FreeRTOS

KORISĆENJE PREKIDNIH RUTINA

KAKO SE KORISTE PREKIDI ZAJEDNO SA FREERTOSOM

ZAMENA KONTEKSTA U OKVIRU PREKIDNE RUTINE

PRIMERI

FreeRTOS KORIŠĆENJE PREKIDNIH RUTINA

- Mehanizam prekida je u potpunosti isti bez obzira da li ste koristili neki RTOS ili pisali ***Bare-Metal*** softver
- Kada **se FreeRTOS koristi zajedno sa prekidima** postoje određene **specifičnosti** koje moramo uzeti u obzir prilikom projektovanja softvera baziranog na FreeRTOS-u
  - U okviru **prekidne rutine** obično postoji potreba za korišćenjem nekih funkcionalnosti FreeRTOS-a
    - Na primer želimo da oslobodimo neki semafor i da na taj način signaliziramo da se desio događaj u sistemu
  - Šta bi se desilo ukoliko postoji potreba da se iz ISR poziva neka **API funkcija** FreeRTOS-a koja, ukoli se poziva iz taska, podrazumeva da task može otići u stanje „Block“?
    - Potrebno je napomenuti da nije moguće blokiranje ISR jer je to svojstvo taskova (softverskog dela)
- U okviru FreeRTOS-a postoje dve vrste API funkcija
  - One koje se pozivaju iz **konkteksta taska**
  - One koje se pozivaju iz **konkteksta prekidne rutine** (*Interrupt safe API functions* – **sufiks *FromISR***)
    - Kako bi povećali vremensku efikanost funkcija koje se pozivaju iz konteksta taska, a za koje postoji potreba da se pozivaju i iz prekidne rutine, uvedene su posebne funkcije
      - Na primer, ukoliko se neka API funkcija poziva iz taska onda se primenjuje jedna logika dok ukoliko se poziva iz prekidne rutine primenjuje se druga logika
      - Da funkcije nisu razdvojene postojao bi veliki ***overhed*** kada se neka API funkcija poziva iz prekidne rutine što nije dobro jer **ISR treba da bude što kraća i efikasnija**
    - API funkcije FreeRTOSa koje se mogu koristiti iz konteksta prekidne rutine imaju isti naziv kao API funkcije koje se mogu koristiti iz konteksta taska ali je na kraju naziva funkcije dodat sufiks **FromISR**
    - Kako budemo uvodili pojedine FreeRTOS API funkcije koje se pozivaju iz konteksta taska tako ćemo i uvoditi **ekvivalentne** FreeRTOS API funkcije koje se mogu pozivati i iz konteksta prekidne rutine

API funkcije za rad sa semaforima pozivaju se iz konteksta prekidne rutine

- Do sada smo se upoznali se xSemaphoreTake i xSemaphoreGive FreeRTOS API funkcijama
- Njihovi ekvivalenti, ukoliko se poziv vrši iz konteksta prekidne rutine, su :

```
xSemaphoreGiveFromISR( xSemaphore, pxHigherPriorityTaskWoken )
```

Naziv parametra	Opis
xSemaphore	Instanca prethodno kreiranog semafora koji se zauzima
pxPriorityTaskWoken	?

```
xSemaphoreTakeFromISR( xSemaphore, pxHigherPriorityTaskWoken )
```

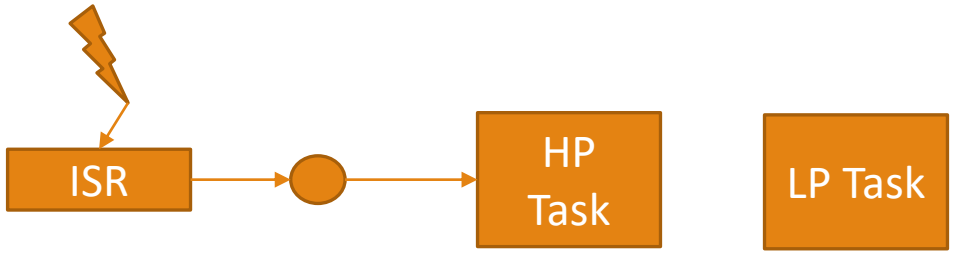
Naziv parametra	Opis
xSemaphore	Instanca prethodno kreiranog semafora koji se zauzima
pxPriorityTaskWoken	?

Šta označava ovaj drugi parametar?

FreeRTOS KORIŠĆENJE PREKIDNIH RUTINA

Zamena konteksta – `pxPriorityTaskWoken` parametar

- Već nam je poznato da neke FreeRTOS API funkcije mogu promeniti stanje taska iz „Blocked“ u stanje „Ready“
  - Na primer `xSemaphoreGive`
- Postavlja se pitanje **kada** FreeRTOS vrši zamenu konteksta u situaciji gde se iz prekidne rutine poziva API f-ja FreeRTOS-a koja dovodi do promene stanja taska iz „Blocked“ u „Ready“ gde taj task ima veći prioritet od taska čije je izvršavanje obustavljeno generisanjem prekida???
  - Na primer „HP Task“ task (task većeg prioriteta) čeka na binarni semafor koji treba da se oslobodi iz prekidne rutine. Na sistemu postoji još i „LP Task“ task (task manjeg prioriteta) koji se konstantno izvršava



Zamena konteksta - `pxPriorityTaskWoken` parametar

- U okviru konteksta prekida se ne vrši automatska zamena konteksta već se prosleđuje informacija o tome da li je potrebno izvršiti zamenu konteksta
- U okviru *FreeRTOS Interrupt safe API* funkcija (one koje se završavaju sa *FromISR*) uvek postoji **opcioni** parametar `pxPriorityTaskWoken` koji se setuje na vrednost pdTRUE ukoliko postoji potreba da se izvrši zamena konteksta do se u suprotnom vrednost setuje na pdFALSE
  - Dakle, u prekidnoj rutini se desilo da task većeg prioriteta, u odnosu na task prekinut usled generisanja prekida, pređe iz stanja „Blocked“ u stanje „Ready“
  - Uočiti sa prethodnih slajdova da ovaj parametar zapravo predstavlja pokazivač i njegovu vrednost zapravo setuje FreeRTOS API funkcija
- Ako želimo da *FreeRTOS Interrupt safe API* funkcija modifikuje vrednost ovog parametra u zavisnosti od toga da li treba izvršiti zamenu konteksta ili ne, pre poziva API funkcije u okviru prekidne rutine vrši se inicijalizacija ovog parametra na pdFALSE.
  - Ako je vrednost ovog parametra nakon poziva API funkcije setovana na pdTRUE i ako želimo da se u okviru prekidne rutine izvrši zamena konteksta onda se **na kraju prekidne rutine** poziva `portYIELD_FROM_ISR(parameter);`
    - Ako je vrednost parameter setovana na **pdTRUE vrši** se zamena konteksta.
    - Ako je vrednost parameter setovana na **pdFALSE ne vrši** se zamena konteksta.
- **Žašto je uopšte bitno da postoji ovakav pristup???**
  - Odložena obrada prekida .... !?

Zamena konteksta - `pxPriorityTaskWoken` parametar

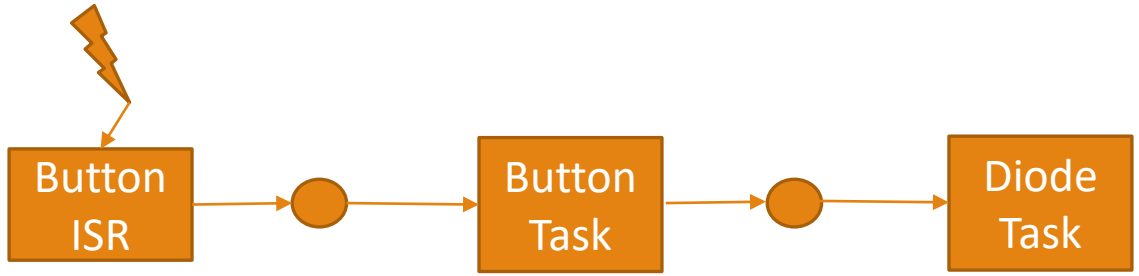
- U okviru konteksta prekida se ne vrši automatska zamena konteksta već se prosleđuje informacija o tome da li je potrebno izvršiti zamenu konteksta
- U okviru *FreeRTOS Interrupt safe API* funkcija (one koje se završavaju sa *FromISR*) uvek postoji **opcioni** parametar `pxPriorityTaskWoken` koji se setuje na vrednost `pdTRUE` ukoliko postoji potreba da se izvrši zamena konteksta do se u suprotnom vrednost setuje na `pdFALSE`
  - Dakle, u prekidnoj rutini se desilo da task većeg prioriteta, u odnosu na task prekinut usled generisanja prekida, pređe iz stanja „Blocked“ u stanje „Ready“
  - Uočiti sa prethodnih slajdova da ovaj parametar zapravo predstavlja pokazivač i njegovu vrednost zapravo setuje FreeRTOS API funkcija
- Ako nam vrednost `pxPriorityTaskWoken` parametar nije bitan, onda se prosleđuje vrednost `NULL`

➢ ZADATAK 9

Modifikovati zadatak 6 tako da se detekcija pritiska tastera, umesto u tasku, implementira u prekidnoj rutini

➢ Zadatak 9 – Rešenje (Projektovanje)

