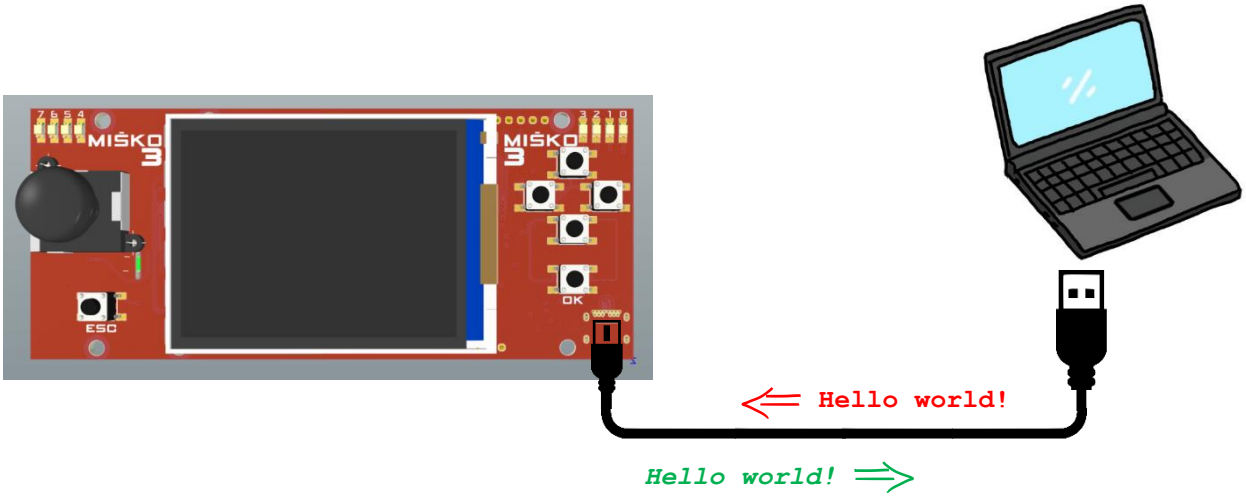
Osnove mikroprocesorske elektronike

Vaja 7: Serijski vmesnik SCI – "polling"

V sklopu te vaje bomo *na podlagi USART periferne enote* implementirali *splošno-namenski serijski vmesnik* SCI (angl. Serial Communication Interface) na *sistemskem* nivoju. Tak *splošno-namenski* serijski vmesnik je močno orodje, saj ga lahko uporabimo za mnogo različnih stvari. Serijski vmesnik lahko uporabljamo za *komunikacijo z drugimi vgrajenimi sistemi, napravami oziroma moduli*, ki za komunikacijo uporabljajo *tekstovne ASCII protokole* (npr. s SIM modulom preko AT protokola, z GPS modulom preko NMEA protokola, z laboratorijskimi instrumenti preko SCPI protokola ipd.). Zelo pogosto pa se serijski vmesnik uporablja kot *diagnostično orodje* pri razvoju vgrajenega sistema tako, da vgrajeni sistem preko SCI vmesnika na računalnik pošilja sporočila, ki vam *v realnem času pomagajo preveriti delovanje vašega vgrajenega sistema* (npr. rezultate meritev, stanja sistema ipd.). In na tak način ga bomo uporabili tudi mi v testnem delu te vaje.



Slika 1 - pri tej vaji oživimo serijski vmesnik USART, ki bo omogočal komunikacijo z računalnikom preko USB povezave

Naloge vaje

Implementirajte *splošno-namenski serijski vmesnik SCI na podlagi USART periferne enote* tako, da izpolnite sledeče naloge:

1. ***Definirajte podatkovno strukturo, ki bo hranila vse potrebne parametre za delo s serijskim vmesnikom SCI.***

   Definirali jo boste s pomočjo naslednjih korakov:

   a) definirajte *tip* strukturne spremenljivke `SCI_handle_t`, ki naj hrani vse potrebne parametre za implementacijo SCI vmesnika na podlagi USART periferne enote

   b) na podlagi zgornjega tipa definirajte *globalno* spremenljivko `SCI`, ki pa bo naša "handle" struktura za SCI vmesnik

2. ***Inicializirajte "handle" strukturo*** `SCI` ***znotraj*** `SCI_init()` ***funkcije.***

   Pri inicializaciji "handle" strukture boste poskrbeli:

   c) da se specificira, katero USARTx periferno enoto bomo uporabili za implementacijo SCI vmesnika

