KATEDRA ZA ELEKTRONIKU

SISTEMI U REALNOM VREMENU

HARIS TURKMANOVIĆ - VEŽBE 2022/23- ČAS 3

RTOS

KOMUNIKACIJA I SIGNALIZACIJA

KOMUNIKACIJA KORIŠĆENJEM REDOVA SA PORUKAMA

QUEUE

KORIŠĆENJE QUEUE-A U FREERTOS-U

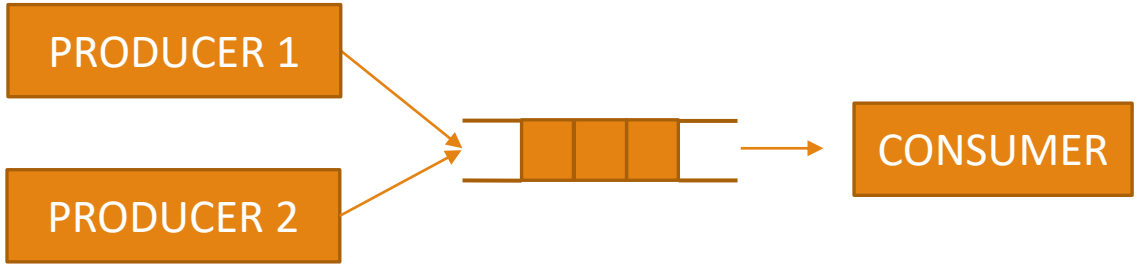
RTOS QUEUE

Red sa porukama

- Jedan od načina da se realizuje komunikacija između taskova, ali i između taskova i prekidne rutine, jeste korišćenjem **objekta kernela RTOS** pod nazivom ***Queue*** (Red sa porukama)
  - FIFO Struktura
    - Podaci se **dodaju na kraj** reda a **čitaju se sa početka** reda
  - ***Thread-safe*** struktura

- Komunikacija bazirana na korišćenju *queue*-a podrazumeva da postoji neko (task ili ISR) koji upisuje elemente u *queue* (***producer***) i da postoji neko (task ili ISR) ko čita ono što je upisano u queue (consumer)

- Može da postoji više producera i više consumera koji koriste isti queue
  - Može da postoji više producera i više consumera koji koriste isti queue
    - Najšeće se javljaju modeli komunikacije u okviru kojih imamo **više *producer*-a** koji generišu podatke koji se obrađuju od strane **samo jednog *consumer-a***



RTOS QUEUE

Red sa porukama

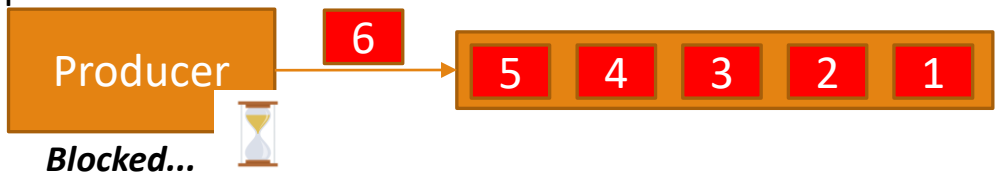
- Pri radu sa *queue*-om kao prvi korak neophodno je **kreirati** *queue*. Obično se prilikom kreiranja specificiraju vrednosti dva parametra
  - Veličina jednog elementa unutar *queue*-a (***Queue Item size***)
  - Maksimalan broj elemenata koji može biti unutar *queue*-a(***Queue length***)
- **Elementi** *queue*-a mogu biti **različiti tipovi podataka**
  - Ugrađeni tipovi podataka (int, char, ...)
  - Korisnički definisani tipovi podataka (strukture)
- Ukoliko u *queue* upisujemo „složenije“ poruke preporučljivo je **kreirati strukturu**, instancirati **objekat te strukture**, **inicijalizovati** polja tog objekta i **upisati** ga u *queue*
  - Dakle, ideja je da ne upisujemo „sirov“ podatak već da dodamo još neke dodatne oznake podatku koje su potrebne kako bi se na prijemnoj strani podatak uspešno obradio
    - Na primer, iz dva taska se upisuju podaci u jedan *queue*. Da bi prijemna strana mogla da identifikuje koji task je proizveo podatke neophodno je enkapsulirati proizvedene podatke koristeći instancu prethodno kreirane strukture
- Upisivanje podataka u *queue* vrši se na jedan od dva načina:
  - Umesto sadržaja podatka u queue se upisuje pokazivač na podatak (prosleđivanje po referenci)
    - ***Brži upis ali moramo čuvati podatak na predajnoj strani sve dok ga prijemna strana ne obradi***
  - Kopiranje sadržaja podatka u queue (prosleđivanje po referenci)
    - ***Sporiji upis ali čim smo podatak upisali u queue ne moramo više da brinemo o njemu***

Šta je bolje???

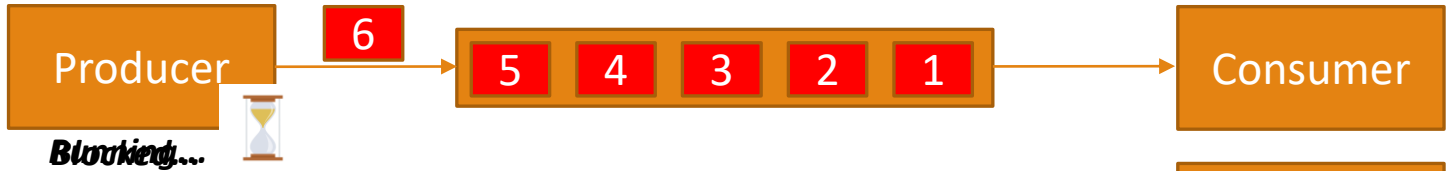
RTOS QUEUE

Sinhronizacija

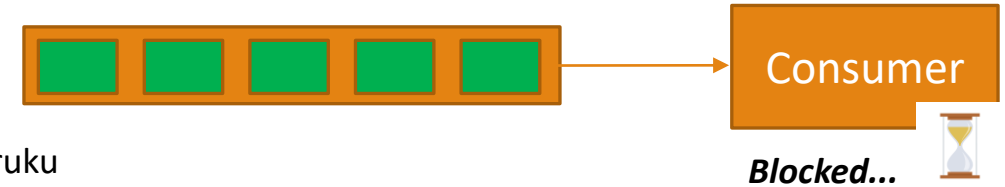
- Blokiranje taska koji vrši operaciju nad *queue*-om je moguće:
  - Ukoliko se upisuje u pun queue



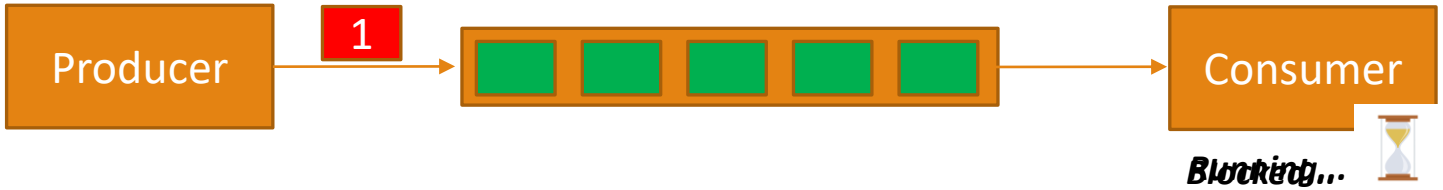
    - Task će se odblokirati kada consumer pročita prvu poruku



  - Ukoliko čita iz praznog queue-a



    - Task će se odblokirati kada producer upiše prvu poruku



- Dakle, pored toga što se queue može koristiti kako bi se ostvarila komunikacije *queue* se može koristiti i kako bi se ostvarila i sinhronizacija između taskova koji čekaju na neki podatak
  - Na primer, kako bi se sinhronizovali taskovi koji rade različitim brzinama

FreeRTOS

QUEUE

API FUNKCIJE

PRIMERI

FreeRTOS QUEUE

API

- Kao prvi korak u procesu korišćenja *queue*-a neophodno je kreirati *queue*
  - Queue se kreira pozivom sledeće FreeRTOS API f-je

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t uxItemSize );
```

Naziv parametra	Opis
`uxQueueLength`	Maksimalan broj elemenata u *queue*-u
`uxItemSize`	Veličina jednog elementa u okviru *queue*-a

  - Ukoliko postoje problemi sa kreiranjem *queue*-a vraća se NULL
    - Najčešće problem postoji zbog **nedovoljne memorije** na platformi. O tome više pri kraju vežbi
  - Ukoliko je *queue* uspešno kreiran vraća se **instanca** strukture (***handler***) ***QueueHandle_t*** koja se koristi kako bi se pristupilo kreiranom *queue*-u

FreeRTOS QUEUE

API

- Upis u ***queue*** je moguće realizovati posredstvom jedne od dve funkcije
  - Funkcijom koja dodaje elemente na početak reda

```
BaseType_t xQueueSendToFront( QueueHandle_t xQueue,
                              const void * pvItemToQueue,
                              TickType_t xTicksToWait );
```

Naziv parametra	Opis
xQueue	Instanca prethodno kreiranog *queue*-a
pvItemToQueue	Pokazivač na podatak koji će biti **kopiran** u *queue*
xTicksToWait	Ukoliko je *queue* pun ovim parametrom se specificira koliko vremena će task provesti u blokiranom stanju

    - F-ja vraća ***pdPASS*** ukoliko je element uspešno pročitan. Ukoliko je *queue* **prazan** i **isteklo je vreme** koje je specificirano da task provede u blokiranom stanju funkcija fraća ***errQUEUE_EMPTY***
  - Funkcijom koja dodaje elemente na kraj reda

```
BaseType_t xQueueSendToBack( QueueHandle_t xQueue,
                             const void * pvItemToQueue,
                             TickType_t xTicksToWait );
```

  - Ukoliko se funkcije pozivaju iz konteksta prekidne rutine potrebno je koristiti njihove implementacije koje se završavaju sa ***FromISR***
    - xQueueSendToBackFromISR i xQueueSendToFromFromISR
    - Pogledati ***FreeRTOS Reference manual***

FreeRTOS QUEUE

API

- Čitanje iz ***queue***-a je moguće koristeći sledeću f-ju

