|  |  |  |  |
| --- | --- | --- | --- |
| Project name | ADC scan using DMA with three channels and with LDR | | |
| Not done | Partially done   (major problems) | Partially done   (minor problems) | Completed |
|  |  |  | *x* |

|  |
| --- |
| We have done the whole homework, here is our explanation:  **Project 3a:**  Firstly,we have configured the board as shown below:  Which is actually identical to the projects 2b and 2c.  Secondly,we have enabled the three channels de ADC as shown below:  And then we configured the ADC as shown below:  Where we have set the “DMA Continuous Requests” to Enabled, “Number of Conversion “ to 3 as we have to measure three channels in this project, “External Trigger Conversion Source” to Timer 2 Triggrt Out event as we will use the timer2 to control the measureing frequency, and in these 3 Rank section, set their channel to Channel 1, Channel Temperature Sensor, Channel Vrefint respectively and each Sampling Time to 480 Cycles in order to give the capacitance enough time to charge/discharge.  Actually,before set the “DMA Continuous Requests” to Enabled, we first need to go to DMA settings and add a DMA request as shown below.otherwise you would find that you can’t set the “DMA Continuous Requests” to Enabled:  Where we have set the Mode to Circular because in this project we need to sample those three datas every s, and in order to save those samples, we have created a array of size 3, with mode Circular, once the DMA reaches the end of the data buffer, it will automatically wrap around to the beginning of the buffer(in our case, the array) and continue the data transfer. And if we use the mode Normal, when the DMA saved all the datas in the first iteration, the DMA will stop transmit data to the memory if we do not restart the DMA, which means this mode will make our implementation much more complicated, therefore we choose the Mode Circular. And we have also set the Data Width to Half Word(16 bits) as our ADC’s sample has a size of 12bits(and also our array is a array of uint16\_t), thus in order to optimize the efficiency, we have set it to Half Words.  Then, we have configured the Timer2 in order to make the ADC sampling frecuency to 1Hz:  As you can see, the configuration is exactly the same as project2b and 2c.  After doing that, we went to the NVIC table:  Where we have enabled manually ADC1 global interrupt.  Until this point, we have done all the configuration on the GUI, then we started programming in the main.c:  we firstly created a global array of uint16\_t in order to save the samplings:  In the main, we have start both Timer2 and ADC in a safe way:  Then, we have implemented the callback function:  In this function,we just transforms the data offered by the ADC to the appropriate form and send them using UART2. The temperature fucntion is from the slide:  And the code works as we expected:  We have noticed that the Ref is exactly 1.21V as we expected.  And we can compare this temperature to the temperature of the thermometer in the room of Hui: Imagen de la pantalla de un celular con letras  Descripción generada automáticamente con confianza baja  Note: Maybe you have noticied that the exercise said that the acquisition is started by software, and actually this is our first version:  Which requires the HAL\_Delay function which is a operation kind of dangerous, so we decided to use timer(the version we have showed so far.)  **Project 3b:**  Firstly, we have configured the board as shown below:    Where we have set the PA0 which is the pin connected to LDR to ADC1\_IN0.  Here is the procedure to find the pin connected to LDR:  In green board schematics.pdf we have find that LDR is connected to 14:  Then we went to the nucleo schematics.pdf:  We found 14 is connected to PA0.  Then we have configured the ADC1 exactly the same as the previous project but in this project we just enabled the channel IN0:  And then we have set the timer2 as shown below:  Where we set the Prescaler to 0 and Period to 83999 as (0+1)\*(83999+1)/83e6 = 1e-3 s = 1ms which is the required sampling frequency. Other settings are exactly the same as the previous projects(2b 2c and 3a).  After doing that, we went to the NVIC table in order to enable the following interrupt:  Until this point, we have done all the configurations on the GUI, then we went to the main.c, we first define a macro and a array to save the samples:  You may ask why the array has a size of 2000 instead of 1000(sampling frequency 1000Hz, send the average every 1s, therefore the size 1000 is sufficient). It is due to the method that the professor provided during the lecture, we will explain it when we get the appropriate point.  In the main, we just start both the timer2 and ADC in a safe way:  Then we created two callback functions, and both of them do the exactly the same thing:  In our project, we encountered a scenario that required the processing of vast amounts of data. To understand the challenges, let's consider an example where the array size is set to 1000. In this scenario, once the DMA completes filling the array, the callback function is triggered to process the data. However, while the CPU processes this data, the DMA is halted, preventing it from placing new data into the array. This leads to inefficiency.To address this, we adopted a strategy suggested by our professor. We split the array into two halves. As soon as the DMA fills the first half, the first callback function is invoked to process this half. Concurrently, the DMA starts filling the second half. Once the entire array is filled, the second callback function is triggered to process the latter half. Meanwhile, the DMA begins refilling the first half. This approach eliminates the waiting time seen in the 1000-sized array scenario, thus significantly boosting efficiency. And the way we calculate the LDR and LUX is from the slide:  Finally, the code works as we expected: |
| Professor comments: |