Complete the “yellow” tabs and delate the phrases in italics.  
You can duplicate the table “Project”, if more than one project are due for the homework.

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A1* | | |
| Homework number: | *11* | | |
| Due date: | *18/12* | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| 1 *Giorgio Donato Carlo* |  |  | *x* |
| 2 *Lenzi Francesco* |  |  | *x* |
| 3 *Lodari Gianmarco* |  |  | *x* |
| 4 *Lanzini Alessio* |  |  | *x* |
| 5 *Chiapparo Lenn* |  |  | *x* |
| Notes:  *Complete in necessary* | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | IR project 1 - Preliminary | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Successfully completed |
|  |  |  | *x* |
| For this project we set TIM2\_CH3 to PWM mode 2 to generate a square wave with a 38kHz frequency and a 50% Duty Cycle; we also configured TIM3 to trigger with a 2400 Hz frequency, the frequency equivalent to the baud rate we decided to use for the infrared communication (2400bps). We connected the IR LED pin PB10 to TIM2\_CH3 and the IR receiver pin PA10 to USART1\_RX. We enabled UART1 in asynchronous mode at 2400 bps and activated its interrupt line.  In the main.c we defined *void IRByteTransmit (char byte*): a function that takes a 8 bit variable, divides it into 8 separate bit by using a progressively shifted mask, and then proceed to transmit them in the following way: it first sends the start bit (‘0’), then it sequentially sends the 8 bit obtained from the conversion and the final bit (‘1’). The transmission frequency is regulated by a flag updated in HAL\_TIM\_PeriodelapsedCalback; every time TIM3 triggers, a flag is set to 1; if we want to send a bit at the established baud rate, we:   1. Set the flag to 0: 2. Activate the PWM if we want to transmit a ‘0’, stop it if we need to transmit a ‘1’ 3. Wait until the flag is set to 1 4. End the transmission   These operations are all executed by the IRByteTransmit function.  To send a string, we defined a similar function that, for each character of the string, calls IRByteTransmit.  In order to receive the bytes, we enable the receiving via uart1 with interrupt mode in the main; then in the HAL\_UART\_RxCpltCallback we immediately restart the receiving process, we save the received Data in a variable and finally proceed to transmit it to the PC via uart2. | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | IR project 1 - Preliminary | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Successfully completed |
|  |  |  | *x* |
| The objective of this project was to scan the Keyboard, send the pressed buttons characters via infrared, and then print the received characters on the led matrix.  In the. ioc interface, we set IR\_LED pin, IR\_RX pin, UART1, TIM3 and TIM2\_CH3 PWM exactly like in the previous project; additionally we configured the necessary pins for the keyboard, we set the SPI pins and we enabled SPI1 and his relative DMA\_TX in normal mode. We set both TIM10 and TIM4 to trigger every 4 ms and enabled their interrupts.  In the code part, we used the exact same logic as the previous project to handle the transmission of bytes. TIM4 is used to scan through the keyboard columns while TIM10 is for the LED matrix ones.  In HAL\_TIM\_PeriodElapsedCallback we set to ‘1’ the elapsedColumnTimer flag when TIM4 triggers; when TIM10 triggers we perform a HAL\_SPI\_Transmit\_DMA to change the enabled column of the led matrix.  In the main() we initialize the keyboards pin, we start TIM10 in interrupt mode, we enable receiving through uart1 in interrupt mode,.  In the while(1) loop we handle the logic of detecting if buttons are pressed: after checking if elapsedColumnTimer is set to ‘1’, when a pressure is detected we call IRByteTransmit with the button position (uint8\_t) as parameter. Then the column index is updated and elapasedColumnTimer is set to 0.  In order to avoid bouncing we set a debounce time equal to 1 column cycle (4\*4ms = 16 ms); that means that to be detected as pressed, we must keep down a button for at least 16 ms.  To print the letters, we defined a simple struct composed by two uint8\_t variables, representing the row and column value to be transmitted. Then to define one letter we use an array of five of these structs, one for each column. In the main, every character is manually inserted into the vector ‘led\_map’ in the following way: the character ‘i’ will be in position ‘i’, so that led\_map[5] contains the information to show a ‘5’ on the led matrix.  In the HAL\_UART\_RxCpltCallback, we save the received data and we change the index of the character that will be shown. | | | |
| Professor comments: | | | |