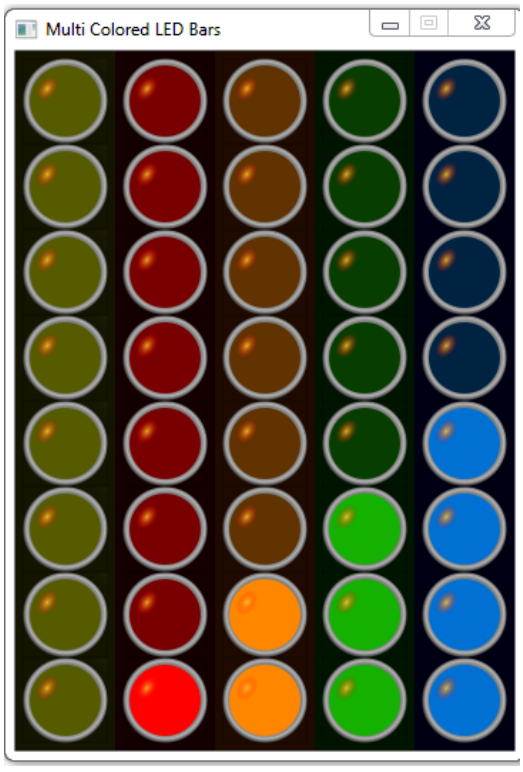
VEŽBA 7 – FreeRTOS

U ovoj vežbi prikazan je rad sa dve dodatne simulirane periferije, LED barom I simulatorom serijske periferije.
Napomena: Koristiti za FreeRTOS solution FreeRTOS_simulator_final.sln i softvere koji simuliraju periferije iz ove lekcije radi realizacije projekata, ne stare verzije koda , bilo je manjih izmena u kodu, ovo je potpuna verzija.

1.1 Led bar

Led bar se pokreće preko aplikacije *LED_bars.exe*, ako se samo pokrene bez ikakvih ulaznih argumenata pojavljuje se 5 stubaca ledovki, u svakom stubcu po 8 LED lampi, kao na sledećoj slici.



Moguće je menjati boju stubac, kao i broj stubca, što se vrši dodeljivanje argumenata prilikom pokretanja programa, npr iz *Command Prompt-a*. Što se tiče boje LED stubca, bira se jedna od 5 boja upisivanje odgovarajućeg karaktera, sledeće opcije su moguće:
Y - žuta
R - crvena

O - narandžasta
G - zelena
B - plava
Zavisno koliko se karaktera navede prilikom pokretanja programa, toliko će biti i stubaca. Maksimum je 8 stubaca. Primer : ako se ukuca **LED_bars.exe RGB** u command prompt, pojaviće se 3 LED stubca, redom crvene, zelene i plave boje. Ako je potrebno da neki stubac bude ulazni, tj da se njegovo stanje može očitavati, potrebno je da argument bude malo slovo. Npr: **LED_bars.exe RgB** će generisati LED bar sa srednjim stubcom (indeksa 1) kao ulaznim, i taj će stubac imati zelenu podlogu.

`int init_LED_comm(void)`	Inicijalizacija led bar. Ako je inicijalizacija uspešna, vraća 0, a ako je neuspešna vraća -1
`int set_LED_BAR(uint8_t b, uint8_t d)`	Kontroliše stanje jednog stubca u LED bara. Parametri: **b** – broj stubca koji se kontroliše, od 0 do 9, zavisno od ukupnog broja stubaca. Prvi stubacsleva ima indeks 0, drugi stubac indeks 1, pa nadalje **d** – podešavanje stanja LEDovki. Gleda se vrednost ovog parametra u bitskom obliku, svaki bit kontroliše odgovarajuću LED-ovku (npr LSB bit kontroliše najdonju LEDovku u redu, a MSB bit najgornju).
`int get_LED_BAR(uint8_t b, uint8_t* d)`	Očitava stanje jednog stubca u LED bara. Parametri: **b**- broj stubca koji se očitava, od 0 do 9, zavisno od ukupnog broja stubaca. Prvi stubac sleva ima indeks 0, drugi stubac indeks 1, pa nadalje **d** – očitavanje stanja LEDovki. Očitava se vrednost ovog parametra u bitskom obliku, svaki LED-ovka setuje stanje odgovarajućeg bita (npr najdonja LEDovka u redu setuje stanje LSB bita, a MSB bita najgornja LEDovka).

