT_TICKS          1
#define configIDLE_SHOULD_YIELD         1
#define configUSE_MUTEXES               1
#define configQUEUE_REGISTRY_SIZE       0
#define configGENERATE_RUN_TIME_STATS   0
#define configCHECK_FOR_STACK_OVERFLOW  2
#define configUSE_RECURSIVE_MUTEXES     1
#define configUSE_MALLOC_FAILED_HOOK    1
#define configUSE_APPLICATION_TASK_TAG  0
#define configUSE_COUNTING_SEMAPHORES   1
```

Algoritmi raspoređivanja taskova

- **Kooperativno** raspoređivanje
  - Svi taskovi su istog prioriteta
  - Promena konteksta se vrši samo kada taskovi prepuste kontrolu
- **Raund-Robin** raspoređivanje
  - Svi taskovi su istog prioriteta
  - *Scheduler* brine o tome da svaki task dobije isto vreme za izvršavanje
- **Preemtion** raspoređivanje
  - Raspoređivanje sa kontrolom pristupa
  - *Scheduler* aktivira task najvišeg prioriteta
- Za potrebe ovog predmeta koristićemo **Preemption** raspoređivanje koje će u slučaju da postoji 2 ili više taskova istog prioriteta nad njima primenjivati **Round-Robin** algoritam

```
#define configUSE_PREEMPTION 1
```
→ Setovanjem ovog makroa na 1 koristimo **Preemption** algoritam raspoređivanja. U suprotnom bi se koristilo kooperativno raspoređivanje

```
#define configUSE_TIME_SLICING 1
```
→ Ukoliko setujemo i ovaj makro na 1 onda se za taskove istog prioriteta primenjuje ***Round-Robin*** algoritam raspoređivanja. U suprotnom bi se koristilo kooperativno raspoređivanje

FreeRTOS

Sistemski tajmer

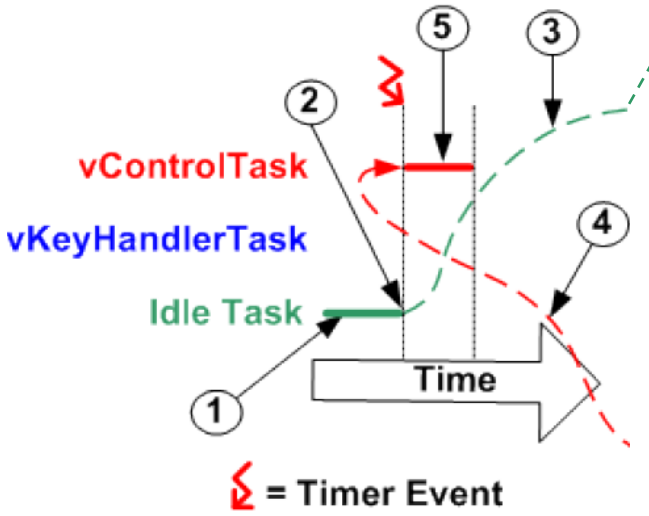
- U većini *embedded* aplikacija, baziranih na *RTOS*-u, neophodno je obezbediti **periodično pozivanje algoritama za raspoređivanje taskova**
- Ova funkcionalnost se u okviru *embedded* platforme realizuje koristeći neki od dostupnih tajmera te *embedded* platforme
  - U okviru **prekidne rutine** odabranog **tajmera** pozivaju se funkcije operativnog sistema u okviru kojih se vrši zamena konteksta ukoliko je to potrebno
  - Tajmer, koji se koristi od strane RTOSa, nazivaćemo **sistemski tajmer**
- U okviru ovog kursa, za potrebe *FreeRTOS*a, koristićemo ***TIMER A0*** mikrokontrolera MSP430F5529

```
/* The MSP430X port uses a callback function to configure its tick interrupt.
This allows the application to choose the tick interrupt source.
configTICK_VECTOR must also be set in FreeRTOSConfig.h to the correct interrupt
vector for the chosen tick interrupt source.  This implementation of
vApplicationSetupTimerInterrupt() generates the tick from timer A0, so in this
case configTICK_VECTOR is set to TIMER0_A0_VECTOR. */
#define configTICK_VECTOR                TIMER0_A0_VECTOR
```

FreeRTOS

Sistemski tajmer

- U okviru prekidne rutine tajmera poziva se scheduler




```
#pragma vector=configTICK_VECTOR
interrupt void vTickISREntry( void )
{
extern void vPortTickISR( void );

    __bic_SR_register_on_exit( SCG1 + SCG0 + OSCOFF + CPUOFF );
    #if configUSE_PREEMPTION == 1
        extern void vPortPreemptiveTickISR( void );
        vPortPreemptiveTickISR();
    #else
        extern void vPortCooperativeTickISR( void );
        vPortCooperativeTickISR();
    #endif
}
```

```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
;-----------------------------------------------------------
```

- Moguće je konfigurisati *FreeRTOS Scheduler* tako da se u okviru prekidne rutine sistemskog tajmera poziva proizvoljna *callback* funkcija definisana u okviru aplikacija.