3. ***Implementirajte sistemske funkcije za delo s serijskim vmesnikom SCI.***

   Do konca boste implementirali boste sledeče sistemske funkcije:

a) funkcije za *pošiljanje* podatkov

- `SCI_send_char()` – funkcija za pošiljanje enega samega *znaka* preko SCI vmesnika
- `SCI_send_byte()` – funkcija za pošiljanje enega samega *bajta podatkov* preko SCI vmesnika
- `SCI_send_string()` – funkcija, ki pošlje *znakovni niz* preko SCI vmesnika
- `SCI_send_bytes()` – funkcija, ki pošlje zaporedje binarnih podatkov preko SCI vmesnika

b) funkcije za *sprejemanje* podatkov

- `SCI_is_data_waiting()` – funkcija, ki preveri, če je v SCI vmesniku na voljo nov prejeti podatek
- `SCI_read_char()` – funkcija, ki *poskusi* prebrati en *znak* iz SCI vmesnika in ga shraniti
- `SCI_read_byte()` – funkcija, ki *poskusi* prebrati en *bajt podatkov* iz SCI vmesnika in ga shraniti

c) *priredite uporabo* `printf()` funkcije tako, da bo sporočila pošiljala preko SCI vmesnika

V sklopu implementacije sistemskih funkcij za delo s serijskim vmesnikom SCI boste poskrbeli tudi za *prireditev uporabe* standardne funkcije `printf()` (angl. `printf()` customization). Namreč, funkcijo `printf()` bomo priredili tako, da bo svoja *formatirana sporočila* izpisovala v SCI serijski vmesnik. To boste storili tako, da boste re-definirali posebno funkcijo `_write()`, ki določi, na kakšen način funkcija `printf()` pošilja svoja sporočila.

Dodatno pojasnilo: spodaj se lahko na kratko podučite o t. i. "`newlib`" *standardni C knjižnici*, ki je namenjena programiranju vgrajenih sistemov. "`newlib`" knjižnica je namreč tista, ki nudi enostavno modifikacijo uporabe `printf()` funkcije preko `_write()` in `setvbuf()` funkcij.

4. ***Spišite testno kodo***, ki bo *preizkusila* in hkrati *demonstrirala* sistemske funkcije za delo z SCI vmesnikom.

V tem delu boste implementirali tri testne funkcije:

a) `SCI_demo_Hello_world()` – funkcija, ki preko SCI vmesnika pošlje sporočilo "`Hello world!`"

b) `SCI_demo_Hello_world_printf()` – funkcija, ki preko SCI vmesnika pošlje sporočilo "`Hello world!`" s pomočjo prirejene funkcije `printf()`

c) `SCI_demo_echo_with_polling()` – funkcija, ki naj demonstrira t. i. "*echo*" funkcionalnost s *pristopom poizvedovanja* (angl. *polling*)

Dobro preučite, kaj ta funkcija počne. Ta funkcija bo namreč ključna za *preučevanje slabosti, ki jo prinese pristop poizvedovanja*, kar pa boste storili v naslednji, zadnji točki vaje.

Vse testne funkcije boste uporabljali v navezi s *terminalskim programom* HTerm, ki vam omogoča povezavo računalnika in Miškota preko serijskega vmesnika.

5. ***S pomočjo testne funkcije*** `SCI_demo_echo_with_polling()` ***pokažite slabost***, ki jo pri sprejemanju podatkov preko serijskega vmesnika prinese *pristop poizvedovanja* ("polling").

To boste storili tako, da boste z računalnika preko terminala pošiljali Miškotu različna sporočila in opazovali, kaj Miškotov "echo" pošlje nazaj na računalnik. Poskusite postopno večati dolžino sporočila. Za testna sporočila lahko uporabite:

a) en sam znak, npr. "`a`"
b) eno samo kratko besedo, npr. "`abc`"
c) daljši stavek, npr. "`Brez muje se še čevelj ne obuje.`"

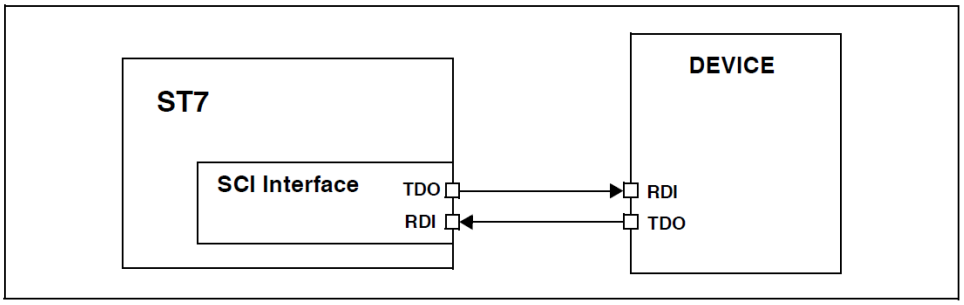
Opazujte, kakšno sporočilo sprejmete nazaj na računalnik. Premislite, kaj se dogaja.

