Creating a driver

In Assignment 2 you are going to create your own driver routines to access the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) port of the AT32UC3A1512 (see Chapter 26 of the datasheet). This will later allow you to send and receive data to and from a PC with serial port. Having such functionality is especially important during the design phase of the embedded system because it allows to transmit various kinds of information (system state, value of variables, error messages, etc.) which is otherwise not accessible.
A driver can be implemented based on polling or based on interrupts. As you know from the lecture, both implementation variants have their pros and cons.
Polling based implementations are simple to implement and the interrupt mechanism does not need to be set up. Main goal of this assignment will therefore be a implementation of a USART driver in polling mode.

Requirements

- The driver shall have the following interface:
  - `void USART_init(volatile avr32_usart_t * usart)`
    This Function initializes the USART driver, here we set up the hardware module.
  - `char USART_getChar()`
    This Function is used to receive one character.
  - `void USART_putChar(char c)`
    This function is used to transmit one character.
  - `void USART_reset()`
    This function resets the driver to a save state (reset all registers that could lead to unpredictable behavior).

  The parameter `usart` points to the USART datastructure of the AT32UC3A1512. This is useful because the processor implements multiple USART modules. The parameter `c` is the character the function transmits.

- The USART driver shall operate at **9600 baud**. It is not necessary to make the baudrate configurable.

- The functions `USART_getChar()` and `USART_putChar(char c)` should not try to access the hardware if the driver was not initialized first.

- A driver should be portable and only the interface functions should be accessible from other parts of the code.
- Packet length should be **8 bit**.
- The driver should sent **no parity bit**.

- The driver should send **one stop bit**.
- The driver should operate in **normal channel mode**.

Solution hints

- Read Chapter 26 of the datasheet thoroughly. The USART module is complex and supports multiple transmission protocols, you will not need all parts. Once you understand which parts (e.g. which registers) you need to configure you should proceed.
- Create a new project. This is done the same way as in Assignment 1.
- In order to have a portable driver you can create a C code file and a header file. You then define all functions which are public, e.g. the interface functions in the header file. Functions which are not meant to be accessible from other modules are defined in the C code file. As always all function bodies are in the C code file. If any other code part wants to access your functionality they need to include your header file. Since the interface functions are defined here they can be accessed, all other functions of your driver are hidden.
- Steps you need to do in the initialization:
  - Reset the driver to avoid unpredictable behavior during initialization.
  - Set the baudrate.
  - Set the message length.
  - Initialize the stop bit.
  - Set the driver to *normal mode*.
  - Enable the communication.
- See the processor datasheet **26.7.1**.

Assignment

Create a driver which fulfills the above requirements. To have a higher level of usability you should define and implement a function which is able to send strings, and one function which is able to receive strings.