- Za razliku od zadatka 6, gde smo u okviru taska implementirali i detekciju pritiska tastera i algoritme za debaunsiranje (obradu pritiska tastera), u ovom primeru ćemo detekciju pritiska tastera izmestiti u prekidnu rutinu dok će algoritmi za dodatnu obradu (debaunsiranje) ostati u tasku.
  - Jedan od jako bitnih zahteva pri projektovanju prekidnih rutina jeste da one moraju biti što kraće po pitanju vremena izvršavanja

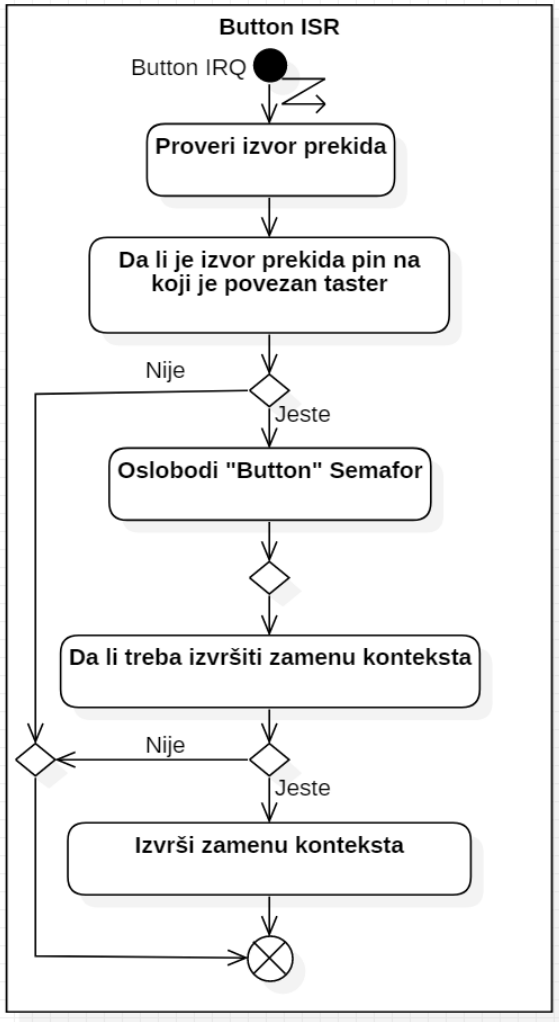


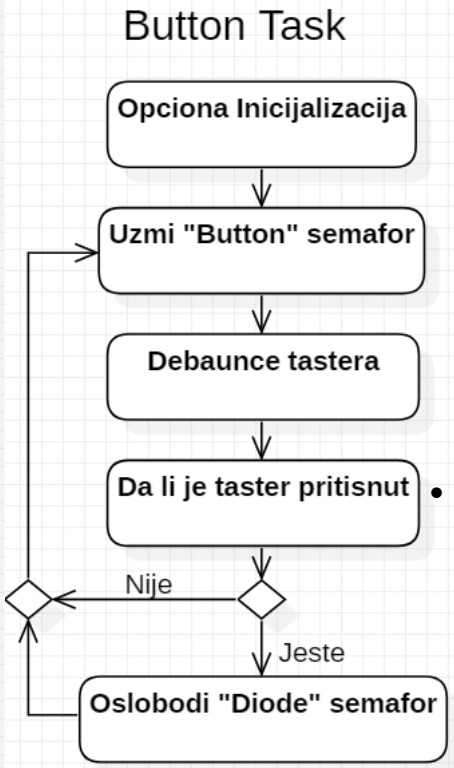
- Kako izgleda raspodela prioriteta po taskovima?

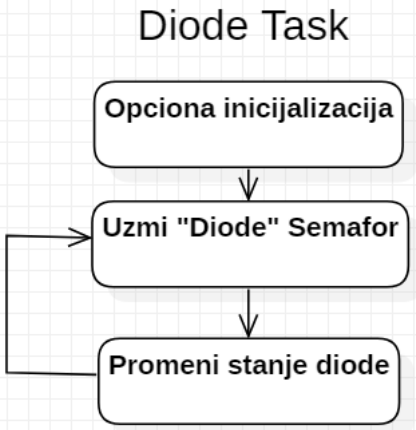
FreeRTOS KORIŠĆENJE PREKIDNIH RUTINA

PRIMERI

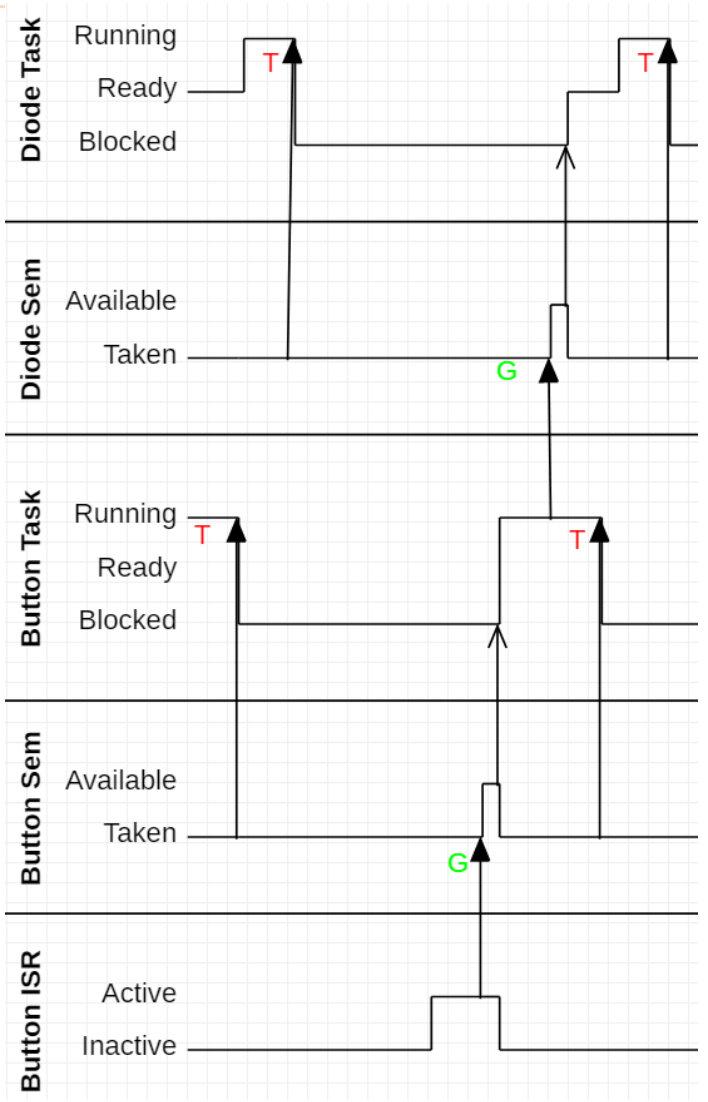
- Zadatak 9 – Rešenje (Projektovanje)







- Za razliku od zadatka 7, gde smo u okviru taska implementirali i detekciju pritiska tastera i algoritme za debaunsiranje (obradu pritiska tastera), u ovom primeru ćemo detekciju pritiska tastera izmestiti u prekidnu rutinu dok će algoritmi za dodatnu obradu (debaunsiranje) ostati u tasku.



- Zadatak 9 – Rešenje (Realizacija)
  - Pogledati kôd ***SRV_2_5*** projekta
- **Zadatak 10** (*Za samostalni rad*)

  Modifikovati zadatak 8 tako da se detekcija pritiska tastera, umesto u tasku, implementira u prekidnoj rutini

RTOS

KOMUNIKACIJA I SIGNALIZACIJA

KOMUNIKACIJA KORIŠĆENJEM DELJENE MEMORIJE

MUTEX SEMAFORI

KORIŠĆENJE MUTEX SEMAFORA U FREERTOS-U

Koncept deljenje memorije

- Pod deljenom memorijom podrazumevamo deo adresnog prostora platforme kome se pristupa iz dva ili više taskova
  - Globalna promenljiva
  - Blok u memoriji
  - Periferija
  - ...
- Deljena memorija se može koristiti kao jedan vid komunikacije među taskovima u sistemu
  - Ovakav pristup podrazumeva da imamo task (*Producer*) koji sa jedne strane **proizvodi podatke** i jedan task (*Consumer*) koji sa druge strane **koristi te podatke**

- Operacija **upisa** podataka skoro **nikada nije atomična** operacija
  - Na primer, čak i jednostavna operacija upisa u memoriju nije atomična

producer.c

```
globalVariable = counter;
```

producer.asm

```
012882:  415F 0004        MOV.B   0x0004(SP),R15
012886:  4F82 38F8        MOV.W   R15,&globalVariable
```

Šta se dešava u slučaju da se operacija upisa u ovom delu **prekida** od strane schedulera koji ***Consumer*** **tasku dodeljuje procesorsko vreme**???