Dodatna pojasnila

Kaj pravzaprav je SCI vmesnik?

Kot že rečeno, SCI kratica pomeni "Serial Communication Interface". Če pobrskate za tem terminom, boste ugotovili, da obstaja oziroma je nekoč obstajal modul oiroma periferna enota za serijsko komunikacijo, ki se je imenuje SCI (poglejte spodaj). Taka SCI periferna enota močno spominja na serijski vmesnik UART.

Figure 1. ST7 and SCI interface set-up



Slika 2 – z besedo "SCI vmesnik" lahko mislimo dejansko periferno enoto za serijsko komunikacijo (vir: AN969 - SCI Communication Between ST7 and PC (STM32))

Vendar mi termina "SCI vmesnik" ne uporabljamo v tem smislu. Mi bomo ta termin uporabljali tako, da *z njim poimenujemo sistem, ki je sposoben preko serijske komunikacije komunicirati* s preostalimi napravami (poglejte izsek spodaj).

> A serial communications interface (SCI) is a device that enables the serial (one bit at a time) exchange of data between a microprocessor and peripherals such as printers, external drives, scanners, or mice. In this respect, it is similar to a serial peripheral interface (SPI). But in addition, the SCI enables serial communications with another microprocessor or with an external network. The term SCI was coined by Motorola in the 1970s. In some applications it is known as a universal asynchronous receiver/transmitter (UART).

Dogovor: termin "SCI vmesnik" bomo mi torej uporabljali za poimenovanje *splošno-namenskega serijskega vmesnika na sistemskem nivoju*, ki pa je lahko na *nizko-nivojskem strojnem nivoju* implementiran z različnimi perifernimi enotami:

- SPI (Serial Peripheral Interface)
- RS232 (Recommended Standard 232)
- RS485
- USART
- ipd.

V našem primeru bomo sistemski SCI vmesnik implementirali na podlagi USART periferne enote mikrokrmilnika.

Razdelitev na tri programske nivoje na primeru vaje z SCI vmesnikom

Da ponazorimo delitev na tri programske nivoje še na primeru tokratne vaje s serijskim vmesnikom:

- funkcije strojnega nivoja bomo uporabili *za nizko-nivojsko upravljanje USART periferne enote,*
- sistemski nivo bo USART periferno enoto uporabil tako, da bo *v sistemu implementiral splošno-namenski serijski vmesnik* za serijsko komunikacijo z drugimi napravami
- aplikacija na koncu semestra pa bi na primer lahko serijski vmesnik SCI uporabila *za izvedbo možnosti igranja igrice v načinu "dual-player"*

Opaziti je potrebno še, kako na poti od strojnega nivoja proti aplikacijskemu nivoju dobiva strojna oprema oziroma sistemski modul *čedalje bolj specifičen namen*. Poglejte spodaj.

strojni nivo	**sistemski nivo**	**aplikacijski nivo**
USART periferna enota	SCI vmesnik	"dual-player" mode igranja preko SCI vmesnika

zelo nizek nivo, zelo blizu strojni opremi mikrokrmilnika;

splošen namen uporabe (npr. USART vmesnik je lahko uporabljen za komunikacijo z GPS modulom, SIM modulom ipd.)

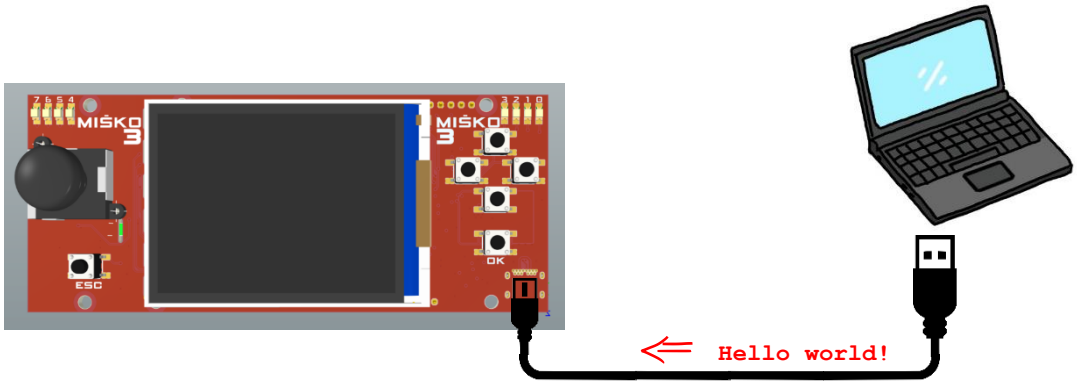
s pomočjo strojnega nivoja smo v sistemu *implementirali možnost serijske komunikacije*

modul je sedaj ***na voljo*** za uporabo, vendar pa ***še nima specifičnega namena uporabe***