```
#define configUSE_TICK_HOOK        1
```

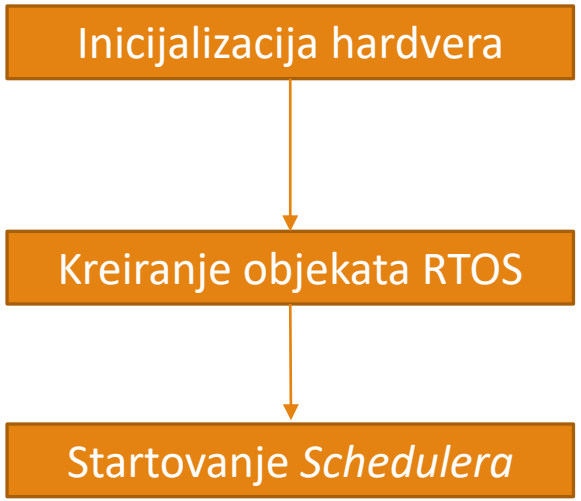
Na svaki istek sistemskog tajmera poziva se korisnički definisana funkcija koja mora imati sledeći potpis

```
void vApplicationTickHook(void);
```

Funkcija se izvršava u okviru prekidne rutine, tako da je poželjno da **bude kratka**, **ne koristi puno steka** i da ne zove API funkcije koje se ne završavaju sa **FromISR**

Tipičan tok izvršavanja *main* funkcije

- Većina aplikacija baziranih na RTOS imaju sledeći tok izvršavanja u okviru *main* funkcije:
  - Pod inicijalizacijom hardvera podrazumeva se podešavanje takta, portova, periferija, ...
    - Nije neophodna inicijalizacija svih periferija, ukoliko je potrebno neke periferije se mogu inicijalizovati i u okviru taska
  - Kreirati objekte operativnog sistema
    - Taskove, semafore, *queue*-ove
  - Startovati scheduler operativnog sistema
    - Ukoliko je sistem dobro inicijalizovan i ukoliko su objekti uspešno kreirani, poziv Schedulera sa programerskog stanovišta predstavlja funkciju iz koje se nikada ne vraćamo.
    - U okviru *FreeRTOS*a se poziv *Schedulera* realizuje pozivom funkcije

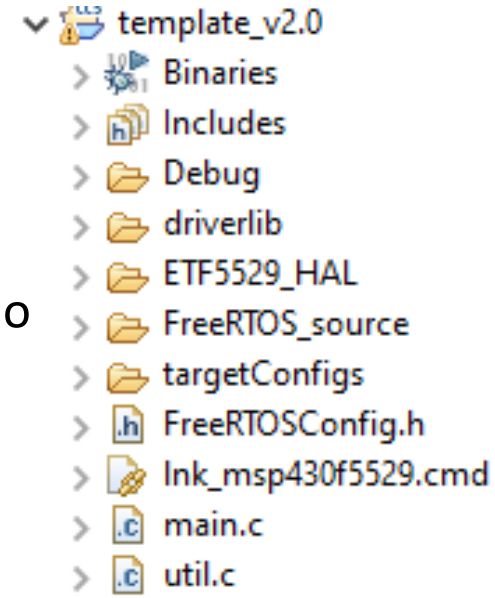



```
/* Start the scheduler. */
vTaskStartScheduler();
```

FreeRTOS

Template projekat za SRV

- Za potrebe vežbi iz predmeta „Sistemi u Realnom Vremenu“ ***FreeRTOS* je portovan na MSP430F5529 platformu.**
- Portovana verzija FreeRTOS-a dostupna je u okviru **SRV_*Template*** projekta koji je moguće preuzeti sa zvaničnog GitHub linka https://github.com/turkmanovic/SRV
  - U okviru zadnjeg dela vežbi iz ovog predmeta biće pojašnjen postupak portovanja FreeRTOS-a na različite platforme
  - Za sada ovaj projekat predstavlja **polaznu tačku** u razvoju embedded aplikacija baziranih na korišćenju *FreeRTOS*-a



- Pre prelaska na narednu sekciju u okviru ove prezentacije, studenti bi trebalo da preuzmu ovaj projekat i pokrenu projekat na dobijenoj razvojnoj platformi.
  - Projekat otvoriti u okviru razvojnog okruženja ***Code Composer Studio***
  - Spustiti kod na dobijenu razvojnu platformu
  - Ukoliko je sve u redu, spremni smo za naredni deo priče ☺
  - Ukoliko postoje problemi, pisati email na haris@etf.rs ili u okviru **četa MS Teams platforme**

Prebačeno u čas 2

FreeRTOS

Taskovi

OSOBINE TASKOVA KARAKTERISTIČNE ZA FREERTOS

FREERTOS TASK API

PRIMERI