VEŽBA – FreeRTOS – 5

U ovoj vežbi radimo sa redovima (*eng. queue)*. U RTOS komunikacija među taskovima se najčešće obavlja pomoću redova . Oni se mogu koristiti za slanje poruka između taskova i između interapta i taskova. U većini slučajeva koriste se kao FIFO (First In First Out) baferi sa novim podacima, koji su zaštićeni od problema sa multitaskingom. Biće prikazani primeri korišćenja redova. Na ovom linku možete dodatno proučiti princip rada redova:

https://www.freertos.org/Embedded-RTOS-Queues.html

Nove API funkcije FreeRTOS koje ćemo koristiti su navedene:

`QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t );`	Kreira red i vraća referencu (*handle*) na njega. Ako vrati nulti pokazivač (`NULL`), kreiranje nije uspelo.Parametri: **`uxQueueLength`** - Maksimalan broj podataka koje red može da drži u bilo kojem trenutku. **`uxItemSize`** - Veličina u bajtovima potrebna za držanje svakog podatka u redu. https://www.freertos.org/a00116.html
`BaseType_t xQueueSend( QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait );`	Ubacuje podatak u red. Podaci su kopirani u red, nisu referencirani. Parametri: **`xQueue`** – referenca reda; **`* pvItemToQueue`**- pointer na podatak koji se stavlja u red **`xTicksToWait`** – maksimalno vreme blokiranja taska, u slučaju da je red pun pa se podaci ne mogu ubaciti https://www.freertos.org/a00117.html
`BaseType_t xQueueReceive( QueueHandle_t xQueue, void *pvBuffer,`	Uzima, tj očitava podatak iz reda. Parametri: **`xQueue`** – referenca reda; **`*pvBuffer`** – pointer na bafer u kojeg se upisuju podaci iz reda **`xTicksToWait`** – maksimalno vreme blokiranja

TickType_t xTicksToWait);	taska dok se čeka na prispeće podatka u red, u slučaju da je red prazan u trenutku pozivanja ove funkcije https://www.freertos.org/a00118.html
BaseType_t xQueueSendFromISR (QueueHandle_t xQueue, const void *pvItemToQueue, BaseType_t *pxHigherPriorityTaskWoken);	Funkcija koja ima svrhu kao i **xQueueSend,** ali se koristi kada se podaci ubacuju u red iz interapta. Parametri: **xQueue –** referenca reda; *** pvItemToQueue-** pointer na podatak koji se stavlja u red *pxHigherPriorityTaskWoken* **–** ovaj pointer ce podesti vrednost promenjive na koju pokazume na pdTRUE ako je slanje podataka odblokiralo task sa višim prioritetom u odnosu na task koji je trenutno aktivan https://www.freertos.org/a00119.html
portYIELD_FROM_ISR(xHigherPriorityTaskWoken);	Ako je prosleđen pdTRUE kao **xHigherPriorityTaskWoken** parametar, onda se izvršava promena konteksta (*context switching*)

Zadatak

1.

a) Kreirati dva taska, jedan za upisivanje u red (svakih 400 ms), a drugi za primanje podataka iz reda. Pored toga, kreirati i jedan softverski tajmer preko koga također upisivati podatke u red (svakih 1100 ms).

b) Prikazati na dva odvojena dela displeja broj puta koliko puta su pisani podaci u red iz taska, a koliko iz tajmera.

2. Umesto softverskog tajmera koristiti externi (simulirani) interapt

Rešenje zadatka 1

a) Kreiramo taskove i tajmer na već poznati način. Jedina razlika u odnosu na ranije kreirane taskove je korišćenje drugog parametra `xTaskCreate`, gde smo tasku za primanje podataka iz niza dodelili string RX, a tasku za upis podataka u red dodelili string TX, ovi stingovi se koriste samo

za indetifikaciju taskova prilikom debagovanja.

```
tH = xTimerCreate(NULL, pdMS_TO_TICKS(1100), pdTRUE, NULL, TimerCallback);
        if (tH == NULL)
                while (1);
        //xTimerStart(tH, 0);



        if (xTaskCreate(QueueReceive_tsk, "Rx", configMINIMAL_STACK_SIZE, NULL, 2, &tA) != pdPASS)
                while (1); // task za primanje podataka iz reda
        if (xTaskCreate(QueueSend_tsk, "TX", configMINIMAL_STACK_SIZE, NULL, 2, &tB) != pdPASS)
                while (1); // task za upis podataka u red

        myQueue = xQueueCreate(2, sizeof(uint32_t));// kreiramo Queue duzine dva uint32_t
```

Za kreiranje reda `myQueue` koristimo funkciju `xQueueCreate`, ovaj red može da čuva dva podatka tipa `uint32_t`, tj. unsigned int.