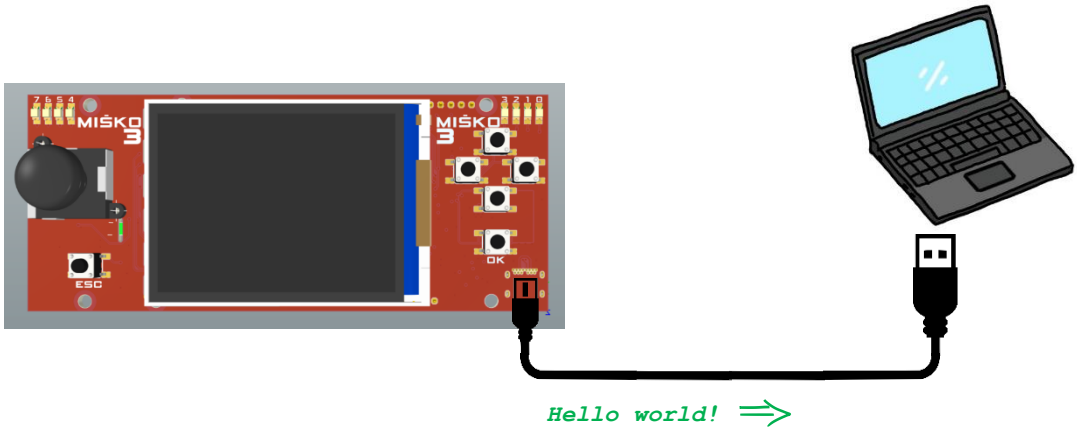
aplikacija uporabi sistemski modul – SCI vmesnik ***za točno določen namen***, ki je aplikaciji pač potreben

Komunikacija v "echo" načinu

Pri prenašanju sporočil preko komunikacijskega kanala je včasih ugodno uporabljati t. i. echo način (angl. echo = odmev), kjer *sprejemna naprava sprejme sporočilo in nato pošlje to isto sporočilo nazaj pošiljatelju*. Na ta način lahko izvajamo detekcijo napak pri prenosu sporočila. Poglejte idejo spodaj.



Slika 3 - računalnik pošlje sporočilo napravi



Slika 4 - in naprava mu odgovori tako, da sprejeto sporočilo pošlje nazaj ("izvede echo")

Če je poslano sporočilo enako "echo sporočilu", potem je prenos uspel; če se pa sporočili razlikujeta, pa imamo očitno napake pri prenosu sporočil.

Postopek pošiljanja in sprejemanje podatka preko USART vmesnika

Za v pomoč pri razumevanju in programiranju SCI funkcij:

Pošiljanje podatka

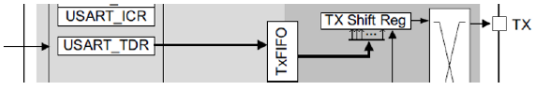
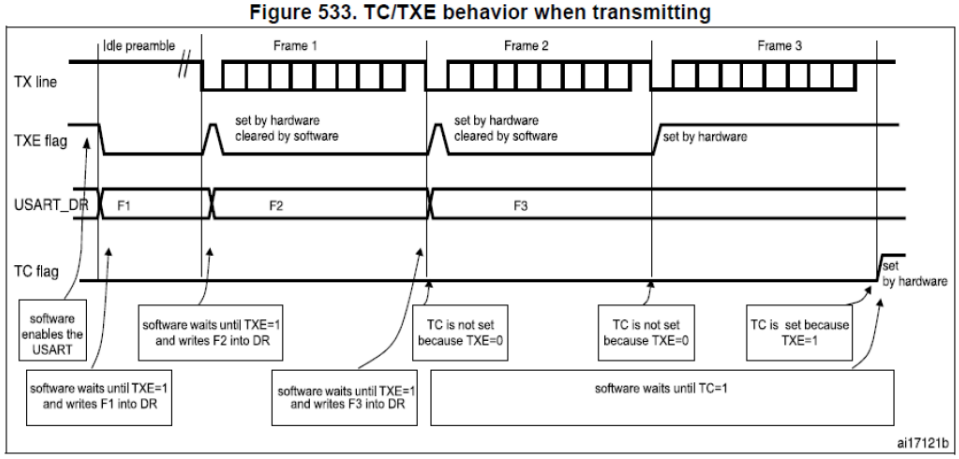


