Embedded and Ambient Systems

Topic of exercise 3

During the exercise, we will use one of the microcontroller's serial peripherals (UART0) to send and receive characters. We will make some of the necessary settings at the register level. However, unlike the previous practice, this time the register-level access will be assisted by the C language header files that are part of the SDK released for the microcontroller. Most of the necessary settings will be made even more conveniently using emlib, which is also part of the SDK. This is a firmware library with functions for managing the peripherals of the microcontroller.

The UART0 peripheral of the microcontroller is accessible for the computer through the Board Controller, which also includes the debugger circuit, where it appears as a virtual COM port. By default, however, the connection between the UART0 peripheral and the Board Controller is not establed. Enabling the connection can be done by setting the PF7 pin into high logical level. So this is a configuration step that has nothing to do with the settings of the UART0 peripheral, but we have to do it anyway.

Task 1

Let's enable the connection between the UART0 peripheral of the microcontroller and the Board Controller! For this purpose, we use the header files available in em_device.h. Since we can adjust the PF7 pin using the GPIO peripheral, we must first ensure that it receives a clock signal!

Hint: during the task, we use the CMU and GPIO pointers available by em_device.h – which point to structures that enable CMU and GPIO peripheral settings - as well as the #defines that produce the appropriate bit mask, which can also be accessed using the above-mentioned header.

The steps to be taken are therefore:

1. **Enable GPIO clock**
   a. **Enable high frequency peripheral clock (HFPERCLK) in general**, because this is what the GPIO also receives (this is enabled after reset, so this step can be skipped). To enable it, the HFPERCLKEN bit of the HFPERCLKDIV register of the CMU peripheral must be set to High.
   b. **Specifically enabling the clock signal going to the GPIO peripheral**. Here, the GPIO bit of the HFPERCLKEN0 register of the CMU peripheral must be set to High.
2. **Set PF7 high**
   a. To do this, you must first define the given pin as an output. Here, the appropriate setting of the MODEL register belonging to port F of the GPIO peripheral is required. Hint: since the GPIO peripheral manages several ports (A-F), the registers that are required to manage the specific ports (and not the entire GPIO peripheral in general) are located in the P array of the structure pointed to by the GPIO pointer. The individual elements of the array are also structures that allow access to the registers required to set the given port. Since in C we index with a number, starting with 0, A-F ports have indexes 0-5. Furthermore, out of the two possible MODEx registers (MODEL and MODEH), you need MODEL because it contains the settings for pins 0-7 (while MODEH is responsible for configuring pins 8-15).
   b. **After that, we need to make sure that PF7 is set to a high value**. This can also be done with the DOUT and DOUSET registers (the latter results in more efficient code).

Task 2

Configuring the UART0 peripheral for 115200 Baud rate and 8N1 frame format (i.e. 8 data bits, no parity, 1 stop bit). The pins used are PE0 (TX, transmit) and PE1 (RX, receive), since these pins are wired on the STK3700 card.

To complete this task, we will use the functions provided by emlib. Since in addition to UART0, the correct setting of the CMU and GPIO peripherals is also required to solve the problem, we will need three modules of emlib: USART[1], CMU and GPIO. The three modules have three source and three header files: em_<module name in lower case>.(c|h).

Hint: the header files can be used by inserting the appropriate #include lines, while the source files must be added to the project. The easiest way to do the latter is to select "Browse Files Here" from the menu that appears when you right-click on the emlib/em_system.c file that already exists in the project. Provided that the project was not created by selecting the "Copy contents" option, only links from the project point to the source files of the libraries used, and consequently the file browser displayed by clicking on the above menu item points to the location of the emlib files. From there, we can drag and drop the necessary files into the project with a simple Drag & Drop operation. In the pop-up dialog window, choose to link the file (compared to the STUDIO_SDK_LOC environment variable).