Task za upis podataka u niz se izvršava u funkciji `QueueSend_tsk`:

```
static void QueueSend_tsk(void* pvParams)
{
                 uint32_t ulValueToSend = mainVALUE_SENT_FROM_TASK;
                xTimerStart(tH, 0);// pokrecemo tajmer
 for (;; ) // beskonacna petlja
               {
                vTaskDelay(pdMS_TO_TICKS(400UL)); // task je blokiran 400 ms
                xQueueSend(myQueue, &ulValueToSend, 0U); // salje podatak iz taska u red myQueue
                }

}
```

Može se videti da je task blokiran u periodima od 400 ms, posle čega upisuje podatak u red `myQueue` preko funkcije `xQueueSend` . Kao prvi agrument u ovu funkciju prosleđujemo referencu reda `myQueue`, drugi argument je adresa podatka kojeg upisujemo u red (vrednost ovog podatka se odmah kopira u red),ovaj podatak je makro `mainVALUE_SENT_FROM_TASK`, tj vrednost 100. Treći argument je maksimalno vreme blokiranja taska, u slučaju da je red pun pa se podaci ne mogu ubaciti, ovde je 0, tj. uopšte se ne čeka da se red isprazni.
Callback funkcija softverskog tamera također šalje podatak u red, samo druge vrednosti, makro `mainVALUE_SENT_FROM_TIMER` tj vrednost 200.

```
static void TimerCallback(TimerHandle_t tmH)
{
         uint32_t ulValueToSend = mainVALUE_SENT_FROM_TIMER;
        xQueueSend(myQueue, &ulValueToSend, 0U);// salje podatak iz tajmera
}
```

b) Task za primanje podataka iz niza se izvršava u funkciji `QueueReceive_tsk`:

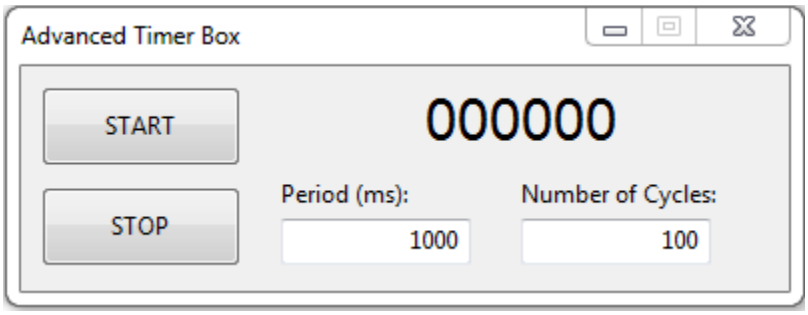
```
static void QueueReceive_tsk(void* pvParameters)
{
	uint32_t ulReceivedValue;
	uint16_t a_num = 0;
	uint16_t b_num = 0;
		for (;; )
	{	xQueueReceive(myQueue, &ulReceivedValue, portMAX_DELAY); // task ceka u blokiranom stanju
                                                                 //dok ne dobije podatak iz Queue

		if (ulReceivedValue == mainVALUE_SENT_FROM_TASK) //ako smo dobili podatak iz taska
                                                 //inkrementira vrednost na zadnje dve cifre displeja
		{
			mxDisp7seg_SelectDigit(myDisp, 4);
			mxDisp7seg_SetDigit(myDisp, character[a_num % 10]);
			mxDisp7seg_SelectDigit(myDisp, 3);
			mxDisp7seg_SetDigit(myDisp, character[a_num / 10]);
			a_num++;
		}
		else if (ulReceivedValue == mainVALUE_SENT_FROM_TIMER)  //ako smo dobili podatak iz
                                        //tajmera inkrementira vrednost na prve dve cifre displeja
		{
			mxDisp7seg_SelectDigit(myDisp, 1);
			mxDisp7seg_SetDigit(myDisp, character[b_num % 10]);
			mxDisp7seg_SelectDigit(myDisp, 0);
			mxDisp7seg_SetDigit(myDisp, character[b_num / 10]);
			b_num++;
		}
		else
		{
			printf("Unexpected message\r\n");
		}
	}
}
```

