|  |  |
| --- | --- |
| **Mark** | **/11** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A5* | | |
| Homework number: | *HOMEWORK 06* | | |
| Due date: | 27/10/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Alessio Spineto |  |  | *x* |
| Riccardo Lamarca | *x* |  |  |
| Sofia Cecchetto |  |  | *x* |
| Annamaria De Togni |  |  | *x* |
| Emma Crespi |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name |  | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  |  |
| **Project 3a**: ADC scan using DMA  We enabled the potentiometer (PA1) analog input channel as ADC1\_IN1. This automatically flags the corresponding ADC channel in *ADC1 Mode and Configuration* menu.  We also enabled the internal channels for the temperature sensor and Vref from the same interface.    We set the ADC acquisition and conversion to be triggered by TIM2 every second, by setting the prescaler to 8399 and the ARR to 9999.    Since we enabled 3 different channels, we set *Number of Conversion* of the ADC to 3 and set the sequence that the ADC must follow (first *Channel 1*, then *Channel Temperature Sensor*, and lastly *Channel Vrefint*). This automatically enables *Scan Conversion Mode*. We enabled DMA continuous requests.      We enabled a DMA request in circular mode with data width set to 16 bit, to make sure that data is correctly stored. We enabled the flag for increment address of the memory so that each value is saved in consequent elements of the buffer.    We enabled the global interrupt both for the ADC and the USART. We used USART in DMA mode with transfer mode USART2\_TX and high priority.    Then we need to initialize the ADC and the timer. In the ADC initialization we need to pass the pointer to where we want to store the data (*adc\_dma\_result*) and the number of elements saved in the buffer (*adc\_channel\_count*).      When the conversion is complete, the ADC will call HAL\_ADC\_ConvCpltCallback(). In this function we converted the values read by the ADC in the different channels to voltages. The temperature sensor voltage is then converted to °C .  To convert the Voltage to temperature we used this formula and these values:    Consequently, we printed all values to serial using *HAL\_UART\_Transmit\_DMA*.  This is a screenshot of the output that we obtained. | | | |
| **Project 3b:** Light Dependent Resistor  First, we located the LDR pin on the schematic, PA0, and enabled it as ADC1\_IN0 in the GUI.    This, automatically flags IN0 in the ADC1 *Mode and Configuration* tab.    We proceeded by configuring TIM2 as to trigger the ADC conversion every 1 ms.      Then, we set TIM2 to trigger the ADC conversion and enabled DMA Continuous Requests.    Then, in the *Analog* section, we also added a new DMA request for ADC1, enabling circular mode and a data width of half word. In this way, data will be correctly saved in a 16 bit dimension.    To allow the transmission of data to the remote terminal, in the *Connectivity* section we enabled a new DMA request, selecting USART2\_TX.    Then, in the NVIC tab, we made sure to enable the ADC1 interrupt, as well as the USART2 one.    Before starting with the coding part, we needed to include the <math.h> and <string.h> libraries, that will be useful later.      Then, we initialized the ADC and the timer. Like in the previous exercise, we store the acquired values in the array *adc\_dma\_result.* The number of values stored in the buffer is 1000, as set with the global variable SIZE.    Since each conversion is performed at a rate of 1ms, at the end of the 1000 conversions *adc\_dma\_result* will be full and 1s will be elapsed. At this point we will be able to transmit the averaged value to the remote terminal and the ADC will automatically start another series of 1000 conversions.  We used the following variables:   * *adc\_dma\_result*[SIZE] is the array of dimension 1000 where we store all the values acquired by the ADC. * *first*[SIZE/2] is an array of dimension 500 where we will store the first half of the values converted by the ADC * *second*[SIZE/2] is an array of dimension 500 where we will store the second half of the values converted by the ADC. This operation is possible thanks to the function *HAL\_ADC\_ConvCpltCallback()*   *HAL\_ADC\_ConvHalfCpltCallback()* is called each time the buffer dedicated to acquisition of data is half full. In our case, will be called after 500 acquired values, and so after 500 ms. These values will be stored inside the array *first*.  *HAL\_ADC\_ConvCmpltCallback()* is called when the *adc\_dma\_result* array is full. The last 500 values will be stored in the array *second*.  At the end of this operation, we can call the *convert()* function.    Inside the *convert*() function we computed the mean value and the conversion of the values acquired by the ADC, turning them into the corresponding resistance using the formula:  Finally, we computed the illuminance:  We then sent the final value to a remote terminal with the function HAL\_UART\_Transmit\_DMA.    This is the output when directing a phone flashlight to the LDR.    **Notes for the Professor:**  We noticed that in both projects the string sent to the remote terminal is not received exactly every second, but it seems to arrive a few milliseconds (about 10 to 20) earlier on every callback.  Below is a screenshot of the remote terminal at the start of execution and after a minute (for project 3a):    We noticed that this also happens using UART transmission in DMA mode in the interrupt of a timer set at a frequency of 1Hz, and we were not able to solve this issue. | | | |
|  | | | |