VEŽBA – RTOS – 4

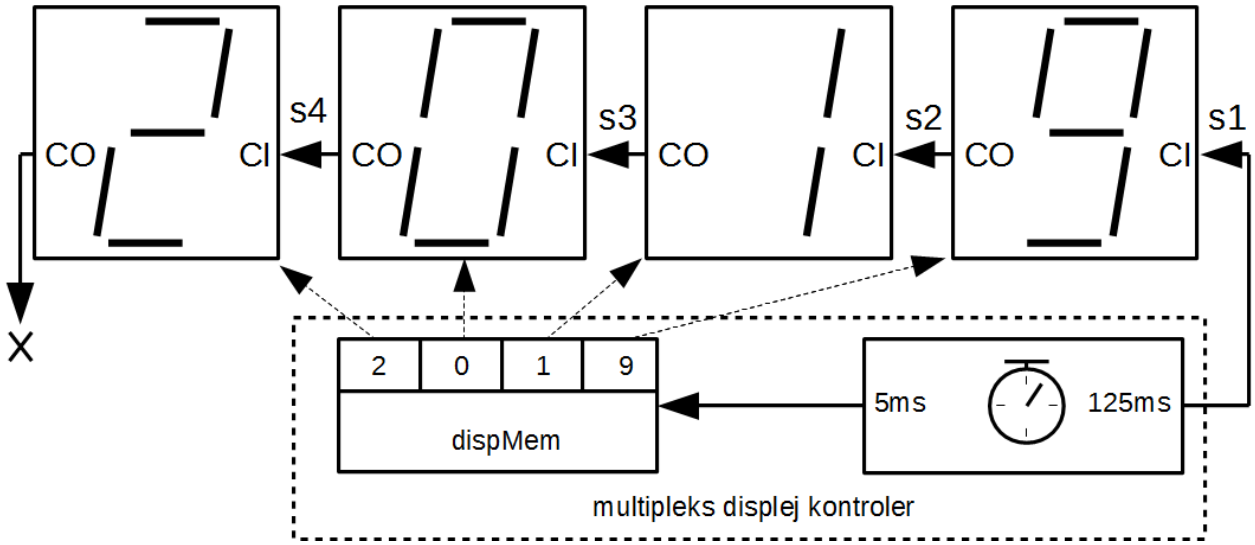
Pregled

U ovoj vežbi nastavljamo se na zadatak koji smo radili u prošloj vežbi.

Uradićemo praktično isti zadatak, ali ovaj put koristeći strukturu koja će sadržati sve podatke o displeju, što je uobičajeni način da se rešavaju problemi u C jeziku sa različitim tipovima podataka koji su logički povezani na neki način, i upotreba struktura predstavlja pravilnu programersku tehniku rada. Na kraju vežbe uspredićemo ovo i rešenje iz prethodne vežbe.

Pre pokretanja debagera u Visual Studio uključite displej preko fajla Seg7Mux.exe,koji je postavljen u 7 vežbi.

Ponavljamo opis problematike prikaza podataka na displeju



Slika 1 - Principska šema sistema brojača sa četiri cifre

Brojač 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).Sistem se sastoji od nekoliko celina.

Sistem se sastoji od nekoliko celina.

Multipleks displej kontroler

Ovaj deo sistema neposredno rukovodi multipleksnim prikazom cifara na displeju. Redom aktivira svaku cifru na period od 5 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.

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.

   b) Kada se dostigne broj 90 na displeju, ugasiti task sa referencom tC. Proveriti koliki je aktivan broj taskova prikazom na 5 cifri displeja.

Izmena zadatka: Funkcija koju pozivaju taskovi treba da ima parametar koji je pokazivač na strukturu koja je data u dodatnim informacijama.

Rešenje zadatka 1.a):

Navesti ćemo razlike u odnosu na prethodno rešenje:

Deklarišemo strukturu koja služi kao parametar tasku

```
typedef struct taskInfo_ {
	SemaphoreHandle_t clk_in;
	SemaphoreHandle_t clk_out;
	unsigned char* disp;
}taskInfo;
```

Može se videti da u ovoj strukturi postoje svi podaci potrebni za prikaz cifre na displeju(pogledati :

- semafor `clk_in` tj. ulazni clock signal koji signaliza da je potrbeno inkrementirati vrednost cifre

- semafor `clk_out` tj. izlazni clock signal koji signaliza da je potrebeno inkrementirati vrednost više cifre
- pointer tipa char `disp`, u kojem upisujemo vrednost koja se treba prikazati na cifri

Prvi deo funkcije `IV_vezba_1` je isti kao i u prethodnom rešenju. Posle kreiranja 4 semafora (na isti način kao i ranije), deklarišu se 4 promenjive tipa strukture `taskInfo` i inicijalizuju se njihove vrednosti

```
taskInfo  tinf1 =
 {
   s1,
   s2,
   dispMem[0]// moglo je i :   dispMem
 };
 taskInfo  tinf2 ={ s2, s3, &dispMem[1] }; //moglo je i : dispMem+1
 taskInfo  tinf3 ={ s3, s4,&dispMem[2] };  ////moglo je i : dispMem+2
 taskInfo  tinf4 ={ s4,  NULL,  &dispMem[3] }; //moglo je i : dispMem+3
```

Može se videti da su ulazni i izlazni clock signali tj. ovde semafori povezani kao na slici 1. Npr za prvu cifru je deklarisan `tinf1` čiji `clk_in` semafor sada pokazuje na `s1`, `clk_out` na semafor `s2`, dok pointer `disp` sada pokazuje na nulti član niza `dispMem`, u kojem čuvamo vrednost koja se treba prikazati na prvoj cifri.