Može se videti da se za očitavanje podataka iz reda koristi funkcija `xQueueReceive`. Kao prvi agrument u ovu funkciju prosleđujemo referencu reda `myQueue`, drugi argument je adresa bafera u kojeg se upisuju podaci iz reda (u ovom slučaju promenjiva `ulReceivedValue`). Treći argument je maksimalno vreme blokiranja taska dok se čeka na prispeće podatka u red, u slučaju da je red prazan u trenutku pozivanja ove funkcije, ovde je podešeno da to vreme bude beskonačno (pomoću `portMAX_DELAY`), tj task će biti blokiran sve dok ne dodje podatak u red.

U zavisnosti da li je podatak upisan u red iz taska ili tajmera, u promenjivu `ulReceivedValue` se iz reda upsisuje podatak vrednosti 100 (tj. makro `mainVALUE_SENT_FROM_TASK)` ili podatak vrednosti 200 (tj. makro `mainVALUE_SENT_FROM_TIMER)` . U slučaju da je primljen podatak vrednosti 100, inkrementuje se promenjiva `a_num` i njena vrednost se prikazuje na zadnje dve cifre displeja. U slučaju da je primljen podatak vrednosti 200, inkrementuje se promenjiva `b_num` i njena vrednost se prikazuje na prve dve cifre displeja.

2. Kao externi simualtor koristimo softver **AdvTimerBox.exe** ,ovaj program treba pokrenuti pre pokretanja debug softvera koji testiramo.
Kao što se može videti na slici dolje, u pitanju je softer koji se jednstavno kontroliše, u polje *Period* se podešava perioda dešavanja interapta, u *Number of cycles* broj interapta koji će se desiti, a pomoću *Start* i *Stop* uključujemo i isključujemo izvršavanje interapta.



Preko funkcije `PortSetInterruptHandler` podešavamo referencu za eksterni interapt. Kada se desi eksterni interapt ,poziva se funkcija myExtSimInterrupt, pošto je to prosleđeno kao argument za drugi parametar funkcije `PortSetInterruptHandler.`

```
PortSetInterruptHandler(portINTERRUPT_EXTSIM, myExtSimInterrupt);
```

Više ne koristimo tajmerski interapt, pa su funkcije vezane za njega obrisane. Ipak, sadržaj tajmerske callback funkcije iz zadatka 1 se skoro u potpunosti poklapa sa sadržajem funkcije `myExtSimInterrupt .`

```
static uint32_t myExtSimInterrupt(void)
{
	const uint32_t ulValueToSend = mainVALUE_SENT_FROM_INTERRUPT;
	BaseType_t xHigherPTW = pdFALSE;

	xQueueSendFromISR(myQueue, &ulValueToSend, &xHigherPTW);
	portYIELD_FROM_ISR(xHigherPTW);
}
```

I ovde se upisuju podaci u red, samo što se to iz interapta mora uraditi sa specijanom funkcijom `xQueueSendFromISR()`. Ovo je funkcija koja ima svrhu kao i **`xQueueSend,`** ali se koristi kada se podaci ubacuju u red iz interapta. Jedina razlika u primeni ove funkcije u odnosu na **`xQueueSend je njen treći parametar,`** ovaj pointer ce podesti vrednost promenjive na koju pokazuje na pdTRUE ako je slanje podataka odblokiralo task sa višim

prioritetom u odnosu na task koji je trenutno aktivan. U interapt funkcijama obično se na kraju funkcije koristi funkcija `portYIELD_FROM_ISR()`, kojom se traži od raspređivača (*eng.* scheduler*)* da se izvrši promena konteksta (*eng. context switching),* ali samo ako je kao argument prosleđen pdTRUE (u ovom kodu ako je `xHigherPTW` jednak pdTRUE). Pomoću ove funkcije ubrzava se oslobađanje resursa sistema, tj raspoređivaču se javlja da je interapt funcija završila sve što se očekuje od nje.

Pokretanjem ovog zadatka može se videti da se vrednost na prve dve cifre displeja neće menjati sve dok se ne pokrene START na **AdvTimerBox.exe,** tj kada taj simulator ne krene sa slanjem interapta.