Koncept deljene memorije

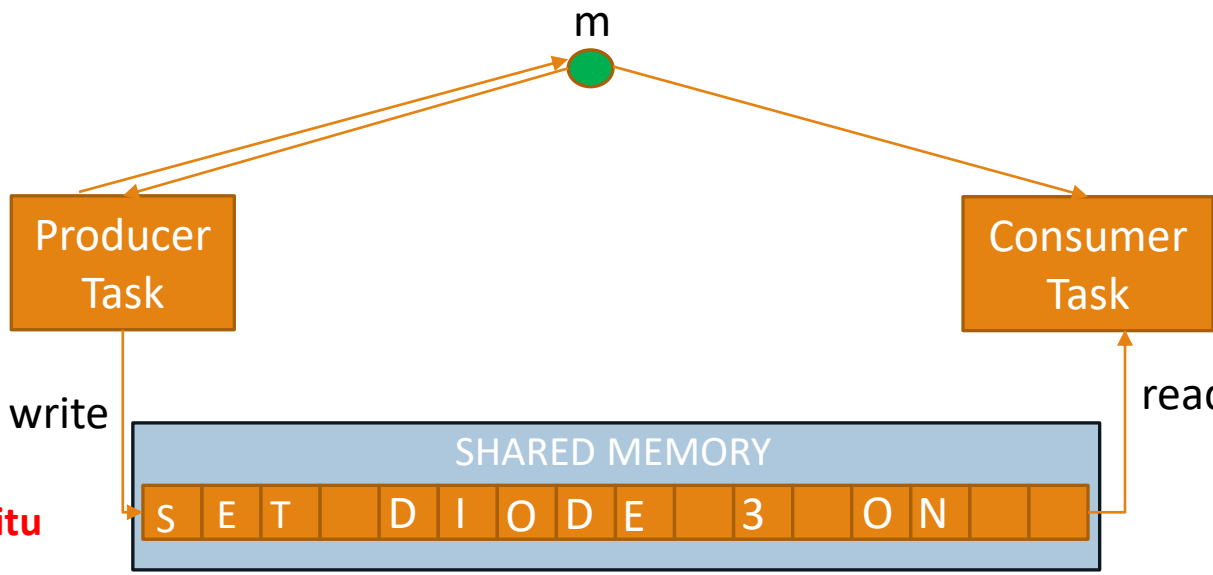
- Pod deljenom memorijom podrazumevamo deo adresnog prostora platforme kome se pristupa iz dva ili više taskova
  - Globalna promenljiva
  - Blok u memoriji
  - Periferija
  - ...
- Deljena memorija se može koristiti kao jedan vid komunikacije među taskovima u sistemu
  - Ovakav pristup podrazumeva da imamo task (*Producer*) koji sa jedne strane **proizvodi podatke** i jedan task (*Consumer*) koji sa druge strane **koristi te podatke**

- Operacija **upisa** podataka skoro **nikada nije atomična** operacija
  - Na primer, čekanje kontrolne poruke preko UART interfejsa.
    - Preko UART treba da stigne poruka „SET DIODE 3 ON“ koja uključuje diodu 3.
    - Na platformi koja prima ovu poruku rezervisan je bafer dužine 15 karakter vrednosti.

Potrebno je obezbediti atomičnost!

Zaštita pristupa deljenoj memoriji

- Mutex semafori omogućavaju atomičnost operacije nad deljenom memorijom u okviru softvera baziranog na RTOS
  - Kada koristimo neki oblik deljene memorije (globalna promenljiva, blok u memoriji, periferiju, ...) treba joj dodeliti ***Mutex*** semafor koji omogućava da u jednom posmatranom trenutku **samo jedan task** ima pristup deljenoj memoriji (*Consumer* ili *Producer*)
  - Ukoliko drugi task (na primer *Consumer*) **pokuša da pristupi** deljenoj memoriji dok prvi task (na primer Producer) obavlja operaciju nad deljenom memorijom, drugi task će se **blokirati**. Drugi task će se odblokirati onda kada **prvi task**, koji je već zauzeo *mutex* semafor, **oslobodi** *mutex* semafor
    - Task koji želi da radi nad deljenom memorijom prvo pokuša da **uzme** *mutex* semafor
      - Ukoliko je semafor već zauzet task će se blokirati
    - Ukoliko je semafor slobodan, što znači da niko nad resursom ne vrši neku operaciju, task može **započeti operaciju nad resursom**
    - Kada task završi operaciju nad resursom, **vraća** semafor
      - Dolazi do odblokiranja taskova koji su se blokirali na ovom *mutex* semaforu



Da li bi umesto mutex-a mogao da se koristi binarni semafor za zaštitu pristupa deljenom resursu??

Mutex vs Binarni semafor

- Mutex je po strukturi dosta sličan binarnom semaforu
- Mutex semafori podržavaju mehanizam inverzije prioriteta
  - Sprečavaju da se task manjeg prioriteta izvrši ukoliko je task većeg prioriteta blokiran na mutexu
  - Pogledati predavanja
- Onaj ko je uzeo mutex semafor mora taj semafor i da oslobodi
  - Ovo nije slučaj sa binarnim semaforom
- Koriste se u okviru različitih mehanizama
  - Binarni semafor se koristi za signaliziranje događaja
  - Mutex se koristi kako bi se ostvarila atomičnost kritičnih sekcija u kodu
  - ...

FreeRTOS

MUTEX

API FUNKCIJE

PRIMERI

FreeRTOS MUTEX

API

- Kako bi omogućili korišćenje *Mutex* semafora u okviru *FreeRTOSConfig.h* fajla moramo podesiti sledeći makro:

```
#define configUSE_MUTEXES                1
```

- Prvi korak pri radu sa *Mutex* semaforom jeste da izvršimo kreiranje *Mutex* semafora korišćenjem sledeće API f-je:

```
SemaphoreHandle_t xSemaphoreCreateMutex( void );
```

  - Ukoliko prilikom kreiranja f-ja vrati NULL to znači da semafor nije uspešno kreiran (verovatno zbog nedostatka resursa u sistemu)
  - Ukoliko prilikom kreiranja f-ja vrednost različitu od NULL to znači da je *Mutex* semafor uspešno kreiran i da ga možemo koristiti

- F-je za zauzimanje i oslobađanje Mutex semafora su iste kao i f-je za zauzimanje i oslobađanje binarnog semafora

➢ ZADATAK 11

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

Pritisak tastera S3 uzrokuje promenu stanja diode LD3 dok pritisak tastera S4 uzrokuje promenu stanja diode LD4.

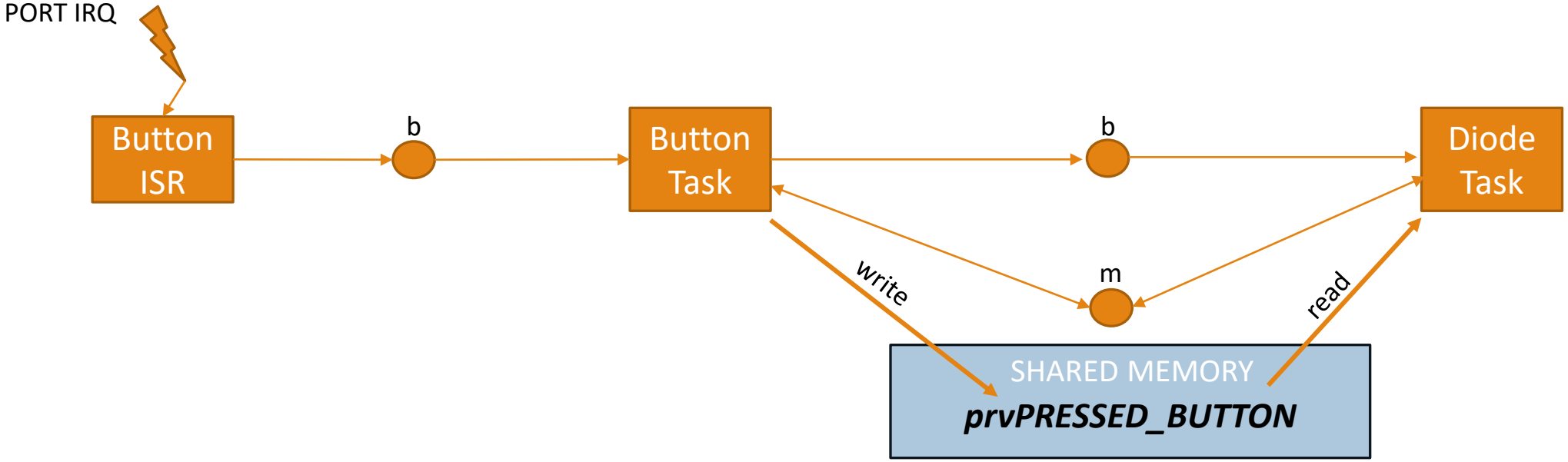
Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena ali i implementirane algoritme. Pored toga, treba priložiti i kôd realizovanog rešenja.