```
BaseType_t xQueueReceive( QueueHandle_t xQueue,
                          void * const pvBuffer,
                          TickType_t xTicksToWait );
```

Naziv parametra	Opis
`xQueue`	Instanca prethodno kreiranog *queue*-a
`pvItemToQueue`	Pokazivač na deo memorije u koju će biti kopiran pročitani podatak
`xTicksToWait`	Ukoliko je *queue* prazan ovim parametrom se specificira koliko vremena će task provesti u blokiranom stanju

  - F-ja vraća ***pdPASS*** ukoliko je element uspešno pročitan. Ukoliko je *queue* **prazan** i **isteklo je vreme** koje je specificirano da task provede u blokiranom stanju funkcija vraća ***errQUEUE_EMPTY***

- Ukoliko se funkcije poziva iz konteksta prekidne rutine potrebno je koristiti njenu implementaciju koja se završava sa ***FromISR***
  - *`xQueueReceiveFromISR`*

- Ovo su neke od najčešće korišćenih f-ja za rad sa *queue*-om. Ostale f-je, kao i njihove detaljne opise, moguće je naći u okviru ***FreeRTOS Reference Manual*** dostupnom na zvaničnom FreeRTOS sajtu.

➢ ZADATAK 16

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Ukoliko preko UART interfejsa stigne karakter „e“ potrebno je uključiti diodu LD3. Ukoliko stigne karakter „e“ dok je dioda LD3 uključena, dioda ostaje uključena. Ukoliko preko UART interfejsa pristigne karakter „d“ potrebno je isključiti diodu LD3. Ukoliko je dioda LD3 bila isključena, a pristikao je katakter „d“, dioda ostaje isključena.

UART je potrebno konfigurisati kao ***9600_8N1***

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

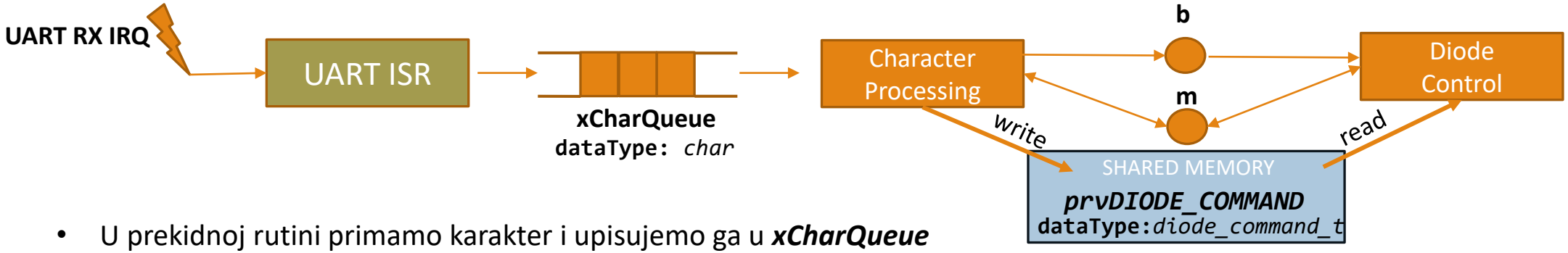
➢ ZADATAK 16 – Rešenje (Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti dve logičke celine
  1. **Prijem i obrada karaktera**
     - Ovu ligičku celinu implementiraćemo kroz prekid UART-a i „Character processing“ task. Karakter ćemo primiti u okviru prekidne rutine UART interfejsa i prosledićemo ga „Character processing“ tasku na dalje procesiranje
  2. **Kontrola stanja diode**
     - Ovu logičku celinu implementiraćemo u okviru „Diode Control“ taska

FreeRTOS QUEUE

Primeri

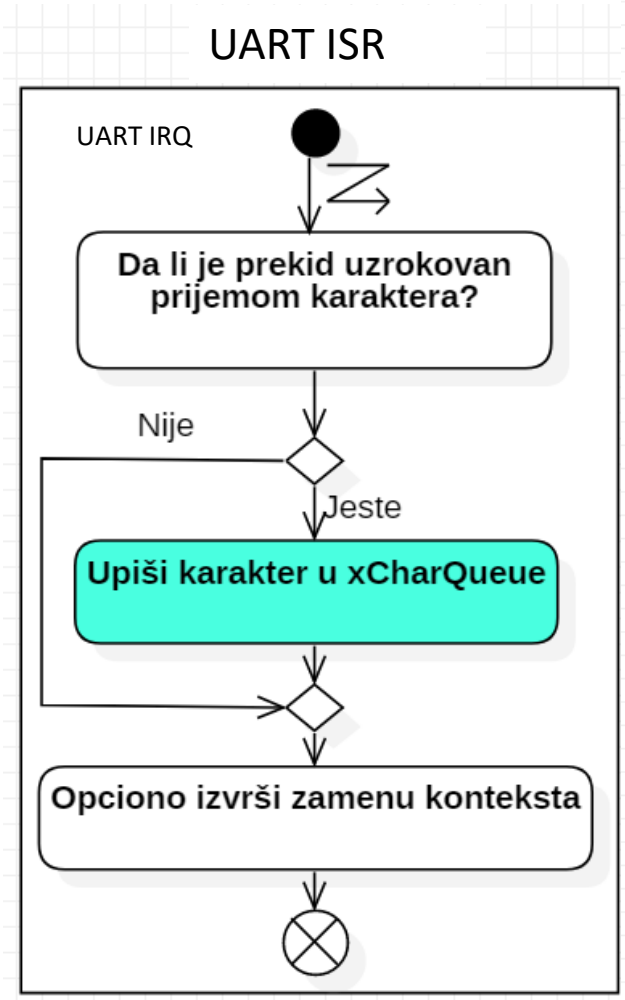
➢ ZADATAK 16 – Rešenje (Projektovanje)

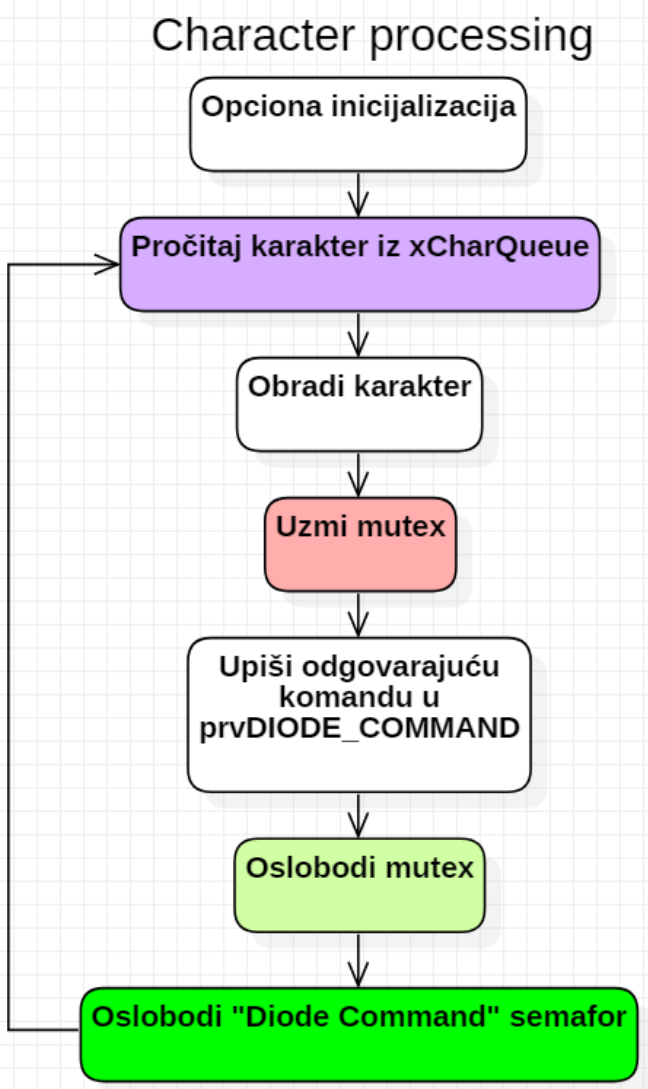


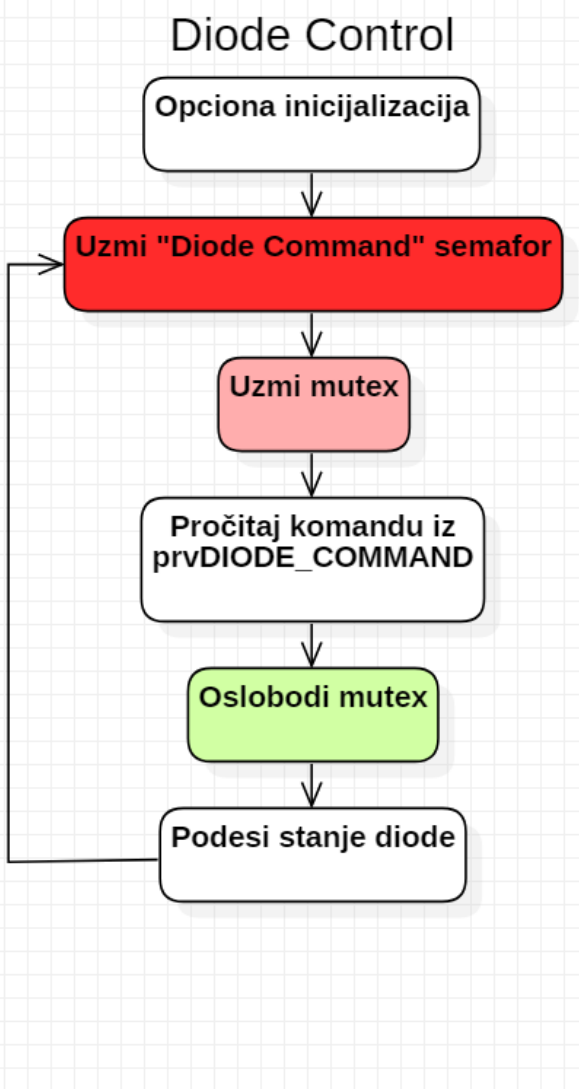
- U prekidnoj rutini primamo karakter i upisujemo ga u ***xCharQueue***
  - ***xCharQueue*** je *queue* koji služi za baferisanje primljenih karaktera.
- „Character Processing“ task ima zadatak da čita karakter po karakter iz *xCharQueue*-a i da shodno pročitanoj vrednosti upiše odgovarajuću komandu u globalnu promenljivu ***prvDIODE_COMMAND***
  - ***prvDIODE_COMMAND*** će biti korisnički definisan nabrojivi tip podataka (enum) ***diode_command_t*** koji će imati dve vrednosti DIODE_COMMAND_ON i DIODE_COMMAND_OFF.
- „Diode Control“ task ima zadatak da kada se oslobodi binarni semafor, na kom je ovaj task blokiran, na osnovu komande ***prvDIODE_COMMAND*** uključi ili isključi diodu
- **Kako izgleda raspodela prioriteta po taskovima za ovakvu realizaciju?**

Primeri

- ZADATAK 16 – Rešenje (Projektovanje)



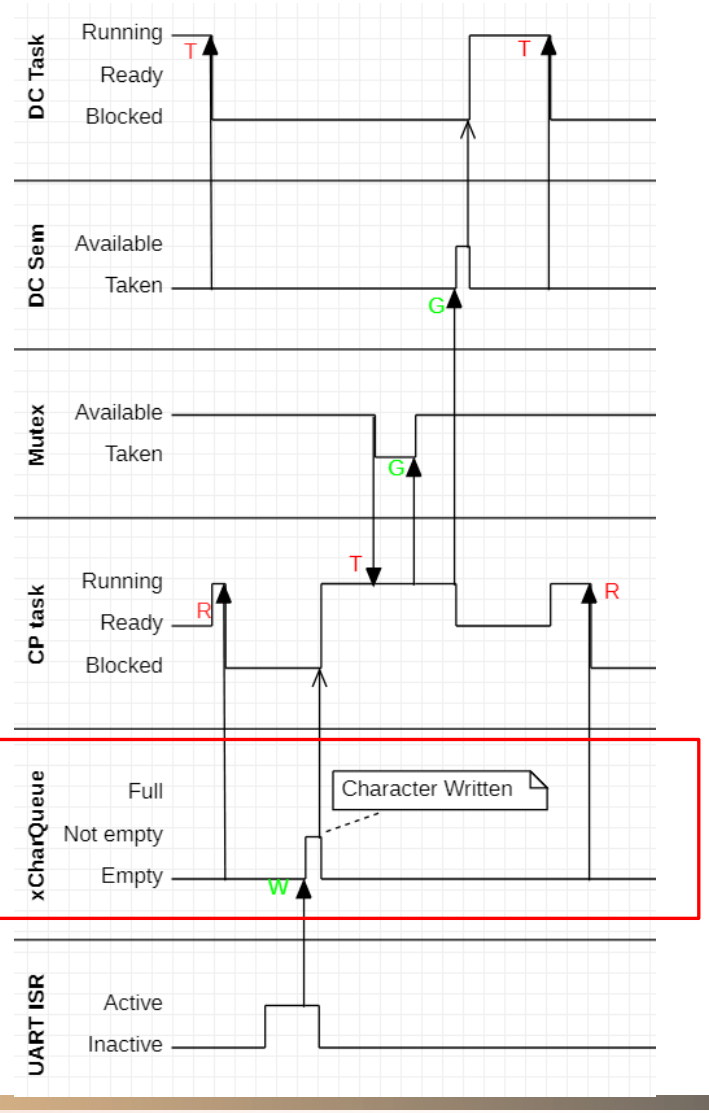




FreeRTOS QUEUE

Primeri

- ZADATAK 16 – Rešenje (Projektovanje)
  - U čemu je razlika između *queue*-a i semafora???
- ZADATAK 16 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_9***



➢ ZADATAK 17

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Ukoliko preko UART interfejsa stigne karakter „e“ potrebno je uključiti diodu LD3. Ukoliko stigne karakter „e“ dok je dioda LD3 uključena, dioda ostaje uključena. Ukoliko preko UART interfejsa pristigne karakter „d“ potrebno je isključiti diodu LD3. Ukoliko je dioda LD3 bila isključena, a pristigao je katakter „d“, dioda ostaje isključena.

Pored kontrole diode, koja se može realizovati posredstvom UART interfejsa, treba omogućiti da se dioda paralelno kontroliše i posredstvom tastera SW3 i SW4. SW3 taster isključuje diodu dok taster SW4 uključuje diodu.

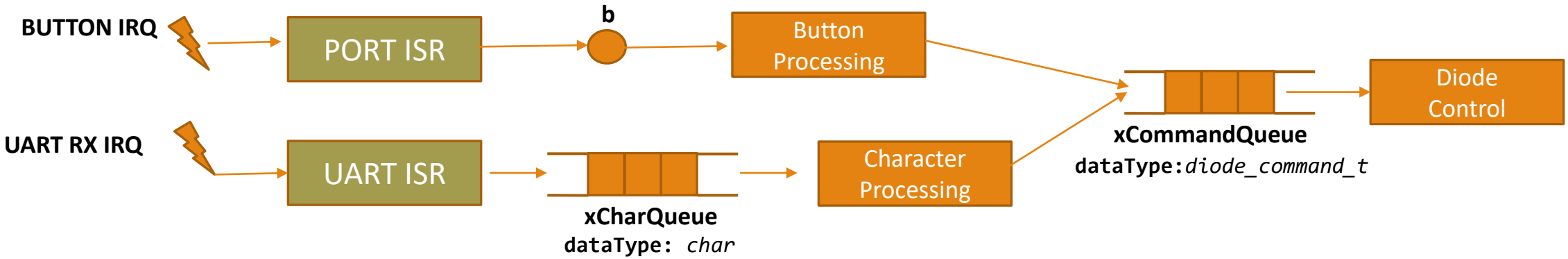
