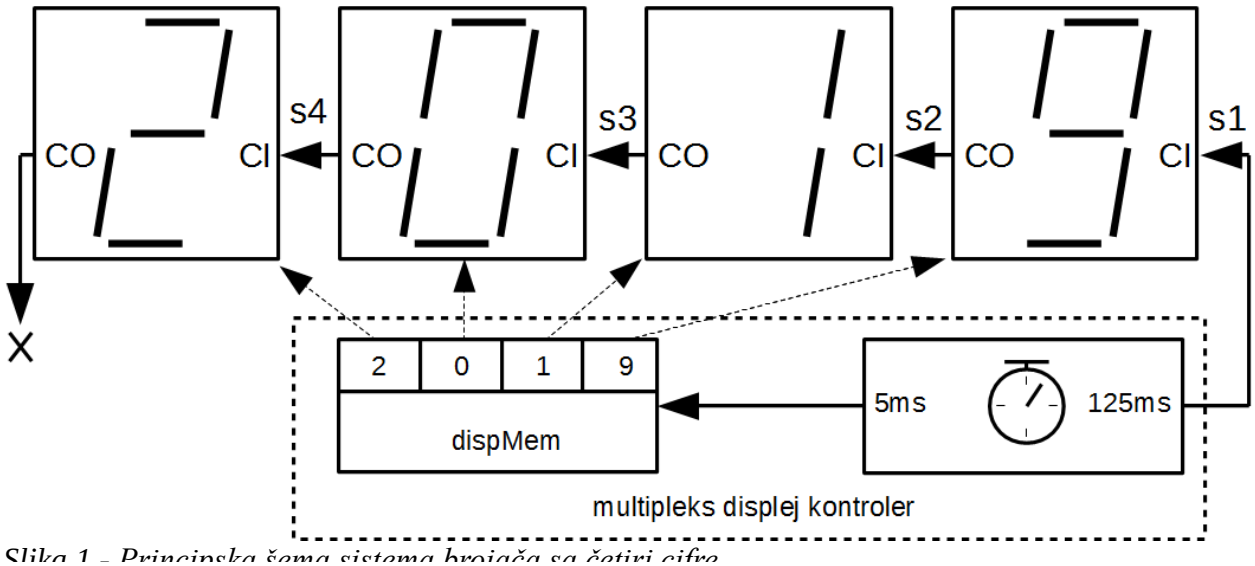
VEŽBA – RTOS – 3

Pregled

Primenom operativnog sistema, moguće je problem razložiti na niz nezavisnih funkcionalnih celina od kojih svaka čini poseban program koji se izvršava paralelno sa ostalima – na istom centralnom procesoru. Ovakvi programi koji se paralelno izvršavaju na istom sistemu uobičajeno se nazivaju nitima (engl. *thread*) ili zadacima (engl. *task*). Osim ako sistem sadrži više procesorskih jezgara, stvarno paralelno izvršavanje niti nije moguće. Međutim, kapaciteti centralnog procesora redovno višestruko nadmašuju one potrebne za obavljanje zadatka, tako da operativni sistem dodeljuje procesorske resurse sukcesivno tokom kratkih vremenskih intervala različitim zadacima što čini da se oni efektivno izvršavaju paralelno.

Niti koje se na opisani način paralelno izvršavaju su u potpunosti izolovane i nemaju nikakvu podrazumevanu vremensku sinhronizaciju ili razmenu podataka sa drugim nitima. Ovi zadaci se moraju posebno rešavati primenom posebnog API-ja koji FreeRTOS stavlja na raspolaganje, a elementi koji se pri tome koriste su semafori (engl. *semaphore*) i redovi (engl. *queue*).

Brojač sa četiri cifre



Slika 1 - Principska šema sistema brojača sa četiri cifre

Na slici 1 prikazan je brojač se četiri cifre koji je predmet ove vežbe. Displej je multipleksni i segmenti svih cifara su direktno kontrolisani istim signalima, a svaka cifra može nezavisno da se aktivira odgovarajućim signalom (videti dodatak). Petu, krajnju desnu, cifru na displeju iz Seg7Mux nećemo koristiti.

Multipleks displej kontroler

Ovaj deo sistema neposredno rukovodi multipleksnim prikazom cifara na displeju. Redom aktivira svaku cifru na period od 20 ms, a na njoj prikazuje cifru koja je upisana u na odgovarajuće mesto u

nizu označenom kao `dispMem`. Dozvoljene su vrednosti od 0 do 9, a mapiranje za 7-segmentni displej se izvodi na osnovu tabele (engl. *look-up table*). Pošto se prikazuju četiri cifre, ukupna perioda prikaza je 80 ms.

