A7 Assignment – ESE516-SPRING 2022

DUE DATE: March 28th 2022 before 11:59pm EST. To be submitted on your Google Drive Folder Please deliver on a folder called A7!

1.) Understanding the Starter Code – answering questions [25 points]

Please download the starter code available at Google Drive:

The starter code is an Atmel Studio 7 project for your SAMW25 that does the following:

- Initialize the UART at 115200 8N1
- Starts the FreeRTOS kernel.
- Reads characters using interrupts from the Uart and adds them to a ringbuffer
    - You can read a character from this ring buffer with the function "SerialConsoleReadCharacter()"
- The system echoes the characters the user writes, so they can read what they are typing!
- Writes characters from another ringbuffer to be sent back via the UART
    - You can send characters by adding them to the buffer using the function "voidSerialConsoleWriteString(char * string)"

Once you downloaded the code, please go through the code. In particular, focus on how the device is reading characters and writing characters.

Please answer the following questions and SUBMIT IN A WORD DOCUMENT WITH YOUR NAME:

1.) What does "InitializeSerialConsole()" do? In said function, what is "cbufRx" and "cbufTx"? What type of data structure is it?

2.) How are "cbufRx" and "cbufTx" initialized? Where is the library that defines them (please tell the *C file they come from).

3.) Where are the character arrays where the RX and TX characters are being stored on at the end? Please mention their name and size. Tip: Please note cBufRx and cBufTx are structures.

4.) Where are the interrupts for UART character received and AURT character sent defined?

5.) What are the callback functions that are called when:

- A character is received? (RX)

- A character has been sent? (TX)

6.) Explain what is being done on each of these two callbacks and how do they relate to the cbufRx and cbufTx buffers.

7.) Draw a diagram that explain the program flow for UART reception – starting at the user typing a character and ending on how that characters ends up in the circular buffer "cbufRx". Please make reference to specific functions in the starter code.

8.) Draw a diagram that explain the program flow for the UART transmission – Starting from a string added by the program to the circular buffer "cbufTx" and ending on characters being shown on the screen of a PC (On Teraterm, for example). Please make reference to specific functions in the starter code.

9.) What is done on the function "startStasks()" in main.c? How many threads are started?

2.) Programming Warm-Up: Debug Logger Module [50 points]

What is a Debug Logger Module:

The Debug Logger Module will be C calls that we can later use in the development of our IoT products to help us debug as well as indicate any system information. They are useful for the programmer to write human-readable strings to a terminal to determine the state of a device. It is the same idea as when we use "print" statements on code running on a PC to print statements about the program.

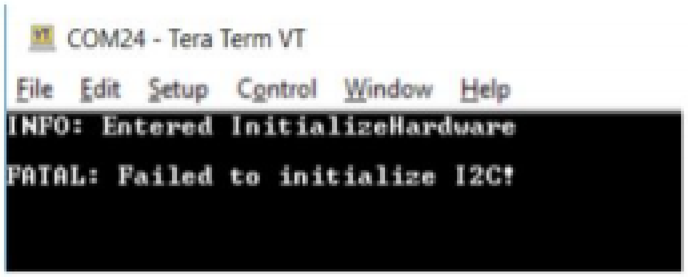
For example, imagine you are implementing a very complicated state machine along with some hardware initialization. With the use of the Logger Module, you could make calls that print to a terminal the current state of the program. Here is an example of how that can be useful:

```
//Attempts to initialize I2C Hardware void InitalizeHardware(void) {
int error = ERROR_NONE;

LogMessage(LOG_INFO_LVL, "Entered InitializeHardware\r\n"); //<Logs message to Terminal

error = InitI2c(void); //Initializes I2C devices if(error != ERROR_NONE) {
LogMessage(LOG_FATAL_LVL, "Failed to initialize I2C!\r\n"); //<Logs message to Terminal }else {
LogMessage(LOG_INFO_LVL, "Initialized I2C!\r\n"); //<Logs message to Terminal } }
```

An example output if the previous code failed would be the following:



```
INFO: Entered InitializeHardware

FATAL: Failed to initialize I2C!
```