UART je potrebno konfigurisati kao ***9600_8N1***

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

➢ ZADATAK 17 – Rešenje (Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti tri logičke celine

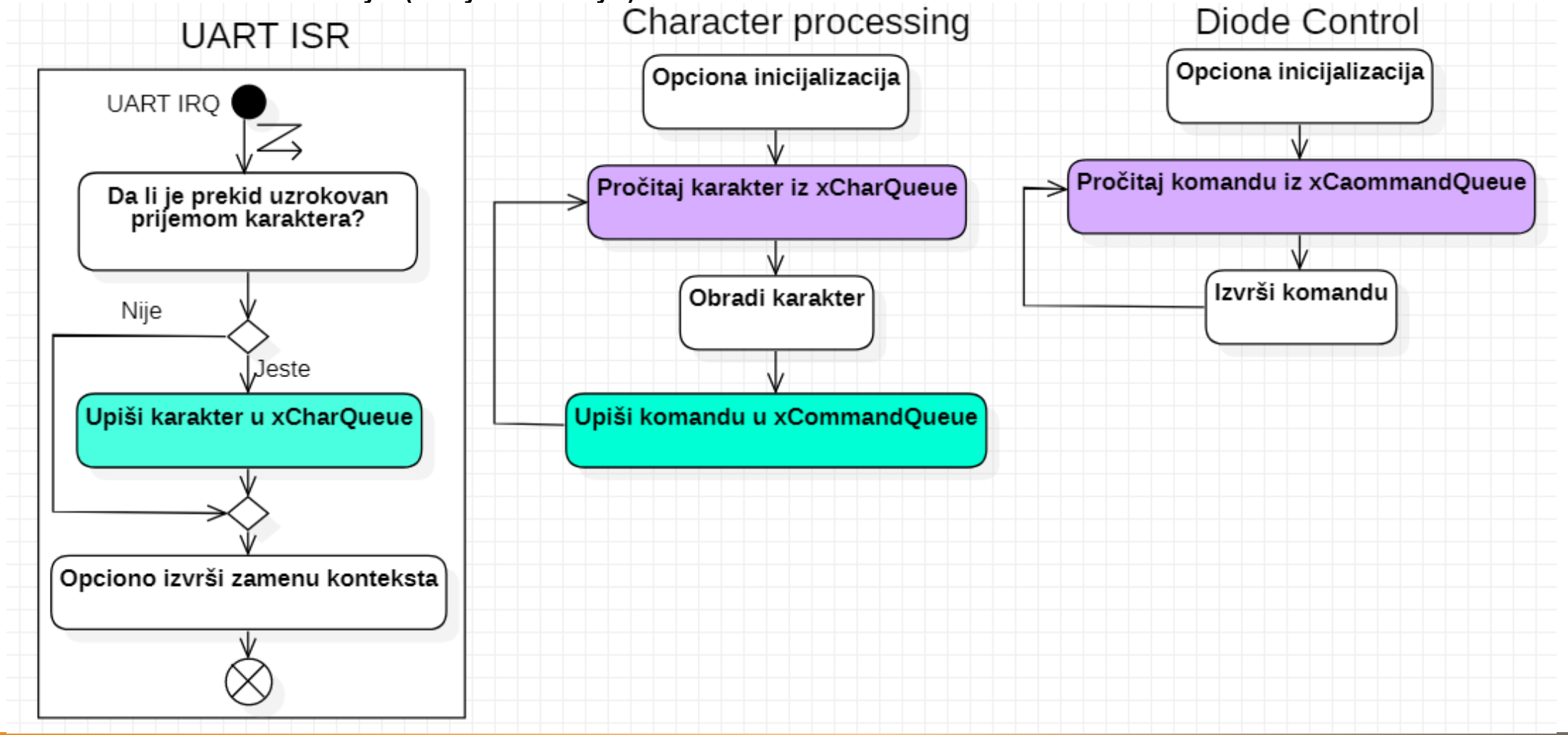
1. **Prijem i obrada karaktera**
   - Ovu ligičku celinu implementiraćemo kroz prekid UART-a i „Character processing“ task. Karakter ćemo primiti u okviru prekidne rutine UART interfejsa i prosledićemo ga „Character processing“ tasku na dalje procesiranje
2. **Kontrola stanja diode**
   - Ovu logičku celinu implementiraćemo u okviru „Diode Control“ taska
3. **Procesiranje tastera**
   - Ovu ligičku celinu implementiraćemo kroz prekid paralelnog porta na koji je povezan taster i „Button processing“ task. Detekciju pritiska tastera izvšićemo u prekidnoj rutini i dok ćemo dalje procesiranje izvršiti u okviru „Button processing“ taska na način na koji smo to pre radili.



FreeRTOS QUEUE

Primeri

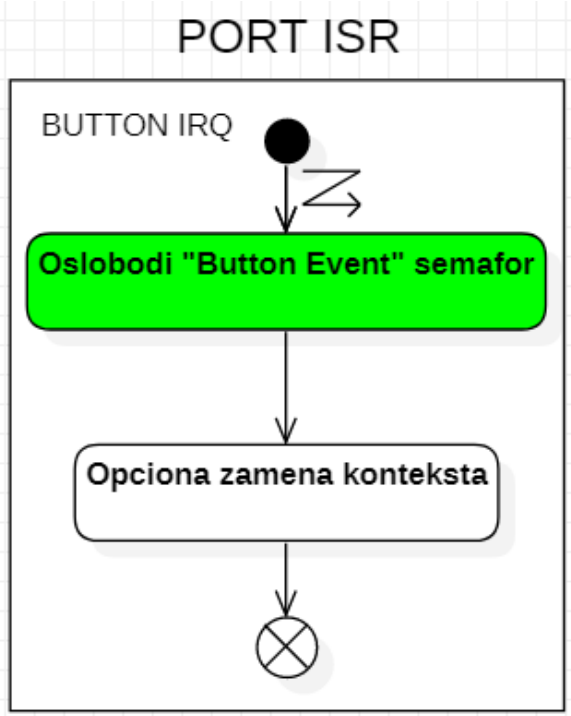
- ZADATAK 17 – Rešenje (Projektovanje)

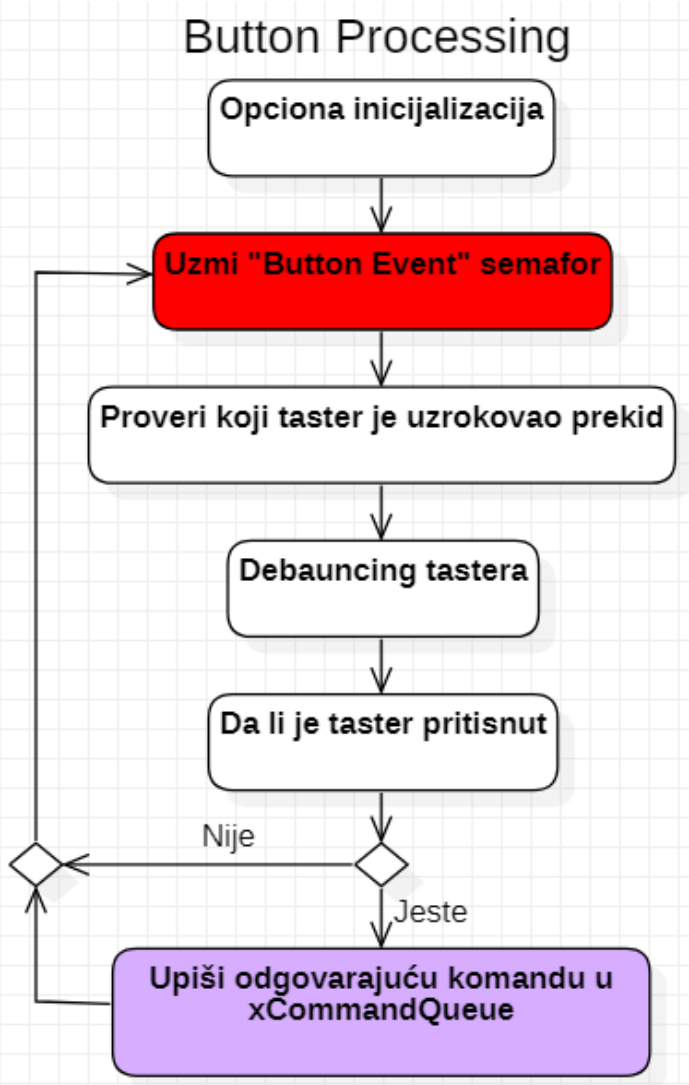


FreeRTOS QUEUE

Primeri

- ZADATAK 17 – Rešenje (Projektovanje)
- ZADATAK 17 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_10***





Primeri

➢ ZADATAK 18

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Na dvocifrenom sedmosegmentnom LE displeju se prikazuje digitalna vrednost nastala konverzijom analogne vrednosti dobijene sa potenciometra PT2. Za konverziju se koristi ADC12 periferija MSP430F5529 mikrokontrolera.

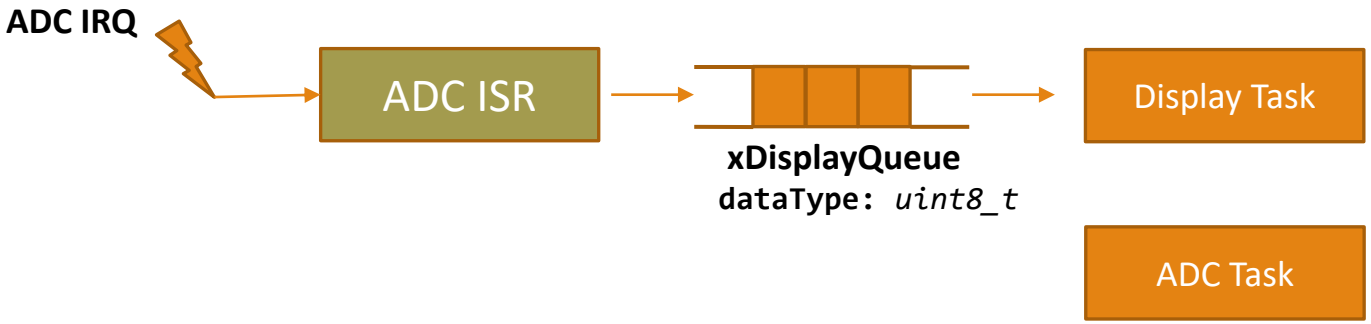
Napomena: Očitavanje AD konvertora se vrši **periodično** sa periodom od **200 sistemskih tikova**. Rezolucija AD konvertora je 12bita, a na eksternom *schield*-u imamo dostupna samo 2 LE sedmosegmentna displeja, pa je potrebno izvršiti skaliranje.

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

➢ ZADATAK 18 - Rešenje(Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine

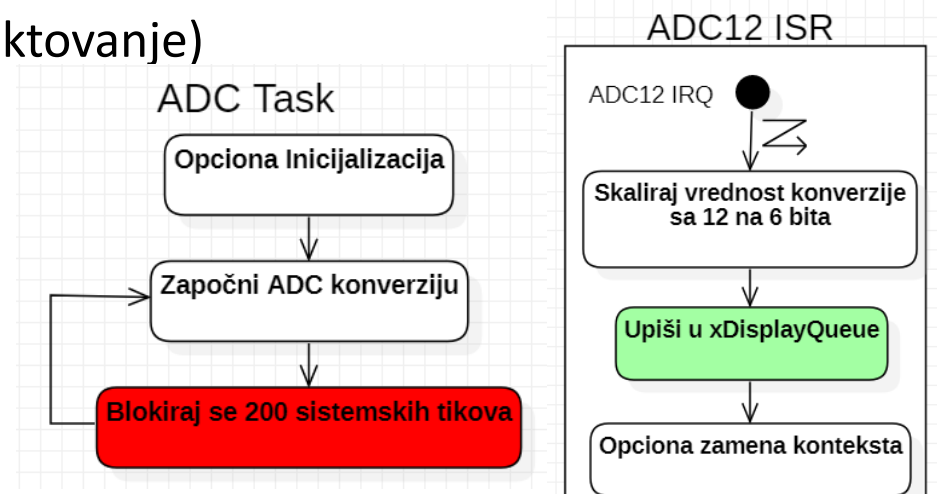
1. **Čitanje vrednosti sa AD konvertora**
   - Ovu logičku celinu implementiraćemo koristeći ADC12 prekidnu rutinu i „ADC Task“ task. U okviru „ADC Task“ taska periodično ćemo započinjati konverziju dok ćemo rezultat konverzije čitati iz prekidne rutine
2. **Ispis na dvocifreni sedmosegmentni displej**
   - Mehanizam multipleksiranja dvocifrenog sedmosegmentnog LE displeja implementiraćemo u okviru „Display task“ taska.

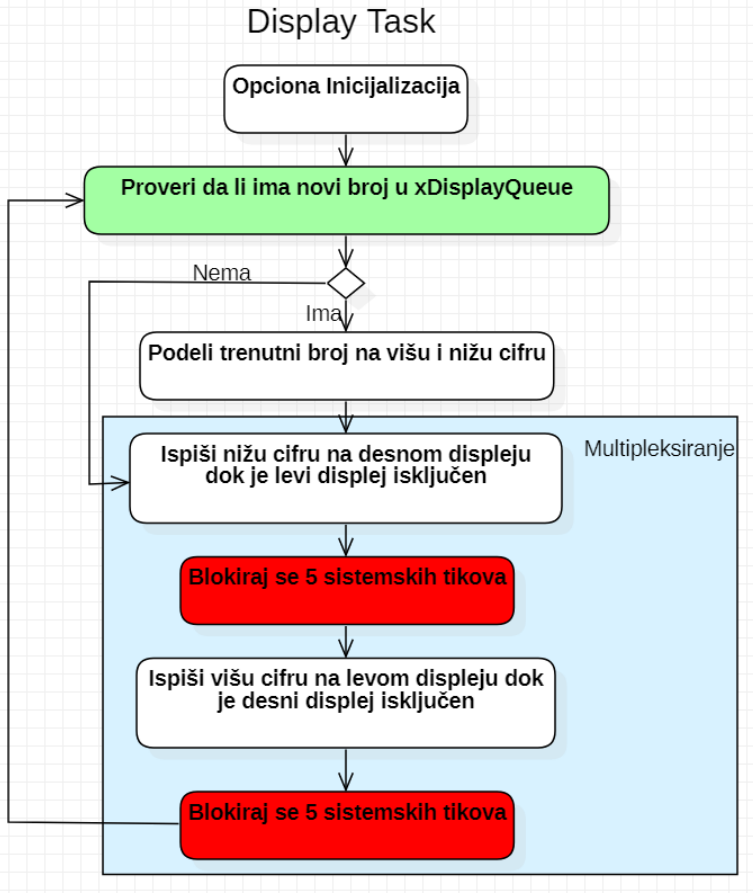


- „ADC Task“ na svakih 200 sistemskih tikova **softverski započinje konverziju**. Kada se konverzija završi generiše se prekidni zahtev koji dolazi od ADC12 periferije i poziva se prekidna rutina dodeljena ovom prekidnom zahtevu. U okviru prekidne rutine čita se **ADC12MEM0** registar i vrednost registra, nakon skaliranja, se upisuje u ***xDisplayQueue***. „Display Task“ task sa periodom od 10 sistemskih tikova vrši čitanje *xDisplayQueue queue*-a kako bi proverio da li je upisana nova vrednost u queue ali se ne blokira ukoliko nema nove vrednosti u ***queue***-u već nastavlja sa mehanizmom multipleksiranja displeja.

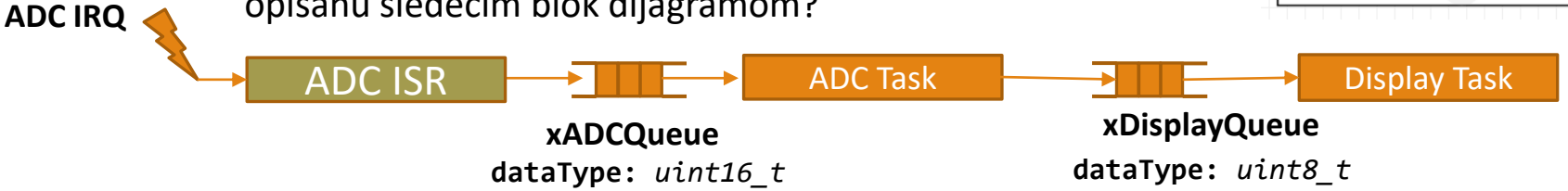
Primeri

➢ ZADATAK 18 - Rešenje(Projektovanje)





- Kako bi izgledali dijagrami aktivnosti za realizaciju zahteva opisanu sledećim blok dijagramom?



- U čemu je prednost ove realizacije u odnosu na prethodnu?

➢ ZADATAK 18 - Rešenje(Realizacija)

- Videti kôd projekta ***SRV_2_11***

Primeri

- ZADATAK 19 (Za samostalni rad)

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Na dvocifrenom sedmosegmentnom LE displeju se prikazuje digitalna vrednost nastala konverzijom analogne vrednosti dobijene sa potenciometra PT2. Za konverziju se koristi ADC12 periferija MSP430F5529 mikrokontrolera. Pritiskom na taster SW3 se putem UART-a rezultat konverzije u ASCI formatu prosleđuje na računar.

Napomena: Startovanje konverzije se vrši **periodično** sa periodom od **200 sistemskih tikova**. Rezolucija AD konvertora je 12bita, a na eksternom *schield*-u imamo dostupna samo 2 LE sedmosegmentna displeja, pa je potrebno izvršiti skaliranje.