Pošto ovaj sistem precizno meri vreme, iskorišćen je da generiše i pobudni signal za brojanje koji iznosi 25 osnovnih perioda, odnosno 500 ms.

Pojedinačni brojači cifara

Svaki brojač upravlja prikazom jedne cifre. Prikazom na displeju upravlja posredno, upisujući vrednost na odgovarajuće mesto u nizu `dispMem`. Svaka cifra reaguje na ulazni klok signal (CI) pod čijim uticajem odbroji jednom na gore. Ukoliko se prikazuje cifra 9, sledeća će se prikazati cifra 0, a istovremeno će se generisati i izlazni klok signal (CO). Taj klok signal će izazvati promenu prikaza na sledećoj cifri prema levo gde će služiti kao ulazni klok. Krajnja leva cifra nema potrebu za generisanjem izlaznog kloka jer više cifre od nje nema.

Svaki brojač treba implementirati kao jedan task.

Klok signali

U prethodnim odeljcima je već opisana namena ovih signala. Treba ih implementirati kao brojačke semafore (engl. *counting semaphore*) sa ograničenjem brojanja na 10. Ovi semafori obezbeđuju sinhronizaciju svih elemenata sistema. Task će biti blokiran brojačkim semaforom samo ukoliko je pokušao da ga uzme (engl. *take*), a njegova vrednost je u tom trenutku bila 0. Na ovaj način, moguća je akumulacija događaja, tj. ako blokirani task ne odreaguje dovoljno brzo i u međuvremenu se još nekoliko događaja da semafor, i oni će biti uvaženi.

FreeRTOS

U narednoj tabeli dat je kratak pregled API funkcija koje su potrebne za rešavanje zadataka ove laboratorijske vežbe:

`SemaphoreHandle_t xSemaphoreCreateCounting( UBaseType_t uxMaxCount, UBaseType_t uxInitialCount );`	Kreira brojački semafor, zadaje maksimalnu i početnu vrednost. Vraća referencu na kreirani semafor. Referenca je `NULL` ako je neuspešno. https://www.freertos.org/CreateCounting.html
`BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );`	Daje semafor čija se referenca navodi kao parametar.
`BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait );`	Pokušava da uzme semafor, blokira u slučaju da je neuspešno. Vreme blokiranja se može ograničiti drugim parametrom. Vraćena vrednost je `pdPASS` ako je semaform uzet ili `pfFAIL` ako je vreme isteklo.
`void vTaskDelete( TaskHandle_t xTask );`	Briše task, i oslobađa memoriju koju je ovaj task koristio, tj njegov stek. https://www.freertos.org/a00126.html

Zadaci

1. a) Implementirati sistem prikazan na slici 1. Voditi računa da treba kreirati 4 taska, a u kontekstu tih taskova se izvršava jedna jedina funkcija. Funkcija treba da proverava koji task ju je pozvao, i u slučaju da je dobila odgovarajući clock signal, tj. brojački semafor, da inkrementira vrednost cifre. Izvršiti osvežavanje ekrana svakih 80 ms, a inkrementiranje cifre koja se prikazuje svakih pola sekunde.
2. b) Kada sistem izbroji 9999 na displeju, obrisati sve taskove

Rešenje zadatka pod a:

Prema uslovima zadatka kreirana su 4 taska:

```
if (xTaskCreate(only_tsk, NULL, configMINIMAL_STACK_SIZE, (void*)0, 2, &tA) != pdPASS)
                while (1);// ID taska 0, referenca tA
        if (xTaskCreate(only_tsk, NULL, configMINIMAL_STACK_SIZE, (void*)1, 2, &tB) != pdPASS)
                while (1);// ID taska 1, referenca tB
        if (xTaskCreate(only_tsk, NULL, configMINIMAL_STACK_SIZE, (void*)2, 2, &tC) != pdPASS)
                while (1);// ID taska 2, referenca tC
        if (xTaskCreate(only_tsk, NULL, configMINIMAL_STACK_SIZE, (void*)3, 2, &tD) != pdPASS)
                while (1);// ID taska 3, referenca tD
```