➢ ZADATAK 11 – Rešenje (Projektovanje)

- Ovaj zadatak je **sličan** zadatku broj 8. Međutim, cilj ovog zadatka jeste da pokaže jedan od praktičnih načina korišćenja mehanizma deljene memorije
- U okviru zadatka moguće je uočiti sledeće logičke celine:
  1. **Detekcija pritiska jednog od taster**
     - Ovaj deo ćemo implementirati u okviru prekidne rutine porta na koji je povezan taster
  2. **Odložena obrada zahteva koji dolazi od porta na koji su povezani taster**
     - Ova logičke celine bavi se detaljnijim procesiranjem izvora prekida sa ciljem da utvrdi koji taster je uzrokovao prekid i da **upiše** tu informaciju u **okviru deljene memorije** implementirane u vidu globalne promenljive ***prvPRESSED_BUTTON***
     - Ovu logičku celinu ćemo implementirati u okviru taska „Button task“
  3. **Promena stanja diode**
     - Čita sadržaj globalne promenljive ***prvPRESSED_BUTTON*** i shodno rezultatu čitanja menja stanje odgovarajuće diode
     - Ovu logičku celinu ćemo implementirati u okviru taska „Diode task“

PRIMERI

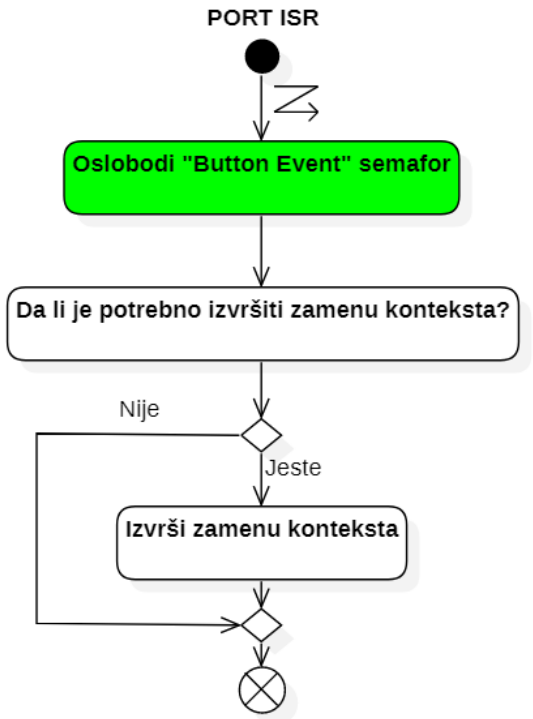
- ZADATAK 11 – Rešenje (Projektovanje)



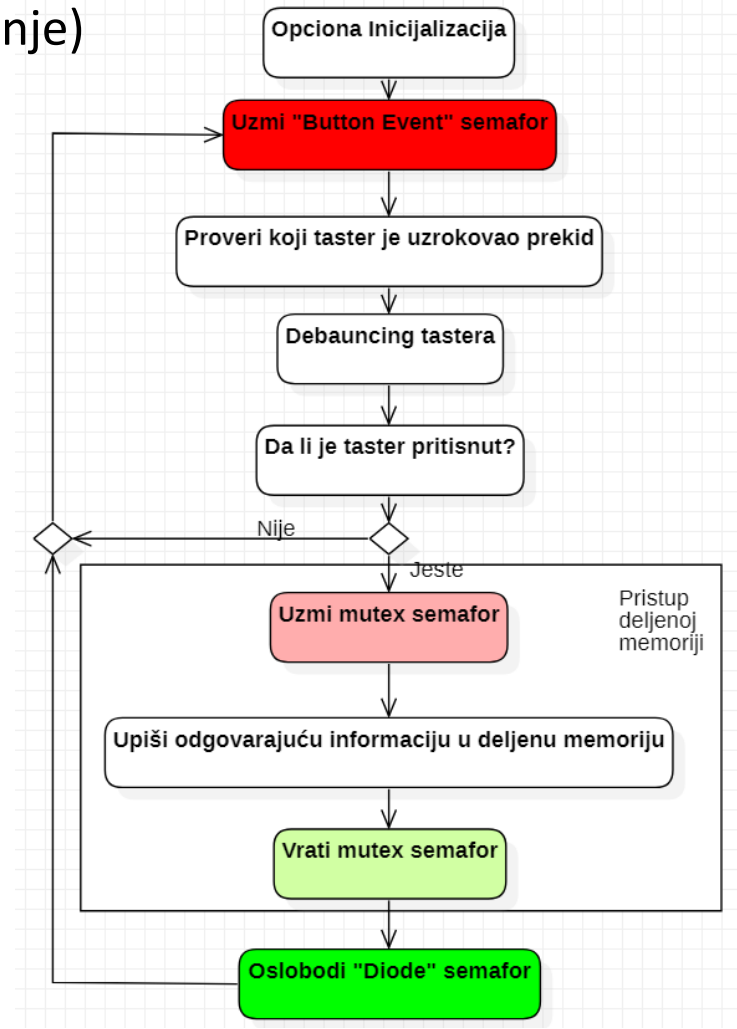
FreeRTOS MUTEX

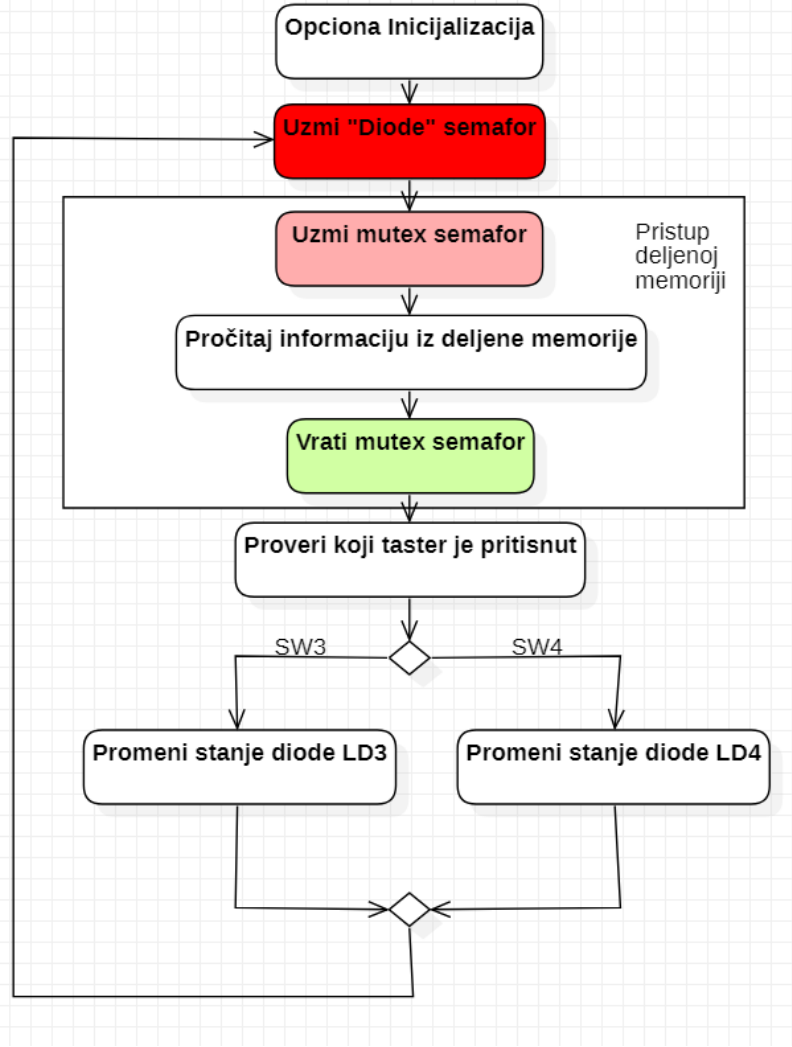
PRIMERI

- ZADATAK 11 – Rešenje (Projektovanje)



Koji task ima veći prioritet?



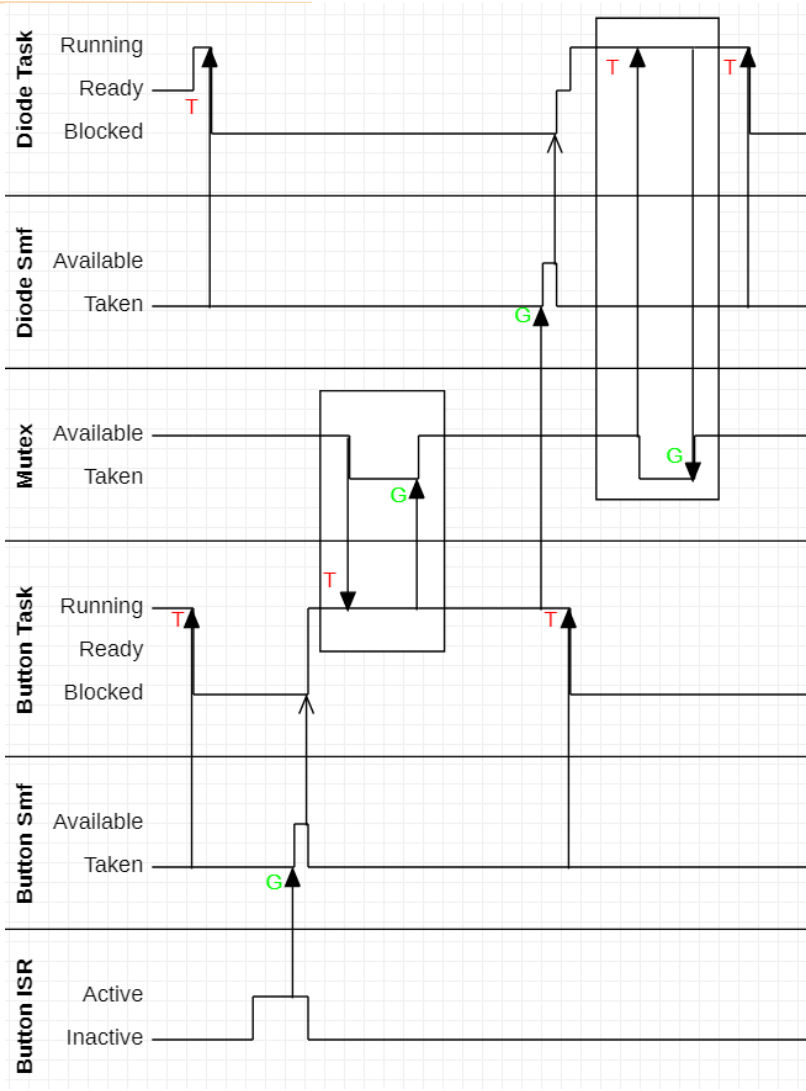


FreeRTOS MUTEX

PRIMERI

- ZADATAK 11– Rešenje (Projektovanje)

- ZADATAK 11 – Rešenje (Realizacija)
  - Videti kod projekta ***SRV_2_6***



➢ ZADATAK 12

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

Periodično (sa periodom od 200 tikova) na UART konzoli računara se ispisuje poruka „* Ovaj ispis se poziva periodično iz „Periodic Task“ taska“. Kada korisnik pritisne taster SW3 na konzoli se ispisuje string „- Korisnik pritisnuo taster“.

