1 Introduction

The goal of this assignment is to create your own driver routines to access the Universal Synchronous/Asynchronous Receiver/Transmitter (UART) port of the Tiva™ TM4C129ENCPDT Microcontroller (see Chapter 19 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 an implementation of a UART driver in polling mode.

2 Driver's Requirements

1. The driver shall have the following interface:
   - *A.* *void UART _init(uint32_t ui32Base)*
     This Function initializes the UART driver, here we set up the hardware module.
   - *B.* *char UART_getChar()*
     This Function is used to receive one character.
   - *C.* *void UART putChar(char c)*
     This function is used to transmit one character.
   - *D.* *void UART reset()*
     This function resets the driver to a save state (reset all registers that could lead to unpredictable behavior).

   You should also implement the following two functions to improve usability of your drivers:

   - *E.* *void UART putString()*
     This function uses the putChar function to write a string
   - *F.* *void UART getString()*
     This function uses the getChar function to read a string.

   The parameter *UART* points to the UART data structure of the TM4C129ENCPDT. This is useful because the processor implements multiple UART modules. The parameter c is the character the function transmits.

2. The UART driver shall operate at 9600 baud. It is not necessary to make the baud rate configurable.
3. The functions *UART getChar()* and *UART putChar(char c)* should not try to access the hardware if the driver was not initialized first.
4. A driver should be *portable* and only the interface functions should be accessible from other parts of the code.

5. Packet length should be *8 bits*.
6. The driver should send *no parity bit*.
7. The driver should send *one stop bit*.
8. The driver should operate in *normal channel mode*.

For this lab it is mandatory that you use Direct Register Access Model. Re-visit section 2.2 in TivaWare Peripheral Driver Library and follow the examples with programming the CR0 register in the SSI module. That is how you should program your uart drivers (A,B,C and D). Your uart driver should only have the two following includes:

```
#include <tm4c129encpdt.h>
#include <stdint.h>
```

And your main should only include your driver, nothing more!

3 Solution hints

- Read Chapter 19 of the TM4C129ENCPDT datasheet thoroughly. The UART 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:
    1. Reset the driver to avoid unpredictable behavior during initialization.
    2. Set the baud rate.
    3. Set the message length.
    4. Initialize the stop bit.
    5. Set the driver to normal mode.
    6. Enable the communication.
    7. Enable the digital register bit for the GPIO pins, page 789
- See the microcontroller datasheet section 19.4.

Note: The step 7 in the initialization in chapter 19 is missing

A tip: CCS has a terminal which can communicate with the TivaWare. Select Launch Terminal/Open Terminal and select Serial. This is simpler than to use a second program (Teraterm etc.) to read/write via the uart.

4 Assignment (2 bonus points if completed before the deadline, which is specified in Canvas for each assignment)

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

Note: The bonus points collected after successfully completing the compulsory assignments before the corresponding deadlines will be added to your score in the final exam. These bonus points could help you in improving your final grade in the course.