|  |  |
| --- | --- |
| **Mark** | **A** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *B5* | | |
| Homework number: | *10* | | |
| Due date: | 17/12/2023 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Ghidini Alessandro |  |  | *x* |
| Latino Francesco |  |  | *x* |
| Luppi Eleonora |  |  | *x* |
| Bravin Riccardo |  |  | *x* |
| Feltrin Elia |  |  | *x* |
| Notes: | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | IR receiver | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  | *x* |
| **Part 1a: Transmit board**  In the graphical interface three timers have been set:  TIMER 1 at 2400Hz: Prescaler at 70-1, autoreload register at 500-1. This timer is used for transmission.  TIMER 2 Channel 3, 38251Hz: Prescaler at 35, autoreload register at 60. The third channel of this timer is configured to generate a PWM with a duty cycle of 50% (Pulse = 30), which is used as an oscillator for the IR transmitter.  TIMER 3 at 100Hz: Prescaler at 8399, autoreload at 99. This timer is used to read the keyboard  Additionally, global interrupts for timers 1, 2, and 3 have been enabled in the NVIC section.  A queue has been implemented with the classic functions init\_queue, isFull, isEmpty, enqueue, dequeue, and front, useful for transmission in case the provided inputs saturate the transmission speed. Although it's unlikely to happen in this case, we implemented this solution with the perspective of easier implementation for the next project.  In the main function, the queue is initialized, and HAL\_TIM\_Base\_Start\_IT(&htim3) is called to start the timer with the callbacks handling the keyboard reading.  The function void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim) has been overridden, managing interrupts for timers 1 and 3. Regarding timer 1, necessary variables are declared:   * If we are at the beginning of transmission, the first byte is dequeued from the queue, and the next bit to transmit is set to 0. * If we are transmitting the byte's content, the next bit to transmit is extracted, and the byte is shifted left for the next extraction and transmission. Additionally, the parity bit is updated. * If the entire byte has been transmitted, the next bit to transmit will be the parity bit. * If the entire byte and parity bit have been transmitted, then the next bit to send will be the stop bit.   Now, timer 2 channel 3 is handled, that acts as an oscillator for the IR transmitter:   * If the bit to send is 1 and PWM is started, then PWM is turned off * If the bit to send is 0 and PWM is off, then PWM is started   according to the transmitting protocol.  Finally, the variables are reset for the next byte if the transmission is completed. If the queue is empty, the timer 1 and, consequently, the entire transmission is stopped.  Regarding timer 3, at each interrupt, a different column of the keyboard is read, as done in previous homework. The only difference is that when a key is read, the transmit function is called with the corresponding symbol on the keyboard.  The transmit function simply inserts the next byte to be transmitted into the transmission queue and starts timer 1 if it's off, as there is no ongoing communication.  **Part1a: Receive board**  In our project, we opted to directly oversee the LED board illumination instead of implementing just the green led light up. To set things up via the graphical interface, we performed the following configurations:     * TIM 10 was configured with a prescaler of 8400-1 and a counter period of 39, utilizing relative interrupts to achieve a frequency of 250Hz. * TIM 11 was set with a prescaler of 8400-1 and a counter period of 5000-1, employing relative interrupts to attain a frequency of 2Hz. * SPI1 was configured with a prescaler set at 4, global interrupts enabled, and a DMA configured for transmission with default parameters. * USART1 was set with a baud rate of 2400 bit/s, even parity bit, and global interrupts. Additionally, PA10 and PA9 pins were configured as USART1\_RX and USART1\_TX, respectively, to facilitate proper reception.   Upon generating the code, we utilized a constant 3-dimensional tensor to store all potential character matrices for display. An ad-hoc implemented queue was employed to keep track of received inputs awaiting display. The critical section for IR reception was placed within the HAL\_UART\_RxCpltCallback, encompassing the initiation of the timer for managing letter changes, enqueueing the currently received letter to the display sequence, and restarting byte reception to continuously listen for new inputs with HAL\_UART\_Receive\_IT.  Within the HAL\_TIM\_PeriodElapsedCallback, LED matrix management was implemented. TIM10 controlled the illumination of subsequent LED columns, while TIM11 handled dequeuing letters for display, along with indexing the letter within the predefined tensor.    In the main function, the queue was initialized, TIM10 was started for LED matrix display, and the reception of the first byte was initiated.  **Part2: Receive and transmit**  To complete the project, we had to cobine the first two parts into one. The only necessary changes were done to ensure that the receiver of the same board did not receive the data being transmitted by itself. To avoid this problem before the transmission of the first start bit in the TIM1 HAL\_TIM\_PeriodElapsedCallback we disable the HUART1 and then reinitialize it once the transmission is completed. Other than that, all the code and setup from the graphical interface is unchanged. | | | |
| Professor comments:  “global interrupts for timers 1, 2, and 3 have” 🡪 why TIM2 interrupt?  Ok! | | | |