Primer korišćenja funkcija za LED bar u kodu.

- Prvo je potrebno inicijalizovati LED bar:

```
init_LED_comm();// inicijalizacija led bara
```

- Led bar ćemo kontrolisati iz zasebnog taska i tajmera

```
/* create a timer task */
        per_TimerHandle = xTimerCreate("Timer", pdMS_TO_TICKS(500), pdTRUE, NULL, TimerCallback);
        xTimerStart(per_TimerHandle, 0);

        /* led bar TASK */
        xTaskCreate(led_bar_tsk, "ST",   configMINIMAL_STACK_SIZE, NULL, SERVICE_TASK_PRI, NULL);
```

- Podešavanje stanja LED-ovki na LEd baru se vrši pomoću funkcije *set_LED_BAR*. Prvi parametar kontroliše broj stubca, od 0 do 9, zavisno od ukupnog broja stubaca. Prvom stubcu sleva pristupa se argumentom 0, pa nadalje.

Drugi parametar kontroliše stanja LEDovki u izabranom stubcu. Gleda se vrednost ovog parametra u bitskom obliku, svaki bit kontroliše odgovarajuću LED-ovku (npr LSB bit kontroliše najdonju LEDovku u redu, a MSB bit najgornju).

```
static void TimerCallback(TimerHandle_t xTimer)
{
        static uint8_t bdt = 0;
        set_LED_BAR(2, 0x00);//sve LEDovke iskljucene
        set_LED_BAR(3, 0xF0);// gornje 4 LEDovke ukljucene

    set_LED_BAR(0, bdt); // ukljucena LED-ovka se pomera od dole ka gore
        bdt <<= 1;
        if (bdt == 0)
                bdt = 1;
}
```

U ovom tajmeru su primeri korišćenja *set_LED_BAR* funkcije. Drugi stubac po indeksu (tj treći stubac sleva) podešava tako da su sve LED-ovke isključene, u trećem stubcu su uključene gornje 4 ledovke, a u nultom stubcu se aktivna LED-ovka pomera sa dole ka gore.

- Očitavanje stanja LED-ovki na LEd baru se vrši pomoću funkcije *get_LED_BAR*. Očitava se vrednost ovog parametra u bitskom obliku, svaki LED-ovka setuje stanje odgovarajućeg bita (npr najdonja LEDovka u redu setuje stanje LSB bita, a MSB bita najgornja LEDovka). Ulazni ledovi su SAMO ulazi, ne mogu se kontrolisati

programski i imaju zelenu podlogu. Svaki ulazni LED se ukljucuje levim klikom miša, iskljucuje se desnim klikom. Stanje SVIH stubaca se moze ocitati. Svaka promena stanja LED-ovki generiše prekid rednog broja 5, te je potrebno podesiti interapt referencu.

```
/* ON INPUT CHANGE INTERRUPT HANDLER */
vPortSetInterruptHandler(portINTERRUPT_SRL_OIC, OnLED_ChangeInterrupt);
/* Create LED interrapt semaphore */
LED_INT_BinarySemaphore = xSemaphoreCreateBinary();
/* led bar TASK */
xTaskCreate(led_bar_tsk, "ST",configMINIMAL_STACK_SIZE, NULL, SERVICE_TASK_PRI, NULL);
```

Preko funkcije `vPortSetInterruptHandler` podešava se da se svaki put kada se klikom miša promeni stanje neke od ulaznih LED-ovki, generiše se interapt broja 5 (što je vrednost makroa `portINTERRUPT_SRL_OIC`), i poziva se funkcija `OnLED_ChangeInterrupt`. Potreban je i binarni semafor radi prosledjivanje informacije iz interapt funkcije.

```
static uint32_t OnLED_ChangeInterrupt(void)
{
	BaseType_t xHigherPTW = pdFALSE;

	xSemaphoreGiveFromISR(LED_INT_BinarySemaphore, &xHigherPTW);

	portYIELD_FROM_ISR(xHigherPTW);
}
```

Može se videti da se u funkciji `OnLED_ChangeInterrupt` samo "daje" semafor `LED_INT_BinarySemaphore`, tj preko ovog semafora javlja se ostatku sistema da se ovaj prekid desio.