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

ZADATAK 20(Za samostalni rad)

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Na dvocifrenom sedmosegmentnom LE displeju se prikazuje digitalna vrednost nastala konverzijom analogne vrednosti dobijene sa jednog od dva potenciometra. Prebacivanje sa potenciometra PT2 na potenciometar PT1, i obratno, vrši se posredstvom tastera SW4. Ukoliko je selektovan potenciometar PT1 uključena je dioda LD3 dok ukoliko je selektovan potenciometar PT2 uključena je dioda LD4. Za AD konverziju se koristi ***ADC12*** periferija MSP430F5529 mikrokontrolera. Pritiskom na taster SW3 se putem UART-a rezultat konverzije u ASCI formatu prosleđuje na računar.

Napomena: Startovanje konverzije se vrši **periodično** sa periodom od **200 sistemskih tikova**. Rezolucija AD konvertora je 12bita, a na eksternom *schield*-u imamo dostupna samo 2 LE sedmosegmentna displeja, pa je potrebno izvršiti skaliranje.

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

ZADATAK 20 – Mogućnost bonus poena

- Kompletno rešenje (**dijagrami koji opisuju realizaciju sa kratkim opisom u PDF formatu i sama realizacija u vidu CCS projekta**) zapakovati u arhivu pod nazivom ***BONUS_GGGG_BBBB.zip*** (gde je BBBB broj indeksa a GGGG godina upisa) i potaviti na OneDrive i link poslati na ***haris@etf.rs*** najkasnije do 26.04.2022. godine.

RTOS

KOMUNIKACIJA I SIGNALIZACIJA

FREERTOS SPECIFIČNI MEHANIZMI

KOMUNIKACIJA I SIGNALIZACIJA DOGAĐAJA

Funkcionalnosti specifične za FreeRTOS

- Do sada smo govorili o opštim mehanizmima komunikacije i signalizacije pojave događaja koji su implementirani u **skoro svim realizacijama RTOS**
  - Semafori (binarni, brojački i mutex) i *queue*-ovi
  - Pokazali smo kako su ovi mehanizmi implementirani i kako se koriste u okviru FreeRTOS-a
- Ova prezentacija predstaviće mehanizme komunikacije i signalizacije pojave događaja specifične za FreeRTOS
  - **Grupa događaja**
  - **Notifikacija**

FreeRTOS

Grupa događaja

SIGNALIZACIJA DOGAĐAJA KORIŠĆENJEM GRUPE DOGAĐAJA

API FUNKCIJE

PRIMERI

FreeRTOS GRUPA DOGAĐAJA

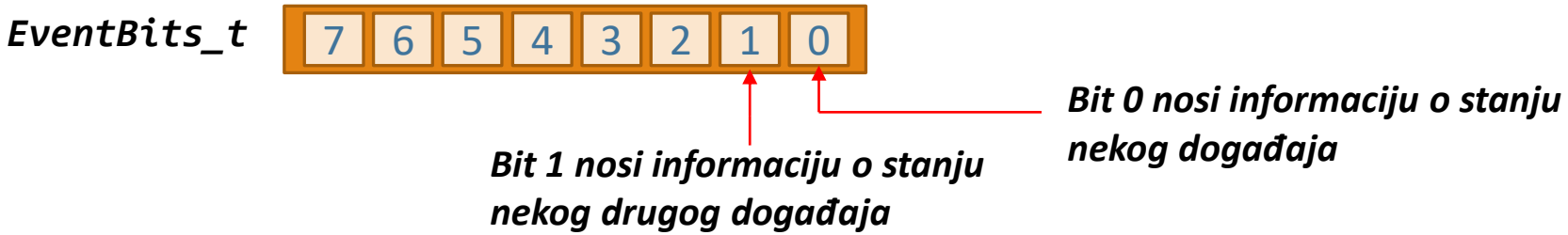
Osobine

- Mehanizam signalizacije pojave jednog ili više događaja **karakterističan za FreeRTOS**
- Mehanizam „Grupa događaja“ omogućava:
  - Sinhronizaciju više taskova
  - *Broadcasting* događaja na više taskova
  - Blokiranje jednog ili više taskova do trenutka generisanja jednog ili **kombinacije** događaja
  - Redukovanje količine memorije koja se koristi od strane FreeRTOS-a jer više semafora, koji su korišćeni za signalizaciju više događaja, sada možemo zameniti samo jednom instancom grupe događaja.
- Korišćenje ovog mehanizma nije obavezno. Ukoliko želimo da koristimo grupu događaja potrebno je u okviru projekta kompajlirati fajl *event_groups.c*

FreeRTOS GRUPA DOGAĐAJA

Realizacija

- Grupa događaja je realizovana kao niz *boolean* vrednosti
  - Informacija o pojavi nekog događaja (***stanje događaja***) čuva se u okviru jednog bita (***Event flag***)
    - Vrednost 1 označava da je došlo do generisanja događaja dok vrednost 0 označava da se događaj nije generisao
  - Ove binarne vrednosti koje čuvaju stanje događaja su deo promenljive tipa ***EventBits_t***



  - Pri projektovanju softvera neophodno je svakom od bita dodeliti odgovarajuće značenje
    - Na primer bit 0 nosi informaciju o tome da li je taster SW3 pritisnut dok bit 4 nosi informaciju o tome da li je taster SW4 pritisnut.
- Broj događaja koji se čuvaju u okviru jedne promenljive tipa EventBits_t zavisi od toga da li je setovan ***configUSE_16_BIT_TICKS***
  - ***configUSE_16_BIT_TICKS*** setovan na ***1*** → ***Koristi se 8 bita***
  - ***configUSE_16_BIT_TICKS*** setovan na ***0*** → ***Koristi se 24 bita***

API f-je

- Kreiranje grupe događaja se realizuje pozivom sledeće API f-je