Figure 533. TC/TXE behavior when transmitting



- Omogoči oddajnik (**TE = 1**)
- Preveri, če je **USART_TDR** prazen (**TXE==1**?)
- Vpiši podatek v **USART_TDR**
  - Avtomatsko postavi **TXE = 0**
  - Ko se podatek dejansko pošlje se postavi **TC = 1**

Sprejem podatka

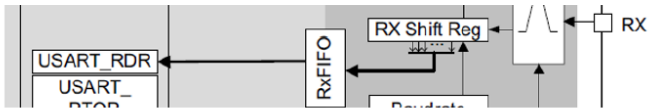
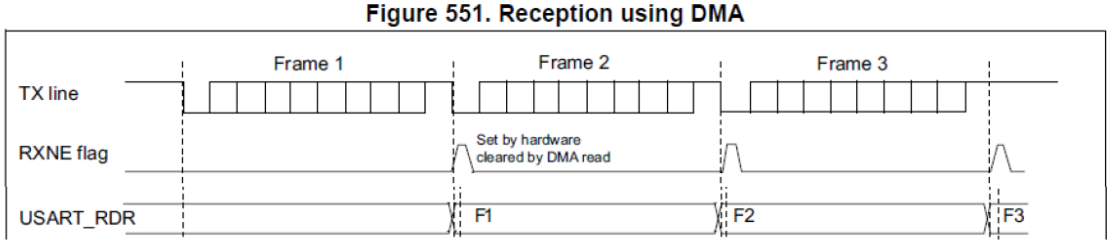


Figure 551. Reception using DMA



- Omogoči sprejemnik (**RE=1**)
- Preveri, če je v **USART_RDR** nov podatek (**RXNE==1**?)
- Preberi podatek iz **USART_RDR**
  - Avtomatsko postavi **RXNE=0**

Poizvedovanje - polling

Standardna C knjižnica `newlib`

Tako imenovana *standardna C knjižnica* `newlib` je nekakšna *standardna zbirka makrojev, definicij in funkcij* za:

- delo z znakovnimi nizi (npr. `strcpy()`)
- delo z matematičnimi operacijami (npr. `sin()`)
- obdelovanje vhodnih/izhodnih podatkovnih tokov (npr. `fputs()`)
- upravljanje pomnilnika (npr. `malloc()`)
- ipd.

Knjižnica `newlib` se od preostalih standardnih C knjižnjic razlikuje v tem, da je namenjena programiranju vgrajenih sistemov (angl. embedded systems). Poglejte izseka spodaj.

1.4.1 The C library (newlib)

newlib implements a version of the C library that is suitable for use in embedded systems. **newlib** supports the most common functions used in C programs, but not the more specialized features available in standard operating systems, such as networking support.

Note: *Wide character support is not enabled in the supplied version of **newlib**.*

newlib assumes a minimal set of OS interface functions (the **syscalls** API). These provide all the I/O, entry and exit, and process control routines required by programs using **newlib**. The syscalls API is implemented either by the **libdtf** DTF library (for ST40 simulator or silicon) or by the **libgloss** library (for GDB simulator).

Slika 5 - vir: UM1399 - ST40 Micro Toolset

2.5.1 Run time library

The toolchains included in STM32CubeIDE contain two prebuilt run time C libraries based on `newlib`. One is the *standard C* `newlib` library and the other is the *reduced C* `newlib-nano`. Use `newlib-nano` to achieve smaller code size. For information about the differences between `newlib-nano` and the standard `newlib`, refer to the `newlib-nano` readme file ([ST-09]), accessible from the *Information Center*.

To select the desired run time library for use in the project.

1. Right-click on the project in the *Project Explorer* view
2. Select menu [**Project**]>[**Properties**]
3. Select [**C/C++ Build**]>[**Settings**] in the *Properties* panel
4. Open the *Tool Settings* tab, select [**MCU Settings**] and configure the [**Runtime library**] setting

Slika 6 - vir: UM2609 - STM32CubeIDE user guide

Če vas knjižnica podrobneje zanima, lahko dokumentacijo v zvezi z `newlib` knjižnico najdete v mapi "dodatno".