Može se videti da sva 4 taska koriste istu funkciju za implementaciju taska, to je funkcija `only_tsk` .U ovom zadatku koristimo ID taska tj dodeljujemo vrednost trećem parameter funkcije *xTaskCreate* , tako da smo im dodelili vrednosti od 0 do 3. Referencu taska,tj zadnji parametar funkcije *xTaskCreate* nećemo koristiti za indetifikaciju taska, već za njihovo brisanje.

```
        s1 = xSemaphoreCreateCounting(MAX_SEM_COUNT, 0);
        if (s1 == NULL) while (1);

        s2 = xSemaphoreCreateCounting(MAX_SEM_COUNT, 0);
        if (s2 == NULL) while (1);

        s3 = xSemaphoreCreateCounting(MAX_SEM_COUNT, 0);
        if (s3 == NULL) while (1);

        s4 = xSemaphoreCreateCounting(MAX_SEM_COUNT, 0);
        if (s4 == NULL) while (1);
```

generišemo i 4 brojacka semafora (Counting Semaphores) preko funkcije `xSemaphoreCreateCounting`. Razlika između binarnog semafora kojeg smo koristili u prošloj vežbi i brojačkog semafora je u tome što su binarni semafori ograničene na vrednosti 0 i 1 (ili zaključane / otključane, nedostupne / dostupne), dok brojački semafori mogu da imaju vrednost veću od jedan. Kao i kod binarnog semafora, funkcija `xSemaphoreGive` se koristi da bi se "dao" semafor, ali za razliku od binarnih kod koji mogu imati maksimalnu vrednost jedan nezavisno koliko puta se pozvala ova funkcija, kod brojačkih semafora njegova vrednost se inkrementuje svaki put kada se pozove ova funkcija. Može se gledati da se brojački semafor inkrementuje za 1 svaki put kada se pozove `xSemaphoreGive`, a dekrementuje kada se pozove `xSemaphoreTake`. Kao i kod binarnih semafora, sve dok je vrednost semafora veća od nula, `xSemaphoreTake` može da "uzme" semafor i time odblokira task u kom se nalazi.

Funkcija `xSemaphoreCreateCounting` ima dva parametra, prvi govori kolika je maksimalna vrednost koju semafor može da dostigne(ovde postavljena na MAX_SEM_COUNT tj. 10), a druga je početna vrednost semafora, ovde postavljena na nulu.

Brojački semafori su u ovm sistemu korišćeni umesto binarnih da bi se obezbedilo da se nijedan poziv iz callback funkcije softverskog tajmera ne preskoči. U slučaju da task koji čeka na semafor iz tajmera sadrži kod kojem s vremena na vreme treba puno da vremena da se izvrši(npr. neku komunikaciju sa ostalim sistemima koja se povremeno dešava), može se desiti da tajmer callback funcija „daje“ semafor koji se ne stigne „uzeti“ iz taska pre nego što tajmer opet pozove callback funciju, tako da brojački semafor dobije vrednsot veću od 1. U tom slučaju task će kada završi sa izvršavanjem dodatnih aktivnosti, „uzeti“ više semafora jedan za drugim, što će na displeju izgledati kao da je odjednom inkrementirao cifre za više vrednosti, ali neće se izgubiti nijedan semafor, tj. u našem slučaju merenje vremena će ostati tačno. U našem slučaju ova situacija se praktično ne može desiti, jer naši taskovi nemaju puno koda za izvršavanje, ali se u praksi sa kompleksnijim programima ovo često dešava.
Kod za funkciju `only_tsk` gde se implementiraju svi taskovi je izlistan. Može se videti da se koristi parametar ove funkcije `pvParams` da bi se indetifikovalo koji task je pozvao funkciju, jer se u ovaj parametar upisuje vrednost ID taska (koji smo dodelili tasku prilikom njegovog kreiranja).

```
static void only_tsk(void* pvParams)
{
	unsigned char number = 0;
	uint32_t task_ID;
	while (1)
	{
		number++;
		task_ID = (uint32_t)pvParams;
		if (task_ID == 0)
		{
			xSemaphoreTake(s1, portMAX_DELAY);
			if (number == 10) { // ako je broj dostigao vrednost 10, treba da se inkrementira
					    //  vrednost vise cifre na displeju
				number = 0;
				xSemaphoreGive(s2);// dajemo takstni signal visoj cifri preko semafora
			}
		}
		if (task_ID == 1)
		{
			xSemaphoreTake(s2, portMAX_DELAY);
			if (number == 10) {
				number = 0;
				xSemaphoreGive(s3);
			}
		}
		if (task_ID == 2)
		{
			xSemaphoreTake(s3, portMAX_DELAY);
			if (number == 10) {
				number = 0;
				xSemaphoreGive(s4);
			}
		}
		if (task_ID == 3)
		{
			xSemaphoreTake(s4, portMAX_DELAY);
			if (number == 10) {
				number = 0;
			}
		}
	dispMem[task_ID] = number;  // upisujemo vrednost broja u niz cija vrednost se prikazuje na displeju


	}
}
```