Sa računarom se komunicira putem UART USCIA1 periferije (pinove P4.4 i P4.5) i brzina komunikacije je ***9600bps.***

Zadatak je potrebno realizovati u dve varijante: bez ijednog mutex semafora i sa potrebnim brojem mutex semafora kako bi se zaštitio pristup zajedničkom resursu.

Za potpuno realizovano rešenje potrebno je priložiti odgovarajuće dijagrame koji što tačnije opisuju ponašanje sistema tokom vremena, dijagrame aktivnosti koji opisuju implementirane algoritme i **rezultat ispisa na konzoli za obe varijante rešenja**. Pored toga, treba priložiti i kôd realizovanog rešenja **za obe varijante**.

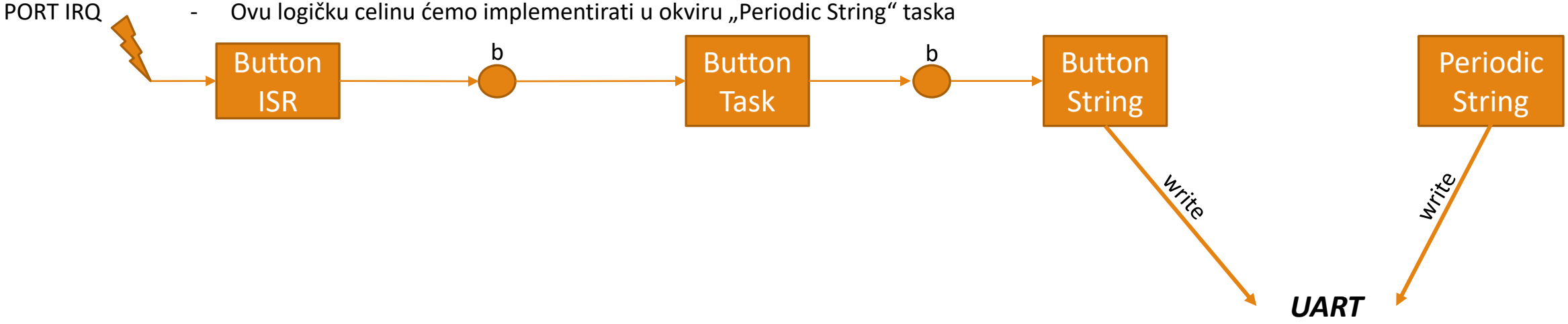
➢ ZADATAK 12 – Rešenje (Projektovanje)

- U ovom zadatku potrebno je realizovati **detekciju pritiska tastera** i **ispis dva stringa** na konzolu putem UART-a. **Ispis jednog** stringa je **periodičan** dok se **ispis drugog stringa** vrši isključivo kada korisnik **pritisne taster**.

➢ ZADATAK 12 – Rešenje (Projektovanje – Varijanta 1)

- Jedna od ideja realizacije jeste da uočimo sledeće tri logičke celine u zadatku

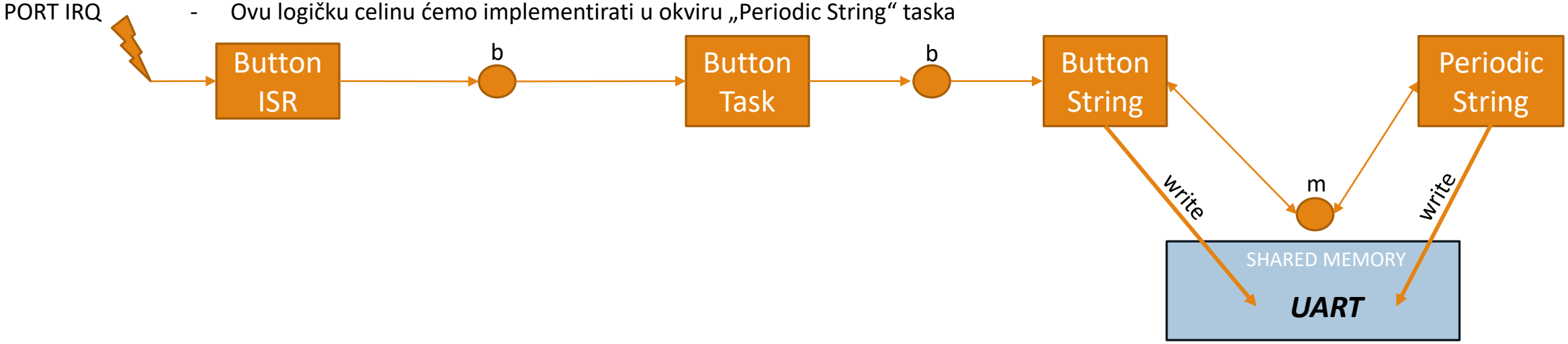
1. **Detekcija pritiska taster**
   - Ovu logičku celinu ćemo implementirati kombinujući **prekidnu rutinu** (u okviru koje ćemo detektovati potencijalni pritisak tastera) i „Button Task“ u okviru koga ćemo nastaviti dalju obradu prekidnog zahteva kako bi smo potvrdili da se pritisak tastera zaista desio.
2. **String koji se ispisuje ukoliko je taster pritisnut**
   - Ovu logičku celinu ćemo implementirati u okviru „Button String“ taska i sinhronizovaćemo ga sa „Button Task“ taskom koristeći binarni semafor
3. **String koji se periodično ispisuje**
   - Ovu logičku celinu ćemo implementirati u okviru „Periodic String“ taska



➢ ZADATAK 12 – Rešenje (Projektovanje – Varijanta 2)

- Jedna od ideja realizacije jeste da uočimo sledeće tri logičke celine u zadatku

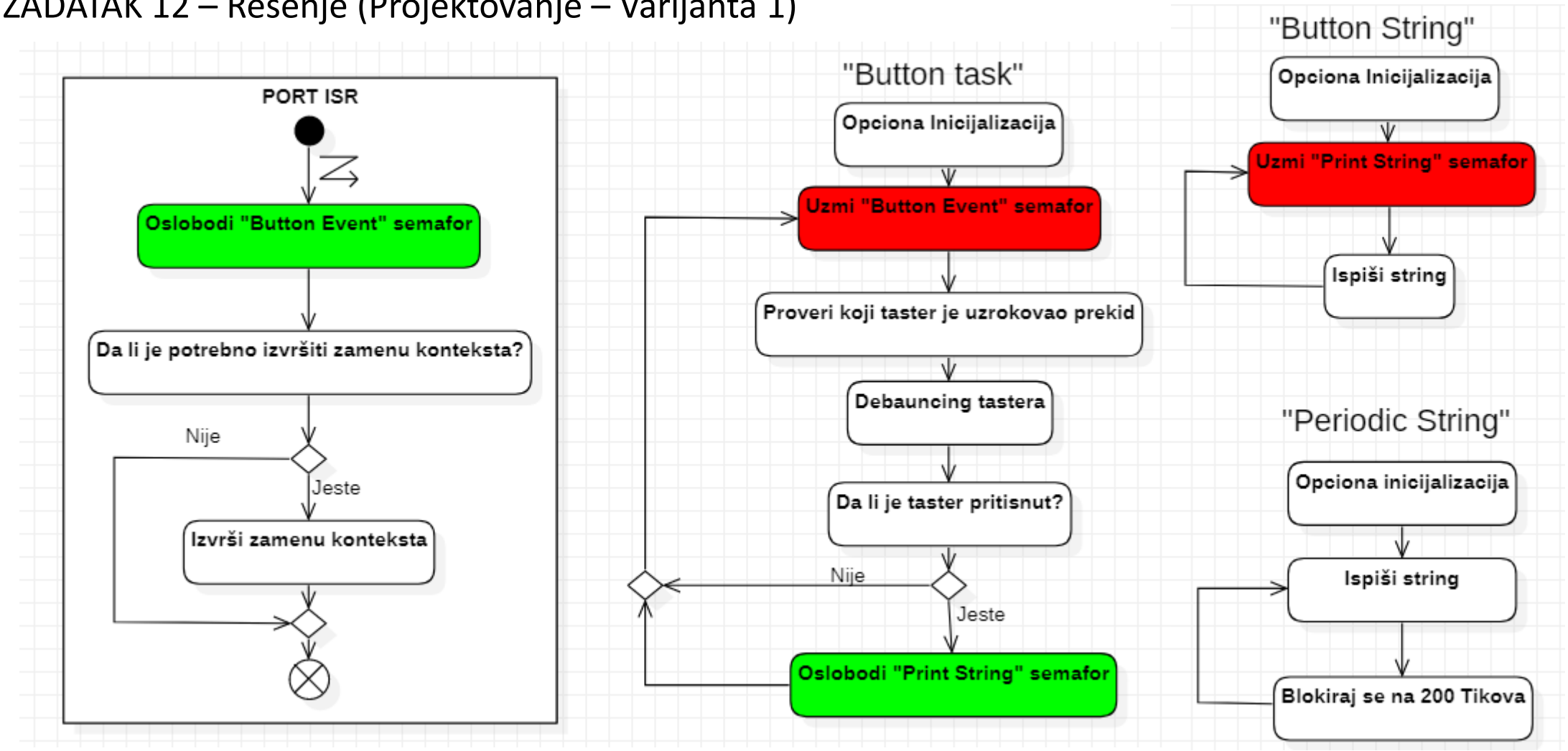
1. **Detekcija pritiska taster**
   - Ovu logičku celinu ćemo implementirati kombinujući **prekidnu rutinu** (u okviru koje ćemo detektovati potencijalni pritisak tastera) i „Button Task“ u okviru koga ćemo nastaviti dalju obradu prekidnog zahteva kako bi smo potvrdili da se pritisak tastera zaista desio.
2. **String koji se ispisuje ukoliko je taster pritisnut**
   - Ovu logičku celinu ćemo implementirati u okviru „Button String“ taska i sinhronizovaćemo ga sa „Button Task“ taskom koristeći binarni semafor
3. **String koji se periodično ispisuje**
   - Ovu logičku celinu ćemo implementirati u okviru „Periodic String“ taska