```
EventGroupHandle_t xEventGroupCreate( void );
```

  - Ukoliko nije moguće kreirati grupu događaja ova f-ja vraća NULL
  - Ukoliko je grupa događaja uspešno kreirana ova f-ja vraća instancu tipa ***EventGroupHandle_t*** koja se koristi za pristup grupi događaja

- Setovanje jednog ili više bita u okviru grupe događaja realizuje se pozivom sledeće API f-je

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
                                const EventBits_t uxBitsToSet );
```

Naziv parametra	Opis
xEventGroup	Instanca prethodno kreirane grupe događaja
uxBitsToSet	Binarna **maska** koja specificira koje bite u okviru grupe bita treba setovati na 1

  - F-ja vraća sadržaj koju promenljiva tipa ***EventBits_t,*** unutar grupe događaja, ima u trenutku poziva ove f-je
    - Ova vrednost ne mora imati one bite koji su specificirani promenljivom uxBitsToSet setovane na 1!
      - Na primer neki task u međuvremenu može setovati iste te bite na 0

FreeRTOS GRUPA DOGAĐAJA

API f-je

- Čekanje da jedan ili više bitova budu setovani na 1, od stane drugog taska ili prekidne rutine, realizuje se pozivom sledeće API f-je

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Naziv parametra	Opis
`xEventGroup`	Instanca prethodno kreirane grupe događaja
`uxBitsToWaitFor`	Binarna **maska** koja specificira bite u grupi događaja na koje čekamo da budu setovani na vrednost 1
`xClearOnExit`	Ovaj parametar može imati dve vrednosti: ***pdTRUE*** ili ***pdFALSE***. Ukoliko ovaj parametar ima vrednost ***pdTRUE*** izvršiće se setovanje bita, definisanih maskom ***uxBitsToWaitFor,*** na vrednost 0. Ukoliko ovaj parametar ima vrednost ***pdFALSE*** biti ostaju nepromenjeni.
`xWaitForAllBits`	Ovaj parametar može imati dve vrednosti: ***pdTRUE*** ili ***pdFALSE***. Ukoliko ovaj parametar ima vrednost ***pdTRUE*** task će ostati u stanju „Blocked“ dok **svi biti,** definisani maskom ***uxBitsToWaitFor,*** ne budu setovani na 1. Ukoliko je vrednost ovog parametra ***pdFALSE*** task će ostati u stanju „Blocked“ dok **bar jedan od bita,** definisanih maskom ***uxBitsToWaitFor,*** ne budu setovani na vrednost 1
`xTicksToWait`	Vreme koje će task provesti u stanju „Blocked“ čekajući da se neki (ili svi) biti setuju na vrednost 1

- Ukoliko se iz f-je izlazi kao posledica setovanja barem jednog bita (ili svih bita) definisanih maskom ***uxBitsToWaitFor,*** f-ja vraća vrednost koju je grupa događaja imala u trenutku kada su se stvorili uslovi da task, koji čeka na bite, bude odblokiran.
  - U slučaju da je ***xClearOnExit*** parametar setovan f-ja vraća vrednost neposredno pre setovanja bita na vrednost 0

FreeRTOS GRUPA DOGAĐAJA

API f-je

- Čekanje da jedan ili više bitova budu setovani na 1, od stane drugog taska ili prekidne rutine, realizuje se pozivom sledeće API f-je

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Naziv parametra	Opis
`xEventGroup`	Instanca prethodno kreirane grupe događaja
`uxBitsToWaitFor`	Binarna **maska** koja specificira bite u grupi događaja na koje čekamo da budu setovani na vrednost 1
`xClearOnExit`	Ovaj parametar može imati dve vrednosti: ***pdTRUE*** ili ***pdFALSE***. Ukoliko ovaj parametar ima vrednost ***pdTRUE*** izvršiće se setovanje bita, definisanih maskom ***uxBitsToWaitFor,*** na vrednost 0. Ukoliko ovaj parametar ima vrednost ***pdFALSE*** biti ostaju nepromenjeni.
`xWaitForAllBits`	Ovaj parametar može imati dve vrednosti: ***pdTRUE*** ili ***pdFALSE***. Ukoliko ovaj parametar ima vrednost ***pdTRUE*** task će ostati u stanju „Blocked“ dok **svi biti,** definisani maskom ***uxBitsToWaitFor,*** ne budu setovani na 1. Ukoliko je vrednost ovog parametra ***pdFALSE*** task će ostati u stanju „Blocked“ dok **bar jedan od bita,** definisanih maskom ***uxBitsToWaitFor,*** ne budu setovani na vrednost 1
`xTicksToWait`	Vreme koje će task provesti u stanju „Blocked“ čekajući da se neki (ili svi) biti setuju na vrednost 1

- Ukoliko se iz f-je izlazi kao posledica isteka vremenskog intervala definisanog parametrom ***xTicksToWait,*** f-ja vraća vrednost koju grupa događaja ima u tom trenutku.

FreeRTOS GRUPA DOGAĐAJA

API f-je

- Setovanje jednog ili više bita u okviru grupe događaja, iz prekidne rutine, realizuje se pozivom sledeće API f-je

```
BaseType_t xEventGroupSetBitsFromISR( EventGroupHandle_t xEventGroup,
                                      const EventBits_t uxBitsToSet,
                                      BaseType t *pxHigherPriorityTaskWoken );
```

Naziv parametra	Opis
xEventGroup	Instanca prethodno kreirane grupe događaja
uxBitsToSet	Binarna **maska** koja specificira koje bite u okviru grupe bita treba setovati na 1
pxigherPriorityTaskWoken	Ukoliko je vrednost ovog parametra setovana na pdTURE tada je potrebno izvršiti zamenu konteksta pre izlaska iz prekidne rutine

➢ ZADATAK 21

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Tasteri S3 i S4 menjaju stanje dioda LD3 i LD4, respektivno. Taster S2 uzrokuju slanje stringa „Taster S2“ sa MSP430F5529 razvojne platforme na PC računar putem UART interfejsa.

Napomena: UART je potrebno konfigurisati kao ***9600_8N1.*** Taster S2 se nalazi na TI razvojnoj ploči (crvena pločica).

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

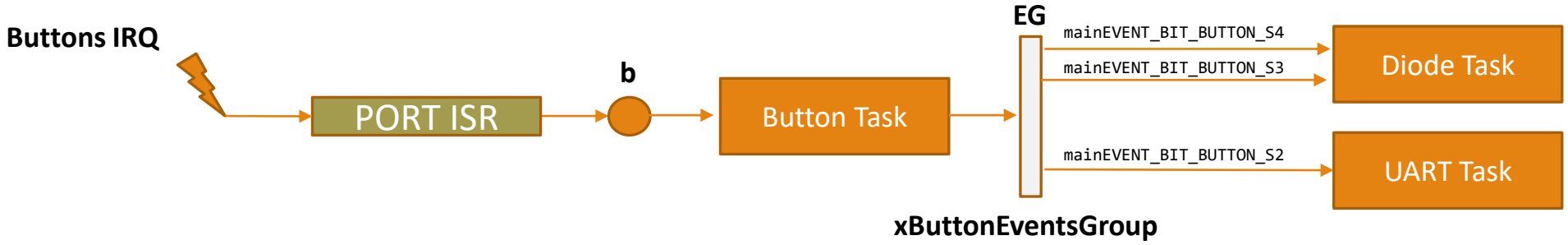
➢ ZADATAK 21 - Rešenje(Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine

1. **Detekcija pritiska tastera**
   - Ovu logičku celinu implementiraćemo koristeći prekidnu rutinu porta i „Button Task“ task. U okviru prekidne rutine porta detektovaćemo da se desio potencijalni pritisak tastera dok ćemo odloženu obradu nastaviti u okviru „Button task“ taska.
2. **Promena stanja diode**
   - Promenu stanja odgovarajuće diode shodno detektovanom tasteru implementiraćemo u okviru „Diode task“ taska
3. **Slanje stringova putem UART interfejsa**
   - Slanje stringa implementiraćemo u okviru „UART task“ taska

➢ ZADATAK 21 - Rešenje(Projektovanje)

- Blok dijagram koji predstavlja korišćene objekte i njihovu međusobnu zavisnost



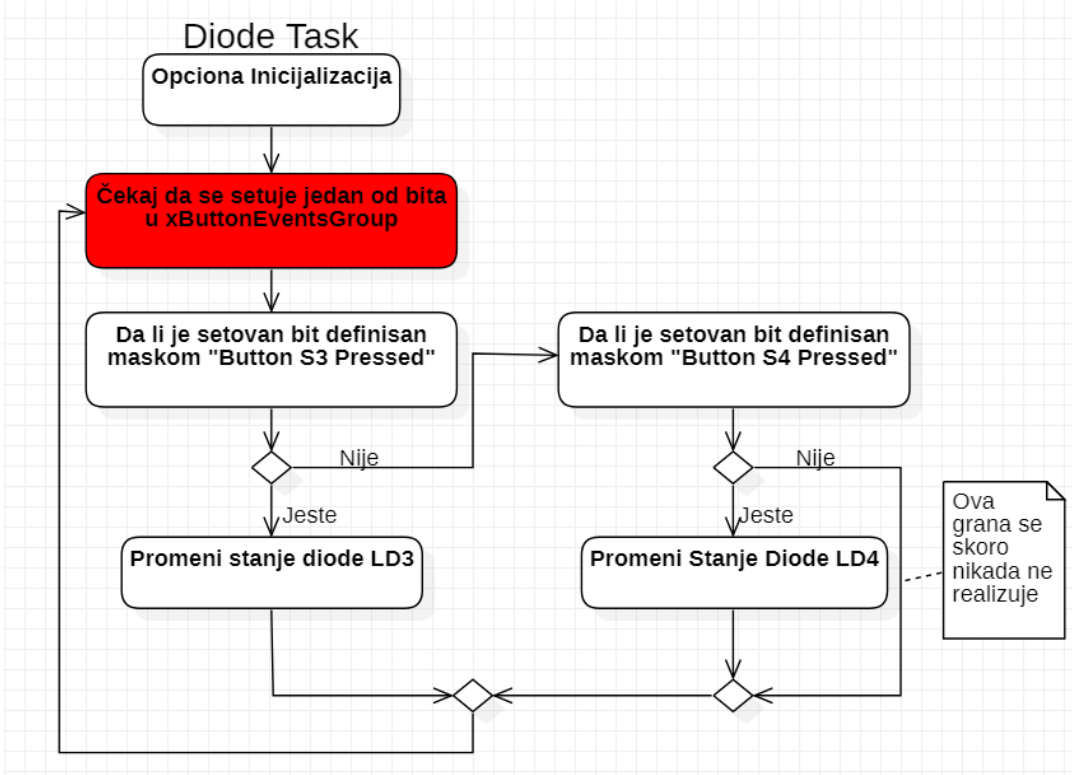
- U prekidnoj rutini porta se detektuje događaj „Potencijalni pritisak tastera“ koji se signalizira „Button task“ tasku posredstvom binarnog semafora **b**. „Button task“ task implementira algoritam debaunsiranja i utvrđuje koji taster je pritisnut. Shodno rezultatu provere, u okviru grupe događaja (EG - ***EventGroup***), setuju se odgovarajući biti na vrednost 1.
- „Diode task“ task čeka na jedan od događaja „Pritisnut taster S3“ ili „Pritisnut taster S4“ koji se signaliziraju posredstvom grupe događaja **EG**
- „UART task“ task čeka na događaj „Pritisnut taster S2“ koji se signalizira posredstvom grupe događaja **EG**