U tasku koji se izvršava od funkciji `led_bar_tsk` dat je primer korišćenja `get_LED_BAR` funkcije.

```
void led_bar_tsk(void* pvParameters)
{
	unsigned i;
	uint8_t d;
		while (1)
	{
		xSemaphoreTake(LED_INT_BinarySemaphore, portMAX_DELAY);
				get_LED_BAR(1, &d);
		i = 3;
		do
		{
			i--;
			select_7seg_digit(i);
			set_7seg_digit(hexnum[d % 10]);
			d /= 10;
		} while (i > 0);
	}
}
```

Svaki put kad ovaj task dobije semafor `LED_INT_BinarySemaphore` (tj. kada se desi neka

promena na ulaznim LED-ovkama), funkcija `get_LED_BAR` očita novonastalo stanje na ulaznom stupcu i upiše tu vrednost u promenjivu `d`. Ta vrednost je u opsegu od `0 do 255`, pošto ima 8 ledovki u stupcu, i ona se prikazuje na 3 cifre 7-segmentnog displeja.

1.2 Sedam-segmentni displej

Može se primetiti da se koriste nove funkcije za 7-segmentni displej, a i sam program za 7-seg displej je izmenjen, potrebno je koristiti novi Seg7_Mux.exe ,skinuti iz ove lekcije.

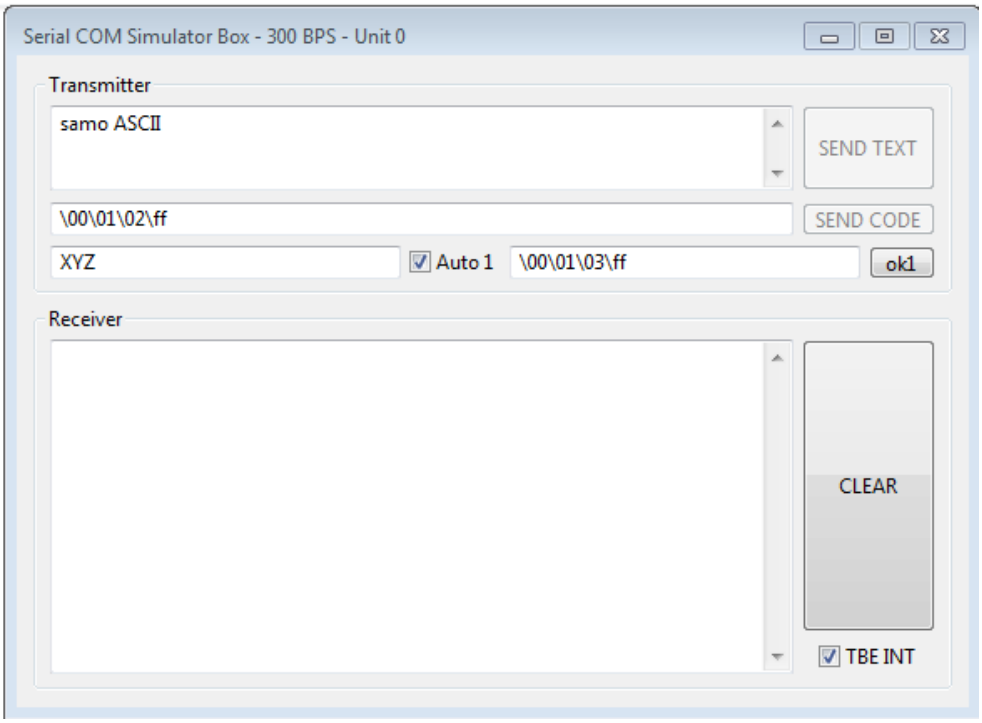
- `select_7seg_digit` umesto `SelectDigit`
- `set_7seg_digit` `umesto SetDigit`

Ove dve funkcije se koriste na isti način kao i prethodne (koje neće raditi sa novim 7 segmentim displejom), samo što imaju po 1 parametar.