FreeRTOS MUTEX

PRIMERI

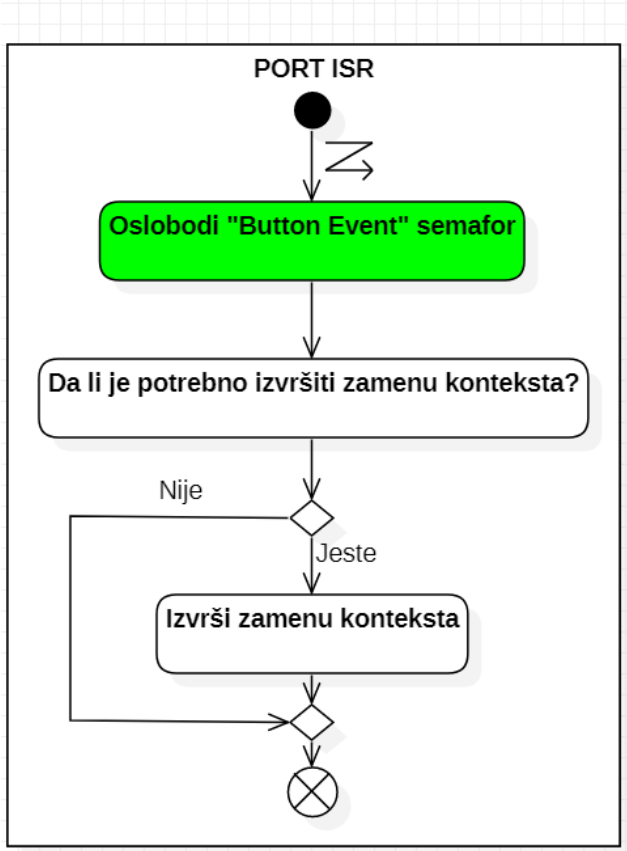
- ZADATAK 12 – Rešenje (Projektovanje – Varijanta 1)

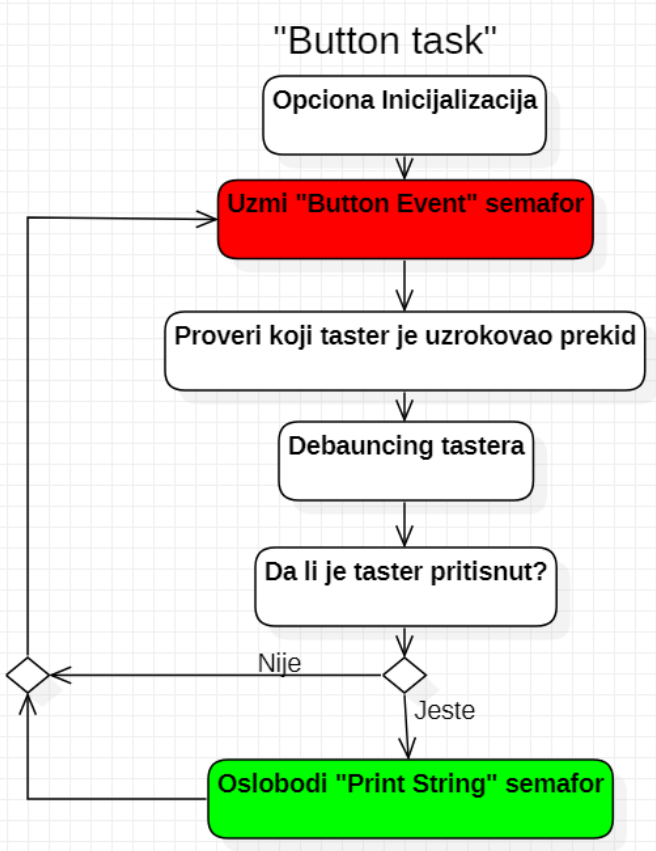


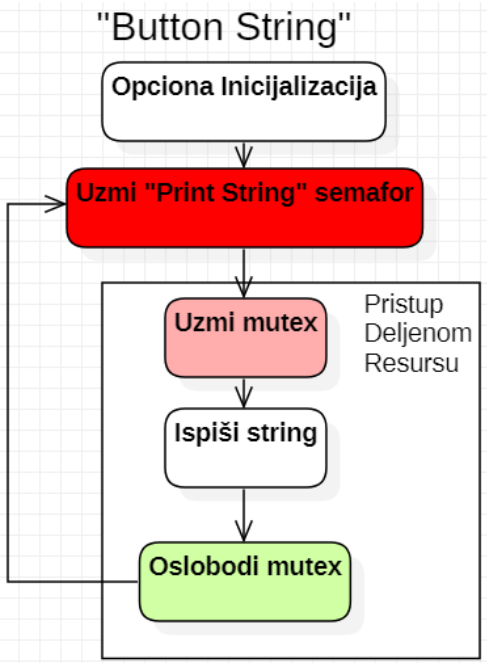
FreeRTOS MUTEX

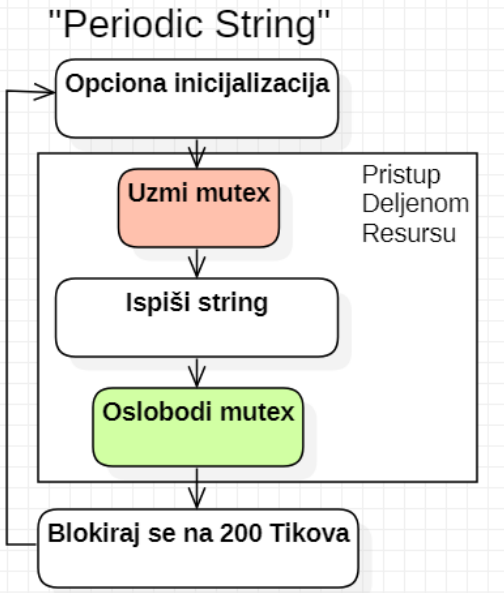
PRIMERI

- ZADATAK 12 – Rešenje (Projektovanje – Varijanta 2)









