Lab 3: UART, ADC and DMA

ECSE 444: Microprocessors

Winter 2020

In this lab, you will learn how to use more peripherals on the STM32L475VG MCU. Note that this lab is divided across two weeks, with demos each week.
In particular, you will:

WEEK 1

- Learn how to use UART to enable communication with a computer.
- Use an ADC to obtain the temperature of the processor core.
- Use the Systick timer interrupt to read the temperature.
- Monitor the temperature in a desktop application.

WEEK 2

- Enable DMA on UART transmissions.
- Use DMA to send 10 temperature values at a time.

Changelog

- 1-Oct Initial release.

Grading

Week 1:

- 30 % UART displays information in correct format
- 40 % ADC temperature values are within range
- 30 % SysTick synchronizes UART at correct interval

Week 2:

- 100 % DMA transmission of UART data

Note: Each week is worth the marks of one lab.

UART

Universal Serial Receiver/Transmitter (UART) is a computer hardware device that enables serial communication with configurable data width and speed. In this lab, the UART interface is already initialized for you in the base-project.

In order to communicate with the board, you will need to download and install MobaXterm, software that supports serial communication (and much more!). Once downloaded, open the software and create a new session as shown in Figure 1.

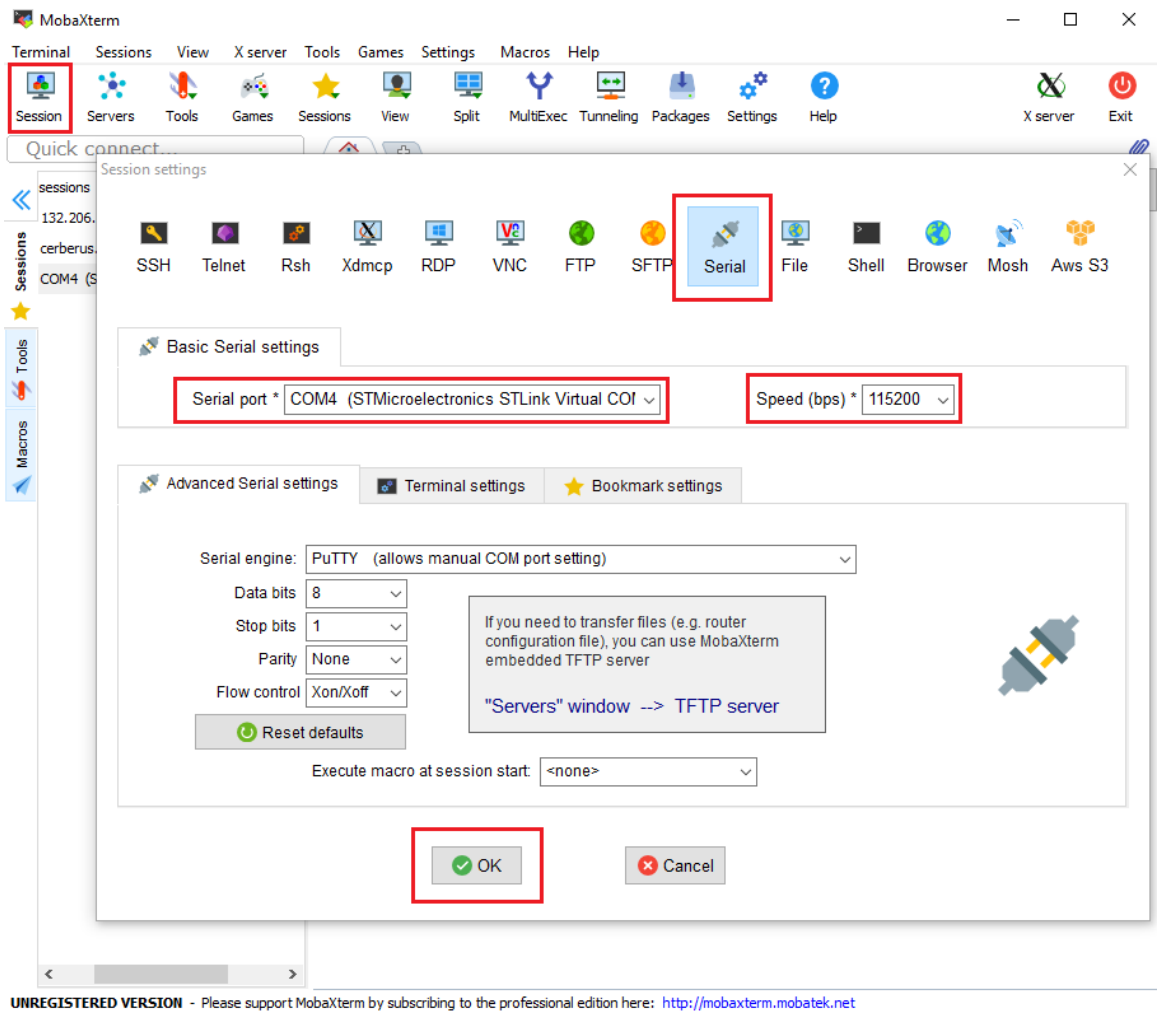

Figure 1: New serial connection in MobaXterm.