- `int select_7seg_digit(uint8_t d)  d-broj cifre koja se setuje, tj bira za upisivanje nove vrednosti`
- `int set_7seg_digit(uint8_t d)  d-vrednost koja se prikazuje na izabranoj cifri`

1.3 Simulacija “serijske” komunikacije

Eksterni softver koji ćemo koristiti za simulaciju serijske je UniCom.exe. Ako se ovaj softver pokrene bez argumenata, pokreće se simulator serijskog terminala na kanalu 0. Ako se doda neki broj kao argument prilikom otvaranja ovog sofvera, pokreće se terminal na kanalu argumenta. Primer: **UniCom.exe 2** pokreće se simulator serijskog terminala na kanalu 2.
Napomena: da bi primeri u ovom kodu radili potrebno je pokrenuti 7-seg displej sa min 7 cifara, tj. iz CMD prompta otkucati **Seg7_Mux.exe 7**



- Prvo je potrebno inicijalizovati “serijsku” komunikaciju, kanal na kom se otvara je određen argumentom koji se prosleđuje init funkcijama:

```
init_serial_uplink(COM_CH);  // inicijalizacija serijske TX na kanalu 0
init_serial_downlink(COM_CH);// inicijalizacija serijske TX na kanalu 0
```

- I slanje i primanje podataka preko “serijske” komunikacije je realizovano preko zasebnih taskova, a potrebni su i odgovarajući semafori. Task za primanje podataka preko serijske ima veći prioritet u odnosu na predajni task, da ne bi doslo do gubljenja podataka prilikom prijema.

```
/* SERIAL TRANSMITTER TASK */
xTaskCreate(SerialSend_Task, "STx", configMINIMAL_STACK_SIZE, NULL, TASK_SERIAL_SEND_PRI, NULL);

/* SERIAL RECEIVER TASK */
xTaskCreate(SerialReceive_Task, "SRx", configMINIMAL_STACK_SIZE, NULL, TASK_SERIAl_REC_PRI, NULL);
r_point = 0;

/* Create TBE semaphore - serial transmit comm */
TBE_BinarySemaphore = xSemaphoreCreateBinary();

/* Create TBE semaphore - serial transmit comm */
RXC_BinarySemaphore = xSemaphoreCreateBinary();
```

Svaka slanje ili primanje podataka sa serijske može da generiše prekide tj interapte, te je potrebno podesiti interapt reference.

```
/* SERIAL TRANSMISSION INTERRUPT HANDLER */
        vPortSetInterruptHandler(portINTERRUPT_SRL_TBE, prvProcessTBEInterrupt);
```

```
	/* SERIAL RECEPTION INTERRUPT HANDLER */
vPortSetInterruptHandler(portINTERRUPT_SRL_RXC, prvProcessRXCInterrupt);
```

Može se videti da se, kao i kod led bara, u interapt funkcijama `prvProcessTBEInterrupt` i `prvProcessRXCInterrupt` samo "daje" odgovarajući semafor `TBE_BinarySemaphore` i `RXC_BinarySemaphore`, tj. preko ovih semafora javlja se ostatku sistema da su se prekid desio.

```
/* TBE - TRANSMISSION BUFFER EMPTY - INTERRUPT HANDLER */
static uint32_t prvProcessTBEInterrupt(void)
{
	BaseType_t xHigherPTW = pdFALSE;

	xSemaphoreGiveFromISR(TBE_BinarySemaphore, &xHigherPTW);

	portYIELD_FROM_ISR(xHigherPTW);
}

/* RXC - RECEPTION COMPLETE - INTERRUPT HANDLER */
static uint32_t prvProcessRXCInterrupt(void)
{
	BaseType_t xHigherPTW = pdFALSE;

	xSemaphoreGiveFromISR(RXC_BinarySemaphore, &xHigherPTW);

	portYIELD_FROM_ISR(xHigherPTW);
}
```

Funkcija u kojoj se izvršava task sa slanjem podataka preko "serijske" je predstavljena.

```
void SerialSend_Task(void* pvParameters)
{
	t_point = 0;
	while (1)
	{

		 if (t_point >(sizeof(trigger) - 1))
		t_point = 0;
		 send_serial_character(COM_CH, trigger[t_point++]);
		 xSemaphoreTake(TBE_BinarySemaphore, portMAX_DELAY);// kada se koristi predajni interapt
		 //vTaskDelay(pdMS_TO_TICKS(100)); // kada se koristi vremenski delay	}
}
```