Što se tiče kreiranja taskova, jedina razlika u odnosu na prethodni zadatak je što ovde trećem parametru funkcije `xTaskCreate` tj ID taska prosleđujemo kao argument adrese strukura `tinf1`, `tinf2`, `tinf3`, `tinf4`, umesto konstanti 0,1,2,3 kao u prethodnom rešenju.

Kod za funkciju `only_tsk` gde se implementiraju svi taskovi je izlistan. I ovde se koristi parametar ove funkcije `pvParams` da bi se indetifikovalo koju strukturu je koji task prosledio.

```
static void only_tsk(void* pvParams)
{
	unsigned char cifra = 0;
	taskInfo* tinf_local;
	while (1) {
		tinf_local= (taskInfo*)pvParams;// castujemo iz pointera  void*  u pointer  taskInfo*
		xSemaphoreTake(tinf_local->clk_in, portMAX_DELAY);
		cifra++;
		if (cifra == 10) { // ako je broj dostigao vrednost 10, treba da se inkrementira vrednost
				   //  vise cifre na displeju
			cifra = 0;
			xSemaphoreGive(tinf_local->clk_out);// dajemo takstni signal visoj cifri preko
							    //semafora
		}
		*(tinf_local->disp) = cifra;  // upisujemo cifru u odgovarajuci clan niza dispMem
	}
}
```

Može se videti da je ova funkcija značajno modifikovana u odnosu na prethodni zadatak, pošto za upravljanje ciframa koristimo strukturu. Ipak, princip rada se nije menjao, kada je prvi task aktivan, čeka semafor s1 preko funkcije:

```
xSemaphoreTake(tinf_local->clk_in, portMAX_DELAY);
```

a kada se vrednost cifre koju prikazuje inkrementuje na 10, šalje semafor s2 sledećem tasku:

```
xSemaphoreGive(tinf_local->clk_out);
```

i upisuje vrednost u odgovarajući clan niza `dispMem` :

```
*(tinf_local->disp) = cifra;
```

Timer callback funkcija nije menjana u odnosu na prethodni zadatak.

Kada je drugi task aktivan, čeka se semafor s2, a prosledjuje se s3, itd..

Može se videti da korišćenjem strukture se dobija nešto manji kod nego u prethodnom zadatku, a za kompleksnije kodove ta razlika može biti značajna. Pored toga, grupisanjem u strukturu svih promenjivih koji su na neki način povezane (ovde informacije oko cifara displeja), dobija se logički razumljiviji kod, što je opet vidljivije na kompleksnijim programima.

Rešenje zadatka 1.b):

Kod za gašenje taska sa referencom *tC* je integrisan u `only_tsk` funkciju:

```
static void only_tsk(void* pvParams)
{
	unsigned char cifra = 0;
	taskInfo* tinf_local;
	while (1) {

		tinf_local= (taskInfo*)pvParams;// castujemo iz pointera  void*  u pointer taskInfo*

		xSemaphoreTake(tinf_local->clk_in, portMAX_DELAY);
		cifra++;

		if (cifra == 10) { // ako je broj dostigao vrednost 10, treba da se inkrementira vrednost vise cifre na displeju
			cifra = 0;
			xSemaphoreGive(tinf_local->clk_out);// dajemo takstni signal visoj cifri preko semafora
		}

		*(tinf_local->disp) = cifra;// upisujemo cifru u odgovarajuci clan niza dispMem

		if (dispMem[1] == 9 && tC != NULL) //ako je dostigao broj 90 i ako već nije
							//obrisan
		{
			vTaskDelete(tC); tC = NULL;
		}
	}
}
```

Može se videti da je posle brisanja taska sa `vTaskDelete(tC)` , potrebno obrisati i njegovu refrencu, jer se to ne vrši automatski.

Očitavanje broja aktivnih taskova je realizovano u tajmerskoj callback funkciji, preko funkcije uxTaskGetNumberOfTasks:

```
mxDisp7seg_SelectDigit(myDisp, 4);// selektovan 5 displej
mxDisp7seg_SetDigit(myDisp, character[uxTaskGetNumberOfTasks()]);// ispisujemo broj taskova
```

Može se videti da se na startu na cifri 5 prikazuje da je aktivno 6 taskova(4 taska koje smo kreirali, 1 tajmerski task i 1 idle task), a posle brisanja taska sa referencom tC na cifri 5 displeja prikazuje se broj 5.