The steps to be taken:

1. **First, a clock signal must be provided to the UART0 peripheral**. For this, we can use the CMU_ClockEnable() function (predefined enum values are available to specify the clock signal to be enabled).
2. **After that, the requested operating mode must be configured**.
    a. **To do this, a USART_InitAsync_TypeDef** type structure has to be created to store the configuration parameters.
    b. **The elements of the structure must then be filled with the appropriate values:**
        i. **baud rate**: can be entered with a number
        ii. **refFreq**: 0 (to generate the baudrate, the frequency of the default HFPERCLK clock signal of UART0 is considered as a reference)
        iii. **databits**: to enter, use the appropriate enum value and not a number, that will not be good!
        iv. **parity**: choose the appropriate enum value
        v. **stop bits**: choose the appropriate enum value
        vi. **oversampling**[2]: select the enum value for 16x oversampling
        vii. **mvdis**[3]: should be false (we don't want to disable majority voting)
        viii. **prsRxEnable**: false (we don't want to use PRS[4]system)
        ix. **prsRxCh**: should be 0 (although the value may be don't care if prsRxEnable is false)
        x. **enable**: choose the enum value that enables both the receiving and transmitting circuit
    c. After filling out the configuration structure, use the USART_InitAsync() function to initialize the UART0 peripheral using the above structure! (Note: for didactic purposes, we have now filled in all configuration options. In general, however, there are #defines loaded with default values to facilitate the filling of configuration structures - in our case, USART_INITASYNC_DEFAULT can now be used for this purpose. It is worth initializing the structure with this when creating it, and then only it is necessary to rewrite the parameters where we want to set the peripheral differently than the default case.)

[1]There are basically two types of simple serial communication protocols: UART (Universal Asynchronous Receiver / Transmitter) and USRT (Universal Synchronous Receiver / Transmitter). The difference between the two is that while the latter uses a dedicated clock signal, the former does not. Since their operation is the same apart from this, in microcontrollers we often find peripherals that can operate in either of the two modes, depending on the setting, and are therefore called USART. Since a clock signal is not required in most cases, we often find peripherals operating only in UART mode in microcontrollers. The device we use has both. Since their handling is very similar, emlib provides only one module called em_usart, which is designed to handle both types of peripherals.

[2]USART receiver circuits usually take multiple samples from one bit (16 by default).

[3]Among the samples taken from one bit, the middle three are usually used as a basis, and a majority vote among them decides if the value is not clear.

[4]PRS (Peripheral Reflex System) is a special data transmission channel of Gecko microcontrollers, where these peripherals can communicate with each other, autonomously, without the intervention of the CPU.

3. After setting the requested mode, the signals of the UART0 peripheral must be output to the pins to be used.
    a. As a first step, using the GPIO peripheral, pin PE0 must be set as output (with an initial value of 1, since this is the rest state of UART communication) and pin PE1 as input. This is necessary because the GPIO peripheral is responsible for setting the mode of the pins, even if they are not used. The function to be used is GPIO_PinModeSet(). The port can be selected using the appropriate enum, while the pin can be chosen by a simple number. From the several possible output and input modes, select push-pull as output, and simple input mode as input (with the help of the appropriate enum). (The value to be written in the output register in the case of output will be the initial value of the pin, so for PE0, choose this as 1! In the case of input, the value that can be written in the output register controls whether an optional filter circuit should be switched on, which tries to get rid of the smaller spikes or not. It is not needed here, so the value of the output register should be 0 for PE1.)
    b. After the pins to be used have been set, the UART0 peripheral must be requested to use them (several possible configurations can be set to assign the RX, TX signals to physical pins). Unfortunately, there is currently no emlib function for this, so we are forced to solve it by register-level access. To complete the task, the appropriate setting of the ROUTE register of the UART0 peripheral is required. The pins we want to use have the "Location 1" output mode. So let's select this with the appropriate #define. Make sure that choosing the right "Location" is not enough in itself, because you also have to say which signals you want to output (of course both). Thus, the enabling bits for the RX and TX signals must also be toggled in the ROUTE register.

Task 3
Let's send a character to the UART0 peripheral! The USART_Tx() function can be used for this.

Hint: in order to display the sent character, we need a so-called terminal program that prints the characters received on the serial port to the screen (and the characters typed). To do this, we first need to know which serial port number belongs to the Gecko board. We can do this in Windows using the Device Manager (under "Ports (COM & LPT)", look for the entry "JLink CDC UART PORT"). A possible terminal program is PuTTY. Set this to monitor the corresponding serial port (with the same communication parameters as the UART0 peripheral)!

Task 4
Let's continuously receive characters from the UART0 peripheral and then send them back! The USART_Rx() function can be used to receive characters.