Your Task: Implement the Debug Logger Module, that when called, prints the message to the terminal. The DebugLogger has these different levels of messages:

```
enum eDebugLogLevels {
            LOG_INFO_LVL = 0, //Logs an INFO + message
             LOG_DEBUG_LVL = 1, //Logs a DEBUG + message
            LOG_WARNING_LVL = 2, //Logs a WARNING + message
            LOG_ERROR_LVL = 3, //Logs an ERROR + message
             LOG_FATAL_LVL = 4, //Logs a FATAL + message (a non-recoverable error)
             LOG_OFF_LVL = 5, //Enum to indicate levels are off
             N_DEBUG_LEVELS = 6 //Max number of log levels
};
```

Our Debug Logger should be able to just print messages that are equal or above a set level given (and set) by the following functions:

```
eDebugLogLevels getLogLevel(void); //Returns the current logging level

void setLogLevel(eDebugLogLevels debugLevel) //Sets the Debug Logger level to the given level
```

Implement the following function, which has the following inputs:

```
void LogMessage(enum eDebugLogLevelslevel, const char *format, ...)
```

- ***enum eDebugLevels*:** Determines the log levels of the message to output. If the level is smaller than the current "logLevel" it is not printed.
    - **const char *format:** Pointer to a array of characters to be printed.
- **Variable List:** the "..." in C denotes a variable list. Please refer to https://www.cprogramming.com/tutorial/c/lesson17.html for more information. In this argument, we expect the variables that you would normally use in a vsprintf (please see example on https://www.tutorialspoint.com/c_standard_library/c_function_vsprintf.htm).

Usage example:

```
setLogLevel(LOG_ERROR_LVL); //Sets the Debug Logger to only allow messages with LOG_ERROR_LVL or higher to
be  printed

LogMessage(LOG_INFO_LVL, "Performing Temperature Test...\r\n", //This should NOT print , since INFO is lower
than  ERROR

LogMessage(LOG_FATAL_LVL,"Error! Temperature over %d Degrees!\r\n", sensorTemperature); //This should print
,  since FATAL is higher than ERROR

LogMessage(LOG_ERROR_LVL,"System error!\r\n", sensorTemperature); //This should print , since they are the
same  level
```

TIPS

***Check the function "SerialConsoleWriteString". You can use that to print to the TX buffer**

*** As said before, check** https://www.tutorialspoint.com/c_standard_library/c_function_vsprintf.htm to understand how to use vsprintf for the variable argument list.

3. Complete the CLI [100 points]

The starter code provides a CLI thread that uses the FREERTOS CLI. This is a code that, when given a terminated string, will execute a corresponding function. You can learn more about this code here:

https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_CLI/FreeRTOS_Plus_Command_Line_Interface.html

For us to be able to use this CLI, we need to finish two unfinished parts on the system:

- `static void FreeRTOS_read(char* character)` on CliThread.c: Please review the CLI thread () to better understand where this function is called. This function should block the thread until there is a character in the reception buffer. If there is a character in the reception buffer, it returns the value in the given argument pointer.

- `void usart_read_callback(struct usart_module *const usart_module)` on SerialConsole.c: This callback is activated when the UART receives a character. The students must write code that adds these characters into a ringbuffer for processing on the CLI thread. The students should design how they will indicate to the CLI thread that a character has been received. For this, please use any of the thread sync. API on https://www.freertos.org/a00113.html

  The special case of a backspace is dealt for you in the CLI thread.

For these two functions, feel free to make any other functions you need.

When you are done, you should be able to open a serial terminal and type "reset" and your device should reset

4. Add a CLI command [25 points]

Check how the command "`xClearScreen`" or "`xResetCommand`" are implemented on "CliThread.c". Looking at how those are implemented, implement the following CLI commands:

"version" → Prints a firmware version, such as "0.0.1". Make sure that the inputs can be modified by changing a #define at the beginning of a file.

A7 RUBRIC

Points	**Rubric**
25 points	**Q1 QUESTIONS on a word document**
75 points	Q2 Logger works
100 points	Q3 CLI works. Code and functions done are also commented using Doxygen (fill out comments on functions. If a new function is used, add a Doxygen block to it).
25 points	Q4 CLI works and “ver” command was added