You will need to use the HAL driver functions *HAL_UART_Transmit()* and *HAL_UART_Receive()* to send and receive characters over the serial connection. Verify that you can transmit and receive by writing the character 'Y' whenever you receive the character 'X'. *Note: The code will need to run in an infinite loop!*

For this section, the task is to write a function, *UART_Print_String()*, that sends a series of characters over the UART connection. The function will take three arguments—a pointer to the UART handle (similar to the HAL drivers), a character array (a character pointer), and an integer which denotes the number of characters in the array—and return an integer with value '1' for success and '0' for failure.

ADC

The STM32L4 family of MCUs comes with three Analog to Digital Converters (ADCs), as well as a crude internal temperature sensor that is used to monitor the internal processor core temperature.

In this task, you will need to use the HAL drivers to initialize the correct ADC and provide the temperature sensor as the input channel. Using the *HAL_Delay()* function in an infinite loop, read the ADC value every 100 ms. You should obtain the ADC value via the **polling** mode (refer to the HAL driver documentation). Once the ADC value is obtained, use the conversion formula from the documentation to obtain the core temperature in degrees Celsius.

Finally, you will need to send the temperature value over the UART connection. Update the temperature value in a character array of the following format: **Temperature = 30 C**

The value 30 will constantly need to be updated with the current temperature. Note that you have to convert an integer into two characters! Send this array over the UART connection using the function created in the previous section. Note that there are 18 character in this array, and to display the next reading on a new line, you can send the newline character '\n'.

Note: To properly view new lines in MobaXterm, an option must be set. Right-click anywhere in the black terminal window of your active serial COM session, and click on the *Change terminal settings ...* option, and check the *Implicit CR in every LF* check-box.

SysTick

The SysTick timer is a core peripheral of the ARM Cortex-M4 family. It is a 24-bit hardware counter that counts down from a configurable load value. When the count reaches 0, a timeout event is generated and the counting begins from the load value again. By knowing the clock frequency of the counter, the load value can be set to create timeout events at desired intervals.

In the base-project provided, the SysTick timeout period has already been set to 1 ms via the *HAL_SYSTICK_Config()* function. Locate this function and verify that you understand how this is achieved. Additionally, the base-project has configured the core clock frequency to 80 MHz and has enabled the SysTick interrupt via the Nested Vector Interrupt Controller (NVIC). A SysTick timeout interrupt is handled in the *SysTick_Handler()* function in the **stm32l4xx_it.c** source file.

In this task, you need replace *HAL_Delay()* and instead use SysTick interrupts to poll the ADC value every 100 ms (10 Hz). To achieve this, you will need to change the SysTick timeout value. Do not write too much code in the *SysTick_Handler()* function. Instead, simply set a software flag (an integer will do) to '1'. This flag will need to be declared as a global variable in **main.c** and as an *extern* global variable in **stm32l4xx_it.c**. In your infinite loop, continuously check if the flag has been set to '1', and if so, immediately set it back to '0' and perform your temperature reading and UART display.

DMA

In this section, you will be introduced to the concept of Direct Memory Access (DMA) by using it to perform the UART transmission. DMA, as the name suggests, enables peripherals to read/write directly to/from memory, thereby removing the burden from the software.

In the first section, you built a function that transmits a character array via UART, in which the

character array is passed via software to the *HAL_UART_Transmit()* function. There are two potential drawbacks to this implementation. First, if the array is large, the function will take a long time to complete, therefore blocking other parts of the code from executing. Second, if this function is called very often, it again takes up valuable program time that could have been used for other tasks.

By enabling DMA on the UART Transmit interface and passing the starting address of the character array and the number of elements, the UART peripheral is now able to read the entire contents directly from memory. The transmission thus only needs to be started in software, following which the program moves on to other tasks while the transmission occurs in the background.

In this task, you are to enable DMA on the UART Transmit interface. Carefully follow the procedure outlined in the HAL driver manual for both the UART and the DMA. Your application should read the temperature every 100 ms using SysTick interrupts, exactly like the previous section. However, instead of updating an array and immediately transmitting it over UART, you should store the temperature values as characters in an array of size 30. For each reading, store 3 characters - the two digits of the temperature and the newline character. Therefore, you will be able to store 10 readings in this array, which will take about 1 second to acquire. After you have finished storing all 10 readings (keep track using a software counter), start the UART DMA transmission and starting acquiring the next 10 readings.

Note for Demo:
Students should demo on their registered time slots. Those demoing on the first lab section should have sampling periods of 100 ms and 10-bits ADC resolution. Groups on the second lab section must have 25 samples per second (instead of 20) and 12-bits of ADC resolution. We are expecting to read a temperature value within 25 to 45 degrees.