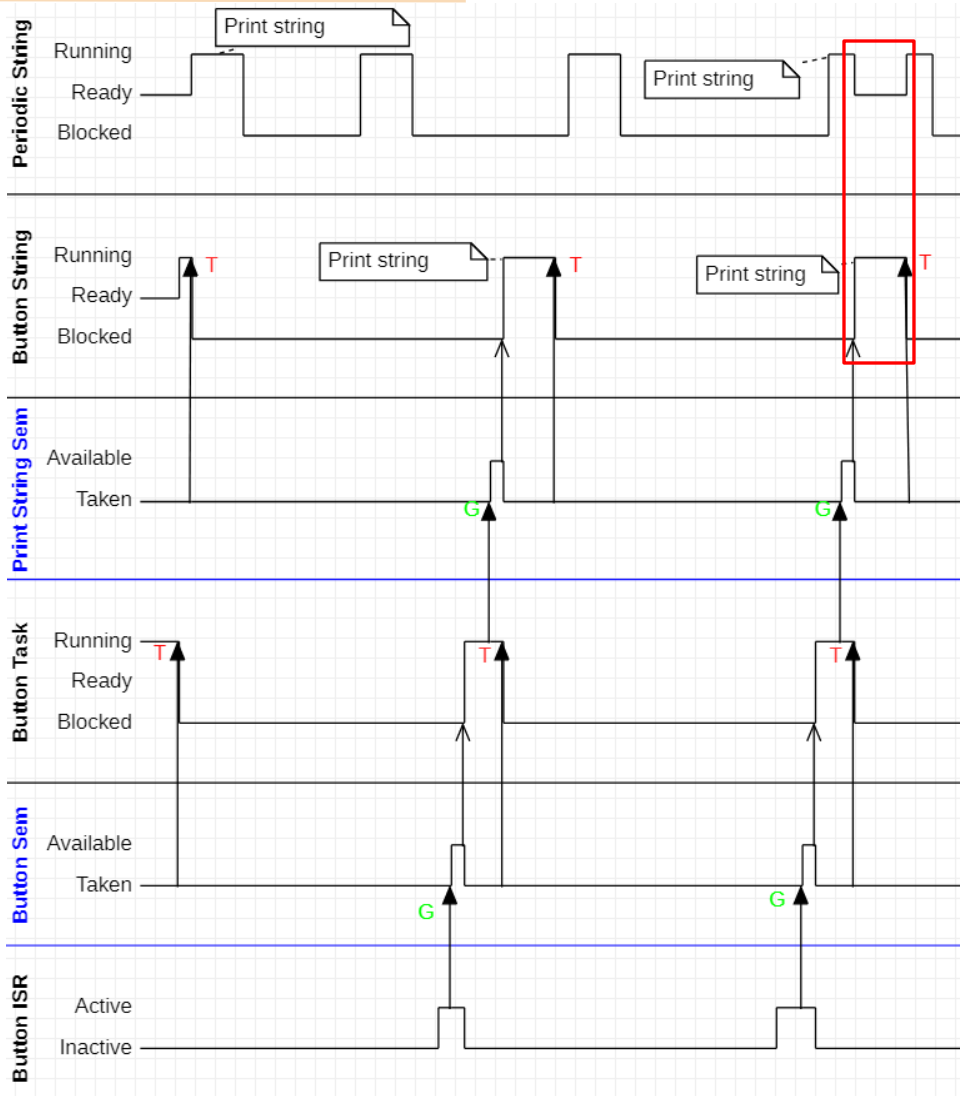
FreeRTOS MUTEX

PRIMERI

- ZADATAK 12 – Rešenje (Projektovanje – Varijanta 1)

```
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Tas- Korisnik pritisnuo taster
k" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se po- Korisnik pritisnuo taster
ziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Pe- Korisnik pritisnuo taster
riodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* O- Korisnik pritisnuo taster
vaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
```

- ZADATAK 11 – Rešenje (Realizacija – Varijanta 1)
  - Videti kôd projekta ***SRV_2_7_v1***



FreeRTOS MUTEX

PRIMERI

- ZADATAK 12 – Rešenje (Projektovanje – Varijanta 2)

```
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
- Korisnik pritisnuo taster
* Ovaj ispis se poziva periodicno iz "Periodic Task" taska
```

- ZADATAK 12 – Rešenje (Realizacija – Varijanta 2)
  - Videti kôd projekta ***SRV_2_7_v2***

FreeRTOS

Softverski tajmeri

OSOBINE SOFTVERSKIH TAJMERA U FREERTOSU

API FUNKCIJE

PRIMERI

FreeRTOS SOFTVERSKI TAJMERI

Osobine

- Softverski tajmeri se u FreeRTOS-u koriste kako bi se **zakazalo izvršavanje** neke funkcionalnosti u određenom vremenskom trenutku ili u slučaju da je potrebno **periodično izvršavanje** određene funkcionalnosti
  - Period softverskog tajmera je vreme proteklo od trenutka startovanja tajmera do trenutka početka izvršavanja tajmerske callback f-je
  - Postoje dve vrste softverskih tajmera u *FreeRTOS*-u
    - ***One-shot timer***
      - Jednom kada se startuje tajmerska *callback* f-ja se poziva **samo jednom** nakon isteka periode tajmera
    - ***Auto-reload timer***
      - Jednom kada se startuje tajmerska *callback* f-ja se poziva **periodično** nakon isteka tajmera.
- Mehanizam softverskog tajmera je implementiran u okviru kernela FreeRTOS-a koji je zadužen i za upravljanje tajmerima.
- Na platformi na kojoj se koriste softverski tajmeri nije potrebno da postoji bilo kakva dodatna hardverska podrška
- Funkcionalnost softverskog tajmera je opciona u FreeRTOS-u. Ukoliko želimo da koristimo ovu funkcionalnost neophodno je:
  1. U okviru projekta uključiti *FreeRTOS source* fajl pod nazivom ***timers.c***
  2. U ***FreeRTOSConfig.h*** fajlu setovati ***configUSE_TIMERS*** makro na vrednost **1**

Tajmerska *callback* f-ja

- F-ja koja se izvršava od strane softverskog tajmera naziva se **tajmerska *callback* f-ja**
  - Funkcionalnost ove f-je definiše korisnik
  - Klasična c-ovska f-ja koja ima sledeću deklaraciju

```
void ATimerCallback( TimerHandle_t xTimer );
```

    - Ne vraća vrednost i kao jedini argument prima instancu tajmera ***TimerHanlde_t*** kome je ova f-ja dodeljena (koji je pozvao ovu *callback* f-ju)
    - Funkcionalnost koju ***tajmerska callback f-ja*** realizuje mora biti jednostavna kako bi se f-ja što pre izvršila.
- Tajmerska *callback* f-ja se izvršava iz konteksta FreeRTOS sistemskog „Deamon“ taska
  - Kreira se automatski pri startovanju FreeRTOS *scheduler*-a
    - Vrednošću ***configTIMER_TASK_PRIOROITY*** makroa u *FreeRTOSConfig.h* fajlu setujemo prioritet „Deamon“ taska
    - Vrednošću ***configTIMER_TASK_STACK_DEPTH*** makroa u *FreeRTOSConfig.h* fajlu setujemo veličinu steka koju želimo da dodelimo ovom tasku
  - FreeRTOS *scheduler* raspoređuje ovaj task kao i bilo koji drugi task u sistemu
    - Treba voditi računa o prioritetu taska kako bi izvršavanje tajmerske f-je bilo moguće
  - Upravljanje „Deamon“ taskom realizuje se posredstvom odgovarajućeg *queue*-a
    - U slučaju kontrole funkcionalnosti tajmera (start, stop, reset, ...), „Deamon“ task čita komande iz **tajmerskog *queue*-a**
      - Kreira se automatski pri startovanju FreeRTOS schedulera
      - Dužina ovog *queue*-a određena je vrednošću ***configTIMER_QUEUE_LENGTH*** makroa u ***FreeRTOSConfig.h*** fajlu
  - Pored kontrole softverskih tajmera ovaj task u FreeRTOS-u služi za implementaciju i ostalih sistemskih funckionalnosti
- **Iz tajmerske *callback* f-je se ne smeju pozivati FreeRTOS api f-je koje mogu izazvati blokiranje taska**
    - U tom slučaju bi došlo do blokiranja „Deamon“ taska

API f-je

- Funkcionalnost API f-ja za rad sa softverskim tajmerom se zasniva na upisu odgovarajućih komandi u tajmerski queue „Deamon“ taska
- Pre korišćenja mora se kreirati softverski tajmer
  - Tajmer se može kreirati pre startovanja schedulera ili iz konteksta taska
  - Kreiranje softverskog tajmera realizuje se pozivom sledeće API f-je

```
TimerHandle_t xTimerCreate( const char * const pcTimerName,
                            TickType_t xTimerPeriodInTicks,
                            UBaseType_t uxAutoReload,
                            void * pvTimerID,
                            TimerCallbackFunction_t pxCallbackFunction );
```

Naziv parametra	Opis
`pcTimerName`	Naziv tajmera
`xTimerPeriodInTicks`	Period tajmera izražen u **sistemskim tikovima**. Ukoliko želimo da umesto sistemskih tikova definišemo **apsolutno vreme** u ms, možemo iskoristiti makro f-ju ***pdMS_TO_TICKS*** koja prima vrednost u ***ms*** a vraća odgovarajući broj u sistemskim tikovima
`uxAutoReload`	Ukoliko je vrednost ovog parametra ***pdTRUE*** kreira se **auto-reload** softverski tajmer. Ukoliko je vrednost ovog parametra ***pdFALSE*** kreira se ***one-shot*** softverski tajmer
`pvTimerID`	Svakom kreiranom tajmeru se dodeljuje jedinsvena vrednost. Posredstvom ovog parametra može se dohvatiti ta vrednost.
`pxCallbackFunction`	Prethodno definisana tajmerska callback f-ja

- Tajmer se kreira u neaktivnom stanju što

API f-je

- Tajmer se kreira u neaktivnom stanju
- Startovanje softverskog tajmera realizuje se koristeći sledeću API f-ju

```
BaseType_t xTimerStart( TimerHandle_t xTimer, TickType_t xTicksToWait );
```

Naziv parametra	Opis
xTimer	Instanca prethodno kreiranog tajmera
xTicksToWait	Broj sistemskih tikova koje će task provesti u blokiranom stanju ukoliko nije moguće startovati tajmer

- Ukoliko je komanda „Startuj tajmer“ uspešno poslata u tajmerski *queue* f-ja vraća pdPASS. U suprotnom vraća pdFALSE
- Ukoliko se tajmer startuje iz prekidne rutine potrebno je koristiti ***xTimerStartFromISR***