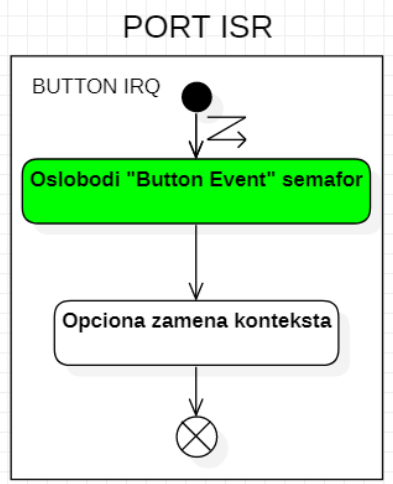
➢ ZADATAK 21 - Rešenje(Projektovanje)

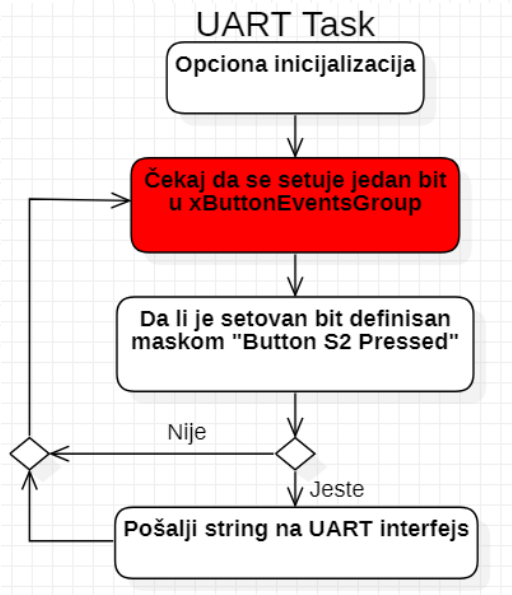
- Predložiti realizaciju koja ne bi koristila grupu događaja.
  - U čemu je razlika?

API f-je

- ZADATAK 21 – Rešenje (Projektovanje)







- ZADATAK 21 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_14***

➢ ZADATAK 22

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Tasteri S3 i S4 menjaju stanje dioda LD3 i LD4, respektivno. Taster S2 uzrokuju slanje stringa „Taster S2“ sa MSP430F5529 razvojne platforme na PC računar putem UART interfejsa dok taster S3 sa ovog mikrokontrolera na PC računar šalje string „Taster S3“.

Napomena: UART je potrebno konfigurisati kao ***9600_8N1.*** Taster S2 se nalazi na TI razvojnoj ploči (crvena pločica).

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

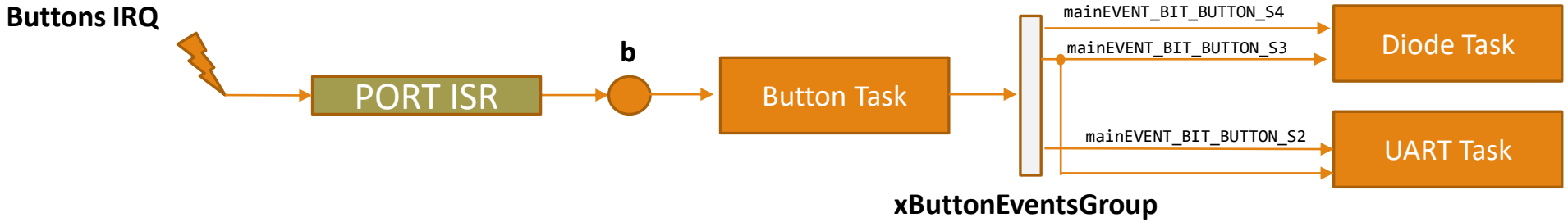
➢ ZADATAK 22 - Rešenje(Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine
  1. **Detekcija pritiska tastera**
     - Ovu logičku celinu implementiraćemo koristeći prekidnu rutinu porta i „Button Task“ task. U okviru prekidne rutine porta detektovaćemo da se desio potencijalni pritisak tastera dok ćemo odloženu obradu nastaviti u okviru „Button task“ taska.
  2. **Promena stanja diode**
     - Promenu stanja odgovarajuće diode shodno detektovanom tasteru implementiraćemo u okviru „Diode task“ taska
  3. **Slanje stringova putem UART interfejsa**
     - Slanje odgovarajućih stringova, shodno pritisku detektovanog tastera, implementiraćemo u okviru „UART task“ taska

API f-je

➢ ZADATAK 22 - Rešenje(Projektovanje)

- Blok dijagram koji predstavlja korišćene objekte i njihovu međusobnu zavisnost



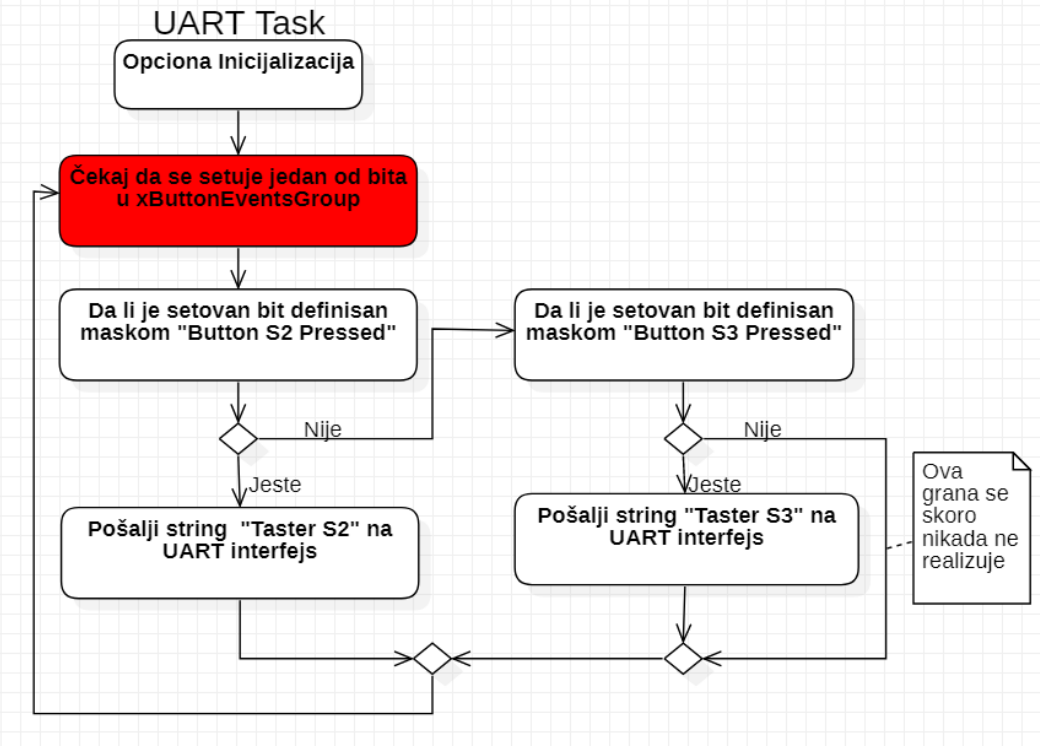
- U prekidnoj rutini porta se detektuje događaj „Potencijalni pritisak tastera“ koji se signalizira „Button task“ tasku posredstvom binarnog semafora **b**. „Button task“ task implementira algoritam debaunsiranja i utvrđuje koji taster je pritisnut. Shodno rezultatu provere setuju se odgovarajući biti u okviru grupe događaja (EG - ***EventGroup***).
- „Diode task“ task čeka na jedan od događaja „Pritisnut taster S3“ ili „Pritisnut taster S4“ koji se signaliziraju posredstvom grupe događaja **EG**
- „UART task“ task čeka na jedan od događaja „Pritisnut taster S2“ ili „Pritisnut taster S3“ koji se signalizira posredstvom grupe događaja **EG**

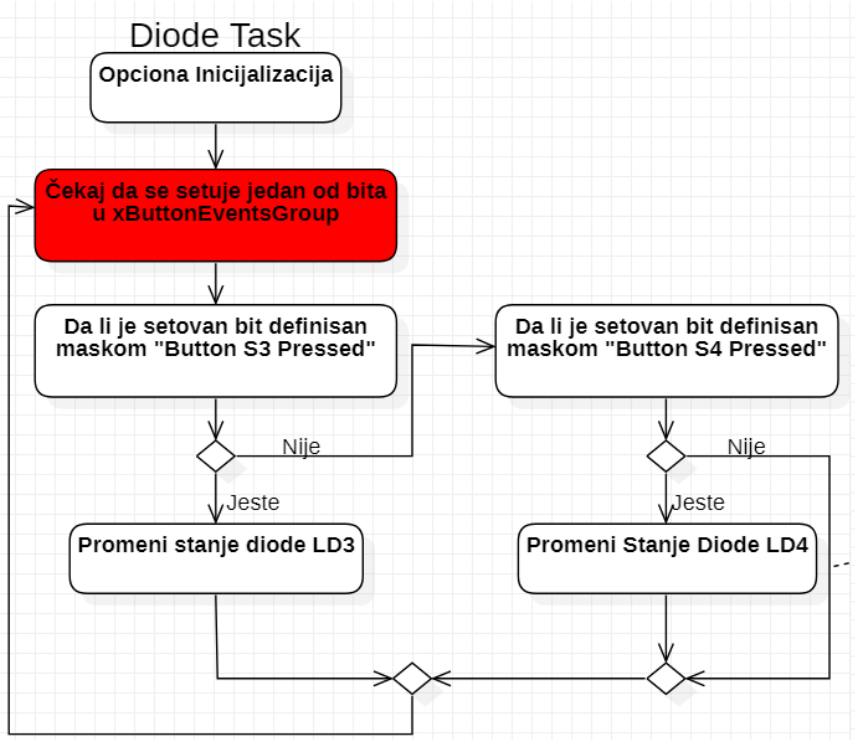
➢ ZADATAK 22 - Rešenje(Projektovanje)

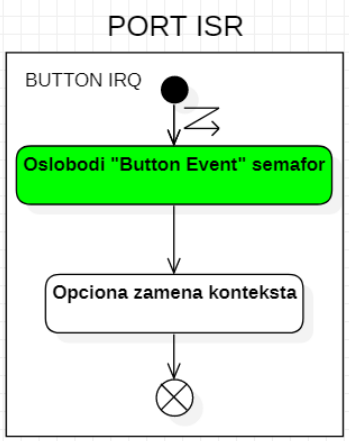
- Predložiti realizaciju softvera koja ne bi koristila grupu događaja. Nacrtati blok dijagram takve realizacije softvera.
  - U čemu je razlika takve realizacije u odnosu na realizaciju u kojoj koristimo grupu događaja?

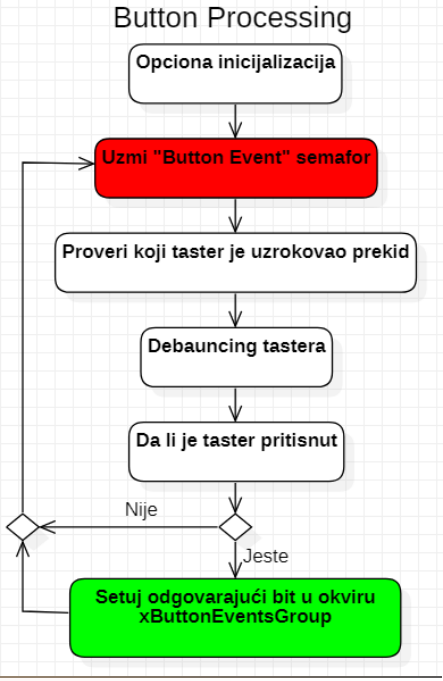
API f-je

- ZADATAK 22 – Rešenje (Projektovanje)









- ZADATAK 22 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_15***

FreeRTOS

Mehanizam notifikacije

SIGNALIZACIJA DOGAĐAJA KORIŠĆENJEM GRUPE DOGAĐAJA

API FUNKCIJE

PRIMERI

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

Osobine

- Sve metode koje se koriste za signalizaciju pojave događaja, kao i metode za komunikaciju, koje smo do sada koristili, podrazumevaju kreiranje zasebnih objekata (*queue*, semafori, grupa događaja)
  - Nekada je ovakav pristup suviše komplikovan
  - Nekada je ovakav pristup suviše spor
  - Postoji veliki *overhead* za vrlo jednostavne stvari
    - Na primer, da bi signalizirali pojavu događaja moramo koristiti ili grupu događaja ili semafor. Svaki od ovih mehanizama podrazumeva kreiranje novog objekta koji zauzima relativno veliki deo memorije koja je veoma vredan resurs u embedded sistemima.
    - Suštinski nam treba informacija da li se događaj desio ili ne(binarno 0 ili 1 – dovoljan 1bit). Kada tu informaciju enkapsuliramo u binarni semafor ili grupu događaja dobijamo značajno veće zauzeće memorije u odnosu na objektivno potrebnu količinu memorije potrebnu da se ta informacija prenese
    - Međutim, do sada smo morali da koristimo ove mehanizme da bi omogućili realizaciju blokiranja taskova, zamene konteksa i ostalih procesa karakterističnih za RTOS
    - Postavlja se ključno pitanje da li je na neki način moguće rešiti ovaj problem „neefikasnosti“ a da sa druge strane naša realizacija ostane u okvirima RTOS?
- Mehanizam notifikacije taska u FreeRTOS-u omogućava direktnu signalizaciju tasku da se desio neki događaj, ili direktnu komunikaciju sa taskom, bez upotrebe dodatnih objekata kernela (queue-ova, semafora, grupe događaja, ...)