Task sa ID 0 kontroliše vrednost na najnižoj, zadnjoj cifri displeja, i upisuje vrednost u nulti član niza `dispMem`, Task sa ID 2 kontroliše sledeću, višu cifru, displeja i tako do taska sa ID 3. Task sa ID 0 dobija semafor s1 od softverskog tajmera, i tada inkrementira vrednost promenjive `number`

za 1, a kada ta vrednost dostigne vrednost 10, upisuje se nula i daje se binarni semafor s2 kao taktni signal tasku sa ID 2 da inkrementira vrednost na sledećoj cifri displeja. Task sa ID 3 nema kome da da taktni signal, pošto kontroliše najvišu cifru displeja.

```
static void TimerCallback(TimerHandle_t tmH)
{
        static unsigned char count = 0;
        static unsigned char secount = 0;

        mxDisp7seg_SelectDigit(myDisp, (3 - count));
        mxDisp7seg_SetDigit(myDisp, character[dispMem[count]]);

        if (count < 3) count++; else count = 0; // count ima vrednost od 0 do 3

        secount++;
        if (secount == 25) { // posle 500 ms
                secount = 0;
                xSemaphoreGive(s1);
                        }
}
```

Tajmerska funkcija `TimerCallback` poziva se svakih 20 ms, promenjiva *count* može imati vrednost od nula do 3 i menja se pri svakom pozivanju ove funkcije. Pošto ova promenjiva kontoliše koji se cifra displeja trenutno osvežava (`mxDisp7seg_SelectDigit(myDisp, (3 - count));`) tj. u koju se trenutno upisuje vrednost, jedna od 4 cifre displeja koje koristimo se osvežava svakih 20 ms, a sve četiri tj. celi displej se osvežava svakih 80 ms. Kod realnih sedmosegmetnih displeja to je neophodno raditi jer su oni najčešće multipleksirani (tj u jednom trenutku u vremenu samo jedna cifra je prikazana), pa se i češće osvežavaju da se ne bi videlo treptanje. Naš simulator displeja ne zahteva osvežavanje, ali je to ostavljeno da bi videli principsko rešenje. Svakih 500 ms ova funkcija inkrementira vrednost semafora s1, tj. daje taktni signal tasku sa ID 0 da inkrementira vrednost najniže cifre displeja.
Napomena: može se videti da u funkciji `only_tsk` koristimo promenjivu `number` da prikaže vrednost na sve 4 cifre displeja, pošto je ovo lokalna funkcija ona ima različitu vrednost za svaki task. Može se reći da svaki task napravi svoju instancu ili kopiju te funkcije, i ta kopija funkcije se izvršava neovisno od ostalih instanci, Svaki task ima stek u kojem čuva stanje svih lokalnih promenjivih funkcije gde je implementiran, i kada je neki task aktivan vrednosti lokalin promenjivih se prepišu iz njegovo steka, tako da različiti taskovi mogu imati različite vrednosti loklanih promnejivih, iako su implemntiirani u istoj funkciji.

Rešenje zadatka pod b:

Kada se neki task treba obrisati , koristi se funkcija `vTaskDelete`. Kada displej odbroji 9999, to je situacija kada je task sa ID 3 odbrojao 10, tako da ćemo proširiti funkciju `only_tsk` u delu koji radi sa taskom sa ID 3. Posle dostizanja broja 9999 na displeju, svi taskovi će biti obrisani, tako da neće više biti promene vrednosti na ciframa displeja.

```
if (task_ID == 3)
                {
                        xSemaphoreTake(s4, portMAX_DELAY);
                        if (number == 10) {
                                //resenja zadatka pod b
                                //vTaskDelete(tA); //brisemo task sa ID 0 preko njegovre reference tA
                                //vTaskDelete(tB); //brisemo task sa ID 1 preko njegovre reference tB
                                //vTaskDelete(tC); //brisemo task sa ID 2 preko njegovre reference tC
                                //vTaskDelete(NULL); //brisemo task sa ID 3 preko NULL jer je to aktivan
task
                        }
                }
```

Funkciji `vTaskDelete` je potrebno upisati kao argument referncu taska kojeg treba obrisati, a ako se ova funcija poziva iz aktivnog taska, može i preko NULL kao što je to u urađeno u ovom kodu za task sa ID 3. Isti rezultat bi se dobio i preko refrence, tj moglo se pisati i `vTaskDelete(tD);`