Može se videti da ovaj task ponavlja karaktere iz niza `trigger`, tj XYZ. Svaki put kada se pozove funkcija `send_serial_character`, šalje se jedan od ovih karaktera, i onda ovaj taske čeka da dobije semafor `TBE_BinarySemaphore` iz predajne interapt funkcije `prvProcessTBEInterrupt`. Ovaj interapt se dobija od terminal serijskog simulatora, ako je uključena kvačica kod TBE INT (ispod CLEAR tastera). U slučaju da je uključen TBE INT, ovaj terminal generiše interapt 30 ms posle primaja karaktera preko kanala serijskog simulatora, da bi simulirao ograničenje u brzini u realnim serijskim komunikacijama. Nije neophodno koristititi ovaj interapt, može se koristiti i vremenski delay, u tom slučaju je

potrebno zakomentarisati `xSemaphoreTake` funkciju, i odkomentarisati `vTaskDelay`.

- Primanje podataka preko “serijske” komunikacije je realizovano u tasku koji se izvršava u sledećoj funkciji :

```
void SerialReceive_Task(void* pvParameters)
{
        uint8_t cc = 0;
        static uint8_t loca = 0;
        while (1)
        {
                xSemaphoreTake(RXC_BinarySemaphore, portMAX_DELAY);// ceka na serijski prijemni interapt
                get_serial_character(COM_CH, &cc);//ucitava primljeni karakter u promenjivu cc
                printf("primio karakter: %u\n", (unsigned)cc);// prikazuje primljeni karakter u cmd prompt

                if (cc == 0x00) // ako je primljen karakter 0, inkrementira se vrednost u GEX formatu na
                                //ciframa 5 i 6
                {
                        r_point = 0;
                        select_7seg_digit(5);
                        set_7seg_digit(hexnum[loca >> 4]);
                        select_7seg_digit(6);
                        set_7seg_digit(hexnum[loca & 0x0F]);
                        loca++;
                }
                else if (cc == 0xff)// za svaki KRAJ poruke, prikazati primljenje bajtove direktno na
                                    //displeju 3-4
                {

                        select_7seg_digit(3);
                        set_7seg_digit(r_buffer[0]);
                        select_7seg_digit(4);
                        set_7seg_digit(r_buffer[1]);
                }
                else if (r_point < R_BUF_SIZE)// pamti karaktere izmedju 0 i FF
                {
                        r_buffer[r_point++] = cc;
                }
        }
   }
```

U ovom primeru realizovano je primanje i parsiranje jednostavnog paketa. Rec 0x00 se koristi kao sinhronizacija, a 0xFF kao oznaka kraja paketa. Podaci se prikazuju nakon pristizanja oznake kraja. Bajtovi podataka su svi razliciti od 0x00 i 0xFF izmedju. Paket očekuje 4 bajta, sinhronizacioni (0x00), dva proizvoljna bajta i KRAJ (0xFF).
U terminalu simulatora serijski podaci u različitim formatima se mogu salti u polju pored dugmeta SEND CODE. Kada se ispred dva broja postavi karakter \, podrazumeva se da sledi hexdecimalni broj, a ako ispred broja ili karaktera ne stavi ništa, pdorazumeva se ASCII karakter. Primeri:

- \00\01\02\ff poslati 0x00, 0x01,0x02, 0xff
- 12\ff poslato acii 1 i 2 (tj dec 49 I 50) , te hex 0xff

Ako je potrebno slati samo ASII karaktere, moguće je koristiti veće polje pored SEND TEXT.

Dva bajta paketa između 0x00 i 0xff direktno se prikazuju na prve dve cifre 7-seg displeja. Druge dve cifre prikazuju hex brojac koji odbroji svaki put kad se detektuje sinhronizaciona rec pristiglog paketa.

Ako je potrebno da terminal automatski šalje podatke, moguće je koristiti AUTO mod terminal. Ako terminal primi odgovarajući triger tj izabrani niz karaktera, upisan u polje levo od **Auto 1**, i ako je uključena kvačica **Auto 1,** terminal će automatski slati karaktere upisane desno od polja **AUTO 1** (ako se menajju ti karakteri potrebno je pritisnuti dugme OK). U slici terminal gore podeseno je da triger bude **XYZ,** svaki put kada terminal primi ovaj niz karatera on šalje podatke \00\01\01\ff (ako je uključena kvačica **Auto 1**).