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

Osobine

- Svaki kreirani task u FreeRTOS-u ima:
  - Notifikacionu vrednost (*Notification value*)
    - 32bitni neoznačeni broj
  - Notifikacioni status(*Notification State*)
    - Može imati vrednosti „Pending“ i „Not-Pending“
    - Kada task **primi notifikaciju** notifikacioni status uzima vrednost „**Pending**“. Kada task **pročita notifikacionu vrednost**, notifikaciono stanje uzima vrednost „**Not-Pending**“.
- Prednosti korišćenja mehanizma notifikacije
  - Signalizacija pojave događaja ili slanje podatka tasku je značajno brže od korišćenja mehanizma semafora, queue-a ili grupe događaja
  - Zauzeće memorije je manje u odnosu da mehanizam signalizacije ili komunikacije sa taskom realizujemo koristeći semafor, queue, ili grupu događaja
- Korišćenje mehanizma notifikacije **nije uvek moguće** zbog određenih **ograničenja** kojih **moramo biti svesni** kada pišemo softver baziran na FreeRTOS-u
  - Nije moguće slanje podatka iz taska u ISR
    - ... što je na primer moguće kada koristimo queue
  - Može postojati samo jedan task koji prima notifikaciju
    - Na primer, kada koristimo grupu događaja možemo imati više taskova koji primaju neku signalizaciju
  - Ne postoji mogućnost baferisanja podataka
    - ... što je na primer moguće kada koristimo queue

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

Osobine

- Svaki kreirani task u FreeRTOS-u ima:
  - Notifikacionu vrednost (*Notification value*)
    - 32bitni neoznačeni broj
  - Notifikacioni status(*Notification State*)
    - Može imati vrednosti „Pending“ i „Not-Pending“
    - Kada task **primi notifikaciju** notifikacioni status uzima vrednost „**Pending**“. Kada task **pročita notifikacionu vrednost**, notifikaciono stanje uzima vrednost „**Not-Pending**“.
- Prednosti korišćenja mehanizma notifikacije
  - Signalizacija pojave događaja ili slanje podatka tasku je značajno brže od korišćenja mehanizma semafora, queue-a ili grupe događaja
  - Zauzeće memorije je manje u odnosu da mehanizam signalizacije ili komunikacije sa taskom realizujemo koristeći semafor, queue, ili grupu događaja
- Korišćenje mehanizma notifikacije **nije uvek moguće** zbog određenih **ograničenja** kojih **moramo biti svesni** kada pišemo softver baziran na FreeRTOS-u
  - Ne postoji mogućnost implementacije *Broadcast*-a
    - ... što je na primer moguće kada koristimo grupu događaja
  - Ne postoji mogućnost blokiranja ako prethodno poslata notifikaciona vrednost nije obrađena
    - Na primer, kada upisujemo u pun *queue* task koji upisuje se blokira dok se ne oslobodi jedno mesto u *queue*-u
- Sva navedena ograničenja ne moraju nužno biti loša ali je bitno da ih uzmemo u obzir pri projektovanju softvera baziranog na FreeRTOS-u

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

API

- Kada se task kreira notifikaciona vrednost je inicijalizovana na nulu
- **Signaliziranje pojave događaja** korišćenjem mehanizma notifikacije se realizuje upotrebom `xTaskNotifyGive` i `xTaskNotifyTake` FreeRTOS API f-ja

```
BaseType_t xTaskNotifyGive( TaskHandle_t xTaskToNotify );
```

Naziv parametra	Opis
`xTaskToNotify`	Instanca prethodno kreiranog taska koja se može dohvatiti kada se task kreira koristeći ***xTaskCreate*** f-ju

- **Inkrementira** notifikacionu vrednost taska koji čeka notifikaciju i menja njegov notifikacioni status u „Pending“ (ukoliko već nije u stanju „Pending“). U slučaju da se poziva iz konkteksta prekidne rutine koristiti *xTaskNotifyGiveFromISR*
- Uvek vraća pdPASS

```
uint32_t ulTaskNotifyTake( BaseType_t xClearCountOnExit, TickType_t xTicksToWait );
```

Naziv parametra	Opis
`xClearCountOnExit`	Ovaj parametar može imati vrednosti **pdTRUE** ili **pdFALSE**. Ukoliko parametar ima vrednost *pdTRUE* onda će se izvršiti setovanje notifikacione vrednosti na 0 pre izlaska iz f-je. Ukoliko parametar ima vrednost *pdFALSE* i ako je notifikaciona vrednost veća od 0, onda se pri izlasku iz f-je **dekrementira** notifikaciona vrednost
`xTicksToWait`	Broj sistemski tikova koliko će task provesti u blokiranom stanju ćekajući da notifikaciona vrednost postane različita od 0

- Vraća notifikacionu vrednost pre setovanja na 0 (u slučaju da je *xClearOnExit* setovan na *pdTRUE*) ili dekrementiranja (u slučaju da je *xClearOnExit* setovan na *pdFALSE*)

API

- Dakle kombinacija *xTaskNotifyGive* i *xTaskNotifyTake* API f-ja pruža istu funkcionalnost kao mehanizam semafora
- Kombinacija API f-ja koje pružaju mogućnost fleksibilnijeg korišćenja mehanizma notifikacije su *xTaskNotify* i *xTaskNotifyWait*
  - Za razliku od kombinacije *xTaskNotifyGive* i *xTaskNotifyTake* koja eksplicitno ne menja notifikacionu vrednost, kombinacija *xTaskNotify* i *xTaskNotifyWait* nudi tu mogućnost

```
BaseType_t xTaskNotify( TaskHandle_t xTaskToNotify,
                        uint32_t ulValue,
                        eNotifyAction eAction );
```

Naziv parametra	Opis
`xTaskToNotify`	Instanca taska koji želimo da notifikujemo
`ulValue`	Način na koji se ova vrednost koristi zavisi od ***eAction*** parametra
`eAction`	Uzima vrednost iz predefinisanog skupa vrednosti: **eNoAction** – Vrši se isključivo notifikacija taska bez modifikacije notifikacione vrednosti. ulValue se ignoriše **eSetBits** – Parametar ulValue predstavlja masku koja definiše koji biti će se setovati. Ova funkcionalnost nudi mogućnost implementacije jednostavnije grupe događaja **eIncrement** – Notifikaciona vrednost se inkrementira za 1. **ulValue** se ignoriše **eSetValueWithOverwrite** – Notifikaciona vrednost uzima vrednost definisanu parametrom **ulValue**. Ne posmatra se notifikacioni status.

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

API

- Dakle kombinacija *xTaskNotifyGive* i *xTaskNotifyTake* API f-ja pruža istu funkcionalnost kao mehanizam semafora
- Kombinacija API f-ja koje pružaju mogućnost fleksibilnijeg korišćenja mehanizma notifikacije su ***xTaskNotify*** i ***xTaskNotifyWait***
  - Za razliku od kombinacije *xTaskNotifyGive* i *xTaskNotifyTake* koja eksplicitno ne menja notifikacionu vrednost, kombinacija *xTaskNotify* i *xTaskNotifyWait* nudi tu mogućnost

```
BaseType_t xTaskNotify( TaskHandle_t xTaskToNotify,
                        uint32_t ulValue,
                        eNotifyAction eAction );
```

Naziv parametra	Opis
`xTaskToNotify`	Instanca taska koji želimo da notifikujemo
`ulValue`	Način na koji se ova vrednost koristi zavisi od ***eAction*** parametra
`eAction`	Uzima vrednost iz predefinisanog skupa vrednosti: **eSetValueWithoutOverwrite** – Notifikaciona vrednost uzima vrednost definisanu parametrom **ulValue** samo ukoliko je notifikacioni status „Not-Pending“. Ukoliko je notifikacioni status „Pending“ dodela notifikacioni vrednosti nije moguća i f-ja vraća ***pdFALSE***

- Primetiti da f-ja isključivo u jednom slučaju vraća *pdFALSE* u svim ostalima *pdTRUE*.
- Primetiti da je f-ja *xTaskNotifyGive* samo jedan specijalan slučaj f-je *xTaskNotify*. Pronaći u FreeRTOS *source* kodu kako je f-ja *xTaskNotifyGive* realizovana!
- U slučaju da se f-ja poziva iz konteksta prekidne rutine koristiti ***xTaskNotifyFromISR*** .

FreeRTOS MEHANIZAM NOTIFIKACIJE TASKA

API

- Dakle kombinacija *xTaskNotifyGive* i *xTaskNotifyTake* API f-ja pruža istu funkcionalnost kao mehanizam semafora
- Kombinacija API f-ja koje pružaju mogućnost fleksibilnijeg korišćenja mehanizma notifikacije su ***xTaskNotify*** i ***xTaskNotifyWait***
  - Za razliku od kombinacije *xTaskNotifyGive* i *xTaskNotifyTake* koja eksplicitno ne menja notifikacionu vrednost, kombinacija *xTaskNotify* i *xTaskNotifyWait* nudi tu mogućnost