- Zaustavljanje softverskog tajmera realizuje se koristeći sledeću API f-ju

```
BaseType_t xTimerStop( TimerHandle_t xTimer, TickType_t xTicksToWait );
```

Naziv parametra	Opis
xTimer	Instanca prethodno kreiranog tajmera
xTicksToWait	Broj sistemskih tikova koje će task provesti u blokiranom stanju ukoliko nije moguće zaustaviti tajmer

- Ukoliko je komanda „Zaustavi tajmer“ uspešno poslata u tajmerski *queue* f-ja vraća pdPASS. U suprotnom vraća pdFALSE
- Ukoliko se tajmer zasutavlja iz prekidne rutine potrebno je koristiti ***xTimerStopFromISR***

➢ ZADATAK 13

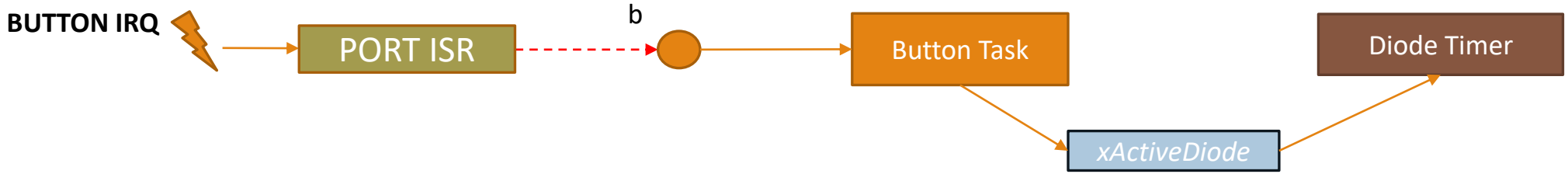
Projektovati i **realizovati** softver baziran na *FreeRTOS*-u koji treba da zadovolji sledeću funkcionalnost:

Pritisak tastera S3 selektuje aktivnu diodu. Aktivna dioda menja stanje sa periodom od 500ms.

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.

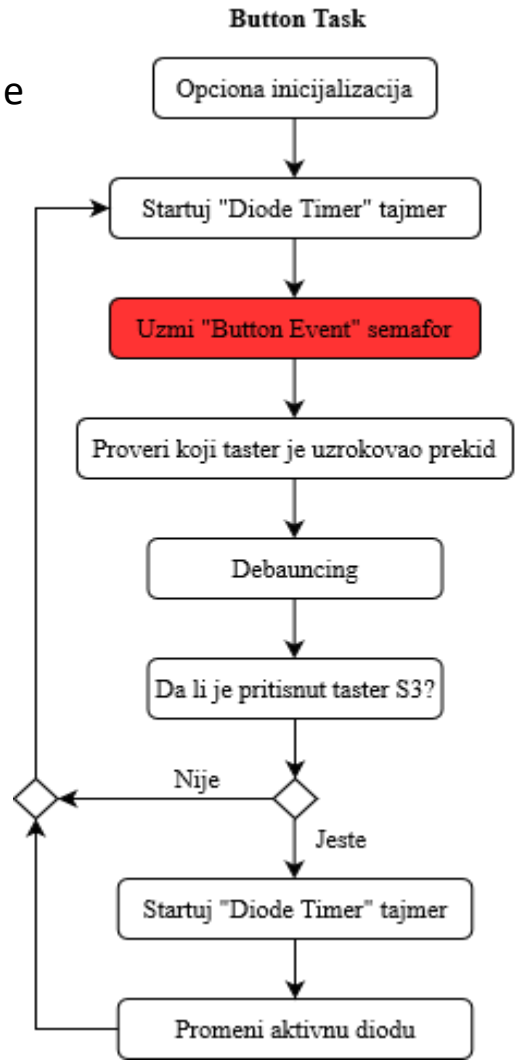
➢ ZADATAK 13 – Rešenje (Projektovanje)

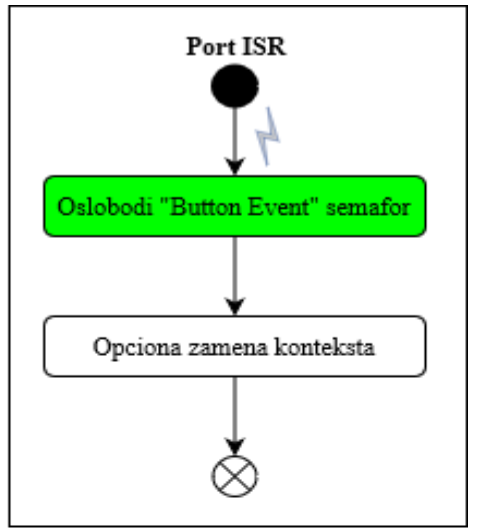
- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine
  1. **Detekcija pritiska tastera S3**
     - Ovu logičku celinu implementiraćemo koristeći prekidnu rutinu porta i „Button Task“ task. U okviru prekidne rutine porta detektovaćemo da se desio potencijalni pritisak tastera dok ćemo odloženu obradu prekida nastaviti u okviru „Button task“ taska.
  2. **Periodična promena stanja aktivne diode**
     - Ovu celinu implementiraćemo u okviru tajmerske *callback* f-je ***prvDiodeTimerCallback*** tajmera ***„Diode Timer“***

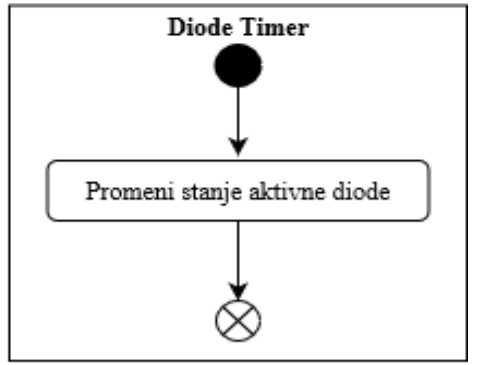


Primeri

- ZADATAK 13 – Rešenje (Projektovanje)
  - Zašto je u ovakvoj realizaciji bitno zaustaviti tajmer pre promene aktivne diode
- ZADATAK 13 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_12***







ZADATAK 14

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

Pritisak tastera S3 uzrokuje povećanje promenljive ***Period*** za korak 2 dok pritisak tastera S4 uzrokuje smanjenje vrednosti promenljive ***Period*** za isti korak. Vrednost ove promenljive može biti u opsegu od 0-30 i predstavlja periodu treperenja diode izraženu u stotinama milisekundi. Inicijalna vrednost promenljive Period iznosi 2.

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 14– Rešenje (Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine

1. **Detekcija pritiska tastera S3 i S4**
   - Ovu logičku celinu implementiraćemo koristeći prekidnu rutinu porta i „Button Task“ task. U okviru prekidne rutine porta detektovaćemo da se desio potencijalni pritisak nekog od tastera dok ćemo odloženu obradu prekida, kao i promenu periode tajmera, uraditi u okviru „Button task“ taska.
2. **Periodična promena stanja diode**
   - Ovu celinu implementiraćemo u okviru tajmerske *callback* f-je ***prvDiodeTimerCallback*** tajmera ***„Diode Timer“***

Primeri

➢ ZADATAK 14

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

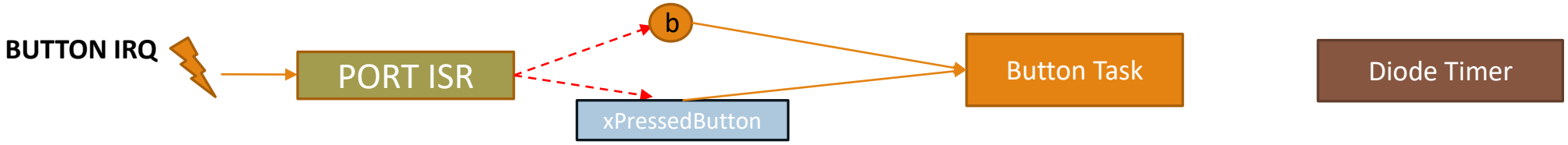
Pritisak tastera S3 uzrokuje povećanje promenljive ***xPeriod*** za korak 2 dok pritisak tastera S4 uzrokuje smanjenje vrednosti promenljive ***xPeriod*** za isti korak. Vrednost ove promenljive može biti u opsegu od 0-30 i predstavlja periodu treperenja diode izraženu u stotinama milisekundi. Inicijalna vrednost promenljive iznosi 2.

xPeriod

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 14– Rešenje (Projektovanje)

- U okviru zahtevanih funkcionalnosti moguće je uočiti sledeće logičke celine
  1. **Detekcija pritiska tastera S3 i S4**
  2. **Periodična promena stanja diode**



Primeri

- ZADATAK 14 – Rešenje (Projektovanje)
  - Za domaći
- ZADATAK 14 – Rešenje (Realizacija)
  - Videti kôd projekta ***SRV_2_13***

Koji problem postoji u realizaciji SRV_2_13?

Primeri

- ZADATAK 15 (Za samostalni rad)

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

Modifikovati zadatak 14 tako da se vrednost promenljive ***xPeriod*** ispisuje I na dvocifrenom sedmosegmentnom displeju.

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.