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry,
                            uint32_t ulBitsToClearOnExit,
                            uint32_t *pulNotificationValue,
                            TickType_t xTicksToWait );
```

Naziv parametra	Opis
`ulBitsToClearOnEntry`	Maska koja definiše koje bite notifikacione vrednosti ćemo brisati na ulazu u funkciju. Može se koristiti makro ULONG_MAX za masku 0xFFFFFFFF
`ulBitsToClearOnExit`	Maska koja definiše koje bite notifikacione vrednosti ćemo brisati na izlasku iz funkcije. Može se koristiti makro ULONG_MAX za masku 0xFFFFFFFF
`pulNotificationValue`	Notifikaciona vrednost pre brisanja. Ukoliko ne koristimo možemo proslediti NULL
`xTicksToWait`	Broj sistemski tikova koliko će task provesti u blokiranom stanju čekajući da notifikaciona vrednost postane različita od 0

- Da bi korišćenje mehanizma notifikacije u FreeRTOS-u bilo moguće neophodno je setovati makro ***configUSE_TASK_NOTIFICATIONS*** na vrednost 1

Primeri

➢ ZADATAK 23

Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Pritiskom na taster S3 očitava se vrednost sa AD konvertora i prikazuje na dvocifrenom sedmosegmentnom displeju. Pritiskom na taster S4 selektuje se jedan od dva kanala na koje su povezani potenciometri PT1 i PT2. Dioda ispod potenciometra na razvojnoj ploči (plava pločica) pokazuje koji potenciometar je trenutno aktivan.

Napomena: Rezolucija AD konvertora je 12bita, a na eksternom *schield*-u imamo dostupna samo 2 LE sedmosegmentna displeja, pa je potrebno izvršiti skaliranje.

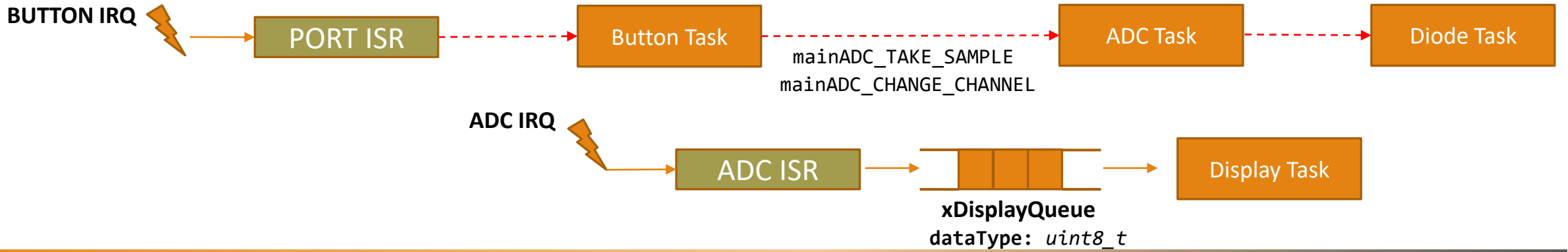
Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, implementirane algoritme, objekte kernela koji se koriste u softveru i njihovu međusobnu zavisnost. Pored toga, treba priložiti i kod realizovanog rešenja.

Primeri

➢ ZADATAK 23 – Rešenje (Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine

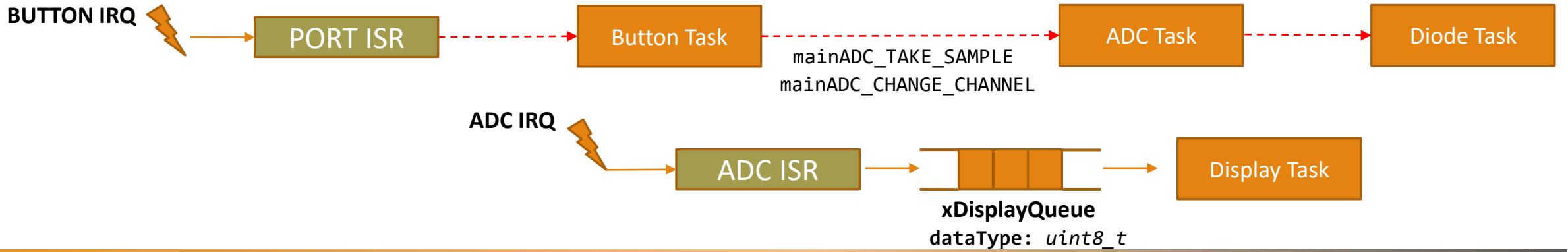
1. **Detekcija pritiska tastera**
   - Ovu logičku celinu implementiraćemo koristeći prekidnu rutinu porta i „Button Task“ task. U okviru prekidne rutine porta detektovaćemo da se desio potencijalni pritisak tastera dok ćemo odloženu obradu nastaviti u okviru „Button task“ taska.
2. **Promena aktivne diode**
   - Implementiraćemo u okviru „Diode task“ taska
3. **Upravljanje AD konvertorom**
   - Start konverzije i promenu aktivnog kanala implementiraćemo u okviru „ADC task“ taska.
4. **Ispis na sedmosegmentni displej**
   - Mehanizam multipleksiranja dvocifrenog sedmosegmentnog LE displeja implementiraćemo u okviru „Display task“ taska.



Primeri

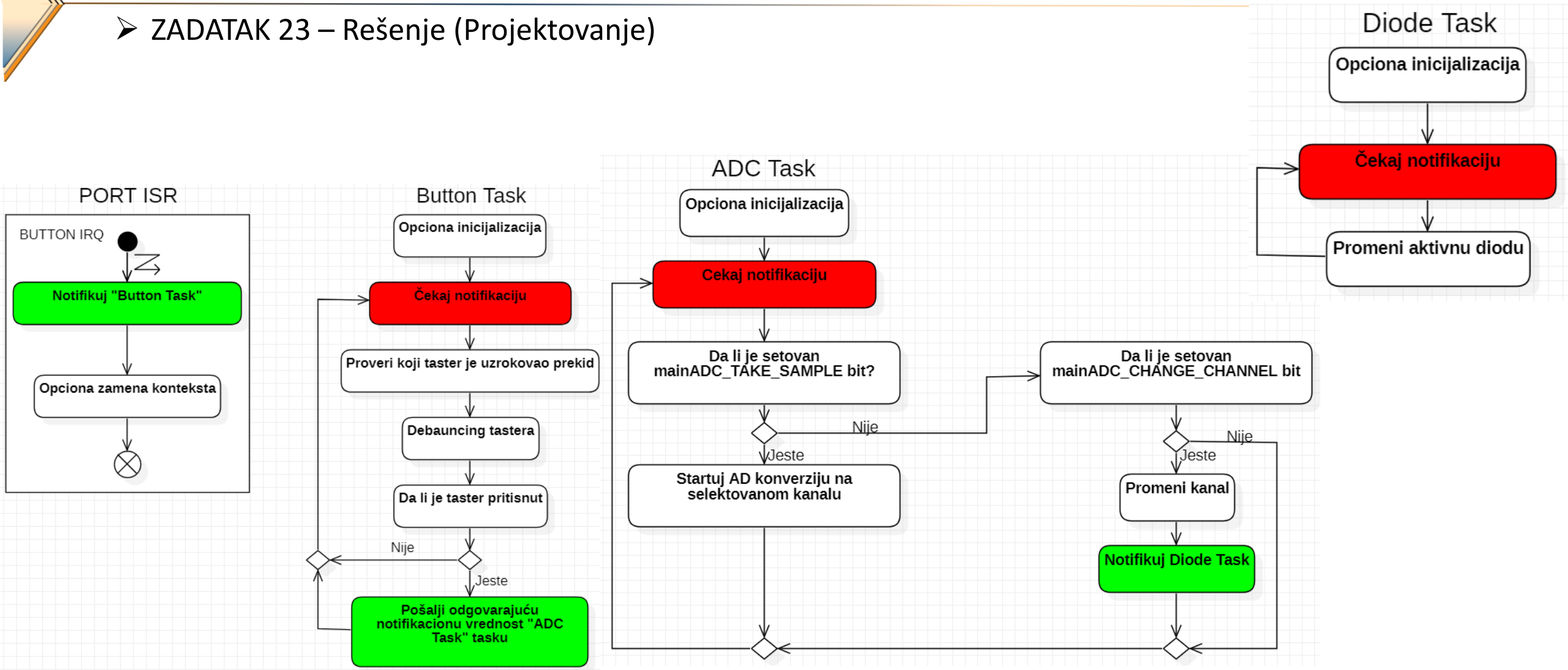
➢ ZADATAK 23 – Rešenje (Projektovanje)

- Prekidna rutina PORT ISR se poziva kao posledica pritiska na jedan od dva tastera S3 ili S4.
- Iz prekidne rutine PORT ISR se notifikuje „Button task“ task u okviru koga nastavljamo **odloženu obradu prekida**. Ovaj task ima zadatak, kao i u ostalim primerima, da **detektuje koji od tastera je pritisnut** i da shodno rezultatu provere pošalje odgovarajuću notifikaciju „ADC task“ tasku. Notifikacija se šalje metodom ***eSetBits*** koja je po funkcionalnosti slična grupi događaja. Ukoliko je „Button task“ detektovao da je pritisnut taster S3, maskom ***mainADC_TAKE_SAMPLE*** se setuje odgovarajući bit notifikacione vrednosti u „ADC task“ tasku. Ukoliko je pritisnut taster S4, maskom ***mainADC_CHANGE_CHANNEL*** se setuje odgovarajući bit notifikacione vrednosti u „ADC task“ tasku.
- „ADC Task“ na osnovu **vrednosti bita u notifikacionoj promenljivoj** vrši startovanje AD konverzije ili promenu aktivnog kanala. Ukoliko se vrši promena aktivnog kanala notifikuje se „Diode task“ kako bi izvršio promenu aktivne diode
- „Diode Task“ čeka notifikaciju od „ADC Task“ taska nakon čega menja aktivnu diodu.



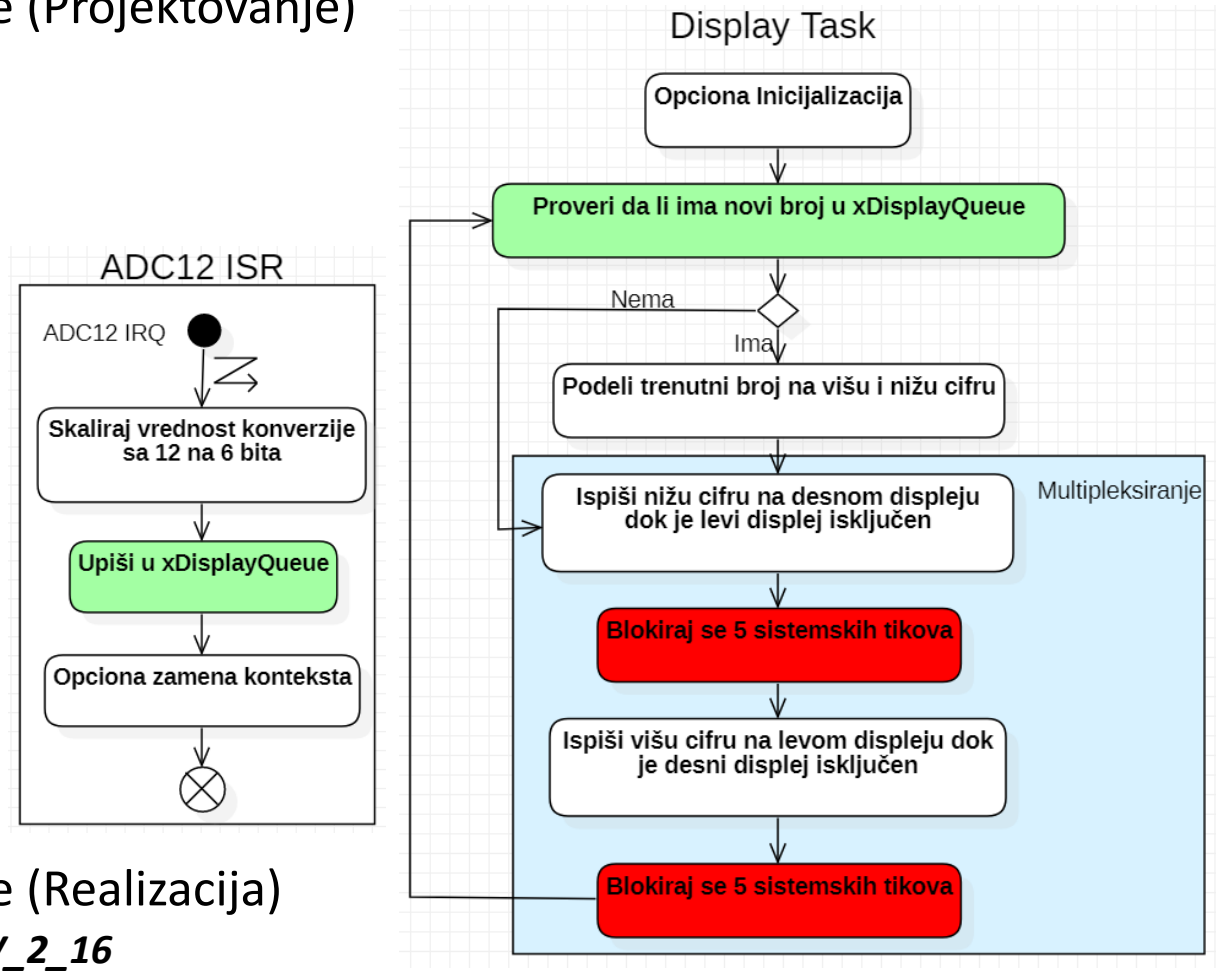
Primeri

- ZADATAK 23 – Rešenje (Projektovanje)



Primeri

- ZADATAK 23 – Rešenje (Projektovanje)



- ZADATAK 23 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_16***

FreeRTOS

Pregled korišćenih simbola

PREGLED KORIŠĆENIH SIMBOLA...